Харківський національний університет радіоелектроніки

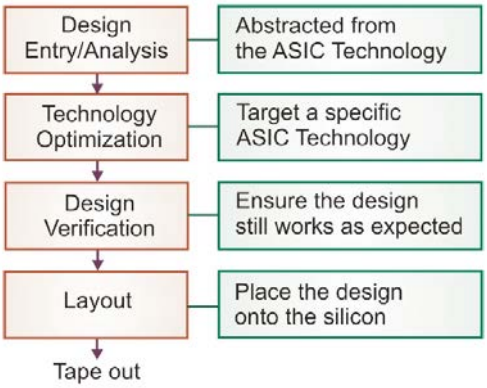
Національний університет "Львівська політехніка"

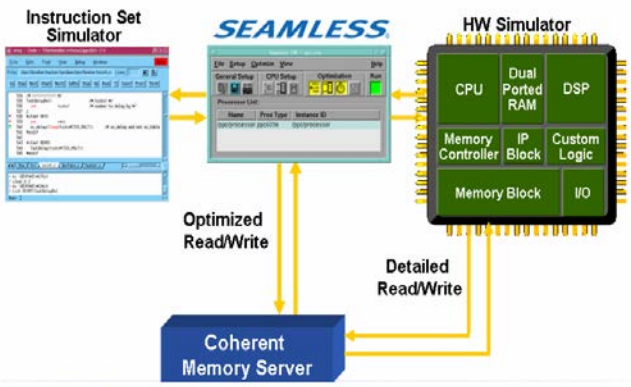
В. Хаханов, Є. Литвинова, О. Матвійків, С. Чумаченко

Технології тестування мікросистем

Підручник

Рекомендовано Міністерством освіти і науки України





№530785-TEMPUS-1-2012-1-PL-TEMPUS-JPCR

Tempus

Навчальний посібник "**Технології тестування мікроситем"** створено для допомоги вищим навчальним закладам України впровадити нову магістерську навчальну програму "Проектування мікросистем".

Посібник "**Технології тестування мікроситем"** створено при підтримці Європейського Союзу за Спільним Європейським Проектом "Curricula Development for New Specialization: Master of Engineering in Microsystems Design" (MastMST), ідентифікаційний номер 530785-TEMPUS-1-2012-1-PL-TEMPUS-JPCR.

Координатор проекту проф. Збігнєв Лісік, Технічний університет м.Лодзь, Польща.

Учасники проекту:

- Національний університет "Львівська політехніка", м. Львів, Україна, координатор проф. Михайло Лобур.
- Київський Національний університет ім. Тараса Шевченка,м. Київ, Україна, координатор проф. Валерій Скришевський.
- Харківський національний університет радіоелектроніки, м. Харків, Україна, координатор проф. Володимир Хаханов.
- Донецький національний технічний університет, м. Красноармійськ, Україна, координатор проф. Володимир Святний.
- Технічний університет м.Ільменау, Німеччина, координатор проф. Іво Рангелов.
- Ліонський Національний інститут прикладних наук, Франція, координатор проф. Александра Апотолюк
- Університет Павії, м.Павії, Італія, координатор проф. Паоло Ді Барба

Посібник схвалено редакційним комітетом (проф. Паоло Ді Барба (Університет Павії) - співголова, проф. Александра Апостолюк(Ліонський Національний інститут прикладних наук) – співголова, члени: проф. Збігнєв Лісік(Технічний університет м.Лодзь), д-р Яцек Подгурські (Технічний університет м.Лодзь), Д-р Януш Возний (Технічний університет м.Лодзь), Д-р Валентин Іщук (Технічний університет м.Ільменау), Д-р Марія-Евеліна Могначі (Університет Павії), Д-р Роберто Галді (Університет Павії)) 6 травня 2016, м. Павія, Італія

Автори висловлюють глибоку вдячність керівництву вищеназваних університетів за всебічну підтримку Проекту.

Textbook "**Microsystems Testing Techniques**" developed to help higher education institutions in Ukraine to introduce new master's educational program "Designing microsystems".

Textbook "**Microsystems Testing Techniques**" was created with the support of the European Union within the Joint European Project "Curricula Development for New Specialization: Master of Engineering in Microsystems Design" (MastMST), identification number 530785-TEMPUS-1-2012-1-PL-TEMPUS-JPCR.

Project Coordinator prof. Zbigniew Lisik, Lodz University of Technology, Lodz, Poland.

Учасники проекту:

- Lviv Politechnical National University, Lviv, Ukraine ,
  Coordinator prof. Mykhailo Lobur.
- Taras Shevchenko National University of Kyiv, Ukraine,
  Coordinator prof. Valeriy Skryshevsky.
- Kharkiv National University of Radioelectronics, Ukraine ,
  Coordinator prof. Vladimir Hahanov.
- Donetsk National Technical University, Krasnoarmiysk,
  Coordinator prof. Volodymyr Sviatny.
- Ilmenau University of Technology, Germany,
  Coordinator prof. Ivo Rangelow.
- Lyon Institute of Applied Sciences, France,
  Coordinator prof. Alexandra Apostoluk.
- University of Pavia, Italy,
  Coordinator prof. Paolo Di Barba.

The Handbook was approved by Editorial Committee (prof. Paolo Di Barba (University of Pavia) - Co-Chair, prof. Alexandra Apostoluk (Lyon Institute of Applied Sciences) – Co-Chair, members: prof. Zbigniew Lisik (Lodz University of Technology), Dr Jacek Podgorski (Lodz University of Technology), Dr Janusz Wozny (Lodz University of Technology), Dr Valentyn lshchuk (Ilmenau University of Technology), Dr Maria Evelina Mognaschi (University of Pavia), Dr Roberto Galdi (University of Pavia) May 6, 2016, Pavia, Italy.

The authors express their deep gratitude to the aforementioned universities for full support of the project.

ВСТУП

Необхідність видання книги обумовлена підвищеним інтересом з боку Hardware індустрії та студентської аудиторії до розробки енергозберігаючої, компактної і спеціалізованої цифрової апаратури на основі SoC і SiP. Часові витрати для вирішення такого завдання вже практично дорівнюють періоду створення аналогічного програмного продукту. В останні роки розходження time-to-market для Software і Hardware Design зникає завдяки технологічним досягненням у наноелектрониці силіконових кристалів і автоматизації процесів проектування виробу на всіх його стадіях (Gate level, RTL, System level). Як наслідок, комп'ютерна і програмна інженерія усе більше інтегруються на основі взаємних інтересів, пов'язаних зі створенням компонентів SoC і SiP, і взаємно збагачуються своїми методологіями, утворюючи Software/Hardware Codesign або Electronic System Level (ESL) Design. Реалізація ідеї – у залізо або програму – залежить від її ринкової привабливості, що формує метрику проекту: швидкодія або гнучкість, спеціалізація або універсалізація, мініатюрні або метрові розміри цифрової системи, високе або низьке енергоспоживання. Вибір очевидний. Проте, існує певний психологічний бар'єр і упереджене відношення до апаратної реалізації ідеї. Із часу виникнення комп'ютера співвідношення необхідного числа фахівців з апаратури й програм підкоряється закону 20/80. Чому тільки 20%? Тому що це складно для інших 80%. Тому що досконалість і надійність апаратури – це 80% успішності проекту. Тому що співвідношення помилок апаратної й програмної частин - не більше 20/80.

Мета видання - підштовхнути читача до думки, що при наявності привабливої для ринку електронних технологій ідеї кожний зможе реалізувати свій спеціалізований і ефективний комп'ютер (Hardware) за час (півроку), що необхідний для програмного (Software) розв язання завдання. Для досягнення мети існують мови опису апаратури, у тому числі й системного рівня: VHDL і Verilog, System C і SystemVerilog. Вони не складніше Software мови програмування C++ і багато в чому з нею подібні, але мають і позитивні відмінності. Головне достоїнство - мови опису апаратури орієнтовані на істотне зменшення часу верифікації проекту, паралелізм і високу швидкодію розв язку проблеми.

Відмінність даної книги від існуючих видань: 1) орієнтація на новітні технології Design & Test для нових конструктивів SoC і SiP; 2) порівняння й аналіз можливостей технологій сервісного обслуговування цифрових систем на кристалах з постановкою актуальних для ринку електронних технологій проблем; 3) стислість і структурованість викладу матеріалу з ненав'язливим впровадженням у текст англомовних термінів; 4) різноманіття прикладів синтезу й аналізу реальних цифрових пристроїв; 5) використання кращих закордонних джерел з проектування й тестування цифрових систем, а також досвіду розробки Hardware Design and Test у Харківському національному університеті радіоелектроніки.

Сутність видання. На тлі збільшення вентильної потужності силіконового кристала, що становить до 700 млн. транзисторів, показати шляхи системного вирішення проблеми створення інфраструктури убудованого сервісного обслуговування функціональних модулів з використанням IEEE стандартів тестопридатного проектування та ESL Design технологій. Системний підхід означає використання мов опису апаратури: VHDL, Verilog, System C, System Verilog, UML, PSL, SVA, OVA; системного програмування: C++, Assembler, C; Web-Дизайну – HTML, Flash, Java, .Net; Internet програмування: PHP, ASP, JSP, Java, .Net, C#; програмування баз даних: C#, Java, .Net. Проте, проблема вибору моделей і методів тестування й сервісного обслуговування, які повинні бути за можливістю автоматично імплементовані у зазначені мови для створення Hardware/Software компонентів SoC і SiP, залишається відкритою. Також у книзі знайшли відбиття цікаві, з погляду авторів, методи синтезу тестів для компонентів і цифрових систем, орієнтовані на побудову аналітичних форм багатозначних таблиць несправностей, що використовуються для пошуку дефектів у складних виробах на кристалах.

Структура видання представлена 11 розділами, у яких відбиті нові технології сервісного обслуговування й адаптовані власні розробки у частині моделювання, синтезу тестів і діагностування цифрових систем у пакетах: 1) технологічні конструктиви для імплементації цифрових систем; 2) стан ринку електронних технологій; 3) основи дедуктивно-паралельного моделювання несправностей цифрових виробів; 4) моделі та методи створення інфраструктури сервісного обслуговування функціональностей SoC; 5) адаптація технологій тестування та діагностування для логічних компонентів SiP; 6) інноваційні технології тестопридатного проектування програмних і апаратних продуктів; 7-8) ієрархічні моделі тестування складних цифрових систем і виявлення дефектів; 9-10) двотактне кубічне зчислення й технології генерації тестів для структурно-функціональних моделей цифрових систем; 11) спеціальні методи синтезу й аналізу послідовностей, що перевіряють, для SoC компонентів.

Найбільш суттєвими й актуальними, з погляду авторів, завданнями в області Hardware/Software Design & Test є: 1) класифікація конструктивних особливостей SiP (SoC); 2) створення ієрархічної інфраструктури сервісного обслуговування цифрових систем у пакетах на основі IEEE стандартів тестопридатного проектування; 3) розробка убудованих технологій реального часу для тестування, моделювання, діагностування й відновлення працездатності кристалів; 4) створення методів діагностування й ремонту компонентів SiP: пам'ять, логіка, програмні й керуючі модулі.

1. ТЕХНОЛОГІЧНІ КОНСТРУКТИВИ ДЛЯ ІМПЛЕМЕНТАЦІЇ ЦИФРОВИХ СИСТЕМ

1.1. Стадії та засоби імплементації ідеї

Основна мета Hardware/Software Design and Test – надати широкому колу користувачів технології створення мініатюрних, спеціалізованих, економічних, мобільних і швидкодіючих цифрових виробів, затребуваних у світі. Тут можна заробити реальні гроші саме шляхом створення власних технологій, не прив'язаних до desk top, ефективність використання яких навряд чи більше 10%. Впровадження або просування нових засобів на ринку електроніки починається з апробації будь-якого методу або алгоритму в програмному виконанні. Після досить повної верифікації проект може бути виставлений на ринку у вигляді IP-core, що гарантує авторські права розроблювача.

Наступним етапом може бути імплементація IP-core у кристал програмувальної логіки, що дозволяє істотно підвищити швидкодію, залишаючи можливість перепрограмування окремих функцій у процесі доробки, виявлення й виправлення помилок. Винятковість технологічного розв язку й ринкова привабливість є критерієм для його імплементації в кристал ASIC. У цьому випадку швидкодія реалізованої системи стає максимальною, але тут вже неможливо виправляти помилки, тому проект повинен бути ідеально коректним. Таким чином, просування нової технології на ринку пов'язане з еволюцією проекту від Software до Hardware (Hardwaring), що стає менш гнучким для модифікації, але більш продуктивним для розв язання проблеми.

Поряд з Hardwaring-процесом існує й зворотна тенденція до створення конструктивних ASIC шляхом їхнього інтегрування із кристалами PLD. Це дає можливість вносити зміни або перепрограмувати вже готовий виріб, імплементований у кристал, що стає гнучким стосовно модифікації проекту й швидкодіючим, оскільки основні функції реалізуються в ASIC. Що стосується ринку CPU як основної частини ASIC, тут відслідковується три рівні ієрархії кристалів: 1) потужні мікропроцесори, що використовуються у серверах і робочих станціях; 2) персональні комп'ютери, де домінують мікропроцесори фірми Intel; 3) мікроконтролери й сигнальні процесори. Перші використовуються для редагування цифрових відеофайлів і speech-to-text трансляторів. Але вже в найближчому майбутньому такі завдання будуть вирішуватися спеціалізованими обчислювальними процесорами. Слід відмітити, що на ринку мікроелектроніки зростає показник спеціалізованої орієнтації на вирішення конкретних завдань – customization. Коли можливості починають перевищувати потреби даної області ринку, конкуренція зміщує акценти з підвищення загальної продуктивності на вдосконалення властивостей, за які користувач готовий платити додаткові гроші. Це – не тільки надійність і зручність, але й швидкодія, замовлені

властивості, енергозбереження, автономність, мініатюризація й можливість перепрограмування. У частині енергоспоживання фірма Intel орієнтує наше майбутнє на багатоядерні архітектури, які значно економичніші за одноядерні процесори. Якщо врахувати, що споживана потужність пропорційна квадрату тактової частоти процесора, то її зменшення у 2 рази з одночасним подвоєнням числа процесорів не знижує швидкодії комп'ютера, при цьому енергетичні витрати практично зменшуються в 2 рази. Площа кристала при таких змінах істотно не збільшується. Проте, процесори із чотирма або більше ядрами складають лише близько 10% процесорів, установлених у комп'ютерних і комунікаційних системах. Одноядерні процесори до цього часу будуть усе ще дуже популярні у вбудованих системах, які займають приблизно половину ринку. Унікальна гетерогенна архітектура PowerQuicc (Freescale Semiconductor) поступово поступиться місцем більше простій двоядерній архітектурі, що претендує на 20% ринку. Причина низьких темпів переходу до багатоядерних процесорів полягає в орієнтації сучасного програмного забезпечення на одноядерні процесори, а також у незадовільному розв язанні проблеми масштабованості операційних систем для симетричних багатопроцесорних архітектур. Крім того, багатоядерні процесори мають власні бар'єри на шляху збільшення продуктивності, пов'язані з вузькими місцями, такими як центральна шина, спільно використовувана кеш-пам'ять і загальний інтерфейс із пам'яттю.

Що стосується кристалів ПЛІС, то вони надають розроблювачам нові можливості для проектування швидкодіючих спеціалізованих убудованих цифрових систем. Доказом перспективності програмувальної логіки служить поява нових, що мають більш досконалу архітектуру, поколінь ПЛІС із убудованими процесорами й зростання обсягу їхнього випуску провідними закордонними виробниками. Аналіз тенденції розвитку архітектур програмувальної логіки дає підстави затверджувати, що найближчим часом основу елементної бази цифрових (реконфігуруємих) систем будуть становити CPLD, FPGA, HardCopy, що доганяють за продуктивністю й вартості ASIC, але не потребуючі витратної тестової та верифікаційної методології для забезпечення масового виробництва мікросхем.

На ринку електронних технологій існує причинно-наслідковий зв'язок між трьома основними компонентами, що впливають на комп'ютерну продукцію (рис. 1.1). Це, насамперед – технологія виготовлення силіконових пластин для кристалів, типи пропонованих на ринку чипів (мікросхем) або кристалів, мови опису цифрової й аналогової апаратури та програмних продуктів, EDA-засоби для проектування систем і мереж на кристалах.

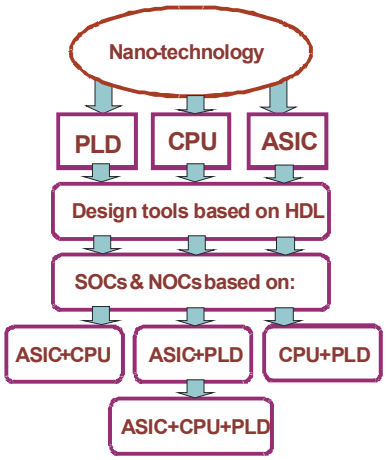


Рис. 1.1. Технології інтеграції для систем на кристалах

Враховуючи, що кожний із трьох основних типів кристалів (PLD, ASIC, CPU) має недоліки, виробники намагаються зменшити їхній вплив, об'єднавши переваги. Так з'являються на ринку цікаві комбінації: {PLD + ASIC} - Lucent Technologies (OR3TP12), Quick Logic (RAM, PCI, DSP), Cyprus Semicomductor (PCI), Altera (Stratix), Xilinx (Virtex® II-VI, Spartan® 4-6). Далі інтегруються такі компоненти як {PLD + CPU} - Atmel (FPSLIC), Trincend (E5, A7), Xilinx (Virtex II Pro), Altera (Excalibur), {ASIC + CPU} - мікроконтролери, {PLD + ASIC + CPU}, які інтегрують швидкодію ASIC, перепрограмуємість PLD, гнучкість і керованість CPU. Приклад, що ілюструє останні досягнення функціональних можливостей PLD, наводиться у табл.1.1, яка надає палітру інструментів сучасним розроблювачам [1].

Таблиця 1.1

Характеристики кристалів сімейства Spartan®-6

Кристал FPGA XC	Осередки логіки	Логічні блоки		DSP 48A1	Блоки RAM		Розмір/Крок виводів, мм
		Slices	RAM, Kb		Kb	Max	
6SLX4	3 366	526	32	4	8	144	20x20/0,5
6SLX25	23 770	3 714	228	38	52	936	13x13/0,8
6SLX150	147 456	23 040	1 358	182	268	4 824	23x23/1,0
6SLX25T	23 770	3 714	228	38	52	936	23x23/1,0
6SLX150T	147 456	23 040	1 358	182	268	4 824	23x23/1,0

Сімейство кристалів Spartan®-6 оптимізовано для їхнього застосування у виробах масового виробництва й забезпечує широкі можливості при відносно низької вартості мікросхем. До нього входять дві підродини, що включають 11 кристалів щільністю від 3 400 до 148 000 логічних осередків зі зменшеним у два рази енергоспоживанням, високошвидкісними та більш гнучкими трасувальними ресурсами. Підродина LX орієнтована для виконання логічних функцій, LXT – для реалізації високошвидкісних

послідовних інтерфейсів. Кристали сімейства Spartan®-6 виконані за відпрацьованою 45-нм КМОП технологією, що дозволяє встановити оптимальний баланс між вартістю, потужністю й продуктивністю. У них використовуються нові, ефективні логічні осередки із шостивходовими таблицями перетворення (Look-up Tables – LUTs) і двома тригерами. Логічні осередки поєднуються парами в секції (slices). Кожний конфігуруємий логічний блок CLB містить 4 таблиці LUT і 8 тригерів (2 секції). Є великий вибір убудованих блоків системного рівня (апаратних ядер). Block RAM має пам'ять ємністю 18 кбіт, що може бути зконфігурований як два блоки по 9 кбіт. Друге покоління блоку цифрової обробки сигналів DSP48A1 складається з 18x18 помножувачів, 48-бітового акумулятора, передсуматора й регістрів для конвеєризації, які можуть функціонувати із частотою 250 Мгц. SDRAM контролери пам'яті забезпечують смугу пропущення до 12,8 Гбіт/с. Є поліпшений блок керування та синтезу сигналів синхронізації (Clock Management Tile, CMT). Кристали забезпечують підтримку SelectIO™ технології та містять високошвидкісні GTP-послідовні прийомо-передавачі з оптимізацією енергоспоживання, а також убудований оконечний пристрій для PCI Express™. Також є вдосконалені режими керування живленням, автоматичного визначення конфігурації й посилений захист проекту від несанкціонованого доступу з використанням алгоритму шифрування конфігураційної послідовності AES і технології Device DNA.

Сімейство кристалів Virtex®-6 оптимізовано для використання у високотехнологічній апаратурі, характеризується поліпшеними властивостями й забезпечує найбільш широкі можливості на ринку FPGA. Використання інноваційної архітектури ASMBL™ (Advanced Silicon Modular Block), що базується на колонках модульних логічних блоків, дозволило створити різні платформи (підродини), націлені на різні області застосування (домени). Кожний домен включає набір додатків, що розділяють загальні ресурси, такі як високоефективна обробка сигналів, убудований процесор, високошвидкісні міжз єднання (рис. 1.2).

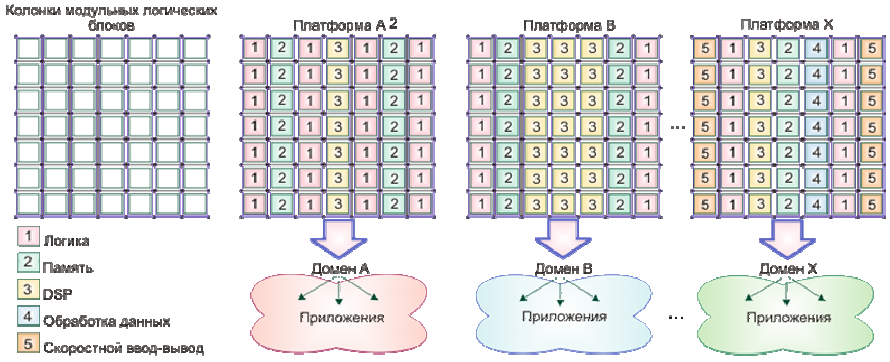


Рис. 1.2. Різноманіття платформ для різних областей застосування

Користувач може вибрати платформу з оптимальним сполученням функціональних можливостей і параметрів для створення спеціалізованої системи на кристалі. ASMBL™ архітектура дозволяє реалізувати технологію монтажу методом переверненого кристала (flip-chip) і усунути обмеження, пов'язані з особливостями традиційного проектування кристалів, наприклад, залежність між кількістю вихідних контактів і розмірами матриці. Архітектура також дає можливість усунути більш тверді обмеження на розміщення ланцюгів живлення й заземлення на кристалі. Як наслідок – значно зменшується час розробки проекту, його вартість і збільшується надійність FPGA.

Дві підродини LXT і SXT включають 9 кристалів різної логічної ємності. LXT оптимізовано для виконання швидкодіючих логічних операцій і реалізації високошвидкісних послідовних інтерфейсів. Кристали SXT насамперед орієнтовані на виконання швидкодіючих операцій цифрової обробки сигналів. Висока продуктивність логічних осередків забезпечується використанням шостивходових таблиць LUT, доповнених здвоєним тригером. FPGA сімейства Virtex®-6 містять широкий спектр убудованих блоків (апаратних IP-ядер), що дозволяють створювати закінчені рішення системного рівня. Block RAM – блок пам'яті ємністю 36 кбіт можна зконфігурувати як два блоки по 18 кбіт. Він також може функціонувати як FIFO і має убудовані ресурси визначення й виправлення помилок.

Третє покоління блоку цифрової обробки сигналів DSP48E1 складається з 25x18 помножувачів, 48-бітового акумулятора, передсуматора й регістрів для конвеєризації, що може працювати на частоті до 600 Мгц. Кристал підтримує SelectIO™ технології із цифровим контролем імпедансу й має також блок керування й синтезу сигналів синхронізації MMCMT. System Monitor орієнтований на зняття характеристик по напрузі живлення, температури усередині кристала й на зовнішніх мікросхемах. У кристалі також є GTX – 6,5 Гбіт/с приймально-передавачі, модуль PCI Express с підтримкою 8-lane Gen1 (2,5 Гбіт/с) і 4-lane Gen2 (5,0 Гбіт/с), TEMAC - 10/100/1000 Мбіт/с Ethernet контролер. Напруга живлення ядра 1В (0,9 В - економічний режим).

Перераховані засоби дозволяють досягти найбільш високого рівня продуктивності й функціональності в цифрових системах, реалізованих на основі FPGA. Виконані по 40-нм КМОП технології, кристали FPGA сімейства Virtex®-6 є реальною альтернативою технології ASIC. Virtex®-6 FPGA являє собою найкращий розв язок для об'єднання можливостей високопродуктивної логіки, цифрової обробки сигналів у цифровій системі на кристалі. Характеристики окремих силіконових кристалів сімейства Virtex®-6 наведені у табл. 1.2.

Сімейство FPGA Stratix® IV [2] компанії Altera виконане за 40-нм технологією й за своїми можливостями перевершує існуючі аналоги.

Кристали характеризуються найбільшою щільністю логічних елементів, найбільшою кількістю прийомо-передавачів, найменшим енергоспоживанням (на 50% нижче, ніж в аналогічних пристроїв FPGA, завдяки використанню технології програмувального енергоспоживання – Programmable Power Technology). FPGA Stratix® IV призначені для реалізації бездротових і DSP додатків, що вимагають високої продуктивності. До складу сімейства Stratix® IV входять 3 підродини, оптимізовані для виконання різних функцій: Stratix IV GX – орієнтовані на прийом і передачу інформації, містять до 531 200 логічних елементів, до 48 повнодуплексних прийомопередавачів, що функціонують на основі технології синхронного відновлення даних (clock data recovery, CDR) і забезпечують швидкість передачі до 8,5 Гбіт/з; Stratix IV E (Enhanced) – інтерфейси пам'яті, містять до 681 100 логічних елементів, до 27 376 Кбіт убудованої пам'яті RAM і до 1 024 убудованих помножувачів 18×18 біт; Stratix IV GT – оптимізовані для швидкісного прийому-передачі інформації, містять до 531 200 логічних елементів, до 27 376 Кбіт убудованої пам'яті RAM, до 1 228 убудованих помножувачів 18×18 біт (табл. 1.3). Послідовні прийомопередавачі підтримують швидкість передачі даних від 2,488 до 10,3125 Гбіт/с. Пристрої Stratix IV GT можна розділити на дві групи залежно від швидкості передачі даних на вході/виході – 100/40 Гбіт/с, які містять відповідно 48 (36) повнодуплексних CDR прийомопередавачів, 24 (12) з яких підтримують швидкість 10,3125 Гбіт/с. Габарити кристалів перебувають у межах від 35 до 43 мм.

Таблиця 1.2

Характеристики кристалів сімейства Virtex®-6

Кристал FPGA XC	Осередки логіки	Логічні блоки		DSP 48A1	Блоки RAM		Розмір, мм / виводів
		Slices	RAM Kb		18 Kb	Max Kb	
6VLX75T	74 496	11 640	1 045	288	156	5 616	23/240
6VLX240T	241 152	37 680	3 650	768	416	14 976	29/400
6VLX550T	549 888	85 920	6 200	864	720	25 920	42,5/840
6VSX315T	314 880	49 200	5 090	1 344	704	25 344	42,5/720
6VSX475T	476 160	74 400	7 640	2 016	1 064	38 304	42,5/840

Таблиця 1.3

Характеристики кристалів сімейства Stratix® IV

Кристал EP	Логічні модулі	Логічні елементи	Пам'ять, Кбит	Убудовані помножувачі	Входів/ виходів
4SGX70 (F1152)	29 040	72 600	7 370	364	480
4SGX290 (F1517)	116 480	291 200	17 248	832	736
4SGX530 (F1932)	212 480	531 200	27 376	1 024	904
4SE110 (F780)	42 240	105 600	9 564	512	480
4SE360 (F1152)	141 440	353 600	22 564	1 040	736
4SE680 (F1760)	272 440	681 100	31 491	1 360	1,104

Інтерес представляє стан ринку електронних технологій у частині top-15 постачальників літографічного обладнання для напівпровідникової індустрії.

Згідно даним маркетингових досліджень компанії VLSI, сумарні доходи першої світової десятки виробників обладнання для виробництва напівпровідників за результатами 2008 року склали $24,5 млрд, що на 25,8% менше, ніж в 2007 році. Лідерство зберігає Applied Materials - $5,877 млрд. За результатами 2008 року голландська компанія ASML перемістилася на другу позицію, що відбиває зростаючу роль літографічного обладнання в напівпровідниковій індустрії. Скоротилися доходи японської Tokyo Electron Ltd. (3 місце) - з $6,3 млрд в 2007 році до $4,3 млрд в 2008, у той час як, що йдуть слідом компанії, KLA-Tencor і Lam Research зберігають за собою четверте й п'яте місце протягом останніх трьох років. Шосту позицію займає компанія Nikon. На сьоме місце в 2008 році перемістилася компанія Canon. Замикають десятку компанії Hitachi High-Technologies, Dainippon Screen і Novellus Systems. За висновками аналітиків VLSI, п'ятнадцять найбільших світових виробників обладнання для виробництва електроніки в сумі контролюють до 70% усього ринку IC-обладнання. У регіональному відношенні top 15 у цей час контролюється на 45,7% компаніями зі США, на 35,6% японськими компаніями й на 18,7% - Європою, за рахунок компаній AMSL і ASMI.

1.2. Структура й технології конструктивів цифрових систем

Представлено основні сучасні конструктиви, використовувані для створення цифрових систем для ринку електронних технологій у порядку зростання їхньої складності. Показано проблеми, що підлягають рішенню в частині сервісного діагностичного обслуговування функціональних блоків.

1.2.1. Система на кристалі - System on Chip (SoC) - спеціалізований цифровий виріб, що конструктивно виконане на силіконовому кристалі (рис. 1.3), яке характеризується низьким енергоспоживанням, малими розмірами, високою швидкодією й що має наступні основні компоненти: процесор (CPU), пам'ять (Reg, ROM, SRAM, DRAM), логіку (UDL), інтерфейс уведення-виводу (I/O, PCI, UART, USB, CAN), перетворювачі інформації (ADC, DSP, MPEG), шини обміну даними й інфраструктуру (I-IP) сервісного обслуговування на основі стандарту IEEE 1500 і технології JTAG.

Функціональні блоки SoC орієнтовані на ефективне рішення актуальної проблеми, на відміну від мікроконтролера, що представляє універсальний обчислювач практично будь-якого завдання. Переваги SoC: прозорість щодо наявності убудованих деструктивних компонентів; висока продуктивність при низькому енергоспоживанні; мініатюрні розміри; ефективне використання різних типів пам'яті; висока надійність; низька вартість готового виробу й високий коефіцієнт використання функціональностей. Плата за описані переваги: істотний час тестування й верифікації проекту (70% від періоду проектування); висока вартість розробки й макетування; складність налагодження й виробництва; інтеграція IP-cores (інтелектуальної власності) з різних джерел.

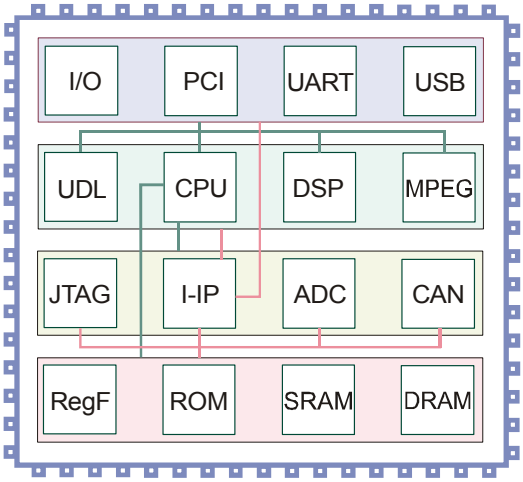


Рис. 1.3. Сучасна структура SoC

IP-Core – складний вже протестований, верифікований і оптимізований функціональний модуль, що може бути багаторазово використаний як компонент при проектуванні більш складних цифрових виробів з метою скорочення часу їхньої розробки [6]. Використання IP-Cores дає: 1) істотне зменшення часу до моменту появи цифрового виробу на ринку; 2) спрощення процесу проектування шляхом створення інтерфейсу для зв'язку системи з IP-Core; 3) мінімізація ризику проектування завдяки включенню вже налагоджених модулів; 4) зменшення часу верифікації всієї системи.

Області застосування SoC: конфігуровані спеціалізовані відкриті репрограмовані вироби; індивідуальні спеціалізовані мініатюрні сервери, орієнтовані на конфіденційність обчислювальних процесів і зберігання інформації під захистом апаратного ключа; інтелектуальна RAM; архітектури для мобільних телефонів; мультимедійні, бездротові й телекомунікаційні функціональності, нейронні, аналогові й квантові обчислювачі. Особливий інтерес ринок проявляє до появи індивідуальних серверів на кристалі, які орієнтовані не тільки на конфіденційне зберігання інформації, але й на захищеність обчислювальних процесів від несанкціонованого доступу. Підключивши сервер, розміром ледве більше Flash Memory до порту USB персонального комп'ютера, користувач виконує характерні для його професійної діяльності обчислення, зберігаючи їх потім на сервері. Таким чином, мікросервер на кристалі можна розглядати як захищену карту Flash Memory з розширеними функціями, орієнтованими на виконання користувачем професійних обов'язків. Актуальність такої концепції підтверджується також низьким коефіцієнтом використання універсального персонального комп'ютера, що дорівнює 5-10 %.

Існуючі проблеми: 1) Зменшення вартості виготовлення SoC кристалів шляхом збільшення виходу придатної продукції. 2) Скорочення періоду появи готового виробу на ринку (time-to-market) і зменшення життєвого циклу готового виробу. Якщо кожні 18 місяців змінюються технології, то термін ефективної служби виробу (3-4 роки) потрібно зменшувати, наближаючи його до 2 років. У цьому випадку компанії, щонайменше, будуть на хвилі використання високотехнологічних продуктів, що означає й привабливість власної продукції для світового ринку. 3) Розглядаючи криву графіка досягнення ефективного виходу придатної продукції на часовій осі (15 місяців), зображеної на рис. 1.4, можна зробити наступний висновок: продукт, що розробляється більше 15 місяців (time-to-market), на ринок ніколи не потрапить. Скорочення даного періоду (зрушення кривій уліво за віссю абсцис), так само як і зменшення числа помилок проекту (підняття кривій нагору за віссю ординат), гарантує високу ринкову привабливість, а значить - додаткові доходи для компанії. Піклуватися про якість і надійність виробу треба, починаючи із системної стадії проектування шляхом створення нових і ефективних методів і засобів тестування й верифікації, що оптимізують процеси, які ставляться до етапу Design for Manufacturability. Yield Learning Curve показує резерви для збільшення виробництва цифрових виробів (production ramp up), особливо шляхом впровадження технологій діагностування й налагодження інтегрованих в I-IP. Надзвичайно важливою тут представляється імплементація стандартів тестопридатності в проект для убудованого тестування, діагностування й ремонту, які істотно підвищують вихід придатної продукції, навіть при наявності дефектів на кристалі [3].

Коментуючи рис. 1.4, слід зазначити: 1) Споконвічно yield має найнижчий рівень. Фаза production ramp up – виготовлення першого кристала – пов'язана з ідентифікацією й діагностуванням проблем (learning), зняттям характеристик, виявленням, аналізом і корекцією (Detection, Analysis and Correction – DAC) дефектів на першому кристалі. Потім yield-крива повільно зростає до раніше вже досягнутого рівня, фіксація якого ініціює початок масового виробництва – volume production. 2) Стиск ramp-up (time to volume) періоду відбувається шляхом нагромадження технічного й технологічного досвіду – yield learning, моделей тестування й налагодження, а також, що особливо важливо, шляхом впровадження інфраструктури сервісного обслуговування I-IP. Графік yield learning може бути істотно поліпшений, якщо стадія yield optimization могла б стартувати в період проектування. Таке можливо шляхом використання знань і особливостей процесу виробництва при проектуванні системи на кристалі. Планувати процес проектування залежно від процесу виробництва (design for manufacturability) або процеси теперішнього часу від майбутнього – значить завжди вигравати! З метою оптимізації проектування й одержання більшого випуску придатної продукції (yield) фабрика повинна виявити yield проблеми як можна на ранній стадії процесу розробки. Збір даних на стадії виробництва, таких як

розподіл дефектів, здійснюється за допомогою блоків убудованої інфраструктури I-IP, які здійснюють моніторинг функціональних модулів.

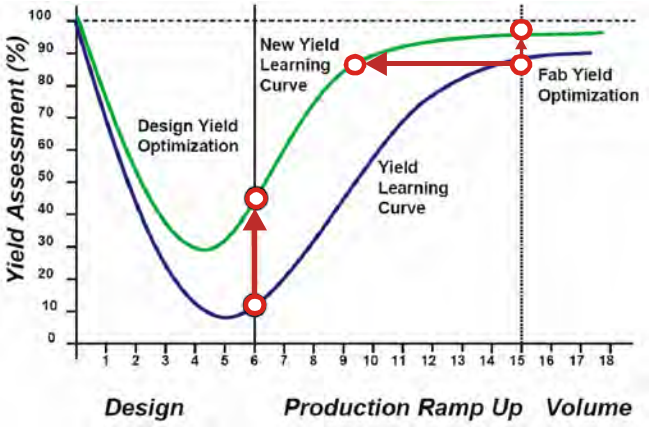


Рис. 1.4. Оптимізація процесу проектування

Інший напрямок підвищення yield - це створення ланцюга поставок у напівпровідниковій індустрії, де існує модель позавиробничої (fabless) розробки, при якій на ринок поставляється IP-Core, не прив'язаний до того або іншого технологічного процесу. При цьому ланцюг поставок включає проектувальників, виробників функціональних IP-Core, фабрики-виготовлювачі кристалів, а також контрагентів з тестування, складання й упакування (packaging) систем на кристалах. У дизагрегированній структурі створення цифрового виробу постачальники (від фабрик) функціональних IP-Core повинні надавати проектувальникам систем на кристалах модель, оптимізовану як під функціональність, так і під виробництво, орієнтовану на високий вихід придатних виробів. Таким чином, просунуті постачальники (підприємства) IP-Core стали брати відповідальність за стадію yield optimization, відповідних функціональних блоків, вбудовуючи необхідну I-IP-інфраструктуру для забезпечення високої якості виробництва.

1.2.2. Мережа на кристалі - Network on Chip (NoC) - спеціалізований мульти-ядерний цифровий виріб конвеєрного типу, що має всі властивості системи на кристалі, доповнене інтелектуальною структурою комунікацій, що підтримує всі архітектури й рівні протоколу паралельного обміну даними офісної мережі.

Як ядро виступає IP-core (процесор або блок пам'яті), що організується в масштабовані сегменти. Такої ієрархії ставиться у відповідність дворівнева система комунікацій і конвеєризації. Підтримка всіх рівнів протоколу обміну даними забезпечує високу швидкодію й пропускну здатність інформаційних каналів. Наявність інтелектуальних мультиядерної архітектури й

комутаторів забезпечує високий рівень розпаралелювання обчислювальних процесів і прийому/передачі даних. Найпоширеніші архітектури представлені на рис. 1.5: a - мережа або матриця, як правило, однакових процесорів, з'єднана числом комутаторів, рівним кількості ядер; b - блочно-функціональна сегментація дворівневої ієрархії мережі; c - багатокутна зірка, орієнтована на ефективний обмін даними між процесорними парами; d - нерегулярна структура дворівневої ієрархії, орієнтована на обслуговування процесорів різних пріоритетів і функціональностей.

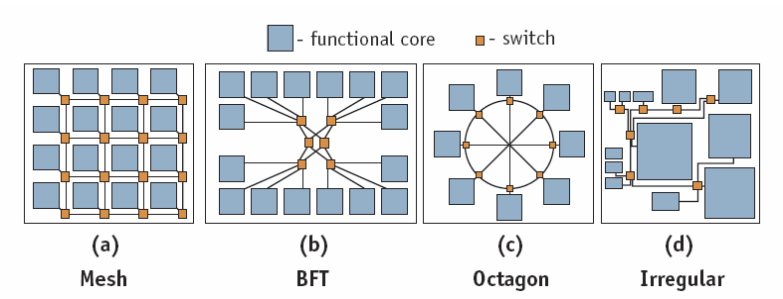


Рис. 1.5. Типові структури NoC

Типова система на кристалі має сьогодні 15-20 (IP-cores) функціональностей. Це поріг її складності й практичного застосування. Далі слідує зона ефективного використання мережі. На відміну від SoC кристал NoC має багаторівневу архітектуру.

Як і глобальна (локальна) мережа, NoC стандарт має сім рівнів протоколу обміну інформацією. NoC структура ефективна не тільки з позиції обчислювальних процесів і обміну даними, але й з погляду енергозбереження й надійності цифрової системи в цілому. Три рівні: транзакцій, транспортний і фізичний використовуються для розподілу IP від даних, переданих по шинах. Перший необхідний для передачі пакетованих сигналів. Другий організує маршрути проходження пакета через систему комутаторів. Третій орієнтований на обмін двійковою інформацією із проводів. Наявність системи пріоритетів, корекція помилок у мережі на кристалі, висока захищеність інтелектуальної власності (IP-core), орієнтація на створення проектів масштабованих систем великої розмірності, ефективність обміну даними, низьке енергоспоживання, висока швидкість виконання операцій, невисока вартість проектування й інших сервісів, характерні для офісної мережі, роблять NoC дуже привабливою технологією для світового ринку, де рівень продажів мережних кристалів у найближчі п'ять років досягне 1 млрд. доларів.

Мережа на кристалі представляється цікавою парадигмою для проектувальників на шляху можливої інтеграції великої кількості

обчислювачів (IP-cores) і блоків пам'яті в одному чипі. Але шлях адаптації SoC технологій до NoC структури й практична імплементація мережі в кристал ставить багато важливих і ще не вирішених проблем перед розроблювачами, що ставляться до методологій проектування, тестування, а також створення нових засобів CAD [4]. Як результат істотного підвищення рівня інтеграції функціональностей виникає проблема підвищення ефективності міжмодульних зв'язків шляхом розробки інфраструктури комунікацій. У цей час існують SoC проекти, що містять мультипроцесори, які застосовуються у TV-тюнерах, бездротових базових станціях, цифровому телебаченні високої роздільної здатності, мобільних телефонах, пристроях обробки зображень (set-top boxes, wireless base stations, HDTV, mobile handsets, and image processing). Згадані пристрої вимагають від розроблювачів ефективних рішень в області проектування регулярної й швидкодіючої комунікаційної архітектури для організації паралельних обчислень на мультиядерних системах у кристалах.

Різні архітектури комунікацій формують типові специфічні шаблони мережних рішень на кристалі. Найцікавіші з них представлені на рис. 1.5. Слід відмітити, що складний кристал SoC може розглядатися як мікромережа множини функціональних блоків, що дозволяє дослідникам запозичити мережні моделі й технології паралельних обчислень для їхнього наступного застосування у архітектурах NoC. Бібліотека «нових» компонентів NoC, що включає комутатори, маршрутизатори, з'єднання й інтерфейси, забезпечує розроблювачеві додаткову гнучкість у проектуванні архітектури NoC на основі процесорних модулів і блоків пам'яті (processor/storage cores). Звичайно, для цього повинні існувати відповідні й досить вивірені засоби й маршрути синтезу й аналізу цифрових мережних систем, що підтримують всі рівні обміну даними, від фізичного до транзакційного.

Істотними характеристиками проекту є енергетичні витрати архітектури міжз’єднань, їхня пропускна здатність, час затримки, а також апаратна надмірність для їхньої реалізації. Всі основні властивості NoC спрямовані на ефективне розв’язання проблеми високопродуктивних міжз’єднань між процесорними компонентами й модулями пам'яті на основі інтелектуальних комутаторів, які створюють комунікаційні зв'язки з високим рівнем абстракції. Проте, задача обміну даними між компонентами мережі (процесори й пам'ять) стає усе більше складною за мірою зростання розмірності системи й узагальненої затримки міжз’єднань. Одним з можливих рішень є подання системи міжз’єднань мультимодульної конвеєрною платформою, де кожний комунікаційний модуль задовольняє за часовими параметрами функціонуванню всієї системи. При цьому NoC архітектура підтримує конвеєрний тип, як внутрішньосегментних перемикань з'єднань модулів, так і міжблочних контактів структури, обумовлений конвеєризацією встановлення зв'язків, перемикань, інваріантних до затримок компонентів мережі.

Тестування NoC. Оскільки мережа походить із системи на кристалі, то природним представляється адаптація існуючих SoC технологій і механізмів тестування (TAM) до аналогічних проблем з обов'язковим використанням методів і засобів тестопридатного проектування (DFT). Стратегія тестування NoC систем орієнтована на вирішення трьох основних проблем: 1) Тестування функціональних модулів і блоків пам'яті й відповідної їм мережі інтерфейсів. Механізм TAM використовується для транспортування тестових наборів з відповідної пам'яті або генератора до модуля, що тестується, або блоку пам'яті. Завдяки регулярності мережі існує можливість одночасного й паралельного завантаження тестових ресурсів для кожного модуля. Таким чином, час тестування може бути зменшено, завдяки використанню просторових характеристик мережі, що дають можливість розпаралелювати процеси подачі тестів на функціональні блоки і їхнє тестування. Спостерігаємість міжз'єднань є порівняно низкою, оскільки вони глибоко убудовані в кристал. Обмеження на число зовнішніх виводів кристала впливають на число вхід-вихідних контактів, які можна використовувати в процесі тестування. NoC інфраструктура повинна бути використана для тестування власних компонентів за рекурсивною процедурою - уже перевірені модулі необхідно використовувати для транспортування тестових наборів до ще не перевірених блоків. Стратегія тестування мінімізує використання додаткових механізмів для транспортування даних до компонентів мережі, що дозволяє зменшити час тестування, завдяки використанню паралельних шляхів для групової передачі даних. Роздільне тестування функціональних блоків і інфраструктури міжз'єднань не є достатньою умовою досяжності необхідної якості тесту. 2) Перевірка інфраструктури міжз'єднань (міжблокових і внутрішньоблокових). Взаємодії між функціональними модулями й структурою комунікацій перевіряється повними функціональними тестами. 3) Тестування інтегрованої системи в цілому повинне охоплювати перевірку вхід-вихідних функцій кожного процесорного елемента, а також всі функції маршрутів проходження даних.

Для забезпечення якості й надійності NoC, за аналогією з SoC I-IP, необхідно створювати інфраструктуру сервісного обслуговування (I-IP) мережі на основі використання IEEE стандартів 11.49 і 1500, що повинна мати наступні компоненти: 1) Моніторинг станів внутрішніх і вихідних ліній у процесі функціонування, верифікації й тестування блоків на основі використання стандарту граничного сканування IEEE 1500 [5,6]. 2) Тестування функціональних модулів, орієнтоване на перевірку дефектів або справної поведінки. 3) Діагностування відмов шляхом аналізу інформації, отриманої на стадії тестування й використання спеціальних методів убудованого пошуку несправностей на основі стандарту IEEE 1500 [6]. 4) Ремонт функціональних модулів і пам'яті цифрових систем на кристалах після визначення місця й виду дефекту при виконанні діагностування. 5)

Забезпечення надійності функціонування виробу за допомогою диверсифікованості й дублювання функціональних блоків.

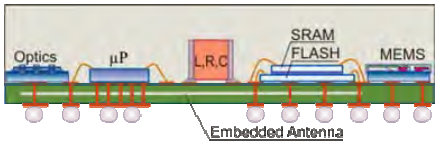
1.2.3. Система в пакеті System in Package (SiP) - це набір кристалів, розміщених на загальній підложці, що представляє собою функціональність, реалізовану в одному корпусі [7]. SiP включає: пасивні компоненти, такі як дискретні елементи; інтегральні пасивні схеми; пасивні елементи, убудовані в підложку, а також компоненти спеціального призначення у вигляді фільтрів на поверхнево-акустичних хвилях, електромагнітні екрани (EMI shields); пакетовані кристали, зовнішні з'єднувачі й механічні елементи.

Традиційне застосування систем у пакеті: керування живленням, сенсорні пристрої, RF і wireless додатки для мобільної телефонії, пристрої для високопродуктивних обчислень, пам'ять, кишенькові комп'ютери - PDA, глобальні системи навігації й визначення місця розташування - GPS, засобу WiFi і Bluetooth™ [6,7].

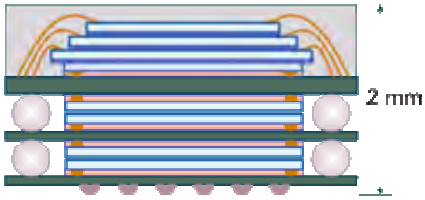
Функціональні особливості силіконових кристалів для виготовлення цифрових виробів можуть бути класифіковані за наступними ознаками [9]: 1) технологія: цифрові, аналогові, RF, оптоелектронні, MEMS, пасивні; 2) частота: від Мгц діапазону для цифрових пристроїв і RF до Ггц для мікрохвильових компонентів; 3) архітектура: аналогові пристрої малої й великої потужності; цифрові структури реального часу, керовані подіями й даними; 4) споживання енергії: портативні пристрої с малим споживанням енергії; 5) інші фактори: форм-фактор, вага, виділення тепла.

Традиційні технології впакування кристалів характеризуються наявністю обмежень за наступними характеристиками [10]. 1) Продуктивність системи на кристалі, обумовлена затримками передачі сигналу між кристалами, поширенням і часом наростання сигналу. 2) Мініатюризація, обумовлена площею посадкового місця корпуса системи на кристалі, зазором між провідниками й кількістю шарів топології. Актуальними є задачі зменшення площі системи в пакеті кристалів і збільшення її продуктивності.

SiP являє собою розвиток більше ранніх технологій упакування MCM (Multichip Module), MCP - Multi-Chip-Package і характеризується наявністю одного або більш кристалів, комбінуванням технологій провідного монтажу й монтажу методом переверненого кристала - виводами до підложки, розміщенням пам'яті й логіки на одній підложці разом з пасивними компонентами, фільтрами й антенами. Різноманіття компонентів MCM модуля показане на рис. 1.6,а.

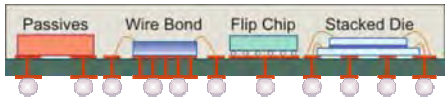


а - Різноманіття компонентів MCM

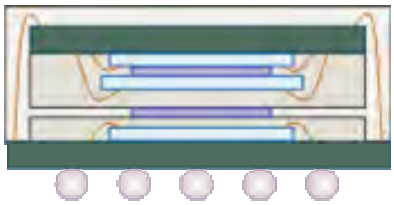


б - Multi Chip Package

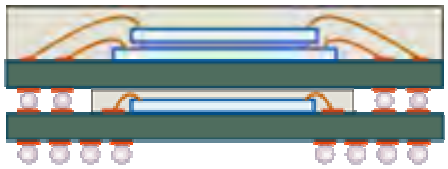
в - SiP на одному кристалі



г - Stacked Die + Flip-chip елементи



д - Package-in-Package Structure



е - Система « Package-on-Package»

ж - Embedded chip package, principle

с - Стік підложок «Thru silicon vias»

Рис. 1.6. Конструктиви цифрових систем у пакетах

У цей час існує п'ять концепцій тривимірного впакування кристалів [11-13]:

1. «Multichip Package» [10, 14-16]. Кристали розміщаються у горизонтальній площині (рис. 1.6,б), можуть накладатися один на одного у вигляді етажерки або утворювати комбіновану 3D структуру з метою забезпечення мінімального значення форми-фактора. Конструкція «Multichip Package» характеризується високою продуктивністю, малими розмірами й вагою, а також мінімальною затримкою поширення сигналу.

2. « System-in-Package» [10, 13, 15-18] заснована на метриці різновидів SiP: 1) Використання компонентів, що допускають розміщення в горизонтальній площині (side-by-side) (рис. 1.6,в); технологія провідного монтажу й термокомпресійного мікрозварювання Wire Bond; монтаж кристалів на гнучкій стрічці, що виконана з полімерного матеріалу (поліїмід) Tape Automated Bonding (TAB) і/або компонентів, що монтуються методом перевернено кристала flip-chip. 2) Використання компонентів, що представляють собою складальні одиниці типу «складений кристал» (Stacked Die) у вигляді BGA (Ball Grid Array) модулів з кульковими виводами або

монтуємих за технологією монтажу Wire Bond, а також компонентів Flip-chip (рис. 1.6,г). 3) Використання убудованих у підложку компонентів. У порівнянні з розміщенням у горизонтальній площині складальні одиниці типу Stacked Die дозволяють збільшити швидкодію, зменшити габаритні розміри, споживну потужність і поліпшити температурний режим у робочій зоні. Окремі кристали можуть бути досить просто переведені зі штатного режиму у режим автономного тестування. Застосування технології монтажу компонентів методом перевернененого кристала дає можливість поліпшити значення форм-фактора, зменшити паразитні ємності й індуктивності між елементами топології.

3. «Package-in-Package» (PiP) [13, 19, 20]. Пакет складається з набору або стека повністю перевірених модулів (Internal Stacking Module, ISM), розміщених у базовому складальному пакеті (Base Assembly Package, BAP) з метою формування єдиного конструктивного виробу (рис. 1.6,д).

4. «Package-on-Package» (PoP) [12, 13, 17]. Являє собою структуру, у якій повністю перевірені пакети встановлені вертикально, один на одного. При цьому ступінь інтеграції може бути різною. Дана концепція набула найбільшого застосування для створення цифрових або аналого-цифрових логічних пристроїв з високою щільністю впакування, розташовуваних у нижньому базовому пакеті, з одним або декількома запам'ятовувальними пристроями великої ємності, розташовуваними у верхньому пакеті [12]. Лідером розробки й впровадження технологій тривимірного впакування кристалів (3D packaging technologies) є компанія Amkor Technology. Приклад конструктивного рішення PoP, при якому верхній пакет - модуль пам'яті - розміщається в корпусі FBGA (Fine Pitch BGA) або Stacked CSP (Chip Scale Package), а нижній - логічний модуль - у корпусі PSvfBGA (Package Stackable Very Thin Fine Pitch BGA) (рис. 1.6,е) [7].

5. «Embedded chip technologies», «Substrate Technology» [10, 21]. Активні або пасивні кристали вбудовуються у підложку (рис. 1.6,ж), де міжз’єднення реалізуються за допомогою мікроотворів. Технологія «Wafer Level» [22-24] дозволяє встановлювати кристал на підложці, підложку на підложці (рис. 1.6,з) і реалізовувати технологію міжз’єднань Thru Silicon Vias (TSV) [25-28, 29, 30]. Wafer Level технологія характеризується високим ступенем інтеграції й, поряд із цим, має ряд недоліків [10]: висока вартість проектування й виробництва цифрової системи в пакеті кристалів; використання компонентів від різних виробників, що виготовляються за різними технологіями; складність процесу відновлення працездатності системи; відносно низький відсоток виходу придатних виробів у порівнянні з іншими технологіями. Стандарт по застосуванню багатокристальних модулів IPC-MC-790 (Guidelines for Multichip Module Technology Utilization) визначає три типи технологій виготовлення підложки: багатошарова (Laminated); керамічна (Cofired) з використанням технології тонких плівок (Thick Film) і підложка, формована технологіями напилювання (Deposited).

Переваги тривимірного впакування кристалів на основі технологій активного використання підложки [10]: 1) Мініатюризація – збільшення щільності впакування й топології, які визначаються в такий спосіб: $P_{eff} = \text{Total Active Silicon Area} / \text{Total Active SiP Area}\ (\%)$, де P_{eff} – щільність упакування; $\text{Total Active Silicon Area}$ – активна площа кристала; $\text{Total Active SiP Area}$ – активна площа цифрової системи в пакеті кристалів SiP; $\text{Wiring Density} = \text{Wiring Length} / \text{Silicon Area}\ (\text{cm}/\text{cm}^2)$, де Wiring Density – щільність топології; Wiring Length – сумарна довжина зв'язків; Silicon Area – площа кристала. 2) Збільшення продуктивності. Для оцінки продуктивності на системному рівні використовується параметр «час циклу»:

$$\text{Cycle Time} = \text{Time for Semiconductor} / \text{Time for Package},$$

де Cycle Time – час циклу; $\text{Time for Semiconductor}$ – час циклу для напівпровідника; Time for Package – час циклу для пакета кристалів. 3) Зменшення вартості пакета за рахунок підвищення відсотка виходу придатних виробів і усунення етапу впакування системи в корпус, а значить і виключення відповідних операцій з виробничого процесу. 4) Поліпшення надійності системи завдяки зменшенню сумарної довжини зв'язків і кількості шарів пакета за рахунок розміщення безкорпусних кристалів безпосередньо на багатошаровій підложці.

Проблеми використання технологій підложки: 1) висока вартість виробництва; 2) складність процесу прийняття рішень (Decision Making Process); 3) неможливість використання існуючих моделей; 4) негарантована якість безкорпусних кристалів; 5) низький відсоток виходу придатних виробів після операцій складання пакета кристалів; 6) складний процес відновлення працездатності; 7) обмежений набір засобів автоматизації проектування (CAD Tools); 8) складність забезпечення оптимального теплового режиму.

Статистика ринку щодо використовуваних продуктів на основі SiP [10]: 33% – замовлені спеціалізовані вироби; 20% – засоби телекомунікації; 20% – компактні енергозберігаючі обчислювачі; 10% – військового або авіакосмічного призначення.

Технології підложки: 58% – керамічна – Cofired; 39% – багатошарова – Laminated; 3% – напилювання – Deposited. Технології монтажу: Wirebond, TAB, FlipChip. Технології забезпечення тестопридатності: 50% – Boundary-Scan; 54% – внутрісхемне зондування контрольних точок (Internal Test Points Probed); 69% – відновлення працездатності дефектних SiP. Типи несправностей: найбільший відсоток дефектів пов'язаний з порушенням працездатності кристалів. Технології автоматизації тестування: 46% – спеціальні тестові набори (Custom Test Sets); 37% – використання тестерів (IC Testers). Напрямки розвитку пакетних технологій: збільшення складності

кристала до 700 млн. транзисторів на кристалі; зменшення площі кристала; збільшення кількості виводів до 10 К; використання кулькових виводів із кроком 50 мкм.

1.3. Технології виготовлення й тестування цифрових систем у пакетах

Загальна стратегія створення й тестування цифрової системи в пакеті (MCM – Multi Chip Module) містить пункти [10], зображені на рис. 1.7.

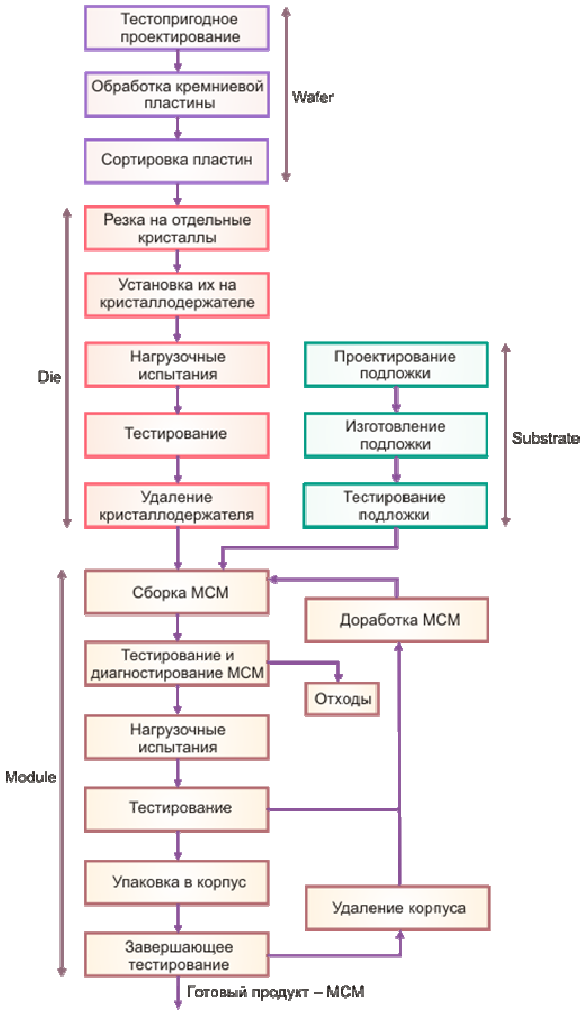


Рис. 1.7. Стратегія виробництва й тестування цифрової системи в пакеті

1) Перевірка 300-мм пластини після її виробництва (виготовлення, сортування, розпилювання). Паралельно у часі виконуються наступні пункти 2 і 3.2) Створення підложки (проектування, виробництво й тестування). 3). Створення кристала (mount in carrier - установка кристалів на носії або

кристалотримачі, випробування на наявність дефектів, повне тестування, видалення носія). 4) Створення (асемблювання) модуля МСМ з раніше створених і перевірених компонентів (підложки - substrate і кристала die): складання, тестування й діагностування модуля, випробування, перевірка після випробування, інкапсуляція компонентів у пакет, остаточне тестування пакета модулів. Тут же виконується відбраковування компонентів у сміттєвий кошик або на переробку, якщо це допускається технологією.

Певний інтерес становить сучасне виробництво кремнієвих пластин. Перший етап – формування монокристала кремнію циліндричної форми. Він вирощується на основі затравочного кристала, що являє собою ідеальну (зроблену) структуру необхідної кристалографічної орієнтації. Затравочний кристал поринає у ванну (тигель) з розплавленим кремнієм. Тигель і кристал обертаються в протилежних напрямках для забезпечення рівномірного розподілу температури й домішок за обсягом розплаву. Потім монокристал повільно витягається з розплаву. При цьому відбувається перехід атомів з рідкої або газоподібної фази речовини у тверду фазу на їхній межі розділу й ріст монокристала. У результаті виходить кремнієвий злиток високої чистоти заданого діаметра. У цей час для виробництва систем на кристалі використовуються заготівлі діаметром 300 мм.

До 2013 року планується перейти на використання 450-мм кремнієвих пластин. Вартість одного нового заводу - 2 млрд доларів. Другий етап - одержання кремнієвих пластин товщиною близько 0,6 мм шляхом розрізування монокристала за допомогою кільцевої алмазної пилки.

Для одержання необхідного значення шорсткості поверхні пластини піддаються додатковій обробці: 1) шліфуванням за допомогою обертових сталевих дисків і абразивного матеріалу, при цьому усуваються нерівності висотою більше 10 мкм; 2) травленням для видалення погрішностей у структурі кристала; 3) поліруванням для одержання дзеркально гладкої поверхні з нерівностями не вище 3 нм. Третій етап - нанесення захисної плівки діелектрика (діоксиду кремнію, $SiO2$), що є гарним ізолятором і виконує захисну функцію при подальшій обробці підложки. Четвертий етап - нанесення фоторезисту з метою формування на підложці захисного рельєфу, що не піддається наступному травленню. П'ятий етап – літографія. Під впливом ультрафіолетового випромінювання, що попадає на певні ділянки захисного шару через маску, фоторезист змінює властивості розчинності. Після опромінення фоторезист піддається прояву. Нова технологія літографії EUV (Extreme UltraViolet - наджорстке ультрафіолетове випромінювання), заснована на використанні ультрафіолетового випромінювання з довжиною хвилі 13 нм, дозволяє формувати лінії шириною до 30 нм. Шостий етап - травлення з метою видалення плівки діоксиду кремнію з ділянок поверхні пластини, які повинні бути піддані подальшій обробці. При виробництві кристалів використовується метод сухого (плазмового) травлення з використанням іонізованого газу (плазми), що вступає в реакцію з

поверхнею діоксиду кремнію. Відбувається руйнування діоксиду кремнію на ділянках, не покритих захисним шаром, після чого частина фоторезисту, що залишилася, видаляється. Сьомий етап - дифузія легуючої домішки шляхом іонної імплантації. Іони домішки «вистрелюються» з високовольтного прискорювача й, маючи достатню енергію, проникають у поверхневі шари кремнію. Таким чином, формується шар напівпровідникової структури. Для створення декількох шарів повторюють етапи із третього по сьомий. Восьмий етап - металізація. При формуванні шарів на окремих ділянках залишають області, які потім заповнюються атомами металу – провідні зони, використовувані для встановлення зв'язків між шарами. Провідні зони формують складну тривимірну структуру міжшарових з'єднань.

Перед розрізуванням кремнієвої пластини на окремі кристали її товщину зменшують (приблизно на 33%) і видаляють забруднення зі зворотної сторони. На тильну сторону підложки наноситься шар спеціального матеріалу, що поліпшує кріплення кристала до корпусу або кристалотримача. По закінченні процесу формування пластини вона піддається тестуванню на установках зондового контролю з метою перевірки якості виконання технологічних операцій і виявлення дефектних кристалів. На контакти кожного кристала накладаються контакти зонда й проводяться електричні тести. Спеціальне програмне забезпечення дозволяє формувати тестові набори й спостерігати стан виходів кристала. Різання пластини на окремі кристали здійснюється за допомогою спеціальної прецизійної пилки. Дефектні кристали відбраковуються. Придатні піддаються навантажувальним випробуванням і тестуванню, після чого вони надходять на ділянку складання багатокристального модуля.

По закінченні процесу складання модулі тестуються і піддаються додатково навантажувальним випробуванням. Потім виконується повторне тестування для визначення технічного стану пристрою. Справні модулі розміщуються в індивідуальному корпусі, що захищає кристал від зовнішніх впливів і забезпечує електричне з'єднання з підложкою, на яку він згодом буде встановлений. Далі впакований модуль піддається завершальному тестуванню, випробуванням при впливі різних температурних, вологісних режимів і електростатичних розрядів у відповідності зі специфікацією пристрою. Несправні модулі відбраковуються. Кристали, що пройшли тестування, надходять на вихідний контроль, завдання якого - підтвердити, що результати всіх попередніх перевірок були коректними, а параметри пристрою відповідають установленим вимогам. Багатокристальні модулі – пакети, що пройшли вихідний контроль, маркіруються й упаковуються для доставки виробів замовникам.

Сучасні мікроелектронні чипи, що розташовуються на друкованій платі, використовувалися в цифрових виробах протягом 40 років. У двовимірному форматі були розроблені чудові засоби для міжз’єднаних кристалів, живлення, охолодження й захисту. Сьогодні система в пакеті забезпечує

різноманіття пакетування для комп'ютерів, космосу й авіації, медицини й військових цілей шляхом перетворення плати у тривимірний пакет кристалів, що зменшує обсяг, підвищує надійність і продуктивність системи в цілому.

Пакетування електронних компонентів розглядається як частина комп'ютерного системного проектування. Найбільш відомої є multichip module (MCM) технологія пакетування, що інтегрує known good dies (KGD) на підложці з високою щільністю розміщення компонентів. На основі даної технології в дев'яності роки минулого сторіччя розроблялися потужні й професійні (high-end) комп'ютерні системи у 2D виконанні. Наприкінці дев'яностих з'явилися 3D реалізації технології MCM з високою щільністю монтажу. Як розширення й розвиток MCM технологія SiP стає широко розповсюдженою, спрямованою на зменшення форм-фактора, що робить її перспективною в найближчі 10 років. Технології тестування пакетів життєздатні, продуктивні, але ефективні тільки для простих пакетів. Що стосується багатокристальних гетерогенних системних структур, що мають логіку, пам'ять, аналогові й радіочастотні компоненти, то їхнє мініатюрне виконання дуже очікуване на ринку, як виріб, що має переваги: невисока вартість і висока продуктивність, незначний час розробки й низькі енергетичні витрати.

Проте, існує ряд проблем в області design and test, які варто вирішувати. Необхідні узагальнені EDA засоби для інтегрування цифрових, змішаних mixed-signal і радіочастотних чипів, що утримуються в SiP. Засоби для проектування й аналізу повинні вміти виконувати моделювання електричних і температурних режимів, а також використовувати KGD як компонент, хоча відсутність придатного кристала затримує проектування інтегрованої системи в цілому. Ще одна проблема, пов'язана з поширенням і імплементацією пасивних елементів у підложку пакета, що повинне виконуватися під керівництвом SiP виробників, які зобов'язані ознайомити проектувальників із сімейством пасивних компонентів, що має високу актуальність для ефективного створення систем у пакетах і їхнього просування на ринок. Зараз існує два напрямки в рішенні проблеми проектування системи в пакеті: розробка чипа й інтегрування пакета. Особливо необхідні нові засоби для пакетування множини кристалів, а також для тестування вже інтегрованої системи. Стратегія тестування SiP відрізняється від перевірки SoC, оскільки система в пакеті інтегрує компоненти від різних виробників. Створення нових технологій тестування SiP є сьогодні пріоритетною проблемою, поставленою в середовищі практичних вчених і дослідників, що ставляться до design-and-test community. Самою перспективною стратегією є використання стандарту IEEE 1500 для убудованого тестування функціональних компонентів. Маючи високу продуктивність і рівнем інтеграції компонентів у невеликому обсязі, пакети містять кілька кристалів, з'єднаних провідними шинами, які негативно впливають на проходження сигналів у високочастотному спектрі.

Інтеграція мультиядерних гетерогенних систем у пакет також ставить багато питань перед design-and-test інженерами. Подолання таких бар'єрів вимагає об'єднання зусиль і засобів з боку фахівців і компаній, що займаються цифровим аналоговим, RF- і тепловим (thermal) проектуванням. Існуючі пропозиції з тестування SiP пов'язані з адаптацією DFT-технології для перевірки убудованих АЦП і ЦАП перетворювачів чисто цифровими тестами, які більше економічні для конвертерів. Таким чином, успіх SiP на ринку залежить від рішення проблем, пов'язаних зі створенням засобів проектування гетерогенних компонентів і їхнього ефективного тестування.

SiP дизайн є привабливою технологією для широкого кола застосувань, таких як: бездротові пристрої, сенсори, графічні перетворювачі, мережні пакетні перемикачі. З погляду проектування й виробництва конструктив SiP пропонує гнучкі технології для створення й виготовлення пакетів з різнорідних матеріалів: германій-кремнієвий кристал (SiGe) – для швидкодіючої логіки, CMOS – для аналогових компонентів і пам'яті. Таким чином, з'являються можливості створювати маленькі за розміром вироби високої продуктивності й випуском придатних виробів. SiP також припускає істотно знизити витрати на тестування, оскільки кожний тип кристала повинен мати власний убудований тестер. На додаток до сказаного, кожний функціональний компонент повинен бути ремонтованим або реконфігуруємим у випадку виявлення дефектів. У цей час SiP дизайн не має інфраструктури сервісного обслуговування (як у SoC) для уніфікованого рішення проблем проектування на основі моделі процесу, орієнтованого на спільне створення кристалів пакета й покриття електричних функціональних, механічних термальних особливостей компонентів проекту.

1.4. Тестопридатне проектування SiP на основі стандарту IEEE 1500

Система в пакеті інтегрує декілька кристалів, зроблених, як правило, від різних постачальників і розроблювачів. Тому тестування SIP-структури істотно відрізняється від перевірки системи на кристалі. Далі розглядаються стратегії, засновані на використанні IEEE 1500 стандарту, які, на думку експертів [6], є перспективними при вирішенні проблем сервісного обслуговування SiP.

Вимоги ринку диктують компаніям розробляти вироби за метрикою: швидше, менше, дешевше. Тому виникли технології, здатні оперувати мініатюрними компонентами, інтегруючи їх у набір функціональних модулів на одному кристалі. Парадигма системи на кристалі гарантовано виграє в цифрових систем на друкованих платах за продуктивністю, розмірами і енергоспоживанням. SoC-виробники можуть розробляти операційні модулі самостійно або купувати готові функціональні блоки (як IP) для наступного асемблювання їх у процесі проектування спеціалізованих цифрових систем. При цьому виробник повинен випускати SoC великими серіями з метою відшкодування витрат інженерної праці, необхідного для розробки кристала.

З 1990-х років концепція SoC постійно зростає в частині її привабливості для ринку електронних технологій, завдяки інтеграції на одному чипі функціональностей у вигляді логіки, пам'яті, радіочастотних модулів і блоків живлення. З боку засобів проектування слід помітити, що інструменти моделювання, верифікації й валідації зробили гігантський стрибок убік складності оброблюваних моделей. Тестування більш складних SoC зажадало таких платформ тестового обладнання, які пропонують більше функціональних можливостей, а також спеціалізованого убудованого тестового обладнання, необхідного для нових маршрутів тестування SoC. Імплементація компонентів в SoC є вибір компромісу між оптимальними технологіями і бізнес-цілями. Тому важливим представляється не тільки якнайшвидше вийти з новим виробом на ринок, але й мати технологічні засоби заміни несправних компонентів.

Переходячи до системи у пакеті, проектувальник і виробник одержують: автономні IP чипи, виконані як чисті кристали, нашаровані один на одного та з внутріпакетними з'єднаннями. Кожний кристал може мати окрему функціональність або бути багатофункціональним, що містить сукупність IP-cores. SiP убрав у себе переваги, як друкованої плати, так і системи на кристалі: 1) Компоненти можуть бути виготовлені окремо друг від друга на основі різних технологій, а потім пакетовані. 2) Це означає, що вся система буде мати високу продуктивність і прості маршрути (фази) проектування. 3) SiP-структура добре масштабується й оптимізується, а також всі функціональні IP-модулі мають надійний захист авторських прав виробника. Стратегія «розділяй і пануй» (divide-and-conquer) тут підходить як для виробника, так і для споживача, які повинні дотримуватися твердого графіка створення продукту – від визначення специфікації до поставки готового виробу. Мільярди цифрових систем у пакетах вже перебувають у руках споживачів; це, насамперед, мобільні комунікаційні вироби й різні види «твердої» пам'яті. Проте, тестування SiP усе ще вимагає пильної уваги з боку вчених і експертів. Активність фахівців спрямована на адаптацію SoC технологій тестування, що використовують регістри граничного сканування як навколишнє середовище (core wrapper) і механізми подачі тестів (test access mechanisms), як складові частини тестопридатного проектування. Для систем у пакетах сценарій тестування відрізняється від описаного вище, оскільки він орієнтований на два рівні опису системи: тестування кристалів окремо, а потім – заключна перевірка асембльованої системи в цілому. Також в SiP є надлишкові нефункціональні компоненти, що забезпечують підвищення керованості й спостерігаємості сигналів, які інтегруються в стратегію тестування системи в цілому. Для цього створюються спеціальні засоби й структури (подібно I-IP для SoC), що обслуговують функціональні блоки системи, включаючи аналогові й змішані – mixed-signal, технологічні міжз’єднання кристалів, а також проектні обмеження й технічні умови, що використовують новий стандарт 1500 Standard for Embedded Core Test (SECT).

Парадигма тестування SiP заснована на передумові існування свідомо справних кристалів – known-good-die. Це означає, що на рівні системи в пакеті не завжди існує можливість протестувати всі функціональності й структурні зв'язки, як це можна зробити на рівні одного кристала. Для цього існує кілька причин: обмежений доступ до всіх тестових режимів кристалів, через існування заборон на понадлімітні витрати за часом і матеріальними витратами. Інші невирішені проблеми ставляться до технології тестування з'єднань усередині підложки, що має стійку тенденцію до ускладнення структури й збільшенню числа внутрішніх компонентів, активних і пасивних. Останні елементи використовуються, у тому числі, для з'єднання окремих кристалів у систему. Крім того, важливо мати повні тести для міжз'єднань на рівні одного кристала, а також тести верифікації, чутливі до параметрів пакетування кристалів.

У цей час SiP архітектури орієнтовані на виконання ручних, обчислювальних і мобільних пристроїв, які вимагають мінімального простору й злиття функціональностей, включаючи цифрові, аналогові модулі й керовані обчислювальним процесом блоки живлення. У такій конфігурації SiP має кілька кристалів різної природи й підтримує наступні режими: 1) Один або кілька обчислювальних пристроїв (процесори, спеціальні обчислювачі, кодеки, периферійні пристрої, з'єднані шинами) здійснюють поставку користувачеві своїх функціональностей. 2) Автономні кристали пам'яті (Flash, E2PROM, DRAM), вироблені роздільно, потім можуть бути асембльовані для одержання більш складних архітектур. 3) Аналогові кристали й чипи живлення надають системам у пакетах можливість взаємодіяти з електромеханічними компонентами, а також здійснювати вивід інформації на екран та аудіосистему. 4) RF кристали надають системі можливість взаємодії із працюючим обладнанням і забезпечують схоронність інформації, отриманої в процесі обчислень. 5) Підложка служить не тільки як структура міжз'єднань кристалів один з одним на основі застосування активних і пасивних елементів, але дана область може бути використана для підвищення повноти тестової стратегії SiP.

Процес виробництва SoC і SiP для стадій створення кристалів має більше загальних моментів, ніж розходжень, що також обумовлює адаптацію для SiP методів SoC тестування. Більше того, методи й механізми перевірки компонентів на рівні тестування кристалів є загальними для SoC і SiP [31-33]. Другий рівень тестування SoC використовує системний тест, орієнтований на перевірку інтерфейсів, міжз'єднань і всієї структури в цілому. Оскільки SiP має багатокристальну архітектуру, маршрут системного тестування складається з наступних пунктів: 1) Визначення вимог до повноти тестування підсистем. Кожний зроблений кристал має власний інтерфейс тестування й строго певний порядок виконання тестів. 2) Визначення вимог до якості перевірки системи. SiP архітектура обумовлює два важливих аспекти. По-перше, тестування кристала може бути

орієнтоване на загальну структуру SiP у взаємодії з іншими компонентами, коли функціональність одного кристала SiP може використовуватися при тестуванні іншого. По-друге, необхідно мати ефективні алгоритми аналізу результатів тестування міжз'єднань кристалів, використовуючи архітектуру інтерфейсів, доступну для підсистем тестування.

Останній рівень виробничого тестування відбиває структурні розходження між SoC і SiP. Тут тестова стратегія, розроблена для всієї системи, переводиться у Automatic Test Equipment (ATE) формат: рівень 1 – опис тестової програми й тестових впливів; рівень 2 – тестові програми, що згенеровані для перевірки міжз'єднань між кристалами й завершальним тестуванням всієї структури SiP.

Стандарт IEEE 1500 SECT є найбільш ефективним рішенням виробничого тестування SoC. У ньому зосереджена стандартизована методологія передачі тестових даних, що не залежить від функціональності системи й дозволяє виконувати «plug-and-play» тестування. Стандарт визначає: масштабовану структуру інтерфейсу тестування для IP-cores, що входять до складу SoC; мова тестування IP-core – Core Test Language – CTL, що дозволяє описати архітектуру тесту й протоколи обміну даними між фірмами виробниками IP-core і системним інтегратором.

На рівні інтеграції цифрової системи на кристалі IP-core, поміщений у тест-інтерфейс (wrapper) IEEE 1500, підключається до спеціальної, визначеної користувачем інфраструктури тестування завдяки наявності механізму тестового доступу (Test Access Mechanism – TAM). Тестування здійснюється через тест-інтерфейс IEEE 1500, за допомогою якого зовнішнє обладнання передає тестові набори й одержує відгуки у встановленому порядку. Використовувані для SoC рішення, засновані на стандарті IEEE 1500, можуть бути поширені на парадигму SiP шляхом застосування представлених у стандарті структур для тестування окремих кристалів, що утворюють систему в пакеті, гарантуючи, таким чином, можливість реалізації тестування на рівні кристала й на системному рівні. Розміщення тест-інтерфейсу на кристалі дозволяє поліпшити якість проекту, незважаючи на те, що часові параметри й апаратурні витрати виробу можуть трохи погіршитися.

Запропоноване в роботі [4] рішення з організації тестового доступу для SiP засновано на ієрархічному підході, представленому в стандарті IEEE 1500. Переваги зазначеного підходу: 1) проста й швидка взаємодія тесту із шарами IP-cores і підсистемами незалежно від типів компонентів, спрощення проектування структури тесту для кожного модуля й приєднання до компонента TAM; 2) ефективна підтримка тестування з'єднань між кристалами в пакеті; 3) визначення стандартного підходу до генерації тестових програм на рівні кристала й SiP у цілому. На рис. 1.8 представлена зразкова конфігурація тесту для SiP, що складається з 2-х кристалів. Тут використовується один послідовний канал для реалізації механізму

тестового доступу (мінімум, певний стандартом IEEE 1500). Запропонована структура може бути використана у двох випадках: коли кристал тестується окремо шляхом його підключення безпосередньо до автоматичного тестового встаткування (ATE) і для тестування функціональності SiP у цілому.

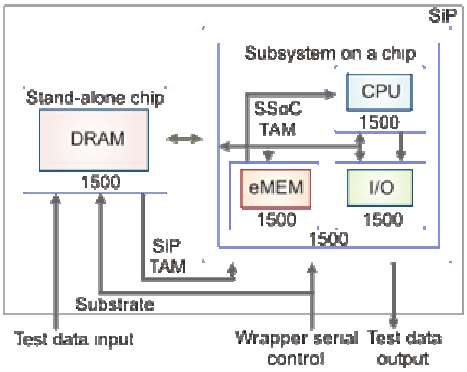


Рис. 1.8. Механізм TAM для двокристальної системи в пакеті

Тест-інтерфейс забезпечує можливість формування тестових даних для кожного IP-core і одержання результатів, а також виконання необхідних перетворень формату даних для передачі їх в TAM, починаючи від уведення тестових наборів і закінчуючи їхнім виводом. Шостирозрядна шина забезпечує керування послідовними структурами інтерфейсу шляхом передачі даних до кожного кристала на основі стандартизованого протоколу. На рис. 1.9 представлена пропонована структура інтерфейсу тестування на основі стандарту IEEE 1500. Стандарт вимагає імплементації мінімального набору структур: 1) Регістр граничного сканування (Wrapper Boundary Register, WBR) діє як boundary scan ланцюг на рівні IP-core. 2) Буферний регістр (Wrapper Bypass Register, WBY), який звичайно складається з одного тригера, що дозволяє направляти тестові дані до іншого IP-core через інтерфейс. 3) Регістр команд (Wrapper Instruction Register, WIR), що приймає кодове слово для активізації одного з можливих регістрів сканування шляхом перемикання мультиплексорів. На рис. 1.9 показані ланцюжки сканування, додані в IP-core, які підключені до системи й додатковому паралельному порту для організації тестового доступу.

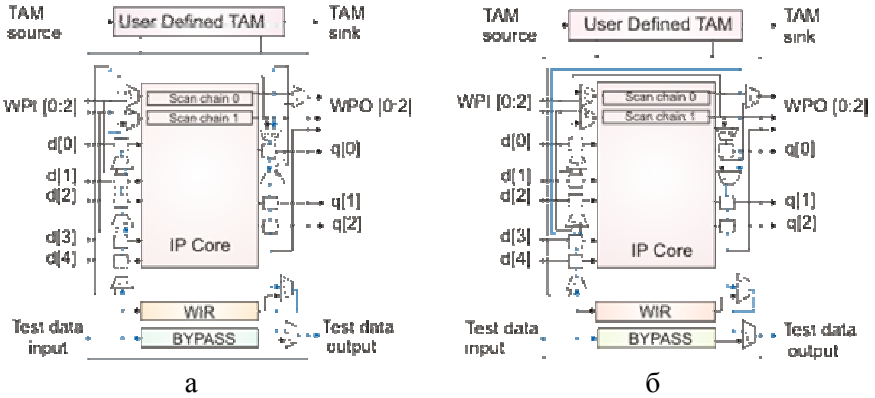


Рис. 1.9. Boundary scan інтерфейс для тестування SiP: а – режим повного доступу до ланцюжка граничного сканування; б – режим тестування міжз'єднань кристалів

Представлена архітектура інтерфейсу забезпечує два режими функціонування, орієнтованих на тестування SiP: перший (рис.1.9,а) - надає повний доступ до ланцюжка граничного сканування, коли є можливість тестування міжз'єднань IP-cores усередині підсистеми на кристалі до й після складання SiP; другий (рис.1.9,б) – орієнтований винятково на тестування міжз'єднань між кристалами й надає доступ тільки до осередку граничного сканування на первинних входах і виходах підсистеми на кристалі. Для представленого приклада лінії: d[3] – d[4] і q[1] – q[2]. Описана конфігурація повторюється для всіх IP-cores підсистеми на кристалі, забезпечує виконання функціональностей інтерфейсу IEEE 1500, що оточує кристал у цілому, без введення додаткового апаратного забезпечення або погіршення характеристик. Вибір конфігурації осередку граничного сканування – завдання рівня тестування пакетованої системи. Незважаючи на те, що розроблений раніше кристал, інтегрований у систему, може згодом втратити структури тестового доступу, що відповідають стандарту IEEE 1500, є можливість одержання доступу до верхнього рівня кристала шляхом використання осередку граничного сканування, що ставиться до іншого кристала системи. Це дозволяє застосовувати тестові набори, призначені для тестування з'єднань між кристалами на основі інтерфейсу IEEE 1500, включеного до проекту SiP.

Архітектура SiP надає велику розмаїтість можливих проектних рішень, у яких використовуються компоненти, виготовлені різними виробниками за різними технологіями. Навіть для кристалів, що задовольняють тестовим структурам стандарту IEEE 1500, множина додаткових факторів може впливати на розробку тестових програм. Залежно від доступного автоматичного тестового встаткування, конфігурації проекту й вимог до нього, може бути використана стратегія з одним або декількома проходами. Для прискорення тестової інтеграції в процесі розробки SiP можуть використовуватися програмні засоби автоматичної генерації тестів. Основні

параметри тестових програм: 1) перевірка сумісності описаних тестових архітектур і існуючих обмежень на доступні ATE і споживану потужність у процесі тестування; 2) швидка оцінка часу тестування SiP для заданої конфігурації тесту; 3) автоматична генерація тестових програм для SiP заданою мовою, включаючи форми сигналів, які отримані й передані в ATE для виконання тесту. Основна перевага використання сервісного програмного забезпечення полягає в можливості автоматичної генерації тестових програм на рівні кристала або системному рівні на основі описів компонентів.

Стратегія тестування SiP (рис. 1.10) припускає виконання кроків:

1. Реалізація індивідуальних тестів для кожного кристала, що входить у SiP. На цьому кроці, відомому як «wafer test» – тест перевірки логічної схеми безпосередньо на пластині, виявляються дефектні кристали, які будуть приводити до неправильного функціонування системи. Тут також виконуються навантажувальні випробування, такі як наробіток на відмову й остаточну ідентифікацію еталонних кристалів. Звичайно цей крок виконується виробником кристала. Якщо ж на підложці встановлюються активні компоненти, виробник повинен виконати заключне складання на еталонній підложці. При цьому виконується процедура тестування квазіпакету кристалів, мета якої – зменшити складність тесту підложки, що звичайно виконується на заключному виробничому тестуванні.

Додаючи проміжні тестові програми до процесу інтеграції SiP, можна виявити дефектні елементи перед заключним виробничим тестом. Прикладом може бути вхідне тестування підложки, що виконується на виробничому встаткуванні. Однак на практиці воно майже не застосовується через ускладнення виробничого процесу й використання різнорідного встаткування на виробничій ділянці. Незважаючи на те, що SiP складається зі наперед придатних компонентів, остаточний продукт вимагає виконання додаткової фази тестування.

Рис. 1.10. Стратегія тестування системи в пакеті

2. Тестування міжз’єднань кристалів дозволяє перевірити коректність зв'язків еталонних компонентів. Вибір тесту для міжз’єднань залежить від технології SiP, оскільки різні виробничі технології приводять до деяких відмінностей у несправній поведінці.

3. Післяпакувальне (post-packaging) тестування виявляє проблеми, які є результатом процесу впакування. На цьому кроці необхідно повторно тестувати кожний кристал, що містить внутрішні структури. У даних умовах використання стандартних структур дозволяє гарантувати доступність для тестування компонентів SiP. Післяпакувальний тест є багатопрохідним і виконується в критичних умовах, наприклад, при високій температурі. Отримана інформація може бути використана при визначенні правил виробничо-орієнтованого проектування SiP (Design-for-Manufacturability, DFM).

4. Повний тест SiP включає підсумкову перевірку працездатності. Мета – визначити остаточну якість системи. Тест SiP забезпечує також зворотні зв'язки із процесом удосконалювання DFM для рівня SiP. Тестування чистих кристалів (bare-die testing) виконується на стандартній підложці й забезпечує інформацію про кореляцію виробничих помилок з електричними несправностями. При цьому необхідно сконцентрувати увагу на можливих

дефектах. Наприклад, міжз’єднання кристалів можуть бути причиною затримки сигналу; з'єднання зварюванням (пайкою) можуть приводити до небажаних електромеханічних несправностей. У цих випадках контактна площадка або кристал, які є причиною відмови, можуть приводити до погіршення параметра yield. Проблема може бути чисто електричною (перехресні перешкоди) або механічною (недотримання мінімального зазору між елементами топології або інших технологічних обмежень).

5. Аналіз несправностей SiP – вирішальний етап розробки системи. Запропонована стратегія тестування забезпечує точну інформацію про дефекти в системі на різних виробничих етапах і дозволяє вивчити їхній вплив на якість продукту на кожному етапі окремо. Однак при аналізі несправностей SiP виникають додаткові труднощі. Наприклад, деякі дефекти виникають під впливом певних умов і не є відтвореними. Для активізації таких дефектів необхідно виконувати мікрозондування кристала аж до демонтажу елементів. Інші труднощі мають відношення до спостерігаємості. Наявність спостережуваних елементів на окремому кристалі є майже обов'язковим. При додаванні зазначених елементів до енергонезалежної пам'яті IP-core також стає спостережуваним, але для інших кремнієвих технологій необхідні додаткові засоби забезпечення спостерігаємості.

Як приклад в [4] розглянута SiP, що розроблена для супутникового зв'язку. Архітектура SiP включає підсистему на кристалі для прямої немодульованої (base-band) передачі, що розроблена за 90nm технологією, і один або два додаткових елементи пам'яті: буферна пам'ять обсягом 128-Mbyte, dynamic RAM і енергонезалежна (E2PROM). До складу системи входять 4 кристали, що реалізують RF інтерфейс і перетворення інформації (рис. 1.11).

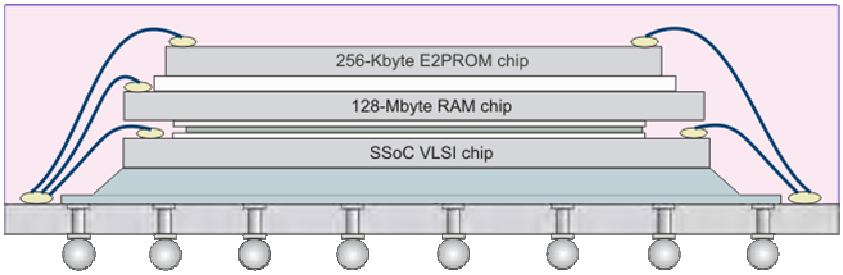


Рис. 1.11. Система в пакеті для супутникового зв'язку

При тестуванні системи були використані рішення, описані вище, і засоби автоматичної генерації тестових програм на всіх етапах виробничого тестування SiP. Тестування підсистем на кристалі виконувалося через порт тестового доступу (TAP). Для структурного тестування SoC необхідна пам'ять великого обсягу, а також аналогові засоби функціональної перевірки убудованих функцій обробки сигналів. На рівні підложки для тестування SoC за допомогою тестера Agilent 93K потрібно 7 секунд на один кристал. Зазначений час збільшується на 25%, якщо система на кристалі включає RF

компоненти. Більше того, система в пакеті піддається випробуванням при різній температурі від 25 до 125°C. Для тестування елементів пам'яті SiP були використані стандартні засоби тестування й DFT. Для пам'яті E2PROM тестування на рівні підложки виконувалося на спеціальному обладнанні з використанням тестера Credence Kalos. Тестування DRAM здійснювалося з використанням стандартного автономного тестера серії Agilent V54 або Advantest. Для розглянутої системи був використаний тестер Agilent 93K, у якому реалізований метод паралельного тестування, заснований на використанні мультипотокового обміну даними. Метод дозволяє подавати кілька незалежних тестів на різні групи виводів паралельно, а також ізолювати окремі кристали при тестуванні або тестувати їх окремо, використовуючи стандартну інфраструктуру.

1.5. Висновки

1. Мільярди компактних комунікаційних, обчислювальних виробів і пам'яті, що використовуються у всіх сферах людської діяльності, дозволяють зробити висновок про повне домінування цифрових систем у пакетах (кристалах) на ринку електронних технологій уже в найближчі п'ять років.

2. Еволюційний шлях перетворення SoC в SiP являє гарний приклад для адаптації існуючих технологій тестування цифрових систем на кристалах до нового конструктивного покоління – цифрових систем у пакетах.

3. Пакет кристалів формує також спектр нових завдань у рамках створення інфраструктури сервісного обслуговування SiP, пов'язаний з тестуванням міжз'єднань кристалів, підложки, а також системи в цілому.

4. Актуальною для ринку електронних технологій представляється проблема створення моделей і методів інфраструктури сервісного обслуговування цифрових систем у пакетах (кристалах) на основі використання стандартів тестопридатного проектування.

1.6. Література

1. Spartan-6 Family Overview. Advance Product Specification.- XILINX.- February 2, 2009.- www.xilinx.com/

2. Stratix IV Device Handbook. - Altera.- 2009. www.altera.com/support

3. Yervant Zorian. Tutorial on EWDTS 2007. Embedding Infrastructure IP for SOC Yield Improvement // Proceedings of the 39th Design Automation Conference.- New Orleans, LA, USA.- 2002.- P. 709-712.

4. P.P. Pande, C. Grecu, A. Ivanov, R. Saleh. Destination network-on-chip // EDA Tech Forum. University of British Columbia, chart the path from SoC to NoC.- June, 2005.

5. IEEE 1500 Web Site. http://grouper.ieee.org/groups/1500/.

6. Appello D., Bernardi P., Grosso M., Reorda M.S. System-in-package testing: problems and solutions // IEEE Design & Test of Computers.- Vol. 23, Issue 3.- May-June, 2006.- P. 203-211.

7. Amkor Technology Web Site. http://www.amkor.com

8.Horgan.SiPorSystem-in-Package.-EDACafe.-September29,2008.- http://EDACafe.com

9.PaulCollander,PekkaLaukkala.SysteminPackageversusSystemonChip//IMN,NokiaNetworks.- Espoo, Finland.- 2000.

10. Zorian Yervant. Test Strategies for System-in-Package // The Plenary Paper of IEEE East-West Design & Test Symposium (EWDTS'08).- Lviv, Ukraine.- October 9-12, 2008.

11. D. Soldo, R. Myoung. System In Package (SiP) and Stacked Package Solutions // Leading Insight, Ansoft. Application Workshop for High Performance Design. - Los Angeles. -2006.

12. L. Smith. 3D Packaging Applications, Requirements, Infrastructure and Technologies // Fourth Annual International Wafer-Level Packaging Conference. - San Jose, California. - September, 2007.

13. The next Step in Assembly and Packaging: System Level Integration in the package (SiP) / Editors: William Chen, W. R. Bottoms, Klaus Pressel, Juergen Wolf // SiP White Paper V9.0.- International Technology Roadmap for Semiconductors. - 2007.

14. Onishi T. SiP Technology Development in Japan // 11th Annual International KGD Packaging and Test Workshop. - Sept., 2004. - Napa, California.

15. Carpenter K., Vardaman J. SiP Emerges. - 2006. - http://www.circuitree.com

16. Masanori Natsuaki, Koichi Nakamura, Akira Takashima. Fujitsu's highly integrated assembly technology for SiP // International Workshop on SiP/SoC Integration of MEMS and Passive Components with RF-ICs. - March, 2004. - P. 13-14.

17. Lenihan T.G., Jan Vardaman E. Worldwide Perspectives on SiP Markets: Technology Trends and Challenges // 7th International Conference Electronic Packaging Technology, ICEPT'2006.- 2006.- P. 1-3.

18. Vardaman E. J., Carpenter K., Matthew L. System-in-Package. The New Wave in 3D Packaging // TechSearch International, Inc.- 2005.- www.techsearchinc.com.

19. Vardaman E. Jan. Trends in 3-D Packaging // TechSearch International, Inc.Technology Licensing and Consulting. - 2004. - www.techsearchinc.com.

20. Karnezos M. Package-in-Package (PiPTM) - 3D Stacked-Package Module // 11th KGD Packaging & Test Workshop. - USA. - Sept., 2003. - http://www.gooddie.net

21. Boettcher L., Manessis D. Embedding of Chips for System in Package realization - Technology and Applications // The 3rd IMPACT and 10th EMAP Joint Conference. - October, 2008.

22. Beyne E. 3D System Integration Technologies // International Symposium VLSI Technology, Systems, and Applications. -April, 2006. - P. 1-9.

23. Balachandran J., Brebels S., Carchon G., Kuijk M., Walter De Raedt, Nauwelaers B.K.J.C., Beyne E. Wafer-level package interconnect options // IEEE Transactions Very Large Scale Integration (VLSI) Systems. - June, 2006. - Volume 14, Issue 6. - P. 654-659.

24. Pieters P., Beyne E. 3D Wafer Level Packaging Approach Towards Cost Effective Low Loss High Density 3D Stacking // 7th International Conference Electronic Packaging Technology, ICEPT'2006. - Aug., 2006. - P. 1-4.

25. Vardaman J. 3-D Through-Silicon Vias Become a Reality // Semiconductor International.- June, 2007. - www.semiconductor.net/article/CA6445435.html

26. Savastiouk S., Siniaguine O., Reche J., Korczynski E. Thru-silicon interconnect technology // Twenty-Sixth IEEE/CPMT International Electronics Manufacturing Technology Symposium. - 2000. - P. 122-128.

27. Di Stefano Th. Wafer-Level Paradigm for Burn-in and Test // Burn-in & Test Socket Workshop. - Mesa, Arizona. - March, 2001.

28. Wolf M.J., Ramm P., Klumpp A., Reichl H. Technologies for 3D Wafer Level Heterogeneous Integration // DTIP. - Nice, France. - 2008.

29. Keigler A., O'Donnell K., Liu Z., Wu B. Enabling 3-D Design // Semiconductor International.- Issue 9, August.- 2007.- www.semiconductor.net

30. Kripesh V., Yoon S.W., Ganesh V. P., Khan N., Rotaru M.D., Fang W., Iyer M.K. Three-Dimensional System-in-Package Using Stacked Silicon Platform Technology // IEEE Transactions on Advanced Packaging.- Vol. 28, No. 3.- 2005.- P. 377-386.

31. Литвинова Е.И. Технологии встроенного тестирования System-in-Package // Радиоэлектроника и информатика.– 2008.– № 3.– С. 37-44.

32. Литвинова Е.И. Технологии встроенного ремонта компонентов System-in-Package // АСУ и приборы автоматики.– 2008.– № 145.– С. 40-48.

33. Литвинова Е.И. Технологии диагностирования и восстановления System-in-Package // АСУ и приборы автоматики.– 2009.– № 146.– С. 4–21.

2. АРХІТЕКТУРИ КОМПОНЕНТІВ ДЛЯ ПРОЕКТУВАННЯ ЦИФРОВИХ СИСТЕМ

Представлено стан ринку електронних технологій, закони та перспективи його розвитку в частині апаратної реалізації цифрових систем на кристалах. Описано основні структури силіконових кристалів, використовуються для реалізації проектів у сучасних середовищах проектування. Показано переваги та недоліки технології Electronic System Level Design (ESL), а також цікаві рішення системного рівня створення прототіпа на основі Transaction Level Modeling (TLM), впроваджувані усіма компаніямі світової топ-десятки в процес проектування цифрових виробів.

2.1. Стан ринку електронних технологій

У теперішній час продукція, що поставляється на ринок електронних технологій, використовується в усіх галузях людської діяльності. Компанії Intel, Motorola, Advanced Micro Devices, IBM, Sun Microsystems і Hewlett-Packard затрачають мільйони доларів на розробку потужних процесорів, що оперують вже 64- і 128-бітовими даними. Щільність розміщення транзисторів у кристалах мікросхем значно перевищила 200 млн на 1 см2 [1]. У мікропроцесорів, як і у інших типів мікросхем, спостерігається експоненціальне зростання можливостей, відповідно до закону Мура [2]. З такою ж періодичністю згадані кристали збільшують свою продуктивність на 100%. Це перетворило розробку універсальних комп'ютерів в потужну лідируючу індустрію на ринку електронних технологій. Однак ситуація у теперішній час змінюється у зв'язку з ростом продуктивності середніх і малих мікропроцесорів. Очевидна ринкова тенденція - протягом найближчих 15 років можливості, пропоновані технічним прогресом і ринком, перевищать потреби користувача. Тому корпорації, націлені тільки на випуск потужних універсальних процесорів, ризикують втратити свою частку прибутку на ринку мікроелектроніки. Щоб цього не сталося, вони реорганізують частину свого виробництва на швидкий випуск замовних (customized) мікросхем.

Потужні процесори застосовуються для тривимірних відеоігор, редагування цифрових відеофайлів і speech-to-text трансляторів. Але в найближчому майбутньому такі завдання будуть вирішуватися спеціалізованими мікропроцесорамі. У більшій частині користувачі застосовують комп'ютери як текстові процесори для доступу до E-mail, Internet. Тому використання для цих цілей високопродуктивних процесорів Itanium і Pentium, Intel, Athlon, Advanced Micro Device, що виконують мільярди операцій з плаваючою комою в секунду, у десятки і сотні разів надлишково. З іншого боку, надійність стає основним показником конкурентоспроможності, за яким виділяються фірми IBM, Compaq і Hewlett-Packard, які отримують основний прибуток на ринку електронних технологій. Другим за значимістю зараз є показник спеціалізованої орієнтації на вирішення конкретних завдань - customization, запропонований компанією Dell. Ринкова конкуренція змушує виробників переходити від ідеї підвищення продуктивності до удосконалення властивостей, за які користу-

вач готовий платити додаткові гроші, якщо можливості фірми перевищують потреби ринку. Цікаві - надійність, зручність і замовні властивості. Такі зміни переживають і інші галузі промисловості: автомобільна, телекомунікаційна, побутова електроніка.

Розробники мікропроцесорів пропонують властивості, що перевищують потреби користувачів. Наприклад, виробники мікросхем створюють більше транзисторів на кристалі, ніж це потрібно для проектувальників комп'ютерів. Ще у 1996 році компанія National Technology Roadmap for Semoconductor звернула увагу на факт, що виробництво транзисторів щорічно збільшується на 60%, при цьому їх використання у нових проектах зростає тільки на 20%. З цієї причини і в області мікроелектроніки відбувається переорієнтація коштів на можливості реалізації тільки необхідних і замовних властивостей. Розвивається модульність пристроїв, у яких зберігається стандартизація інтерфейсів між підсистемами. Такий підхід вже реалізований в нижніх рівнях проектування мікропроцессору у вигляді системи на кристалі (System-on-chip або SoC), що складається з багаторазово використовуваних IP-(Intellectual Property)-модулів. Останні мають різні розміри і функції, від простих, що реалізують інтерфейси, до складних 64-бітових мікропроцесорів, які мають декілька мільйонів транзисторів. Компанії (Tensilica, ARC Cores, Hewlett-Packard і STMicroelectro-nics) почали пропонувати вибір компонентів і властивостей у саміх мікропроцесорах. Вказати необхідну множину команд можна через Web-сайт за допомогою мови високого рівня, такого як Verilog. Сімейство Hewlett-Packard HP / ST Lx дозволяє визначити число команд і операцій, виконуваних паралельно, а також об'єднати Digital Signal Processing з іншими корисними для людини функціями на одному кристалі. Таким чином, прогрес в мікроелектроніці диктує компаніям: 1) дотримуватися стратегії модульності проектів, що складаються з багаторазово використовуваних і реконфігурованих IP блоків та кристалів; 2) надавати можливість реалізації системи на одному кристалі або в пакеті, створюючи замовні системи SoC; 3) скорочувати енергоємність, час проектування та впровадження у виробництво мікропроцесорів і систем SoC; 4) надавати широкі можливості для тестопридатного проектування і верифікації цифрових систем, істотно зменшують час створення готового виробу.

2.2. Мобільний спеціалізований цифровий виріб

Вже близько п'яти років існує стійка тенденція на ринку електроніки до створення портативних цифрових виробів, призначених для виконання необхідних для людини функцій. Вони об'єднуються в поняття PDA (personal digital assistant) - персональний цифровий помічник: мобільний телефон, MP3-програвач, оглядач електронної пошти, цифрова камера, аудіосистема, супутникове радіо, переносний телевізор, ігрова приставка. Виникає питання, чи можна створити універсальний цифровоий виріб (universal digital assistant) з перерахованими вище функціями, який автоматично налаштовується на протокол передачі, прийому і перетворення інформації; самостійно виконує upgrade конфігурації; тривалий період часу працює автономно, на батарейках [3].

Ядром всіх сучасних цифрових виробів є мікропроцесор. За останні 25 років він став доступний не тільки комп'ютеру, але вже впровадився в пральні машини, цифрові камери, НВЧ-печі і тостери. Однак будь-який найпотужніший мікропроцесор програє за продуктивністю або швидкодією спеціалізованій мікросхемі. Сучасні мікропроцесори можуть виконувати мільярди інструкцій в секунду з частотою синхронізації 4 ГГц. Це на три порядки вище швидкодії перших мікропроцесорів у персональних комп'ютерах зі споживною потужністю 80 Вт і більше. При цьому розсіювана потужність на 1 см2 у останніх була в п'ять разів вище, ніж на поверхні кухонної електроплити. Але все це не мало великого значення, поки комп'ютери були стаціонарними і підключалися до електричної мережі. Високий попит на портативні комп’ютери змусив виробників більше приділяти уваги зниженню споживної потужності мікропроцесорів [4]. При цьому акцент проектування змістився з оцінки продуктивності на одиницю площі до продуктивності на 1 Вт. Мікропроцесор, з точки зору витрат електроенергії, став неефективним. Тому він втратив позицію пристрою, що виконує всі основні функції, і перейшов в розряд керуючого ядра. Мультипроцесорні структури від Intel (multicore architecture) є не тільки підвищення продуктивності, а й революційне рішення пр-блеми енергозбереження.

Виходячи з пропозицій ринку мікроелектроніки, найбільш ефективними з позиції швидкодії є: ASIC - Application-Specific Inegrated Circuit - спеціалізовані мікросхеми, розроблювані однією фірмою для конкретного виробу; ASSP - Application Specific Standard Product - спеціалізований пристрій, який може бути використаний багатьма розробниками. Наприклад, Analog Devices, Infineon, Motorola, Qualcomm і Texas Instrument випускають мікросхеми ASSP, які будь-який виробник може використовувати для створення, наприклад, мобільних телефонів. Такі мікросхеми можуть бути в тисячі разів ефективніше за вартістю і продуктивністю на 1 Вт, ніж мікропроцесори. Незважаючи на згадані переваги, спеціалізовані мікросхеми не мають гнучкості реконфігурування. Внаслідок високої вартості масок кристала розробка і випуск нової мікросхеми коштує до 1 мільйона доларів. Свого роду компромісом є програмовані логічні інтегральні схеми (ПЛІС або PLD), що мають продуктивність спеціалізованих мікросхем і гнучкість мікропроцесорів. Такі вироби складаються з двох шарів: перший містить конфігуруємі для конкретної функції лінії і елементи, другий - пам'ять (ре-) конфігурації або структури цифрового виробу.

Якщо розглядати тільки перепрограмовані мікросхеми, що використовують як пам'ять реконфігурації SRAM, то тут структура функцій проекту завантажується один раз в момент ініціалізації, після включення живлення. Окремі чіпи надають можливість для неповної реконфігурації, коли частина елементів і зв'язків може бути змінена, в той час як решта залишаться незмінною. Інтерфейсна мікросхема повинна мати можливість адаптації до різних протоколів без впливу на інші її функції. Цікаві для розробників цифрових пристроїв і динамічно реконфігуровані чіпи [5-7], коли будь-які зв'язки та елементи можуть змінюватися під час роботи мікросхеми. Але й такі

кристали вже не підходять для реалізації універсального цифрового пристрою. Вони повинні підвищити свою швидкодію і знизити енергоспоживання. При цьому за свою гнучкість реконфігурації ПЛІС розплачуються деяким зменшенням продуктивності: 20 транзисторів програмованого кристала припадає на один транзистор ASIC. Пам'ять реконфігураціі може займати до 70% загального числа транзисторів мікросхеми. Довгі лінії зв'язку вентилів і транзисторні програмовані коммутаційні матриці значно уповільнюють швидкість виконання операцій в PLD. На практиці складний цифровий пристрій, реалізований на ПЛІС, являє собою виріб, що займає велику площу на кристалі, з порівняно низькою швидкодією і високою споживаною потужністю. До того ж воно може бути досить дорогим (близько $ 1000), що робить PLD неефективним для створення портативних комп'ютерів.

Компанії Altera і Xilinx, займаючись усуненням позначених недоліків, розширили можливості PLD з метою забезпечення їх конкурентоспроможності з мікропроцесорами і сигнальними процесорами. Останні розробки ПЛІС з високошвидкісними вхід-виходами, блоками пам'яті, помножувателями і вбудованими мікропроцесорами становлять серйозну альтернативу потужним стаціонарним виробам. На жаль, ці додаткові властивості істотно збільшують споживану потужність і роблять їх непридатними для створення портативних пристроїв. Тим не менше, тільки PLD в найближчому майбутньому нададуть можливості для розробки універсального цифрового пристрою. Для цього необхідно скоротити надмірність входів і з'єднань, прискорити і спростити процес повної або часткової реконфігурації. У цьому випадку функції управління залишилися резидентними, решта завантажуються в міру потреби. Слід також помітити, що компанії-виробники мікросхем намагаються зберегти статус-кво для користувачів, тому вони не прагнуть кардинально міняти архітектуру чіпів. Процес просування на ринок нової мікросхеми складний, пов'язаний з певним фінансовим ризиком і вимагає великих матеріальних витрат, що можуть собі дозволити тільки великі компанії, такі як Intel, Motorola, NEC, Nokia і Texas Instruments, що займаються розробками в області реконфігурованих процесорів і цифрових систем.

Цікавий напрямок підвищення ефективності використовуваної площі кристала пов'язаний з динамічною реконфигурацією функцій. У пам'яті находяться описи декількох корисних функцій, кожна їх яких може бути налаштована на кристалі за допомогою супервізора за один цикл. При цьому здійснюється економія енергоспоживання, завдяки виконанню в конкретний момент тільки однієї функції. Крім того, часткова (повна) динамічна реконфигурація дозволяє підвищити ефективність використання площі кристала в припущенні, що всі функції будуть затребувані на тривалому часовому інтервалі. Компанії-розробники (start-up companies), такі як QuickSilver, Elm Technology, IBM, Matrix Semiconductor, Tezzaron Semiconductor, Ziptronic, працюють над створенням багатошарової мікросхеми для портативних пристроїв, в якій внутрішні з'єднання виконуються тисячами вертикальних ліній. Це дозволить значно знизити споживану потужність і підвищити

швидкодію виконання функцій. Інший шлях удосконалення ПЛІС - заміна SRAM-пам'яті, яка вичерпала себе та що має високу споживану потужність, на ефективнішу, енергонезалежну і швидкодіючу [8]. Найбільш перспективними є: магніторезистивна пам'ять, ферроелектрична і пам'ять на аморфних напівпровідниках. Використання такої пам'яті дозволить також поліпшити і функції захисту інформації при відключенні енергоживлення. Таким чином, PLD дозволяє вирішити завдання створення портативного універсального цифрового пристрою. При цьому основною функцією мікропроцесора буде управління [9].

2.3. Закони розвитку ринку електронних технологій

За час, що минув між відкриттям електрона і появою кристалів, які містять більше 500 мільйонів транзисторів на кристалі, було зроблено множина математичних та евристичних спостережень, змістовних висловлювань та стійких прогнозів. Однак 90% всіх тверджень (законів) - констатує письменник наукової фантастики Теодор стежа (Sturgeon's Law) - є повною нісенітницею. Найвідомішим із них є закон, сформульований військово-космічним інженером Едвардом Мерфі (Edward A. Murphy Jr.): Якщо існують дві або декілька можливостей, і одна з них може привести до катастрофи, то хто-небудь її вибере. Півстоліття активного використання напівпровідникових приладів призвело до створення емпіричних законів, п'ять з яких - у достатній мірі затребувані на ринку [10].

Закон Мура (Moore's Law). Число транзисторів у мікросхемах щорічно подвоюється. Закон Мура був представлений в документах компанії Intel 38 років тому Гордоном Муром (Gordon E. Moore). Ще в 1965 році він зазначив, що немає технічних перешкод для удосконалення існуючих технологій (фотолітографії) і передбачив, що через 10 років число транзісторів у мікросхемах досягне 65000. Однак до кінця 1975 мікросхеми мали чи десяту частину передвіщеного обсягу. Період часу для подвоєння числа транзисторів насправді склав 17 місяців. До 1985 року даний інтервал досяг 22 місяців, а до 1995 року - 32 місяці. Потім намітилося скорочення періоду до 18-24 місяців, який дійсний і в даний час. Фахівці компанії Intel оцінюють теперішній період часу подвоєння числа транзисторів в мікросхемі, рівний 18 місяцям. Економічна основа закону Мура: вартість компонента для простих схем зворотно пропорційна їх числу. Але зменшення вартості припиняється при збільшенні складності схем. Таким чином, може настати момент, коли число транзисторів у схемі нарощувати економічно недоцільно. Тому якщо закон Мура перестане діяти, то це відбудеться з економічних, а не з технічних причин. Експоненціальна тенденція зростання продуктивності і числа транзисторів в мікросхемі не може існувати постійно, але її зміна може бути відстрочено. За песимістичними прогнозами напівпровідники будуть переважати, принаймні, найближчі 15 років. Потім очікується перехід до нової технології, яка може не мати означених економічних властивостей (напівпровідників), проте це відбудеться не скоро. Використання закону Мура також ставить питання: чи не змінюється його дія від опису до передбачення? Закон просто описує реальність чи впливає на майбутнє? Так чи

інакше, але електронна промисловість використовує закон для складання планів розвитку (roadmap) і прогнозування конструктивних особливостей чіпів. Це - мінімальна ширина ліній, максимальний розмір пластини (wafer), точність інструментів, чистота виробничих приміщень. Виробники напівпровідникових пристроїв і надалі будуть збільшувати число транзисторів на кристалі до тих пір, поки це економічно доцільно і дозволено для існуючих технологій виробництва мікросхем.

Закон Рока (Rock's Law). Вартість обладнання для випуску напівпровідникових кристалів подвоюється кожні 4 роки. Іноді його називають другим законом Мура, проте сам він приписує авторство Артуру Року (Arthur Rock), інвестору Intel. Проте, на практиці виявилося, що вартість обладнання залишається постійною, а зростає його продуктивність в процесі модернізації, яка вимірюється числом випущених на годину пластин (wafer) (20 штук - 1990 роки, 60-90 - наші дні). Завдяки зрослої продуктивності без збільшення вартості обладнання ціна компонента на кристалі змінилася від 1 цента за один транзистор до 1 долара за мільйон. Як це впливає на вартість персонального комп'ютера?

Закон Макроуна (Machrone's Law). PC, який ви хочете купити, завжди коштує $ 5000. Закон сформульований в 1984 році Біллом Макроуном (Bill Machrone), оглядачем журналу PC Magazine. Побудувавши графік вартості комп'ютерів з 70-х по 90-і роки, він звернув увагу, що їх ціна в зазначеному часовому інтервалі залишається постійною і рівною $ 5000. В даний час ця цифра зменшилася до $ 3000. Зараз прийнятний для роботи комп'ютер з повною конфігурацією і периферією коштує близько $ 1000. Зменшення вартості комп'ютерів призвело до значного збільшення числа користувачів PC. При цьому кожна нова машина подключається до локальної мережі або до Internet. Які переваги дає мережа для користувача?

Закон Меткафа (Metcalfe's Law). Зростання значущості мережі визначається квадратом числа її користувачів. Сформульований Робертом Меткафом (Robert Metcalfe), винахідником стандарту Ethernet і засновником компанії 3Com Corporation у 1980 році. На відміну від попередніх тверджень, цей закон не може бути визначений кількісно, оскільки значимість або цінність інформації важко оцінити числом. Вона різна для конкретних користувачів. Небажані для вас клієнти (contaminant) знижують значимість мережі. Такими можуть бути маркетингові фірми, що поширюють непотрібну інформацію, spam e-mail або несправні вузли, порушують маршрутизацію і призводять до втрати переданих даних. В даний час відзначені 4 проблеми мереж: 1) насиченість (saturation) - нові її клієнти не можуть принести в неї нову інформацію; 2) какофонія (cacophony) - непередбачуваність взаємодії між групами клієнтів в процесі телеконференцій, дискусій, що породжують сотні повідомлень; 3) кластеризація (clustering) - розбиття клієнтів на групи, кожна з яких використовує тільки частину мережі; 4) пошук (search) - стає все більш часовитратним і дорогим, тому більша частина информаційних багатств мережі залишається недоступною на практиці для користувача. Існують деякі шляхи часткового вирішення даних проблем: 1) підключати до мережі або-

нентів, що володіють цінною інформацією; 2) розділяти мережу на фрагменти, наприклад, за тематикою переданої інформації; 3) вилучати з мережі контаменанти (contaminant), що засмічують мережу, або фільтрувати генеруєму ними інформацію.

Закон Вірта (Wirth's Law). Зниження швидкості програмного забезпечення відбувається швидше зростання швидкості апаратури (Software is slowing faster than hardware is accelerating). Сформульований у 1995 році Ніклаусом Віртом (Niklaus Wirth), автором мови програмування Паскаль. Ця ж ідея раніше звучала в анонімному жарті на манер біблійного вірша: «Groves giveth, and Gates taketh away - що сад приніс, рот забрав». Завдяки Енді Гроуву (Andy Grove) швидкість апаратури зростає, а Білл Гейтс (Bill Gates) збільшує складність додатків, яка уповільнює їх швидкодію. Текстові редактори 70-х років вимагали 8000 байт ОЗП, а для сучасних аналогів необхідний обсяг пам'яті збільшився у 1000 разів. Ці програми не можна було б використовувати за відсутністю підвищення швидкодії «заліза» в тисячі разів. Слід пам'ятати, що більшість функцій програмних додатків є надлишковими і ніколи не будуть затребувані користувачем. Але чому він це терпить? Існують 2 причини: 1) здивування неосвіченого користувача обчислювальними здібностями програми, який не в змозі відрізнити корисні функції від даремних; 2) зацікавленість компаній, що створюють програмне забезпечення. Корисні властивості, такі як колір, що випадають іконки роблять програми доброзичливими, тому вони не потребують складної документації. Навіть непрофесіонал може навчитися працювати з такою програмою, але, зробивши це, він попадається на споживчий гачок на довгий час. Крім того, вимога до більш швидкого випуску програми на ринок не дає програмісту часу на її ретельне доопрацювання, в тому числі, і з метою зменшення її розміру.

Звичайно, евристичні закони і правила не обмежуються п'ятьма згаданими. Ще один, що стосується пропускної здатності Internet - закон Якоба Нільсена (Nilsen's Law) - був опублікований у 1998 році. Він говорить, що доступна для професійних користувачів пропускна здатність мережі зростатиме на 50% щорічно, але розробники Web-сайтів не отримують цієї переваги, яка буде компенсуватися збільшенням числа клієнтів і уповільненням програмного забезпечення.

Підтвердженням життєвості наведених законів може служити дослідження, проведене провідним світовим експертом в області Design & Test доктором Є. Зоріаном (Yervant Zorian, Virage Logic). Займаючись аналізом ринку та виробництва цифрових і мікропроцесорних систем, він приходить до наступних узагальнень [11]: 1) Кількість транзисторів на кристалі зростає швидше, ніж їх необхідний обсяг для проектування складних цифрових систем. 2) Інвестиції в технологічний процес перевагають над капіталовкладеннями у створення нових цифрових і мікропроцесорних систем. 3) Матеріальні витрати на розробку програмного забезпечення вбудованої системи (embedded system) складають 80% від її вартості. 4) Витрати на тестування мають експоненційну залежність від обсягу (вартості) випуску продукції. 5) Число інженерів-верифікаторів (тестерів) не менше, ніж у 2 рази перевищує

кількість проектувальників цифрових і мікропроцесорних систем. Кожна теза має методологічне значення для перспектив розвитку радіоелектронного виробництва і ринку країн, що розвиваються, оскільки через два-три роки їх компанії з необхідністю повинні враховувати дані тенденції в плануванні своєї діяльності.

2.4. Універсалізація чи спеціалізація?

На основі часткового аналізу ринку електронних технологій, який стосується проектування і виробництва цифрової та мікропроцесорної техніки, можна сказати, що кожен з пристроїв (мікропроцесор, PLD, ASIC) є цілком самодостатнім. За необмежений час на кожному зі згаданих кристалів можна реалізовувати цифрову систему. Але проблема полягає в тому, щоб потім її можна було вигідно продати. І тут вступають в силу критерії, що впливають на кінцевого споживача: енергоспоживання, швидкодія, корисні властивості, час проектування, вартість виробу, його реконфігурованість. Дані критерії не претендують на повноту, але вони істотно впливають на циклічні зміни вектора уподобання ринку убік універсалізації або спеціалізації цифрових систем. Можна припустити, що якісно графік розвитку ринку за згаданими поняттями описується спіральною моделлю, представленою на рис. 2.1. На кожному наступному циклі або часовому інтервалі, рівному восьми рокам, ринок має більш високий рівень універсалізації або спеціалізації по відношенню до попереднього періоду. Слід вважати, що фаза теперішнього часу відповідає активізації ринку, спрямованої в бік спеціалізації (третя чверть на спіральній моделі розвитку ринку), оскільки фірми, що випускають універсальні комп'ютери, за останні три роки терплять суттєві збитки.

Рис 2.1. Спиральна модель розвитку ринку

Інший цікавий висновок пов'язаний з остаточним захопленням ринку технологією спільного проектування апаратної та програмної частин цифро-вої системи (рис. 2.2). При цьому особливостями сучасних систем проектування є наступні методології [12-17].

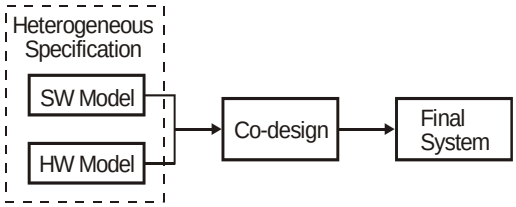


Рис 2.2. Гетерогенна модель процесу сумісного проектування

1) Підтримка декількох способів формалізації специфікацій - диверсифікація, що дає можливість: 1) застосовувати метод опису, який є більш прийнятним для адекватного зображення функцій фрагмента проекту; 2) використовувати як компоненти вже веріфіковані моделі та IP-core, мова опису яких відрізняється від застосованої у поточному проекті.

2) Можливість проведення валідного аналізу поведінки компонентів в реальному масштабі часу і повторне застосування верифікованих проектів (design reuse).

3) Наявність кількох методів створення специфікацій необхідно для проектування складних систем, що складаються з гетерогенних компонентів зі змішаними властивостями: аналогово-дискретні, програмно-апаратні, електронно-механічні. Це необхідно і для розробки електронної системи декількома групами проектувальників, які використовують різні методи формалізації специфікацій або раніше створених моделей компонентів, поданих іншими мовами опису.

4) Існуючі засоби hardware-software co-design системного рівня використовують широкий спектр мов опису, серед яких домінують C ++, VHDL, Verilog, System Verilog, System C.

5) Загальні для всіх провідних фірм стадії розробки: створення неформальної моделі всієї системи в цілому; формалізація специфікації проекту; розробка архітектури та розподіл системи на фрагменти, заснована на наявності конкретних технологій проектування та інженерному досвіді проектувальника; остаточний варіант архітектури системи є результатом вдалих і невдалих послідовно виконуваних ітерацій доопрацювання формальної специфікації; розробка hardware і software для компонентів, що складають проект, здійснюється паралельно або незалежно.

2.5. Архітектури систем на кристалах програмованої логіки

Розглядається елементна база для створення цифрових систем на кристалах у вигляді програмованих логічних пристроїв (Programmable Logic Devices - PLD). Пропонуються архітектурні рішення, орієнтовані на зменшення часу проектування цифрових виробів на основі використання сучасних систем автоматизованого проектування, мов опису апаратури, технологій верифікації, налагодження і діагностування таких пристроїв.

2.5.1. Вступ.

Проектування цифрових пристроїв (ЦП) на основі програмованої логіки надає цілий ряд нових можливостей фахівцям, пов'язанним з розробкою сучасних цифрових систем обробки інформації. Головною відмінною особливістю програмованої логіки є можливість їх гнучкого налагодження на виконання заданих функцій самим користувачем. Мікросхеми програмованої логіки вдало поєднують гнучкість замовних BIC і доступність стандартної логіки. Сучасні кристали PLD мають низьку вартість ($ 10), високу швидкодію (професійні серії мікросхем працюють на частоті 500 МГц), широкі функціональні можливості (один програмований чіп може замінити кілька сот корпусів традиційної «жорсткої» логіки), багатократність перепрограмування, низьку споживану потужність, компактність і мініатюрність.

Процес проектування ЦУ на основі НВІС програмованої логіки полягає у описі його функціонування на вхідній мові використованого програмного засобу, виконані автоматизованого синтезу, проведенні моделювання, верифікації та настройки архітектури кристала програмованої логіки. При цьому час розробки (time-to-market) навіть складних проектів може бути порівнянним з програмними проектами і становити кілька місяців (тижнів).

2.5.2. Розвиток мікросхемотехніки.

Становлення програмованої логіки почалося з появи програмованих постійних запам'ятовуючих пристроїв (ППЗП - Programmable Read Only Memory - PROM) на початку 70-х років. Спочатку PROM використовувалися виключно для зберігання даних, пізніше їх стали застосовувати для реалі-зації логічних функцій. Незручність використання PROM в якості логічних перетворювачів полягає в тому, що логічні функції перед записом в PROM необхідно приводити до досконалої діз'юктівной нормальній формі. Крім того, ємність PROM не дозволяла реалізувати функції великого числа змінних. Спеціально для реалізації систем булевих функцій великого числа змінних були розроблені програмовані логічні матриці (Programmable Logic Arrays - PLA). Їх можна вважати першими програмованими логічними пристроями (Programmable Logic Devices - PLD). Удосконалення архітектури PLA призвело до появи програмованих матриць логіки (Programmable Array Logics - PAL), які на довгі роки визначили найбільш популярну архітектуру PLD. Подальше вдосконалення технології виробництва інтегральних схем, підвищення ступеня інтеграції, успіхи у створенні корпусів з великим числом зовнішніх висновків на початку 90-х років привели до можливості реалізації на одному кристалі декількох PAL, поєднуваних програмованими з'єднаннями. Подібні архітектури одержали назву складних PLD (Complex PLD - CPLD) [18-19]. Основу всіх розглянутих вище пристроїв складають програмовані матриці. Тому їх ще називають програмованими логічними пристроями, що мають матричну структуру. Паралельно з PLD також розвивалися архітектури вентильних матриць (Gate Array - GA), або матриць логічних осередків (Logic Cell Array - LCA), у російськомовній літературі отримали назву базових матричних кристалів. Перші вентильні матриці були полузамовленими. Вони програмувалися під час виготовлення, що стримувало їх широке практичне використання. Однак в 1985 р. фірма Xilinx

випустила вентильні матриці (Field Programmable Gate Array - FPGA), які стали успішно конкурувати з CPLD [20]. В даний час спостерігається бурхливий розвиток архітектур CPLD і FPGA (щороку з'являються нові покоління цих пристроїв), їх зближення, придбання ряду загальних властивостей, зниження вартості, підвищення швидкодії і функціональної складності. Таким чином, тенденції розвитку архітектур програмованої логіки дозволяють припустити, що найближчим часом основу елементної бази цифрових систем будуть складати CPLD і FPGA, які інтегруються з вбудованими процесорами (жорсткими або гнучкими) для управління і компонентами ASIC з метою істотного підвищення швидкодії. Перевага використання програмованої логіки в цифрових системах стає особливо очевидним, коли необхідно швидко розробити дослідний зразок виробу. Стають можливими також коригування проекту в процесі його розробки або реконфигурація структури пристрою при його функціонуванні [21-22].

2.5.3. Характеристики компонентів програмованої логіки.

Сучасні PLD мають такі властивості [21]. 1. Низька вартість, яка впритул наближається до вартості стандартних IC. 2. Висока швидкодія. Застосування передових технологій призвело до зниження затримки проходження сигналу через PLD до 1 нс на один логічний елемент, що дозволяє досягати частоти функціонування пристроїв вище 500 МГц. 3. Високий ступінь інтеграції. PLD мають достатньо регулярну структуру, що дозволяє застосовувати 0,45nm технологію і досягати понад 500 млн вентилів на кристал. 4. Функціональна потужність, завдяки якій PLD можуть замінювати сотні корпусів «жорсткої» логіки, а складні PLD - тисячі корпусів, що дозволяє реалізувати на одному чіпі всю або деяку частину цифрової системи, наприклад співпроцесор. 5. Універсальність. На одному і тому ж PLD можна будувати різні функціональні вузли: комбінаційні, реєстрові, синхронні, асинхронні, арифметичні. 6. Багаторазовість програмування. Залежно від технології виробництва і типу настроюваного елемента число перепрограмування PLD може не обмежуватися або досягати декількох тисяч. 7. Високий відсоток виходу придатних виробів. Завдяки передовим технологіям і наявності вбудованих засобів тестопридатного проектування, PLD відрізняються винятковою надійністю, що виражається в близькому до 100% виході придатних виробів після програмування. 8. Захист інформації від зчитування. Всі сучасні PLD мають біт захисту, установка якого перешкоджає зчитуванню настройки PLD. Біт захисту скидається тільки при перезапису всієї інформації про налаштування PLD. 9. Скидання регістрів при включенні живлення. Ця властивість дозволяє однозначно визначати початковий стан прихованих елементів пам'яті при синтезі послідовних пристроїв і відмовитися від процедури ініціалізації на початку роботи пристрою. 10. Передзавантаження регістрів. На додаток до попередньої властивості PLD допускають попереднє завантаження внутрішніх регістрів певними значеннями. Це дозволяє здійснити процедуру ініціалізації послідовних пристроїв шляхом переведення їх в якийсь стан. Однак на практиці дана харак-

теристика частіше використовується для тестування PLD та перевірки поведінки пристрою у заборонених (illegal) станах. 11. Сумісність за входами і виходами з TTL-логікою. Більшість PLD, виконаних за технологіями, відмінними від TTL, залишаються сумісними з TTL-логікою за коефіцієнтами розширення входів і виходів, рівням напруг. 12. Різноманітність корпусів PLD. Одна і та ж мікросхема PLD може бути упакована в різні типи корпусів: штирові - DIP, планарні - SOIC, PGA, PQFP, TQFP, J-lead. 13. Можливість реконфігурування внутрішньої структури PLD в процесі функціонування для досягнення високої швидкості вирішення складних задач обробки даних [22]. 14. Відкритість і прозорість архітектури є домінуючим фактором для створення критичних технологій, що не залежать у майбутньому від компаній виробників.

2.5.4. Класифікація PLD.

Кристали, програмовані користувачем, найбільш кращі для створення прототипу. За їх допомогою розробники можуть значно скоротити цикл проектування і виробництва, а також поліпшити якість виробів, завдяки перевірці функціональних можливостей, ще на етапі проектування, коли внесення змін до системи легко і відносно дешево. Ці переваги і змусили розробників спеціалізованих мікросхем (ASIC) звернутися до використання PLD. Останні мають також і недоліки: нераціональне використання площі кристала, відсутність ефективних методів проектування складних пристроїв. Крім того, характеристики аналогічних пристроїв, виконаних на базі ASIC, зазвичай більш кращі. Однак ці недоліки не настільки відчутні порівняно з виграшем, одержуваним за рахунок скорочення часу розробки цифрових пристроїв на базі PLD. Тому в даний час програмовані користувачем кристали вважаються найбільш перспективною елементної базою. Це підтверджується численними статистичними дослідженнями і незмінним збільшенням обсягу їх виробництва.

В свою чергу програмовані користувачем ІС можна розділити на мікропрограмні та ІС програмованої логіки відповідно до двох основних підходів до проектування ЦП: мікропрограмним і аппаратним. Перший підхід передбачає побудову ЦП на базі деякого універсального елемента (мікропроцесора, мікрокомп'ютера, мікроконтролера), який спеціалізується завантажуваною до оперативного запам'ятовуючого пристрою (ОЗП) або зашивали у перепрограмоване постійне запам'ятовуючий пристрій (ППЗП) програмою (мікропрограмою). Недоліком такого підходу є невисока швидкість роботи пристрою. Однак при необхідності, кристал PLD може бути легко перенастроєний на інший алгоритм роботи шляхом заміни програми в ОЗП або в ППЗП.

Характерною особливістю PLD є можливість їх настроювання на заданий алгоритм функціонування шляхом зміни своєї внутрішньої структури. Проектування ЦП на основі PLD полягає у визначенні настройки за заданим специфікаціям і програмуванні мікросхеми на спеціальному обладнанні, що

зветься программатором (programmer). Побудовані таким чином ЦП відрізняє, перш за все, висока швидкість роботи, низька вартість і малі терміни проектування. Перші промислові PLD характеризувалися однократністю настройки, яка здійснювалася шляхом перепалювання плавких перемичок (fuse-link). Однак з появленням перепрограмованих PLD з електричним і ультрафіолетовим стиранням цей недолік усувається і за способом свого використання вони нічим не поступаються мікропрограмним ІВ. У теперішній час, згідно архітектурним особливостям, PLD прийнято ділити на три великі класси [21]: стандартні PLD (SPLD), складні PLD (CPLD) і вентильні матриці (FPGA) (рис.2.3). Всі PLD можна також розділити за типом настроюваного елемента [20] (див. рис.2.3): програмовані за допомогою статичного ОЗП (SRAM), за допомогою електрично стираємого постійного запам'ятовуючого пристрою (або ультрафіолетове стирання); за допомогою Antifuse-технології.

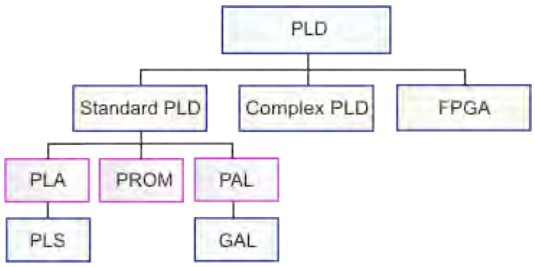


Рис. 2.3. Класифікація PLD за структурною організацією

Для SPLD затримка проходження сигналу з входу на вихід - постійна. Тому при проектуванні на цих пристроях можна не виконувати часове моделювання сигналів. Для представників класу CPLD затримка змінна, але легко обчислюється. В FPGA затримка повністю залежить від шляху, за яким проходить сигнал від входу на вихід. При зміні трасування міжз'єднань змінюються і затримки сигналів.

2.5.5. Архітектури кристалів програмованої логіки.

PLD - багатоцільовий чіп для реалізації цифрових систем. Він містить певну сукупність логічних елементів, які можуть бути з'єднані між собою різними способами. У загальному випадку, PLD може бути представлено у вигляді «чорної скриньки», яка містить логічні елементи і програмовані ключі [23]. Останні дозволяють програмувати з'єднання логічних елементів один з одним і з елементами, що знаходяться в інших мікросхемах, для реалізації необхідних логічних схем.

PLA-програмовані логічні матриці. Першим представником класу програмованих логічних пристроїв можна вважати програмовані логічні матриці. Їх загальна структура зображена на рис. 2.4.

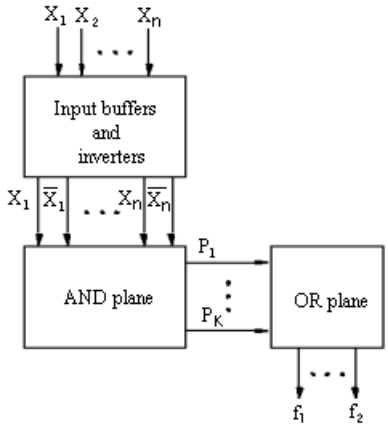


Рис. 2.4. Загальна структура PLA

Основна ідея полягає в реалізації логічних функцій, зображених у вигляді в ДНФ. Для цього PLA містить сукупність елементів I, виходи яких подаються на входи набору елементів АБО. Входи PLA $x_1,\ldots,x_n$ заведені через набір буферів (Input buffers and inverters), які забезпечують подачу як прямих, так і інверсних вхідних сигналів на блок схеми, що називається матрицею I (AND plane). На виході матриці I формуються терми $P_1,\ldots,P_k$, кожен з яких може бути конфігурован для здійснення кон'юнкції вхідних змінних $P_1,\ldots,P_k$. Отримані терми служать вхідними сигналами для матриці АБО (OR plane), на виході якої формуються значення функцій $f_1,\ldots,f_m$. Кожна така функція реалізує будь-яку диз'юнкцію термів $P_1,\ldots,P_k$. Таким чином, за допомогою PLA можна реалізовувати логічні функції, записані в ДНФ. Внутрішня структура фрагмента PLA наведена на рис. 2.5. PLA має три входи, чотири внутрішніх терма і два виходи. Кожен кон'юнктор матриці And має шість входів, що відповідають прямим та інверсним значенням трьох вхідних сигналів. Кожне з'єднання до матриці And програмовано (programmable connection). Будь-які не підключені до матріці And входи не впливають на значення вихідного терма відповідного кон'юнктора. Матриця And формує терм P_1, як кон'юнкцію вхідних змінних x_1 і x_2. Отже, $P_1 = x_1x_2$.

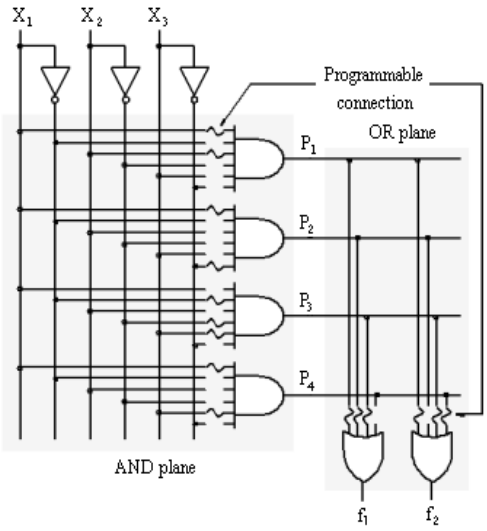


Рис. 2.5. Структура PLA

Аналогічно, $P_2 = x_1\bar{x}_3$, $P_3 = \bar{x}_1\bar{x}_2x_3$ і $P_4 = x_1x_3$. Програмовані з'єднання існують також і для матриці Or. Вихід f_1 пов'язаний з термами P_1, P_2 і P_3. Тому на першому виході отримаємо реалізацію функції $f_1 = x_1x_2 + x_1\bar{x}_3 + \bar{x}_1\bar{x}_2x_3$. Аналогічно $f_2 = x_1x_2 + \bar{x}_1\bar{x}_2x_3 + x_1x_3$. Таким чином, на рис. 2.5 зображена PLA, запрограмована для реалізації описаних вище функцій. Однак матриці And і Or можна запрограмувати й інакше для реалізації інших функцій вхідних сигналів x_1, x_2, і x_3. Єдине обмеження, що накладається на такі функції, це розрядність матриці And (в даному прикладі на виході матриці And можна отримати тільки 4 терма). Комерційні PLA мають набагато вищу розрядність. Типові PLA включають 16 входів, 32 проміжні шини (терми) і 8 виходів. При розгляді структури програмованих кристалів використовується також інший стиль зображення схеми. Прикладом тому може служити рис. 2.6. Тут показані програмовані з'єднання при реалізації функцій f_1 і f_2, зображених на рис. 2.6. PLA, програмовані одноразово, застосовуються для заміни мікросхем стандартної логіки. PLA зі сполуками, програмованими багаторазово, часто включаються до складу HBIC типу мікропроцесорів. В цьому випадку значення логічних функцій можуть бути отримані мікропроцесором набагато швидше, ніж при програмної реалізації даних функцій.

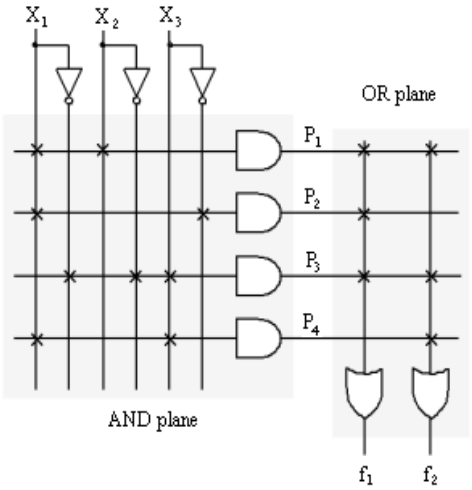


Рис.2.6. Загальноприйнятне зображення структури схем PLA

PAL - програмовані матриці логіки. В кристалі PLA обидві матриці And і Or - програмовані. В кристалі PLD матриця And є програмованою, а матриця Or - фіксованою. Виготовити такі пристрої простіше, ніж PLA, - вони мають меншу вартість і поліпшені характеристики, що сприяє їх високої популярності. Приклад PAL з трьома входами, чотирма термами і двома виходами подано на рис. 2.7.

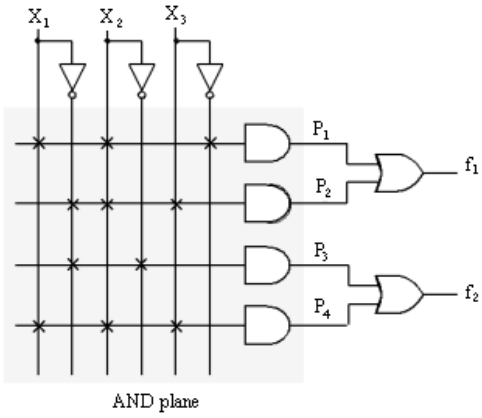


Рис. 2.7. Приклад типової структури PAL

Терми P_1 і P_2 апаратно з'єднані з першим елементом Or, P_3 і P_4 – з другим елементом. Ці з'єднання не можуть програмуватися. PAL реалізує дві логічні функції $f_1 = x_1x_2\overline{x_3} + \overline{x_1}x_2x_3$ і $f_2 = \overline{x_1}\,\overline{x_2} + x_1x_2x_3$. У порівнянні з PLA (див. рис. 2.6), PAL має меншу гнучкість програмування. PLA дозволяє використовувати чотири терми як входи для елементів Or, а елементи Or в PAL

мають тільки два фіксованих входи. Для компенсації цього недоліку виробники PAL випускають чіпи різного розміру, з різним числом входів і виходів, а також входів до вентилів Or. Виходи всіх раніше розглянутих нами PLD з'єднувалися безпосередньо із зовнішніми виходами мікросхеми. В сучасних чіпах ці виходи надходять на зовнішні виводи через осередки, в яких можуть використовуватися тригери різних типів, трістабільні буфери, різні логічні елементи і ряд інших компонентів. Приклад комірки наведено на рис. 2.8 [23].

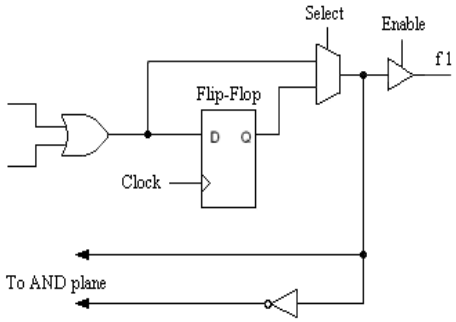


Рис. 2.8. Приклад комірки мікросхеми PAL

У ній використовується D-тригер, мультиплексор 2x1 та тристабільний буфер. D-тригер служить в якості елемента пам'яті і дозволяє зберегти значення функції, отриманої на виході PAL. Тригер стробірується сигналом Clock, який може бути як зовнішнім, так і внутрішнім. Мультиплексор дозволяє за сигналом Select або подавати на вхід буфера сигнал прямо з виходу елемента Or, що характерно при побудові схем комбінаційної логіки, або використовувати значення, збережене D-тригером. Крім того, сигнал з виходу мультиплексора може бути поданий назад на матрицю And, що дозволяє реалізовувати за допомогою PAL складніші логічні функції. Крім цього, виходи мікросхеми можуть бути переведені у високоімпедансний стан за сигналом Enable, який управляє вихідними буферами. Багато компаній об'єднують PLA, PAL і деякі інші подібні з ними структури в окремий клас Simple PLD.

CPLD - комплексне програмований логічний пристрій. Складається з блоків, розташованих на одному кристалі. Кожен з них з'єднаний з блоками вводу-виводу, що здійснюють високошвидкісний обмін з іншими мікросхемами. Крім того, всі блоки зв'язані між собою внутрішніми паралельними шинами. Окремо взятий блок подібний до мікросхеми або PLA, або PAL (PAL-like block). Приклад CPLD наведено на рис. 2.9. Чіп включає чотири таких блоку, які пов'язані між собою внутрішніми шинами і також з'єднуються з модулями вводу-виводу. На рис.2.10 наведено приклад внутрішньої структури PAL-like блоків та їх з'єднання в CPLD.

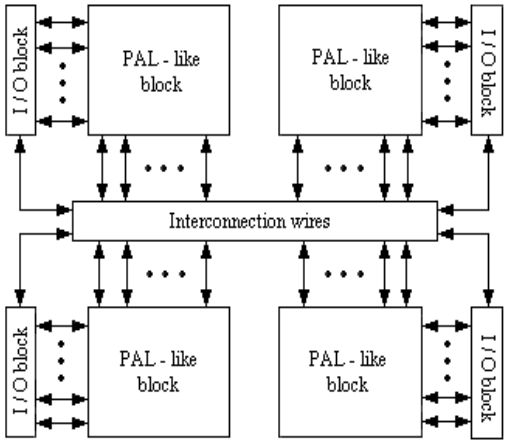


Рис.2.9. Структура мікросхеми CPLD

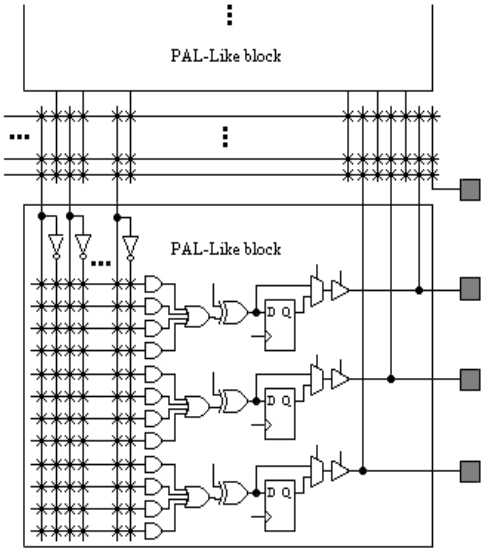


Рис. 2.10. Внутрішня організація блока мікросхеми CPLD

Кожен блок включає 3 осередки. До складу кожного з них входить 4-входовий елемент Or. Вихід цього елементу подається на вхід логічного елементу Xor. Інший вхід вентиля Xor може програмуватися на подачу нуля або одиниці. Якщо вхід запрограмований на подачу логічної одиниці, то даний вентиль буде інвертувати значення, знімається з виходу діз'юнктора. Якщо вхід налаштований на подачу 0, то вентиль передаватиме значення вихідного сигналу без зміни. Осередок також включає тригер, мультиплексор і буфер із трьома станами. Буфер дозволяє налаштовувати висновки чіпа як на введення, так і на вихід сигналів. Щоб вихід мікросхем-

ми міг стати вхідним, досить перевести відповідний буфер у високоімпедансний стан. Подальший шлях проходження сигналу може бути запрограмований. На схемі це вказується символом X, що стоїть на перетині вертикальних і горизонтальних ліній. Необхідно зауважити, що якщо висновок запрограмований на прийом інформації, то він, природно, не може бути використаний як вихідний, і відповідно вся комірка, що стоїть перед ним, виявиться незадіяною. Виготовлювачі вказують зазвичай максимально можливе число вхідних і вихідних ліній.

FPGA - вентильні матриці. За своєю структурою відрізняються від SPLD і CPLD, - не містять матриць Or і And [23]. Замість цього FPGA включають спеціальні логічні блоки для реалізації необхідних функцій. Загальна структура зображена на рис.2.11. Кристал FPGA містить три типи елементів: логічні блоки, блоки вводу-виводу для з'єднання з зовнішніми виходами чіпа і програмовані ключі для створення з'єднань між блоками. Логічні блоки розміщуються у вузлах решітки вертикальних і горизонтальних провідників, за допомогою яких можна створювати самі різні зв'язки між блоками. З'єднання є програмованими і здійснюються шляхом настройки внутрішніх з'єднувальних ключів (interconnection switches).

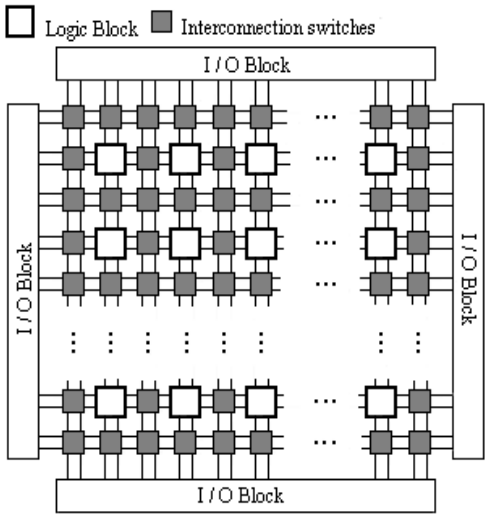


Рис.2.11. Загальна структура чипа FPGA

З'єднання з блоками вводу-виводу також програмуються. У різних серій FPGA реальне число програмованих ключів значно відрізняється, що необхідно враховувати при виборі фірми-виробника та конкретного сімейства мікросхем. Кожний логічний блок в FPGA зазвичай має невелике число входів і один вихід. Типовий логічний блок є функціональним генератором, званим LUT (Look Up Table). Кожен LUT (рис. 2.12), містить запам'ятовуючі осередки, які використовуються при реалізації невеликої логічної функції.

Одинична комірка здатна зберігати значення однієї логічної змінної, 0 або 1. Розмір LUT визначається числом входів, яке залежить від типу мікросхеми. LUT (див. рис. 2.12) здатний реалізувати будь-яку логічну функцію від двох змінних.

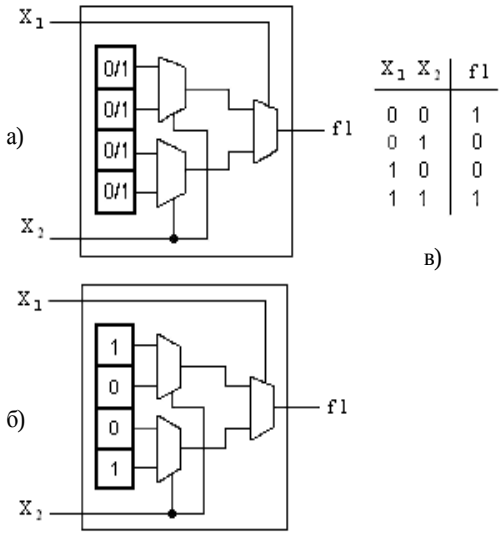


Рис. 2.12. Структура двовходового LUT

Оскільки таблиця істинності функцій двох змінних має чотири рядки, цей LUT має чотири запам'ятовуючих осередки. Один осередок відповідає значенням виходу в кожному рядку таблиці істинності. Вхідні змінні x_1 і x_2 використовуються як адресні входи трьох мультиплексорів, які залежно від значень x_1 і x_2 передають зміст однієї з цих чотирьох запам'ятовуючих осередків на вихід LUT.

Для програмування LUT необхідно лише зберегти таблицю істинності реалізованої функції в осередках пам'яті блоку. Тоді при $x_1 = x_2 = 0$ вихід LUT матиме значення самого верхнього запам'ятовуючого осередку, який зберігає значення функції для набору $x_1x_2 = 00$. Точно також для всіх наборів x_1 і x_2 значення, збережене у запам'ятовуючому осередку, відповідає становищу у таблиці істинності, подається на вихід LUT. На рис.2.13 показано тривходовий LUT.

Він має вісім запам'ятовуючих осередків і може реалізувати будь-яку функцію трьох змінних. LUT сучасних чіпів FPGA зазвичай має чотири або п'ять входів, які вимагають 16 і 32 запам'ятовуючі осередки відповідно. Як і у випадку з PAL-структурою, крім схем реалізації булевих функцій кожен LUT доповнюється елементами, якими оснащений осередок PAL. На

рис.2.14 показано підключення додаткового D-тригера, який може зберігати значення функції, реалізованої LUT.

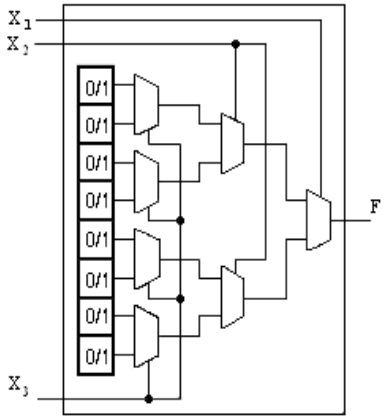


Рис 2.13. Тривходовий LUT

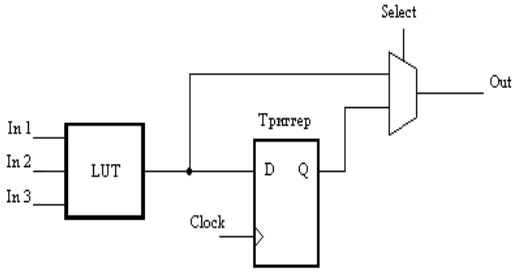


Рис.2.14. Підключення тригера у логічному блоці FPGA

Тригер керується сигналом Clock, який може бути як внутрішнім, так і зовнішнім. Мультиплексор дозволяє вибирати D-тригери при побудові реєстрових схем або відключення їх від схеми при проектуванні комбінаційної логіки. FPGA можна використовувати тоді, коли логічні блоки запрограмовані для реалізації необхідних функцій, а лінії з'єднань - для реалізації необхідних взаємозв'язків між геологічними блоками і блоками вводу-виводу. Запам'ятовуючі осередки LUT в FPGA енергозалежні. Це означає, що вони втрачають свій вміст всякий раз при відключенні харчування. Отже, чіп FPGA повинен програмуватися щоразу при включенні харчування. Зазвичай дані, необхідні для програмування, зберігаються у мікросхемі ППЗП, що розташована на тій же платі, що і чіп FPGA. Дані, збережені у ППЗП, автоматично переписуються в комірки пам'яті FPGA при увімкненні живлення. Фрагмент структури чіпа FPGA, який запрограмований для реалізації необхідної логічної функції, зображений на рис.2.15.

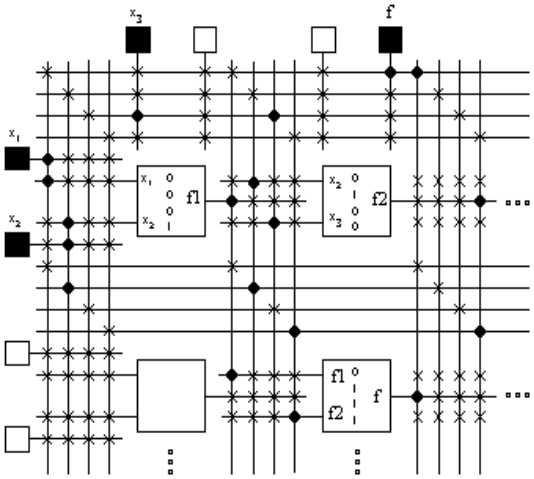


Рис. 2.15. Пример структуры запрограммированного чипа FPGA

FPGA має блоки LUT з двома входами і чотири програмованих лінії з'єднань. Кожен активний ключ з'єднує вертикальну і горизонтальну лінії і показаний символом - жирною крапкою. Перші два блоки LUT запрограмовані на реалізацію функцій $f_1 = x_1 x_2$ і $f_2 = \bar{x}_2 x_3$. Третій LUT реалізує функцію $f = f_1 + f_2$.

Таким чином, CPLD і FPGA є сучасним класом кристалів, які складають конкуренцію в проектуванні спеціалізованих цифрових систем, завдяки достоїнствам: високій швидкості проектування; низькій вартості розроблюваних пристроїв; високій швидкодії, низькому енергоспоживанню, гнучкості до виправлення помилок, відкритості архітектури для реалізації критичних проектів. Застосування CPLD особливо ефективно при реалізації керуючих і інтерфейсних схем, а FPGA - при реалізації алгоритмів цифрової обробки сигналів.

2.6. Технології проектування цифрових систем на кристалах

Розглядаються методи і засоби створення вбудованих цифрових систем на кристалах. Наводиться аналіз існуючих систем моделювання, верифікації, синтезу та імплементації. Дається класифікація чіпів з по-зиції створення енергетично економних і високопродуктивних цифрових систем.

2.6.1. Вступ.

Цифрові системи (ЦС) прийому-передачі, управління, збору, зберігання, перетворення та аналізу даних в даний час застосовуються в усіх галузях людської діяльності (виробництво, наукові дослідження, побут, офіс, транспортні та телекомунікаційні засоби). У зв'язку з цим до ЦС все більш зростають вимоги щодо зменшення вартості, розмірності,

енергоспоживання, апаратурною надмірністю. Очевидно, що універсальні, потужні персональні комп'ютери, які споживають велику кількість енергії і мають значні розміри і вартість, не можуть конкурувати на ринку спеціалізованих пристроїв. Що ж натомість?

Відповіддю на дане питання можуть бути конструктиви: Embedded Systems (вбудовані системи), Systems on Chip (SoC - системи на кристалах) [24], Networks on Chip (мережі на кристалах), System in Package (системи в пакетах) [25]. Іншими словами, щоб домогтися збільшення продуктивності, зменшення споживаної потужності і фізичних розмірів, систему необхідно розміщати не на платі, а на кристалі. При проектуванні SoC проблема функціональної верифікації та тестування є однією з найскладніших: за оцінками провідних зарубіжних фірм та фахівців товариства IEEE вартість її вирішення становить 50 - 70% від сукупних витрат повного циклу розробки кристала [26,27]. На ринку мікроелектроніки для виготовлення цифрових кристалів найбільш популярними є замовні BIC (cell based ASIC) і програмовані логічні інтегральні схеми - ПЛІС (PLD - CPLD, FPGA) [28]. На базі ASIC і PLD існують наступні напрямки розробки цифрових систем (рис. 2.16).

ASIC (Application Specific Integrated Circuits) - спеціалізовані інтегральні схеми, які виграють у ПЛІС за енерговитратами і продуктивністю, але значно програють за часом розробки і виробництва. Крім того, тут слід враховувати високу вартість виробництва підкладки для кристалів ASIC, яка доцільна тільки при великих партіях, починаючи з 50 000 мікросхем [29].

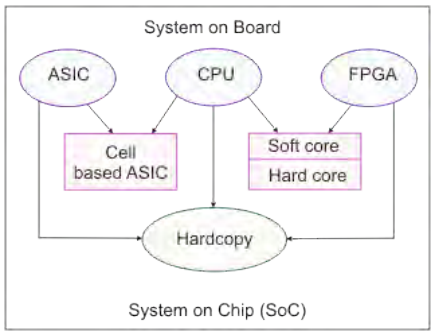


Рис. 2.16. Напрямок розвитку цифрових систем

Традиційно структурно-функціональний опис пристрою на кристалі (ASIC або ПЛІС) зображується у вигляді мнгожини взаємопов'язаних логічних примітивів [30]. Всі автоматизовані засоби верифікації та тестування були орієнтовані на налагодження тільки логічних схем, зображених схематично або за допомогою мов опису апаратури. Але коли мова йде про наявність мікропроцесорів всередині системи на кристалі, то функціональність визначається не тільки логічними елементами, виконаними у силіконі, а й від програмного забезпечення, реалізованого у процесорах системи на

кристалі [31]. Тому для верифікації та тестування таких SoC потрібні спільні зусилля розробників апаратних і програмних компонентів системи. Такі ж вимоги висуваються до моделей, методів, алгоритмів та засобів проектування, які повинні бути у змозі вирішувати завдання спільного тестування та верифікації гетерогенних частин системи [32]. Враховуючи сказане, вельми актуальною видається проблема розробки нових моделей, методів, алгоритмів та програмно-апаратних комплексів спільного тестування і верифікації на стадії проектування цифрових систем і мереж на кристалі.

Мета аналізу - надання початківцям розробникам цифрових систем на кристалах інформації про перспективні напрями імплементації моделей, методів і алгоритмів, які можуть бути затребувані на ринку електронних та інформаційних технологій. Завдання: 1) Порівняння конструктивів інтегральної схемотехніки, що поставляється на ринок EDA-технологій для реалізації цифрових систем. 2) Перспективні маршрути проектування, тестування і верифікації створюваних програмно-апаратних комплексів.

2.6.2. Спеціалізовані інтегральні схеми (IC) – ASIC.

Це домінуючі кристали на ринку мікроелектроніки, які використовуються для реалізації спеціалізованих і універсальних цифрових та цифро-аналогових схем. ASICs є лідерами за місткістю кристала, його продуктивністю, витратам електроенергії і ціною за одиницю виробу при масовому виробництві [29]. Існує кілька різновидів ASICs, що розрізняються за архітектури, а отже, за робочими характеристикам і циклами розробки: Gate Array (GA) - вентильна матриця; Embedded Array (EA) - полузамовлені IC; cell based ASIC (Standard cell - SC) - замовні IC з вбудованими осередками [33]. GA належить до сімейства ASIC і характеризується незначним (тижні) часом проектування та реалізації на кристалі за найнижчими цінами. Технологія виготовлення: 1,0 - 0,045 µm; споживана напруга 5 - 1,8V; кількість доступних до використання вентилів - від 1 тис. до 700 млн, кількість виходів від 20 до 1024, робоча частота понад 500 МГц, кількість шарів 2 - 4. Розробник отримує вже готову підкладку bulk wafers з матрицею транзисторів і за допомогою засобів проектування задає зв'язки транзисторів між собою (routing), що формує функції чіпу. Підкладки виробляються з різною кількістю транзисторів, що дає споживачам свободу вибору за обсягом корисних функцій, які можна включити до кристалу. GA використовується для створення спеціалізованих великих інтегральних схем (BIC). Тут доступний ряд вбудованої у кристал пам'яті. Так, фірма NEC (www.nec.com) для GA пропонує RAM, ROM, синхронні і асинхронні, одно-і двопортові, розмірністю до 8 Кбіт [29]. Функціональні блоки (IP-cores) можуть бути також використані в GA. Але зазвичай це обмежена кількість малих за розміром осередків. Їх набір різний для виробників і сімейств GA. Компанією NEC підтримуються IP-cores: DMA-контролер, блок управління, програмний лічильник, блок переривань, прийомо-передавач (UART с

FIFO), PCI. EA поєднують позитивні якості GA - малий час проектування і можливість використання функціональності на системному рівні, як у Standard Cell [33]. Швидкодія проектування досягається тим, що деякі пластини (wafer) виготовляються паралельно процесу проектування EA. Технологія виготовлення 0,09 - 0,032 µm.

EA - це високопродуктивна полузамовлена IC (вентильна матриця), яка містить у собі базові матриці транзисторів і дозволяє використовувати вбудовані елементи пам'яті і IP-cores, як і у cell-based IC [33]. При створенні проектів з великим обсягом пам'яті полузаказних IC є більш оптимальним рішенням за ціною за одиницю. Але при цьому зростає вартість розробки через необхідність вирішення задачі розміщення матриці транзисторів на базовій підкладці cell-based IC (спеціалізованій IC). Використання вентильного «моря» або безканальної архітектури IC робить процес розробки і подальше з'єднання транзисторів практично рівним тривалості фази проектування SoC в GA. Монолітну підкладку VLSI (BIC) можна повторно використовувати, змінюючи тільки логічну частину під час проектування та верифікації, як і у кристалів технології GA.

SC - замовні IC, які дозволяють оптимально проектувати логічні осередки, блоки пам'яті ROM і RAM, CPU і аналогові компоненти на одному кристалі. SC надають розробникам більш гнучку технологію проектування у порівнянні з GA і можуть бути оптимізовані під потреби замовника LSI. Технології виготовлення 0,35 - 0,18 µm. В даний час вже заявлені технології 0,13 - 0,09 µm. Така оптимізація дозволяє досягти ще більшої функціональності чіпа і меншого споживання електроенергії у процесі експлуатації.

Різниця між типами IC полягає у кількості складних і простих функціональних компонентів, блоків або осередків на кристалі, з яких будується цифровий пристрій [33]. Так, для GA це транзистори і деякі елементи пам'яті; для замовних високоінтегрованих IC - це компонент з розширеною функціональністю, що дозволяють розробляти SoC.

На рис. 2.17 наведені деякі функціональні модулі, що можуть бути інтегровані у кристал (інтерфейси, процесори, периферія, блоки пам'яті, аналогові компоненти).

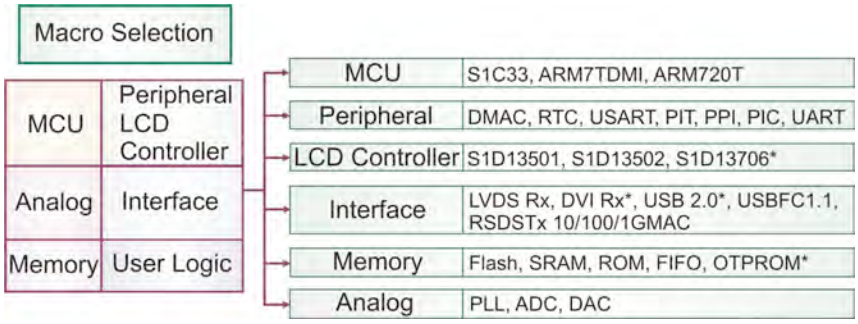


Рис. 2.17. Функціональні модулі, що інтегруються у кристалл

2.6.3. Технології та маршрути процесу проектування.

На рис. 2.18 наведено схему повного циклу розробки GA і полузамовлених мікросхем. Перший етап - системний рівень проектування, визначаються параметри та компоненти SoC і паралельно починається розробка модулей RTL рівня та виготовлення базової підкладки, на якій будуть розташовані всі компоненти цифрового пристрою [34]. Після верифікації моделі системи виконується імплементація проекту шляхом переносу (плавлення) необхідних зв'язків.

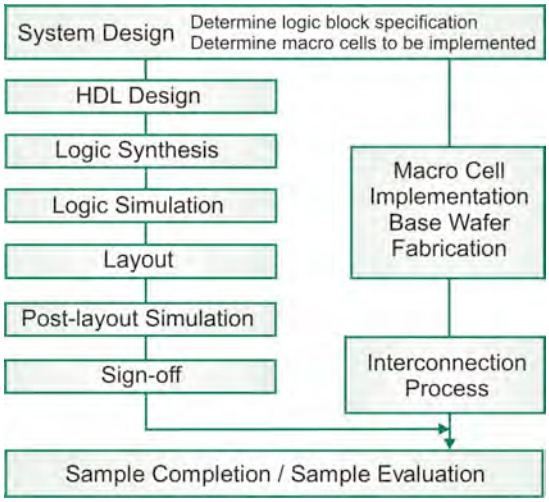


Рис. 2.18. Цикл розробки полузамовлених ІС

На рис. 2.19 зображено процес проектування замовної ІС, де відмінність полягає у відсутності виробництва підкладки кристала. Розробка розпочинається з системного рівня, під час якого підраховується кількість вентилів і макрокомірок (macro cells). Потім під даний проект здійснюється виробництво основної пластини кристала (www.epson.com). Далі на кристалі конструюються макрокомірки шляхом перетворення вентильного «моря» методом плавлення з'єднань в логічні блоки необхідної функціональності. Як і у

випадку з GA, паралельно з виробничими операціями виконуються стандартні етапи проектування, починаючи с опису проекту і закінчуючи моделюванням кінцевого Netlist і виправленням HDL опису, якщо це необхідно.

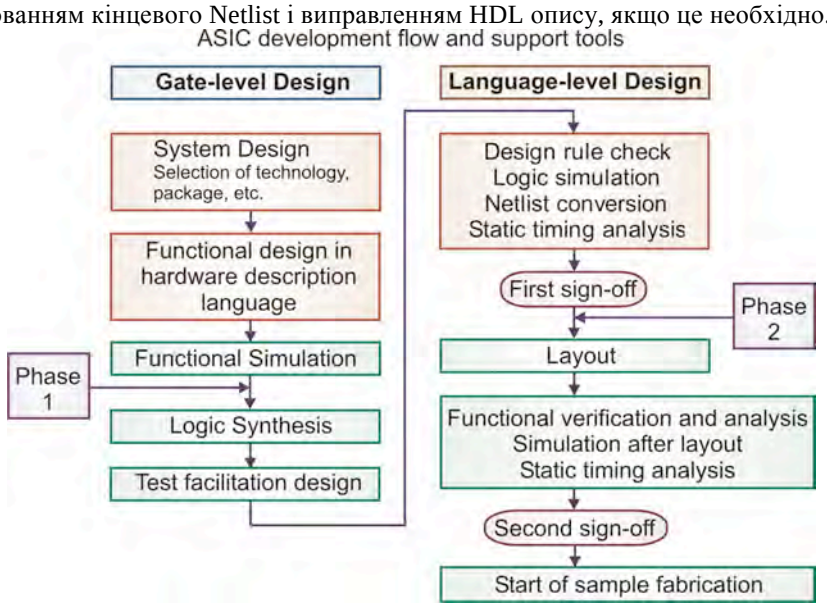


Рис. 2.19. Процес розробки та верифікації ASIC

Потім робоча версія проекту «заморожується» і вона ж імплементується у кристал шляхом плавлення з'єднань між транзисторами на підложці. Після «заморожування» коду Netlist створюються пробні екземпляри мікросхеми. Оскільки базова підложка має властивість повторного використання, то вартість зміни логіки на кристалі аналогічна витратам використання GA-технології [33]. Представлену схему розробки ASIC-проекту можна спростити до чотирьох етапів, показаних на рис. 2.20.

Проектування на вентильному рівні. Введення схемного опису - це традиційно витратна за часом фаза проектування, яка використовується як для проектування IC, так і для програмованих логічних пристроїв. Для введення схемного опису застосовується GUI (Graphical User Interface) - графічний інтерфейс, що дозволяє вибрати необхідну кількість і типи вентилів і/або примітивів і реалізувати зв'язок між ними. При такому підході до проектування можна виділити 4 основних етапи (www.k-micro.com) (рис. 2.21): 1) Вибір відповідного пакета проектування і бібліотеки сімейства ASIC, куди передбачається імплементувати проект. 2) Створення проекту пристрою з використанням відповідних вентилів і примітивів з вибраної бібліотеки та зв'язків між ними. 3) Додавання до проекту вхідних і вихідних (I/O) контактів пристрою. 4) Генерація кінцевого Netlist з розташуванням логічних примітивів на кристалі і зв'язків між ними. Netlist є компактним текстовим

універсальним форматом зображення схеми в примітивах бібліотеки обраного кристала, який підтримується мовами Verilog, VHDL і може бути використаний як для імплементації, так і для timing verification. Netlist змінив формат EDIF, що застосовується тільки для фази Place and Route. Проектування цифрових систем на рівні вентилів виправдано, якщо розмірність пристрою не перевищує 20000 вентилів [30].

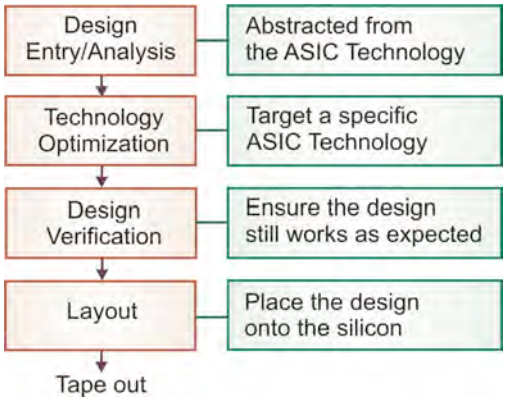


Рис. 2.20. Спрощений процес проектування ASIC

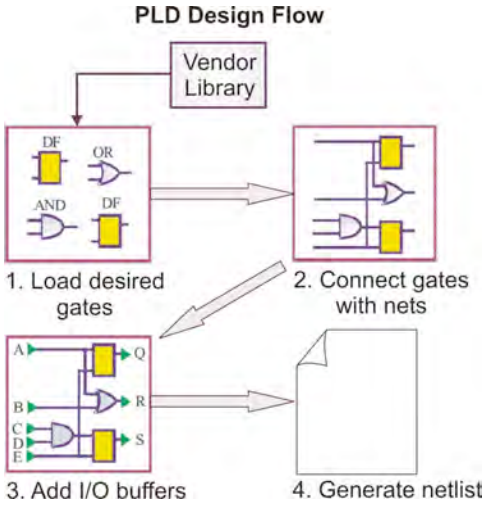


Рис. 2.21. Маршрут проектування PLD

Іншим недоліком такого проектування є неможливість імплементації проекту в інший кристал, оскільки необхідно замінювати всі примітиви одного виробника на елементи з бібліотеки іншого постачальника мікросхем.

Проектування на алгоритмічному рівні (High-Level Design). Тут використовуються мови опису апаратури (Hardware Description Language - HDL) для зображення функціональної поведінки проекту без ідентифікації струк-

тури. Основними стандартизованими HDL є VHDL, Verilog, System C і System Verilog (www.cadence.com, www.magma.com). Ведеться робота зі стандартизації SystemVerilog (www.accelera.org). Мови опису апаратури є ідеальним середовищем, що дозволяє повторно використовувати готові проекти або їх частини. Після опису пристрою на поведінковому рівні повинен бути виконаний його синтез у схемне подання вентильного рівня, що забезпечує задану функціональність і імплементацію в обраний тип кристала. Переваги HDL-опису: це зручне і швидке написання проекту; легке внесення змін; виняток процедури з'єднання схемних блоків між собою; можливість отримання декількох альтернативних рішень залежно від параметрів синтезу; вибір різних типів кристалів після Design Entry. Реально для проектів більше 10 000 вентилів немає альтернативи мовам опису апаратури.

Оптимізація синтезу під технологію виготовлення кристала. HDL опис цифрових систем, як правило, не містить вентилів, має бути перетворено в логічну схему примітивів вибраної серії кристалів. Дана процедура виконується програмами синтезу (Synopsys - Design Compiler). Результуючий netlist після такої операції залежить від бібліотеки обраного кристала (FPGA, CPLD, ASIC) і від критеріїв оптимізації (розмірність проекту або швидкодія). Автоматизація сінтезу дозволяє легко експериментувати з різними параметрами оптимізації, серіями мікросхем та їх виробниками для отримання декількох варіантів рішень, у той час як на вентильному рівні проектування існує один варіант (www.synopsys.com).

Другим завданням синтезу пристрою під вибрану технологію виготовлення кристала є забезпечення тестопридатності. Застосовувана у теперішній час тестова методологія внесення нефункціональної надлишковості (IEEE JTAG boundary scan 11.49, P1500) [35] при розробці SoC є стандартом для більшості виробників і лабораторій. Вона включає імплементацію на кристал засобів управління і спостереження в режимі тестування функцій (Synopsys DFT Compiler), Bilt-in Self Testing (BIST) для перевірки пам'яті, логіки, несправностей, таких як single stack-at-fault, Iddq (Synopsys TetraMAX, Cadence / IBM TestBench) [36-39]. Для визначення виробничих дефектів в ASIC вводиться регістр сканування даних (рис. 2.22), який суттєво підвищує якість (до 95 - 100%) тесту і зменшує його довжину. При цьому апаратура сканування незначно впливає на продуктивність системи і збільшення площі кристала. На ринку існують десятки автоматизованих пакетів, що виконують повний цикл Design for Test, єдиним недоліком яких є висока вартість і використання коштовних зовнішніх тест-станцій, наприклад, від фірм Advantest і HP (www.advantest.com, www.hp. com). Після тестопридатного синтезу розробник отримує пристрій на основі примітивів обраного виробника та серії кристала, що включає функціональність і блоки для тестування виробничих дефектів, готові до імплементації на кристалі. Але перед виконанням останньої процедури необхідно верифікувати проект на коректність роботи з метою усунення функціональних помилок, які можуть привести до небажаного повторення фази Place and Route.

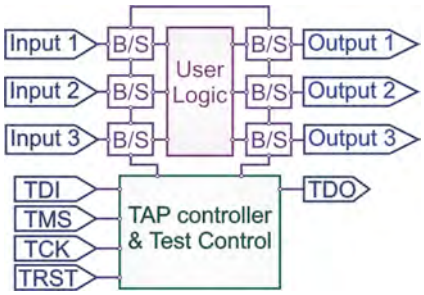


Рис. 2.22. Застосування стандарту IEEE 1149.1 для SoC Design

Верифікація проекту (Design Verification). Тут основні фази і програмні засоби для їх реалізації можна перерахувати у наступному вигляді: 1) Logical Design Rule Check - Synopsys (DC Ultra, Design Compiler), K-Micro tools; 2) Formal Verification - Synopsys (Design VERIFYer, Formality), Verplex (Tuxedo-LEC); 3) Static Timing Analysis - Synopsis (Prime Time); 4) Gate-Level Simulation - Mentor (ModelSim), Cadence (VerilogXL, NC-Verilog), Synopsys (VCS); Cadence Products are golden simulators for K-Micro; 5) Power Estimating (if necessary) - Synopsys (Power Compiler) [40-42]. Зі зростанням мікропроцесорних технологій розмірність ASIC систем збільшується з кожним роком [43]. Разом з тим, для перевірки всіх часових характеристик мультимільйонного проекту необхідно синтезувати тест великої розмірності (сотні тисяч векторів), що є трудновиконуваною задачею в рамках обмеженого часу проектування ASIC [44]. Тим не менш, на ринку з'являються засоби - static timing analysis (STA), що дозволяють виконати аналіз часових характеристик за короткий період часу, якщо проект виконано за STA технології [45-47]. Слід відзначити високу вартість спеціалізованих засобів оцінки статичних часових параметрів проекту після синтезу / імплементації (Synopsys PrimeTime). Це пояснюється великими витратами переробки підложки в разі помилки.

Formal Verification - FV (Verplex). Засоби формальної верифікації призначені для статичної перевірки на відповідність проекту з вже відлагодженим його поданням на попередній стадії розробки [48]. При проектуванні з використанням HDL засоби формальної верифікації необхідні для перевірки еквівалентності веріфікованого RTL проекту з післясинтезним вентильним Netlist. У загальному випадку без логічного моделювання за короткий інтервал часу можна визначити еквівалентність проекту на різних стадіях розробки. Поєднання технологій STA і FV дозволяє значно підвищити продуктивність проектування ASIC великої розмірності. Популярні маршрути, що поєднують моделювання, верифікацію і часовий аналіз, подано на рис. 2.23.

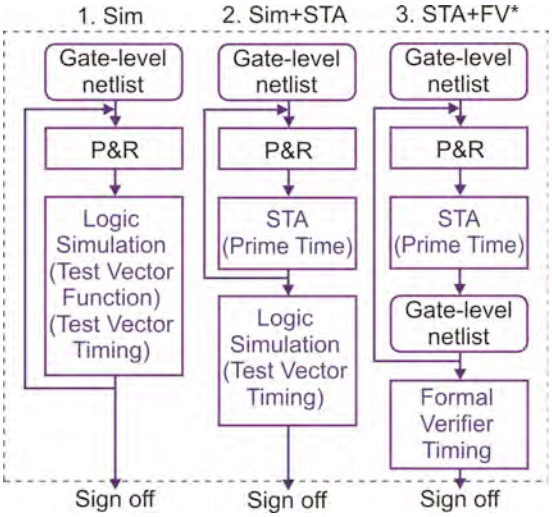


Рис. 2.23. Маршрути верифікації проектів

Планування проекту в кристалі (Layout). Тут основні фази та існуючі програмні засоби для їх реалізації можна представити у наступному вигляді: 1) Floor Planning - Cadence (First Encounter); 2) Power Analysis - Synplicity (Fortify RealPower, Fortify Power Planner); 3) Clock Tree Synthesis [49, 50] - Cadence (CTGen, First Enconter); 4) Place & Route [51, 52] - Cadence (Silicon Ensemble DSM, First Encounter + NanoRoute Ultra); 5) Interconnect RC Analysis - Synopsis (Star-RCXT); 6) Noise Analysis [53, 54] - Sequence (Physical Studio); 7) ATE Rule Check - K-Micro tools; 8) DRC & LVS - Mentor (Calibre) (www.mentor.com); 9) Static Timing Analysis - Synopsis (Prime Time, Customer Sign-Off); 10) TestProgram Generation - K-Micro tools [55].

Ієрархічний процес проектування. Використовується для ASIC великої розмірності і дозволяє зменшити кількість ітерацій, а також час виконання проекту. Увага приділяється спільній роботі команд: 1) розробка RTL проекту; 2) розміщення його на кристалі, починаючи зі стадії Design Entry. При цьому виникають нові завдання: hands-on design partitioning - проект розділяється на блоки з урахуванням функціональності, структури проекту, його розмірності та кількості блоків (size balancing of partitions) [56]; оптимізація параметрів всередині кожного модуля, аналіз часових і синхронізуючих характеристик блоків; розподіл і планування бюджету часу з розробки блоків (timing budget allocation for the partitions); інтегрування блоків і подальша оптимізація, а також верифікація всіх основних параметрів у межах об'єднаної системи.

Висновки. На рис. 2.24 наведені характеристики GA і cell-based IC за часом виробничого циклу [29]. Розробка RTL моделі однакова для обох технологій, але GA виграє за сукупним часом (тестування + виробництво). Дана обставина зменшує виробничий цикл майже вдвічі порівняно з cell-based IC. Для GA, EA, cell-based IC на відміну від PLD додається великий блок - те-

стування, що займає до половини часових і матеріальних витрат повного циклу проектування [35]. Для проектів великої розмірності використовуються потужні і коштовні інструменти логічного моделювання (фірми Cadence, Synopsys), апаратурні прискорювачі, оскільки час одного циклу моделювання може досягати кілька днів (тиждень) при регресивному тестуванні (Axis, Mentor Graphics, Cadence), коли верифікація проекту здійснюється з поверненням від складних тестів до простих при виявленні дефекту.

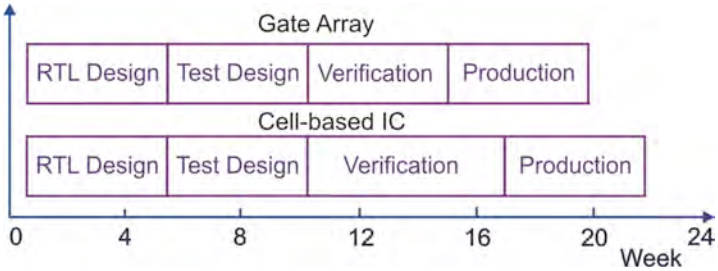


Рис. 2.24. Аналіз виробничих циклів

При розгляді існуючих засобів проектування SoC на PLD інтерес становить програмні пакети фірм-лідерів (XILINX, ALTERA) з виробництва пропрограмованих кристалів. Поява на EDA-ринку програмних продуктів Xilinx® ISE і Altera® Quartus набагато спростило процес розробки проектів для програмованої логіки на основі високорівневих мов опису апаратури (VHDL, Verilog). Загальна схема проектування, включаючи завдання верифікації, зображена на рис. 2.25.

Процес розробки PLD SoC включає кроки: 1) Design Entry - опис проекту на схемному або вентильному рівні, або за допомогою HDL-мов [57, 58]. 2) Синтез - перетворення вихідного опису у вентильний Netlist обраного сімейства CPLD, FPGA. 3) Імплементація - розміщення цифрового проекту на кристалі (тип корпусу, робоча частота, інші параметри) з прокладкою міжз'єднань між блоками. Netlist проекту є опис пристрою вентилями певного сімейства кристалів. Після перевірки на функціональність Netlist готовий до імплементації на кристалі: оптимізація, трансляція у фізичні елементи чипа, перевірка дизайну на достатність компонентів, CLK буферів в обраному чипі. При завершенні імплементації генерується звіт, де крім попереджень і помилок міститься інформація про використання блоків, входів / виходів, що дозволяє визначити оптимальність розміщення проекту в кристалі.

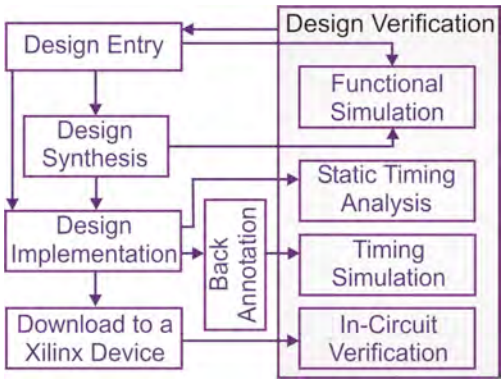


Рис. 2.25. Загальна схема проектування PLD SoC

2.6.4. Верифікація CPLD, FPGA кристалів.

Верифікація CPLD, FPGA проекту - це перевірка правильності функціонування об'єкта шляхом аналізу його реакцій, отриманих в результаті його моделювання на тестових впливах (Testbench) [59-61]. Критерії оп-оптимальної верифікації: 1) Покриття тестовими наборами усіх значень сигналів на лініях схеми. 2) Покриття тестом всіх рядків HDL коду. 3) По-покриття тестом true-false гілок в операторах умови. 4) Повна відповідність пристрою специфікації проекту. 5) Покриття тестом компонентів ге-терогенной системи, оскільки існуючі методи перевірки RTL-проекту слабо орієнтовані на верифікацію SoC з програмним забезпеченням. 6) Перевірка на відповідність специфікації і коректності функціонування SoC на всіх рівнях зображення проекту [62].

Головна умова для виконання верифікації - наявність стандартизованих форматів даних, які дозволяють: 1) багаторазово використовувати (re-use) вже налагоджені проекти або його частини; 2) застосовувати програми моделювання, синтезу та імплементації різних виробників без зміни опису проекту, який може бути перекомпільований з підключенням інших бібліотек примітивів для досягнення більш оптимального результату. Верифікація практично на всіх стадіях проходження проекту означає моделювання на основі його опису та вхідних тестових послідовностей (Testbench). Результат виконання процедури - часові діаграми значень внутрішніх та вихідних ліній, які необхідно перевірити на відповідність критеріям оптимальності верифікації. У правому стовпчику (див. рис. 2.25) подані етапи верифікації: 1. Functional Simulation [63] - перевіряється правильність введеної моделі на функціональність щодо вихідної специфікації. Виконується як для HDL-опису, так і для вентильного рівня. Розробник повинен переконатися - синтез пройшов без помилок і функціональність не змінилася при переході від HDL до вентильного подання проекту. 2. Static analysis - після імплементації проекту з'являється інформація про часові параметри схеми. У разі виявлення помилок

(неприпустимо великі затримки) можна змінити HDL код, перекомпілювати його і виконати повторне моделювання для перевірки функціональності. Потім виконати синтез вентильного подання (Netlist) з HDL-коду та провести моделювання з метою визначення коректності зміненого проекту. Ітеративна верифікація і корекція проекту до усунення всіх дефектів займає порядку 50% часу доведення проекту до необхідного рівня якості. 3. Timing verification - після імплементації вентильного подання виконується моделювання Netlist проекту з урахуванням всіх часових характеристик елементів і зв'язків між ними з метою визначення коректності функціонування майбутнього пристрою. 4. In circuit verification - останній етап верифікації SoC на робочих частотах після программування CPLD, FPGA. Використовується прототип плати зі стандартним набором компонентів та інтерфейсом для зв'язку з робочою станцією, яка емулює зовнішнє середовище для SoC. Debug проекту на цьому етапі верифікації проблематичний через обмежений доступ до внутрішніх ліній CPLD, FPGA кристала для виявлення та локалізації помилок (www.avnet.com).

Висновки. ПЛІС технологія виключає стадії виробництва і тестування силікону. PLD-кристали купуються вже готовими і вільними від дефектів виробництва. Створити SoC - значить запрограмувати кристал під необхідну функціональність, виключаючи помилки проектування. Тому верифікація проекту шляхом моделювання структурного HDL опису, вентильного Netlist, часового опису параметрів логічних блоків обраної серії FPGA або CPLD є переважаючою за матеріальними і часовими витратами при проектуванні SoC. Автоматизація цього сегмента робіт є найбільш важливою проблемою в світі на сьогоднішній день. Більше 90% всіх PLD-проектів мають розмірність менше 1 млн еквівалентних вентилів з бюджетом, що не перевищує 50 000 доларів. Тому EDA-ринок для ПЛІС відрізняється більш повільними і дешевими програмами моделювання, достатньо простими при їх використанні (Mentor Graphics, Model Sim ® and Aldec, Active-HDL ®).

2.6.5. Intellectual Property (IP-) Cores.

IP-Core - складний вже протестований, верифікований і оптимізований функціональний модуль, який може бути багаторазово використан як компонент при проектуванні більш складних цифрових виробів з метою скорочення часу їх розробки [64]. Він приєднується до проекту на системному рівні при виконанні етапу Design Entry. Переваги використання IP-Cores: 1) Суттєве зменшення часу до виходу кінцевого продукту на ринок. 2) Спрощення процесу проектування шляхом створення інтерфейсу для зв'язку системи з IP-Core. 3) Мінімізація ризику проектування завдяки включенню вже налагоджених модулів з готовим Testbench. 4) Зменшення часу верифікації всієї системи. 5) Передбачуваність функціональності і продуктивності створюваного проекту. 6) Спрощення специфікації проекту, завдяки використанню повністю готових до застосування модулів системного рівня (PCI шина, DSP фільтри) [65].

Інтелектуальна власність. Цифрові IP-Cores, які можуть бути легко інтегровані в будь-яку ASIC технологію, вже доступні з чисельних джерел, представлених в Internet. Але більш цінними і ефективними є рішення, які залежать від технології виробництва чипа (аналогові, змішані сигнали, PHYs, CAM) і максимально використовують всі її переваги - value-added IP (VA-IP). Такі VA-IP предоставляють наступні переваги проектувальнику: 1) зменшення витрат проектування шляхом виключення фази розробки функціональних блоків VA-IP; 2) зменшення часу створення проекту завдяки можливості використання стандартних силіконових технологій. Формат і властивості надаваних VA-IP: 1) VHDL / Verilog моделі для моделювання пристроїв під час проектування; 2) тест-чіпи для верифікації правильності роботи IP в силіконі; 3) гарантія коректності часових параметрів при розташуванні IP-модулів на кристалі; 4) статистичний аналіз часових параметрів модулів для перевірки продуктивності; 5) високе покриття несправностей, що поставляються Testbench (www.viragelogic.com).

Конвертація FPGA в ASIC. При розвитку технологій виробники кристалу пропонують власникам FPGA рішень переходити на ASIC-технологію [66]. Конвертація в ASIC приносить такі переваги: 1) зменшення енергоспоживання; 2) зниження вартості виробництва однієї мікросхеми при великих обсягах випуску мікросхем; 3) зменьшення займаної площі на платі шляхом створення одного ASIC кристала замість декількох FPGA; 4) виключення витрат на PROM, необхідних для зберігання конфігурації FPGA. Для прийняття рішення про перехід з FPGA- на ASIC-технологію слід враховувати моменти: 1) Чи покриє економія на вартості виробництва партії мікросхем часові та матеріальні витрати зміни технології. 2) Вибір виробника і типу ASIC: менш коштовну EA з вентильним "морем" та обмеженим набором вбудованих в кристал компонентів або більш продуктивну, але дорогу технологію Cell based ASIC. При переході на ASIC-технологію файл схемного опису FPGA-проекту з усіма специфічними FPGA-конструкціями конвертується в еквівалентні ASIC-структури (CoreGen / LPM / on-chip multipliers, PLL, DLL, DCM, DCI / XCITE / Terminator, programmable slew-rate I / Os, вбудована на кристалі RAM, on-chip CAM (content addressable memory - асоціативна пам'ять)). Після конвертації необхідно визна-лити відповідність отриманого ASIC вихідного FPGA шляхом використання повних Testbenches для функціональної верифікації, яких, як правило, для невеликих FPGA проектів у замовників немає. Іншим рішенням функціональної верифікації отриманого ASIC може бути використання пакету CoreSim, який автоматично виконує такі процедури: 1) Генерація векторів для робочого кристала FPGA та визначення вихідних реакцій для подальшої перевірки FPGA netlist на ідентичність поведінки. 2) Перевірка часових параметрів ASIC, які не повинні перевищувати їх аналогів в FPGA або на друкованій платі. 3)

Тестування ASIC для перевірки правильності його функціонування шляхом подачі отриманих вхідних послідовностей на кристал.

Як ринок EDA вирішує задачу конвертації FPGA в ASIC? Слід відокремлити одного з лідерів на ринку FPGA - компанію Altera. Для останніх найбільш продуктивних серій FPGA (Stratix ™, Stratix II, APEX ™ 20KC і APEX 20KE) фірма пропонує створення кристала з твердою логікою і зі структурою проекту FPGA - Hardcopy ASIC, причому без додаткових витрат на доопрацювання (design for test, заміна FPGA примітивів на ASIC). Altera гарантує: збільшення продуктивності в середньому на 50%, зниження енергоспоживання на 40% у порівнянні з FPGA-кристалом, успіх при імплементації проекту в перший силікон; збереження FPGA-архітектури та netlist, що гарантує відсутність змін у функціональності цифрової системи; надання двох- і трирівневої металевої технології у залежності від серії FPGA. Існує кілька варіантів конвертації FPGA в ASIC (рис. 2.26): 1) «1 to 1» - проект з FPGA переноситься в ASIC без змін, але при цьому для споживача зменшується вартість мікросхеми при масовому виробництві; 2) «n to 1» - інтеграція системи, що складається з декількох кристалів FPGA, в проект на одному силіконі дає виграш за швидкодією, габаритами, енергоспоживанням; 3) «n in 1» - перенесення кількох незалежних кристалів FPGA з плати в один кристал ASIC, кожен з FPGA буде динамічно адресованим після імплементації і виготовлення.

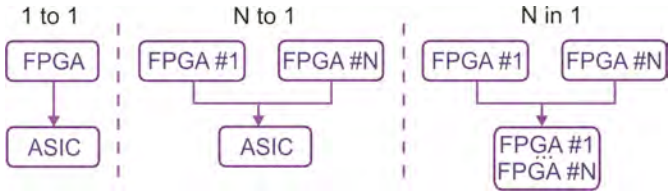


Рис. 2.26. Варіанти конвертації FPGA в ASIC

Автоматизація проектування цифрових систем на кристалах досягла рівня, коли ASIC design flow за витратами інженерної праці наближається до FPGA flow (RTL-flow), коли без втручання проектувальника можна виконати проект на FPGA- або на ASIC-кристалі.

System-on-Chip [67]. Основні переваги системи на кристалі по відношенню до її аналогу, розміщеному на платі: низьке споживання електроенергії; компактні габарити і мінімальна вага; невеликий час розробки; розширена функціональність. Типові компоненти системи на кристалі: процесор (32 біт, технологія RISC); периферія (RTC, LCD контролер, DMA); блок пам'яті; інтерфейси (USB, LVDS, 10 / 100base TMAC); аналогові пристрої (DAC, PLL, ADC). Процесори, які використовуються у SoC, розподіляються на дві групи по реалізації в силіконі: softcore-процесори [68] - синтезуються і імплементуються на PLD; hardcore-процесори - логічна структура не змінюється під

час проектування і жорстко реалізована як примітив на FPGA-кристалі (Xilinx Virtex II Pro - мається 4 IBM PowerPC).

2.6.6. Проектування і верифікації SoC.

SoC на PLD-кристалах. При проектуванні SoC з подальшою реализацією на PLD необхідно розробити модель пристрою і описати її на RT-рівні. Другим кроком є формування тестових впливів (Testbench) для перевірки функціональності розробленої моделі SoC за допомогою програм логічного моделювання [30]. З виникненням концепції системи на кристалі і методології її імплементації в PLD-проекти з'являється новий компонент - програмне забезпечення, яке як і RTL-код впливає на функціонування системи. Виникає питання про інструменти верифікації, які забезпечують моделювання та відлагодження гетерогенної системи, що складається з HW-SW компонентів. Традиційні програми логічного моделювання не підходять для цього за рядом причин: відсутність коштів для налагодження програмного забезпечення [69]; низька продуктивність аналізу проектів, що містять велику кількість рядків програмного коду. Виробниками PLD (Xilinx + Altera = 90%) запропонована методологія роздільної розробки апаратної і программної частин SoC і з'єднання їх разом тільки на етапі створення кри-кристала (рис. 2.27). Іншими словами, тільки при програмуванні PLD-кристала n-й версією hardware можливе завантаження k-ї версії програмного забезпечення в PLD прототип з метою спільної верифікації шляхом під-ключення відладчика програмного забезпечення зі стандартними засобами (покрокове виконання коду, доступ до вмісту регістрів процесора, пам'яті, логічних змінних).

Недоліки методології: до початку налагодження програмного забезпечення апаратна частина повинна бути доведена до стадії закінченого пристрою; при цьому її верифікація є трудомістким процесом, пов'язаним з обмеженим доступом до внутрішніх ліній кристала апаратного прототипу і високими витратами на його створення. Сьогодні на PLD ринку з'явилися softcore про-цесори для створення систем на кристалі (PicoBlaze, MicroBlaze від Xilinx і Nios від Altera). Слід зазначити гнучкість архитектури периферійної шини Avalon для Nios процесора, який є лідером в цьому сегменті ринку. З появою останньої серії Virtex II Pro FPGA, де проектувальникам надаються чотири потужних RISC hardcore IBM PowerPC процесора на кристалі, почалася нова ера розвитку SoC на PLD.

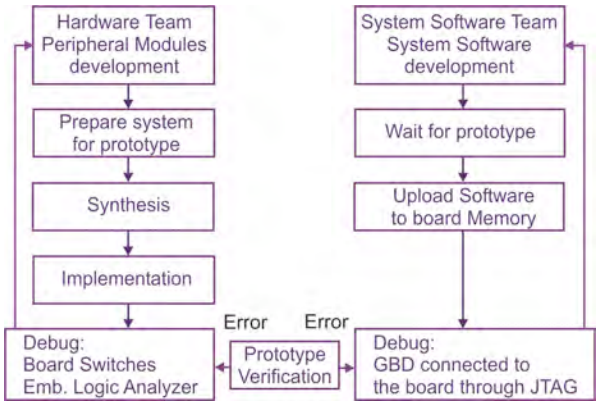


Рис. 2.27. Роздільна розробка частин SoC

SoC на ASIC-кристалах. Інтеграція апаратної та програмної частини системи після виробництва кристала (апаратної частини) - економічно ризикована справа. Можуть виникнути такі проблеми, які приводять до затримки виходу проекту на ринок і до великих матеріальних витрат: 1) програмна частина не повністю закінчена або налагоджена до початку етапу інтеграції, що вимагає додаткового часу на розробку і налагодження вже після створення кристала; 2) налагодження апаратної частини усередині великої мікросхеми є трудомістким і часовитратним завданням через обмежений доступ до внутрішніх ліній чипу; 3) важко локалізовані апаратні помилки доводиться виправляти шляхом великих змін програмної частини системи, або робити коригування HW, що призводить до необхідності повторення циклу проектування, верифікації та виготовлення нового кристала.

Для усунення перерахованих вище ризиків була запропонована спільна верифікація (co-verification) проекту. Вона полягає в інтеграції программної і апаратної частин системи ще до створення першого зразка чипа за допомогою програмного пакета Seamless (Mentor Graphics) [70]. Переваги використання технології спільної верифікації для груп проектувальників і тестерів шляхом створення віртуального прототипу (упродовж місяця) замість його апаратного аналога: 1) прискорення доопрацювання та налагодження програмної частини, що дає часовий виграш близько 3-4 тижнів; 2) простота і швидкість налагодження та локалізації помилок в апаратній частині проекту, яка моделюється в логічному симуляторі; 3) гарантована 100% вірогідність успіху першого силікону, оскільки SW і HW повністю верифіковані до моменту початку виробництва. Структура системи спільної верифікації Seamless (Mentor Graphics) представлена на рис. 2.28 і складається з наступних основних компонентів: програма логічного моделювання; емулятор процесора і пам'яті; модуль зв'язку та синхронізації між собою перших двох компонентів.

Апаратна частина проекту за винятком процесора і пам'яті, які моделюються за допомогою програмних емуляторів, відлагоджується в HW-Simulator. Програмні емулятори в 10 - 1000 разів збільшують швидкість моделювання процесора і пам'яті. Емулятор процесора дозволяє використувати додаткові налагоджувальні інструменти: breakpoints C ++ / Assembler, виконувати покрокове налагодження коду, мати доступ в будь-який момент часу до вмісту регістрів процесора і змінних, які недоступні при моделюванні процесора в HW-simulator. Основна проблема спільної верифікації в Seamless - координація роботи програми логічного моделювання з швидкодією близько 10-30 инструкций за секунду і пакета ISS (Instruction Set Simulator) [71], який може обробляти за секунду до 500 000 команд. Мінусом системи Seamless є її орієнтація тільки на ASIC ринок і підтримку hardcore процесорів, а softcore Xilinx MicroBlaze процесор - один з лідерів на FPGA ринку - залишається осторонь. Пакет Seamless - 60000 доларів, доповнений моделью процесора (6000 - 22000 доларів), орієнтований на ASIC ринок, де вартість кожного проекту вимірюється мільйонами доларів [72]. Для розробників проектів на FPGA, де більшість пакетів проектування поширюється виробниками кристалів безкоштовно, практично неможливо використовувати пакет Seamless через високий ціновий бар'єр.

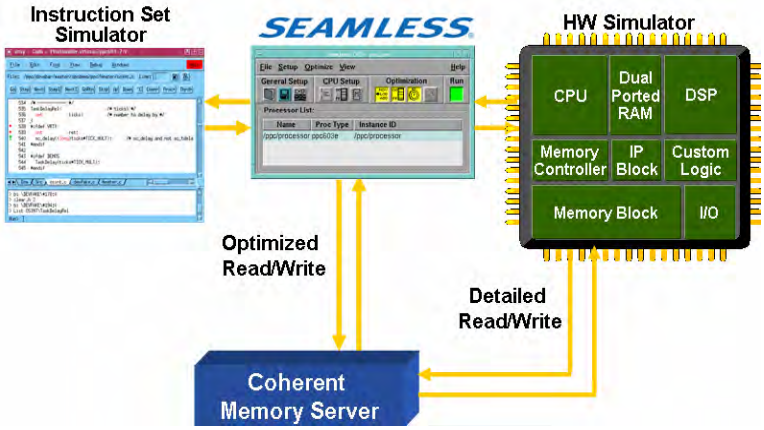


Рис. 2.28. Структура Seamless-пакету для верифікації

Висновки. Вочевидь велике зростання PLD-ринку, особливо з приходом в нього процесорів (softcore, hardcore), з можливістю конвертації FPGA проекту в ASIC. Отже, сегмент ринку, займаний ПЛІС, буде тільки розширюватися і відвойовувати місце, як у інтегральних схем, так і у программного забезпечення. Для виробництва 500 одиниць цифрового виробу економічно вигідно використовувати FPGA-кристали. Для великих партій більше 50 000 починає вигравати cell-based IC технологія за ціною за одиницю виробу при покупці на ринку. Системи на кристалі раніше були лише привілеєм IC, але з появою останніх серій ПЛІС з процесорами (PowerPc, ARM hardcore,

PicoBlaze, MicroBlaze, Nios) SoC стали повноправними учасниками ринку ПЛІС. Зростання ринку SoC обумовлений збільшенням вимог до продуктивності і надійності цифрових систем, а також до зменшення їх вартості. Раніше ASIC був практично єдиною технологією, що дозволяє розробникам розміщувати мультипроцесорні системи на одному кристалі. Абсолютно нові можливості надані інженерам з появою останніх серій FPGA. Щоб повністю використовувати ці переваги, необхідні нові методології та автоматизовані засоби, які дозволять підвищити продуктивність розробки. Однією з таких ASIC-методологій, яка стає все більш актуальною для FPGA, є апаратно-програмна спільна верифікація. Слід також зауважити, що розробка автоматизованих засобів проектування для використання потужностей, закладених в останніх серіях FPGA, значно відстає від нанотехнологій. На ринку EDA відсутні ефективні засоби, що дозволяють виробляти HW-SW спільну верифікацію до створення апаратного прототипу системи з метою виключення помилок на ранніх стадіях проектування, що призводить до зменшення загальних витрат на налагодження проекту.

2.7. Методологія системного проектування TLM

Основною концепцією TLM [73] (Transaction-Level Modeling) є створення моделі рівня деталізації, достатньої для вирішення окремої задачі проектування компонентів системи. Аналіз тільки окремих деталей дозволяє значно підвищити швидкість верифікації моделей. Концепція TLM є незалежною від будь-яких мов, однак базується на мовах високого рівня програмування, наприклад, таких як SystemC або SystemVerilog. Transaction Level Modeling (TLM) пропонує bit-true, address-map accurate, cycle-less моделі. TLM визначає транзакцію (SoC подія), як миттєву передачу даних (комунікацію) або синхронізацію між двома модулями, визначеними hardware / software системною специфікацією. Термін TLM визначений для відповідного ESL [75] (Electronic System Level) рівня проектування. Передбачається використання мов програмування високого рівня для моделюровання апаратури на високому концептуальному рівні, не вдаючись у деталі її реалізації. До таких мов можна віднести SystemC, SpecC, Hpascal, System Verilog, HardwareC.

Моделі абстракції. Основною концепцією TLM є те, що коммунікації між системами і підсистемами і їх функціональність можуть бути розроблені і деталізовані незалежно один від одного. За принципом деталізаціі комунікацій і функціональності моделі можуть бути поділені на Un-timed (UT), Approximately-timed (AT), Cycle-timed (CT) (рис. 2.29) [74]. Моделі, що підтримують детальний потактовий (cycle-timed) опис функціональності і комунікацій, є моделями рівня передачі даних (RTL - register-transfer level). Моделі без використання деталізації часу при описі функціональності і комунікацій (un-timed-моделі) називаються SAM (System Architectural Model)-модель. SAM-модель є засобом комунікації між групами проектувальників алгоритмів, програмного забезпечення та апаратури. До решти чотирьох моделей, зображених на рис. 2.29 і позначених TLM, відно-сятся моделі з

апроксимованим часом (approximately-timed). Вони використовують статичний або розрахунковий час, або часові вимоги підсистем.

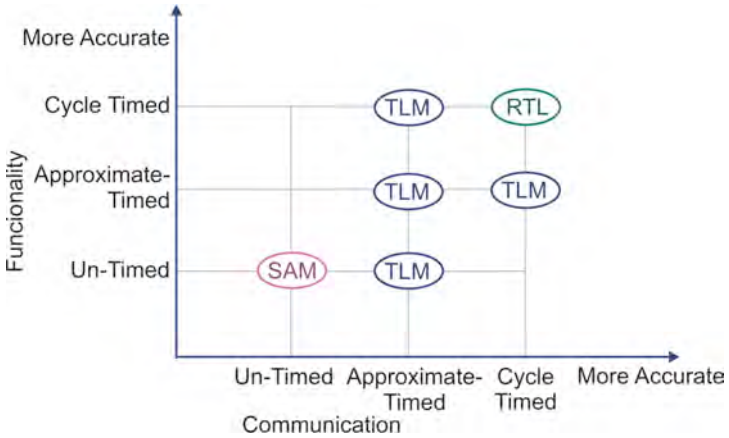


Рис. 2.29. Моделі абстракції зображення цифрових пристроїв

На початковій стадії проектування немає необхідності точного опису процесу передачі даних через шину, тому використовується модель, де цей крок виконується за одну подію або транзакцію (component-assembly model). На наступному етапі проектування може виявитися важливим знання точного числа тактів при передачі інформації для кожного сигналу всередині блоків, однак процес передачі даних по шині розглядається при цьому як одна транзакція або подія (моделі bus-arbitration або cycle-accurate). Коли специфікація стає повністю визначеною, шина моделюється за допомогою транзакцій або подій для зміни сигналу (bus functional or RTL model).

Методологія проектування з TLM. Нова технологія проектування [75, 76] складається з двох частин: стандартної від RTL до створення або прошивки мікросхеми (layout flow) та додаткової від системного рівня до RTL. Проектування починається зі створення системної специфікації, що подається у паперовому вигляді. Грунтуючись на специфікації, системний проектувальник виконує розподіл архітектури пристрою на hardware / software складові. Наступним етапом є TLM. Одним з призначень TLM є забезпечення фази паралельної розробки hardware / software. В цьому випадку TML служить як універсальна вихідна точка для software і architecture команд проектування, для раннього проектування програмного забезпечення та структурного аналізу архітектури. Вона також використовується командами тестування для розробки середовища верифікації та тестів, таким чином забезпечуючи їх готовність на момент розробки RTL-платформи. Між тим проектувальники hardware створюють RTL-проект для системи, розробленої на основі SoC RTL-платформи. Таким чином, досягається hardware / software спільне проектування, що є одним з основних відмінностей нової методології від кла-

сичної. Оскільки доступною є RTL платформа, з її допомогою також можуть бути вирішені завдання: перевірка сумісності при виконанні очікуваних параметрів, hardware верифікація, низкоурівнева інтеграція software з hardware. Ці завдання вирішуються паралельно з кроками емуляції, синтезу та аналізу результатів імплементації.

Потрійна абстракція. Нова методологія проектування визначає структуру потрійний абстракції: функціональний вид SoC; архітектурний вигляд SoC; мікроархітектурний вигляд SoC [74]. 1) Функціональний вигляд SoC (SoC Functional View). Найвищий рівень абстракції, вивчає поведінку системи з точки зору користувача. Функціональний вигляд розробляється без внесення будь-яких деталей реалізації (архітектури чи адресного простору). Модель являє собою програми, що реалізують алгоритм функціонування пристрою. Параметри продуктивності системи описуються окремо у вигляді паперової специфікації. 2) Архітектурний вигляд SoC (SoC Architecture View). Наступним, більш низьким етапом проектування, є архітектурний вигляд SoC, який реалізується TLM-моделями. Крім того, архітектурний вигляд застосовується інженерами верифікації як довідкова модель, яка виступає "золотим еталоном" для генерування функціональних тестів, використовуваних для верифікації пристрою на подальших стадіях проектування. 3) Мікроархітектурний вигляд SoC - це самий нижній рівень потрійної абстракції. Містить всю інформацію для виконання часового моделювання за сінхротактами (cycle-accurate). Загальноприйнятою практикою для цього рівня є використання RTL-моделей, створених за допомогою мов опису апаратури, таких як VHDL або Verilog. В [75] ілюструються ситуації використання мікроархітектурного вигляду SoC, в [76] описуються шляхи застосування багаторівневих моделей для створення проектів.

У загальному випадку сучасний процес проектування може бути подано схемою, зображеною на рис. 2.30. Проектування починається зі створення Product Requirement Document (PRD), що складається замовником або маркетинговою групою. На основі PRD розробляється САМ, на цьому етапі вносяться необхідні зміни та уточнення в PRD. Потім SAM перетворюється в один з видів TLM-моделей, наприклад, спочатку в component assembly модель, а потім в bus arbitration модель. При виборі відповідної мови для SAM така модель може бути повторно використана (reuse) та шляхом деталізації перетворена в TLM.

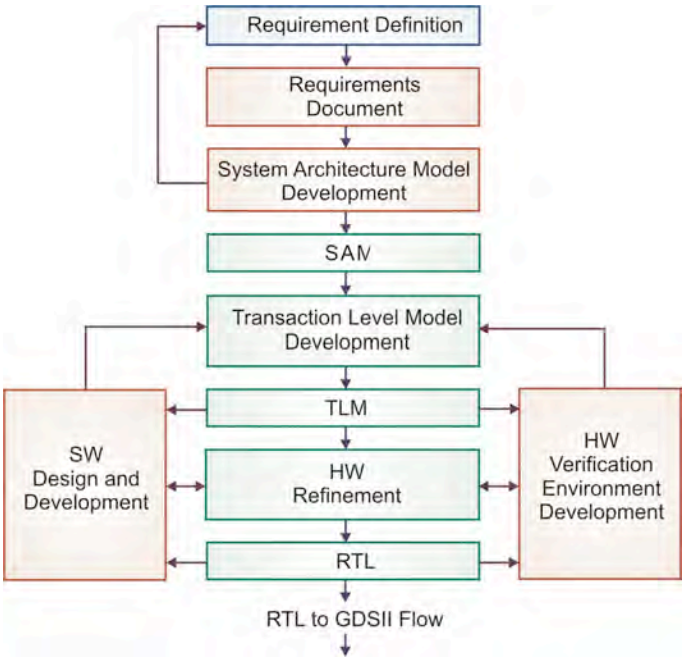


Рис. 2.30. Сучасні етапи проектування SoC

Завдання, які вирішуються за допомогою TLM-моделей: 1) Визначення властивостей їмплементації, таких як HW / SW partitioning; HW partitioning між ASIC, FPGA і board реалізаціями; розробка архітектури шини; визначення або вибір співпроцесора. 2) Розробка платформи для системного програмного забезпечення. 3) Створення «золотого еталона» для функціональної верифікації апаратури. 4) Розробка мікроархітектури апаратури і основ для створення детальної специфікації hardware. До переваг використання TLM-моделей відносяться: можливість більш ранньої розробки software; раннє і якісне створення Testbench для виконання функціональної верифікації; наявність єдиного і безперервного шляху проектування від вимог користувача до розробки детальної специфікації hardware і software.

Мови проектування SoC. Різні мови розроблялися для різних аспектів проектування систем. C / C ++ переважно використовуються для розробки програмного забезпечення вбудованих систем. Мови опису апаратури (HDLs), VHDL і Verilog застосовуються для проектування і синтезу цифрових схем. Мова Vera призначено для функціональної верифікації складних спеціалізованих інтегральних схем (ASIC). Нова мова SystemVerilog є доповненою версією мови Verilog, призначеною для вирішення декількох завдань апаратно-орієнтованого системного проектування. Matlab та інші подібні інструменти і мови, такі як SPW і System Studio, широко використовуються для збору системних вимог і розробки алгоритмів цифрової обробки сиг-

налів. На рис. 2.31 зображені галузі використання мов системного проектування [79]. SystemC [77] був створений для проектування SoC - пристроїв на ESL - рівні (electronic system-level) для розробників, що використовують C / C ++. Стандарт: IEEE Std. 1666-2005 SystemC (Open SystemC Initiative (OSCI)) [78]. SystemC - це єдина мова для проектування та верифікації, яка дозволяє подавати архітектурні та інші атрибути моделей системного рівня у формі класів мови C ++ з відкритим вихідним кодом. Це дає можливість виконувати проектування і верифікацію на системному рівні, незалежно від будь-яких деталей реалізації апаратури і програмного забезпечення. Також мається можливість виконання спільної верифікації з RTL-проектами. Високий рівень опису дозволяє значно швидше і продуктивніше виконувати аналіз вибору компромісного рішення для архітектури, ніж це можна було зробити на RT-рівні. Більш того, верифікація системної архітектури та інших системних атрибутів відбувається швидше, ніж для точної за часовими параметрами або розташуванню зовнішніх контактів RTL-моделі.

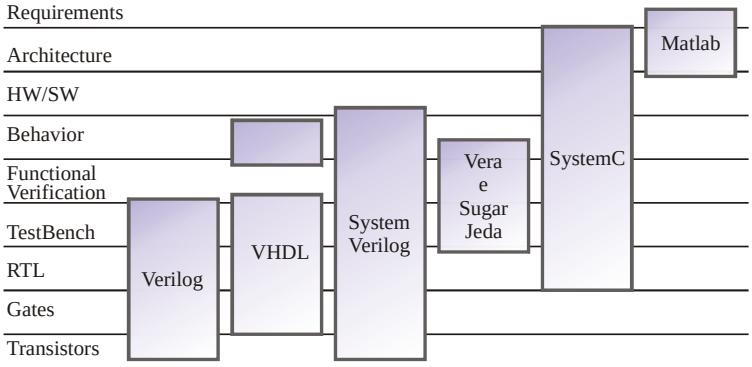


Рис. 2.31. Області застосування мов системного проектування

SystemVerilog - перша мова опису та верифікації апаратури (HDVL), яка є розширенням стандарту IEEE 1364 Verilog-2001 [81] і дозволяє значно підвищити продуктивність проектування проектів, що складаються з великого числа вентилів, з великою нагруженностью шини. Стадарт: IEEE Std. 1800-2005 SystemVerilog (Accellera). SystemVerilog призначений головним чином для створення безперервного процесу проектування SoC, починаючи від поведінки і закінчуючи GDSII-описом. Тим не менш, SystemVerilog також підтримує створення TL-моделей. Таке співпадіння властивостей верифікації між SystemC і SystemVerilog є основою для зв'язків між системним рівнем і реалізацією кристала. SystemVerilog's Direct Programming Interface (DPI) дозволяє викликати функції C / C ++ / SystemC, і навпаки, виконувати спільне моделювання Verilog-моделей з SystemVerilog і SystemC блоками. SystemVerilog підтримує повне середовище верифікації, використовуючи Constraint Random Generation, Assertion Based Verification і Coverage Driven Verification [80]. Він може бути застосований для HW-проектування та верифікації; SystemVerilog може бути частиною середовища

Verilog or VHDL Testbench (TB), частиною проекту або зовнішнього модуля. Наприклад, проектувальник може використовувати можливості Random Generation, включивши деякі з властивостей SystemVerilog Random в поточному TB без необхідності цілого середовища верифікації. Таким же чином інженери верифікації можуть включити деякі ассерції, як частина повного симулятора кристала, для верифікації правильності шини або протоколів.

Методологія верифікації SystemVerilog грунтується на трьох типах блоків, які можуть застосовуватися окремо або всі разом: 1) Stimuli. Проект, що використовує автоматично генеровані сценарії з псевдовипадковими послідовностями (constrained-random - CR) генерування тестів. 2) Check. Перевірка поведінки проекту (ассерція) і вихідних даних (scoreboard) на предмет верифікації правильності операцій. 3) Measure. Метрика функціонального покриття для забезпечення зворотного зв'язку при генеруванні та аналізі виконання процесу верифікації. Використання методів верифікації, які включають процес моделювання, налагодження та оцінки покриття, називається Coverage Driven Verification.

VHDL - це абревіатура від англійського виразу VHSIC Hardware Description Language. В свою чергу, VHSIC походить від назви програми Very High Speed Integrated Circuit (високошвидкісні інтегральні схеми). Ця програма, фінансована Міністерством Оборони США, ставила за мету розвиток нового покоління високошвидкісних інтегральних схем. Перша версія мови була подана у 1985 році. Згодом її було передано IEEE для стандартизації. У 1987 році мову було затверджено як стандарт IEEE 1076-1987. Через п'ять років її було розглянуто повторно, внаслідок чого нова версія 1076-93 [81] містить ряд додаткових можливостей. Наступні версії стандарту були випущені у 2000 і 2002 роках [82]. VHDL може бути застосовано для створення функціональних і RTL-моделей, користується популярністю серед фахівців з автоматизованого проектування (CAD) електронних систем [83-95]. Для синтезу був розроблений пакет IEEE 1164 [89]. На практиці великий виробник систем автоматизованого проектування підтримує VHDL.

Verilog - мова опису апаратури, розроблена у 1985 р Філіпом Мурба (Philip Moorby) [96], яка потребувала простого, наочного і ефективного способу для опису цифрових схем, моделювання та аналізу їх функціонування. Мова стає власністю Gateway Design Automatiion, а потім Cadence Design Systems. Перший стандарт Verilog з'явився IEEE у 1995 році [97]. Останнє оновлення мови виконано у 2005 р. [98]. Verilog має таку ж, а може й більшу, популярність, як і VHDL, широко використовується у промисловості при проектуванні ASIC-пристроїв [99-103].

Класифікація інструментів проектування ESL-рівня. У 2004 р. International Technology Roadmap for Semiconductors (ITRS) визначив ESL (Electronic System Level), як рівень над RTL, для hardware і software проектування [104]. ESL-рівень являє собою поведінковий (до розподілу на HW / SW) та архітек-

турний рівні, а також він дозволяє збільшувати продуктивність на 200000 вентилів у людино-рік. Переваги використання ESL полягають: у підвищенні рівня абстракції представлення системи; в можливості використання більш високого рівня для reuse проектів; в підтримці безперервного ланцюга проектування, починаючи з верхнього абстрактного рівня. Проектування являє собою послідовність повторюваних кроків від абстрактного рівня до створення готового продукту. Розроблена Sangiovanni-Vincentelli і Martin [105, 106], платформенно орієнтована концепція (Platform-based design (PBD)) представляє кожен крок процесу проектування за допомогою Y-діаграми (рис. 2.32), де ліва гілка описує функціональність розроблюваної системи (тобто поставлену перед проектувальником задачу), а права - елементну базу (платформу), на якій планується виконати проект. Нижня гілка відповідає реалізації заданої функціональності на вибраній платформі, яка містить: 1) бібліотеку елементів, у тому числі IP-cores блоки та комунікаційні структури, а також правила об'єднання елементів; 2) методи оцінювання характеристик, пов'язаних з кожним елементом, наприклад, споживану потужність або затримку поширення.

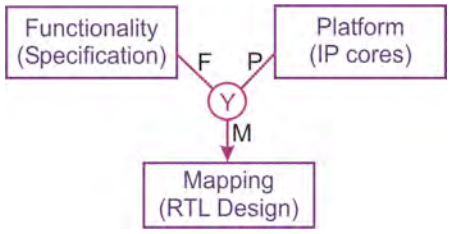


Рис. 2.32. Y-діаграма ESL проектування

Кожне допустиме з'єднання елементів платформи називається екземпляром платформи (platform instance). Функція mapping вибирає компоненти системи і визначає функціональність кожного елемента, які реалізують повну функцію цілої системи, можливо з надмірністю. У проектувальників є можливість оптимізувати цей процес згідно множині вихідних даних і констант, визначених вимогами вартості або якості. Такий підхід є узагальненням процесу проектування, який проектувальники використовують роками на окремому абстрактному рівні. Наприклад, таким же чином може бути інтерпретований процес синтезу, який включає наступні кроки: 1) функціональність проекту представляється за допомогою RTL-коду або булевих рівнянь; 2) платформі відповідає бібліотека вентилів або складніших логічних блоків; 3) операція mapping, для даної ситуації, - це крок логічного синтезу, який реалізує функціональність як з'єднання венти-лей, оптимізуючи згідно множині вихідних даних: розмір, споживану потужність, часові параметри. Така система поглядів описує єдину методологію проектування і тому підходить для визначення місця сучасних інструментів і методик проектування, що надає можливість інтеграції їх в єдину методологію системного проектування [107, 109, 110].

Класифікація класів інструментів виконується за Y-діаграмою з використанням додаткового критерію класифікації, що відноситься до рівня абстракції, на якому вони працюють: 1) Bin F - функціональне зображення проекту без архітектурної реалізації та фізичних параметрів, подібних часу або споживаної потужності. До цієї категорії відносяться інструменти, які маніпулюють, моделюють і формально або неформально аналізують функціональний опис проекту. Сюди відносяться інструменти, які часто служать для швидкого захоплення проекту та створення його специфікації без будь-яких згадок деталей реалізації. На цьому рівні опис може містити поведінковий (паралельний) код [105, 108] або принципи створення комунікацій (комунікаційні протоколи). Деякі інструменти обробляють тільки один MoC - наприклад, кінцеві автомати FSM. Інші є більш універсальними - обробляють множину MoC або не мають обмежень. 2) Bin P становить бібліотеку модулів для реалізації функціонального опису. Модулями є: процесори, блоки пам'яті, сопроцесори, FPGA, користувальницькі блоки апаратури і міжз'єднання (шини, мережі), а також зв'язуюче ПЗ: операційні системи для процесорів, арбітражні протоколи для шин. Ця категорія містить підтримку платформ або платформних компонентів, а також інструментів та мов, які дозволяють описувати, маніпулювати або аналізувати елементи платформи. 3) Bin M являє перенос функціональності в апаратуру (mapping), який виконується проектувальником або автоматично. Містить інструменти, що виконують накладення функціонального опису на елементи платформи, включаючи можливості їх розвитку та виконання кроку синтезу для переходу на більш детальний рівень опису. Інструменти цієї групи дуже різноманітні залежно від різних стилів проектування, MoC і підтримуваної області застосування. Для классифікації таких інструментів використовується поняття «метаклас», що позначається комбінацією букв F, P і M. 4) Metabin FP. У цю категорію входять мови проектування, які можуть представляти одночасно функціональність і архітектуру. Зазвичай вони висловлюють алгоритми, різні стилі комунікацій і структур для різних MoCs. 5) Metabin FM. Являє інструменти, які забезпечують комбінацію функціонального опису та можливості аналізу, плюс mapping і можливості синтезу. В цьому випадку архітектура системи зазвичай фіксована. Цей недолік гнучкості часто компенсується високою якістю результатів, що досягаються реалізації. 6) Metabin PM. Містить інструменти, які комбінують архітектурні засоби та mapping. Ці інструменти мають тісний зв'язок між коштами, які вони підтримують, і способами реалізації функціональності цих коштів. Вони вимагають використання інших засобів для деяких аспектів системного проектування. 7) Metabin FPM. До цієї категорії відносяться всі інструменти, які повністю підтримують всі три аспекти PBD-парадигми. Зокрема, Metropolis підтримує цю парадигму, покриває всі bins на всіх рівнях абстракції.

Критерії класифікації за рівнями абстракції: 1) Системний рівень (S) - відповідає гетерогенним проектам, які використовують різні моделі обчислення MoCs (models of computation) для зображення функції, моделі та реалізації; 2) рівень компонентів (C) - являє підсистеми, що містять однотипні

компоненти; 3) рівень імплементації (I) містить в собі фінальний крок проектування, коли команда проектувальників виконує завершення роботи.

2.8. Практична значимість анализу технологій проектування

ПЛІС надає розробникам нові можливості для проектування швидкодіючих спеціалізованих цифрових систем на кристалах. Доказом перспективності програмованої логіки є поява нових, що мають більш досконалу архітектуру, поколінь ПЛІС з вбудованими процесорами, а також зростання обсягів їх випуску провідними закордонними виробниками. Аналіз тенденції розвитку архітектур програмованої логіки дає підстави стверджувати, що найближчим часом основу елементної бази цифрових (реконфігуруємих) систем становитимуть CPLD, FPGA, HardCopy, що наздоганяють за продуктивністю та вартістю ASIC, але при цьому не вимагають витратної тестової і верифікаційної методології для забезпечення масового виробництва мікросхем. Інтерес для вчених і практиків становить і співвідношення економічних показників, що створюють цілісну картину світового ринку EDA. Вклад двох провідних компаній Xilinx і Altera, що формують практично весь ринок PLD (FPGA, CPLD) кристалів, складає 86%. При цьому частка Field Programmable Gate Array на ринку - 2,8 міліардів доларів, а Complex Programmable Logic Device - 0,5. Що стосується виробників засобів EDA, то 95% ринку електронних технологій формується четвіркою лідерів: Cadence, Synopsys, Mentor Graphics, Magma. Ці фірми обслуговують і визначають сучасну політику технологій проектування цифрових систем і мереж на кристалах.

Для споживачів ринку електронних продуктів третіх країн інтерес становить табл. 2.1, яка ілюструє переваги застосування FPGA в порівнянні з ASIC.

Таблиця 2.1. Переваги застосування FPGA у порівнянні з ASIC

Characteristic	FPGA	ASIC
Час виходу на ринок	Short	Long
Мінімальна партія	None	High
Цена одиниці в партії	High	Low
Витрати на підложку	None	High
Реконфігуруємість	High	None
Продуктивність	Medium	Very High
Щільність вентилів	Medium	Very High
Енергоспоживання	High	Low
Складність розробки	Medium	Very High
Складність тестування	Low	High
Тривалість циклу розробки	Hours	Months

До характеристик, наведених у таблиці, слід додати факт прозорості PLD (теоретична відсутність програмно-апаратних закладок деструктивного характеру) порівняно з ASIC, що дає можливість використовувати FPGA для апаратурної реалізації критичних технологій (атомна енергетика, космічні та авіаційні системи, банківські та спеціальні засоби захисту інформації).

Випуск останніх серій FPGA Xilinx Spartan 6, Virtex 6 і Altera Stratix місткістю понад 10 мільйонів вентилів з вбудованими IBM PowerPC і ARM процесорами забезпечив новий рівень схемотехніки, необхідний для проектування цифрових систем і мереж на FPGA-кристалах. Апаратні можливості, надані нанотехнологіями на 300 мм силіконовій пластині (в найближчому майбутньому - 450 мм), вимагають переходу на новий рівень розвитку засобів проектування, де одним з критичних і часовитратних процесів є функціональна верифікація, що займає до 70% від загального часу створення цифрової системи. За наявністю процесора ц проектованій системи головним об'єктом аналізу стає узгодження функціонування апаратних і програмних блоків. Для їхньої спільної верифікації необхідно мати спеціалізовані засоби та методології, що забезпечують ефективне налагодження проектованих систем і мереж на FPGA. Спільна HW / SW верифікація успішно використовується для налагодження проектів на ASIC. Підвищений інтерес на ринку EDA до імплементації SoC в FPGA, що містять процесори, визначає актуальність завдання розробки аналогічних засобів для налагодження цифрових систем на ПЛІС.

Сучасна стратегія проектування, наведена на рис. 2.33, грунтується на використанні бібліотек вже існуючих і верифікованих компонентів вигляді IP-cores, що входять в NoC-компілятор.

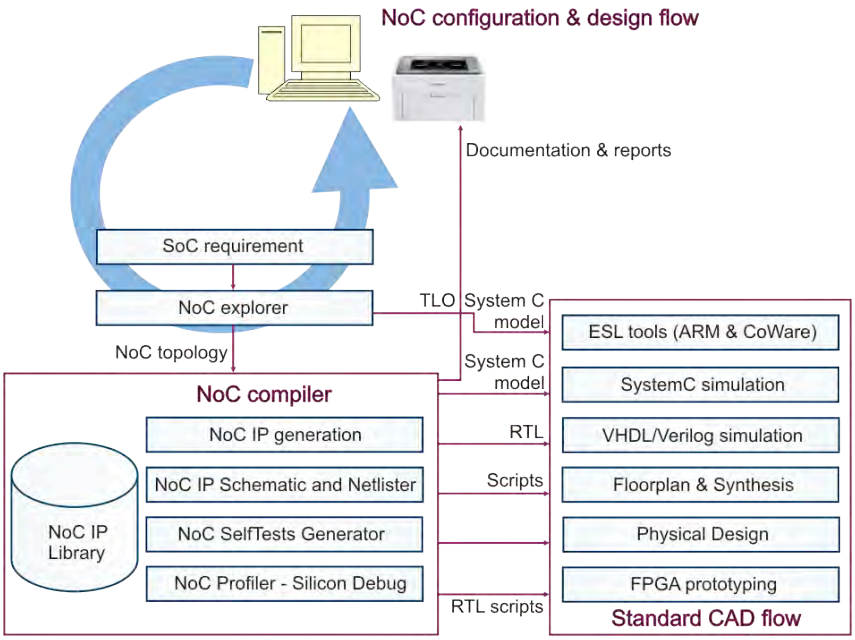


Рис. 2.33. Стратегія проектування цифрових систем на кристалах

Основна ідея ESL-технології заснована на ітеративному процесі отримання оптимального покриття специфікації вже існуючими компонентами з електронної бібліотеки готових апаратних рішень IP-cores. Компілятор також містить засоби для синтезу нових компонентів NoC, наступного генерування NoC IP netlist і синтезу тестів для кожного модуля. Результат на виході компілятора - розводка і прототипування компонентів на силіконовому кристалі. Власне маршрут проектування використовує останні досягнення в світі електронних технологій, що містять ESL Design, заснований на паралельній розробці програмних і апаратних компонентів, що використовує TLM (Transaction Level Modeling) для створення та верифікації інтерфейсних модулів. При описі проекту використовуються системні (SystemC, SystemVerilog) і апаратно-орієнтовані мови регистрового рівня (VHDL, Verilog) для моделювання, верифікації та синтезу компонентів на всіх трьох рівнях проектування (System-, RT-, Gate-Level), з наступною фізичною імплементацією і прототіпуванні проекту у кристал FPGA. Ітеративна процедура проектування повинна бути обмежена рівнем системного проектування, що призводить до суттєвої економії часових і матеріальних витрат при корекції помилок. Це значить, що в процесі верифікації на найвищому рівні мають бути враховані вимоги, що стосуються основних параметрів і характеристик проходження проекту аж до виготовлення продукту на кристалі. Така технологія відома під назвою Design for Manufacturability.

Історично склалася ситуація, коли Європа і Америка володіють технологіямі в області електроніки, а країни, що відносяться до Східній Європі, традиційно сильні теоретичними дослідженнями і математичної культурою. Тому науковий потенціал країн колишнього СРСР стає все більш затребуваним на ринку електронних технологій в частині математичних рішень і створення Software. Відомий вислів К. Маркса "Наука тільки тоді досягає досконалості, коли їй вдається користуватися математикою" можна було б продовжити, наприклад, так: а ефективність математики доводиться шляхом її впровадження в практику. Справді, історія розвитку радіоелектронних технологій показує, що ефект від будь-якої їх практичної реалізації не менш, ніж на 50% залежить від математичного забезпечення (теорії, моделі, методи, алгоритми). Практично будь-яке наукове дослідження пов'язане з математичною формалізацією розглянутих процесів і явищ, перетворюваних на загальноприйняті поняття моделей і методів. Максимальною мірою це відноситься і до дискретної частини радіоелектроніки, яка в найближчі роки покриє на 90% позначену галузь. Інакше, цифрове виконання радіоелектронних пристроїв, які раніше були тільки аналоговими (приймачі, передатчики, частотні перетворювачі, фільтри), призводить до такого співвідношення ринку дискретних і аналогових пристроїв, коли останні будуть виконанувати роль інтерфейсів в форматі АЦП- і ЦАП- виробів.

Звідси випливає досить цікавий висновок: XXI століття буде століттям розвитку телекомунікаційних, але цифрових систем, в яких комп'ютер буде виконувати роль розосередженого в просторі спеціалізованого компонента, орієнтованого не тільки на виконання традиційних задач (перетворення, зберігання інформації), але і на прийом і передачу інформації в різних форматах, в тому числі і за допомогою безпровідних засобів зв'язку. Досить сказати, що сучасні SoC інтегрують не тільки АЛУ, пам'ять, пристрій управління, порти введення і вивода, а й обов'язково приймач з передавачем, що дозволяє будувати з таких цеглинок ефективні макросистеми, розосереджені у просторі. Для побудови таких систем необхідно мати спеціалізовані обчислювачі, спрямовані на високоефективне рішення конретних проблем шляхом розробки алгоритмів і їх подальшої імплементації в універсальну машину або на силіконовий кристал, виконаний у вигляді ASIC або PLD.

Слід зазначити, що еволюція (життєвий цикл) практичної реалізації математичної ідеї (закінченого рішення) проходить кілька стадій: 1) Програмна реалізація на універсальній машині. Тут перевіряється спроможність алгоритму, виявлення помилок, його ефективність. 2) Створення IP-core, як інтелектуальної власності, інваріантної стосовно її подальшої імплементації в кристал, описаної на HDL-мові. На даній стадії розробляються засоби тестування і верифікації програми. 3) Імплементація IP-core в кристал PLD, як заявка на міжнародне визнання ефективності

вирішення конкретної проблеми. Випуск кристалів в малих серіях для продажу і подальшого вивчення попиту. 4) Імплементація IP-core в кристал ASIC, коли ефективність вирішення завдання визнана на ринку математичних ідей та електроніки, а також визначено необхідність тиражування кристала у великих серіях. 5) Апаратурна реалізація рішення імплементується в математичний співпроцесор універсальної машини. Це рівень вищих математичних досягнень, які удостоюються честі бути доступними всьому людству.

В останні роки домінуючою є тенденція до мікромініатюризації виконання цифрових проектів, завдяки впровадженню мікро- і нано-технологій. Тайвань вже задекларував про готовність підприємств до випуску 28 nm кристалів, тоді як 20% ринку використовує 45 топологію мікросхем. Board практично повсюдно замінюється на Chip, більш того, в кристал імплементується і мережа обчислювальних пристроїв, з обміном інформації за IP-протоколом. Разом з тим мікромініатюризація дає можливість непомітно вбудовувати цифрові системи в традиційні вироби повсякденного попиту. Так з'являються wearing (embedded) systems - системи, вбудовані в одяг і взуття, які вирішують проблему енергоспоживання за допомогою тепловіддачі тіла людини. Таким чином, мікромініатюризація дає новий імпульс до розвитку вбудованих спеціалізованих цифрових систем, які в 50% випадків своєю функціональністю можуть замінити коштовні універсальні комп'ютери, виграючи до того ж в розмірах, мобільності і енергоспоживанні.

Кожен заключний тезис має методологічне значення для перспектив розвитку радіоелектронного виробництва і ринку країн, що розвиваються, оскільки всі компанії світу обов'язково враховують позначені тенденції для планування своєї інвестиційної діяльності.

2.9. Література

1. Bass M.J. and Cristensen C.M. The future of the Microprocessor Business. IEEE Spectrum, April 2002.– P. 34-39.

2. Philip E. Ross. 5 Commandments. IEEE Spectrum. December 2003. – P. 30-35.

3. Nick Tredennock, Brion Shimamoto. “Go Reconfigure”. IEEE Spectrum. December 2003. – P. 37-40.

4. Benini L., Macii A., Macii E., Omerbegovic E., Poncino M., Pro F. Energy-Aware Design Techniques for Differential Power Analysis Protection // Design Automation Conference (DAC 03). – 2003. – P. 36-42.

5. Gupta B., Borgatti M. Different Approaches to Add Reconfigurability in a SoC Architecture // Design Automation and Test in Europe, 2003. – 2003.– P. 398-402.

6. Blodget B., McMillan S., Lysaght P. A Lightweight Approach for Embedded Reconfiguration of FPGAs//Design Automation and Test in Europe (DATE 03). 2003. – P. 399- 401.

7. Walder H., Platzner M. Online Scheduling for Block-Partitioned Reconfigurable Devices // Design Automation and Test in Europe (DATE 03). – 2003. – P. 290-296.

8. Geppert L. The New Indelible Memories // IEEE Spectrum. – 2003. – P. 49-54.

9. M. Borgatti, L. Calм, G. De Sandre, B. Forкt, D. Iezzi, F. Lertora, G. Muzzi, M. Pasotti, M. Poles, P.L. Rolandi (STMicroelectronics) A Reconfigurable Signal Processing IC with Embedded FPGA and Multi-Port Flash Memory // Design Automation Conference (DAC 03). – 2003. – P. 691-696.

10. Amlani I., Zhang R., Tresek J., Nagahara L., Tsui R. K. Manipulation and Characterization of Molecular Scale Components // Design Automation Conference (DAC 03). – 2003. – P. 276-278.

11. Don Edenfeld, Andrew B Kahng, Mike Rodgers, and Yervant Zorian. 2003 Technology Roadmap for Semiconductors // Computer, Janvary, 2004. p. 47-56.

12. Yoo S., Jerraya A. Introduction to Hardware Abstraction Layers for SoC // Design Automation and Test in Europe (DATE 03). – 2003. – P. 336-338.

13. Nikolaos S. Voros, Sanchez Luis, Alonso Aleandro, Alexios N. Birbas, Michael Birbas, Ahmed Jerraya. Hardware-Software Co-Design of Complex Embedded Systems. Design Automation for Embedded Systems. Boston: Kluwer Academic Publishers. – 2003. – P. 5-34.

14. Boehm B. A Spiral model of Software Developement and Enhancement. Science of Computer Programming. – 1988. – Vol. 8. – P. 231-274.

15. Vercauteren, S., J.V.D. Steen, and D Verkest. Combining Software Synthesis and Hardware-Software Interface Generation to Meet Hard Real-Time Constraints. DATE’99. – 1999. – P. 177-187.

16. Hard, D. StateCharts. A Visual Formalism for Complex Systems. – 1987. – Vol. 21. – P. 77-82.

17. Jerraya A.A. System Level Synthesis SLS. TIMA Laboratory. Annual Report 2002. – P. 65-75.

18. Data Book. Altera, 1999.

19. The Programmable Logic Data Book. Xilinx, 1999.

20. Programmable Logic Data Book and Design Guide. Actel, 1999.

21. Соловьев В.В., Васильев А.Г. Программируемые логические интегральные схемы и их применение. Мн.: «Беларуская навука», 1998. – 266 с.

22. Стешенко В. Школа разработки аппаратуры цифровой обработки сигналов на ПЛИС // Chip News. – 1999. – N8. – С. 2-6.

23. Brown S., Zvonko Vranesic. Fundamentals of Digital Logic with VHDL Design. The McGraw Hill, 2000. – 530 p.

24. Benoit Clement, Richard Hersemeule, Etienne Lantreibecq, Bernard Ramanadin, Pierre Coulomb, Francois Pogodalla Fast. Prototyping: a System Design Flow Applied to a Complex System-On-Chip Multiprocessor Design. Proceedings of the Design Automation Conference. – 1999. – P. 420-424.

25. Nollet V., Marescaux T., Verkest D., Mignolet J.-Y., Vernalde S. Operating-System Controlled Network on Chip. Proceedings of the Design Automation Conference. – 2004. – P. 256 - 259.

26. Fournier L., Arbetman Y., and LevingerM. Functional verification methodology for microprocessors using the Genesys test-program generator. Proceedings of the 1999 Design, Automation and Test in Europe Conference. – 1999. – P. 434–441.

27. Fine S. and Ziv A. Coverage directed test generation for functional verification using Bayesian networks. Proceedings of the 40th Design Automation Conference. – 2003. – P. 286–291.

28. Synplicity Inc. White Paper. Architecture and Automation: Optimized Design Flow for NEC Electronics' ISSPs. 2003. Synplicity. – 7 p.

29. NEC White paper. Gate Array Solution. Document No. A16841EJ2V0PF00 (2nd edition). March 2004 N CP(K). NEC Electronic Corporation 2004. – 9 p.

30. Karen Parnell, Nick Mehta. Programmable Logic Design Quick Start Handbook. – August 2003. – 210 p.

31. Serge Leef. A Methodology for Virtual Hardware/Software Integration. Mentor Graphics Corporation white paper. – 2003. – 6 p.

32. Passerone C., Lavagno L., Chiodo M., Sangiovanni-Vincentelli A. Fast hardware/software co-simulation for virtual prototyping and trade-off analysis. Proc. Design Automation Conference. – 1997. – P. 389–394.

33. EPSON Electronic Devices white paper. ASIC Gate Array/Standard Cell Embedded Array. – 2004. – 39 p.

34. Benoit Clement, Richard Hersemeule, Etienne Lantreibecq, Bernard Ramanadin, Pierre Coulomb, Francois Pogodalla. Fast Prototyping: a System Design Flow Applied to a Complex System-On-Chip Multiprocessor Design. Proc. Design Automation Conference. – 1999. – P. 420-424.

35. Bulent I. Dervisoglu. A Unified DFT Architecture for Use with IEEE 1149.1 and VSIA/IEEE P1500 Compliant Test Access Controllers. Proceedings of the Design Automation Conference. – 2001. – P. 53-58.

36. Agrawal V.D., Kime C.R., Saluja K.K. A Tutorial on Built-In Self-Test, Part 1: Principles. IEEE Design & Test. – 1993. – Vol. 10, No.1. – P. 73-82.

37. Bardell P.H., McAnney W.H., Savir J. Built-In Test for VLSI: Pseudorandom Techniques, John Wiley & Sons. – 1987. 232 p.

38. Wohl P., Waicukauski J. A., Patel S., Amin M. B. (Synopsys, Incorporated). Efficient Compression and Application of Deterministic Patterns in a Logic BIST Architecture. Proc. of the Design Automation Conference. – 2003. – P. 566-569.

39. Negreiros M., Carro L., Susin A. A. (Universidade Federal do Rio Grande do Sul). Ultimate Low Cost Analog BIST. Proceedings of the Design Automation Conference. – 2003. – P. 570-74.

40. Chowdhury S. and Breuer M. A. Optimum Design of IC Power/Ground Nets Subject to Reliability Constraints, IEEE Trans. of Computer-Aided Design. Vol. 7, No. 7. July 1988. – P. 787-796.

41. Stark D. and Horowitz M. Techniques for Calculating Currents and Voltages in VLSI Power Supply Networks. IEEE Tran. of Computer-Aided Design. Vol. 9, No. 2, February 1999. – P. 126-132.

42. Wu S.-W.(Elan Microelectronics Corporation), Chang Y.-W.(National Taiwan University). Efficient Power/Ground Network Analysis for Power Integrity-Driven Design Methodology Proceedings of the Design Automation Conference. – 2004. – P. 77-180.

43. Wojciech Maly. IC Design in High-Cost Nanometer-Technologies Era. Proceedings of the Design Automation Conference. – 2001. – P. 9-14.

44. Fine S. and Ziv A. Coverage Directed Test Generation for Functional Verification using Bayesian Networks. Proc. Design Automation Conference. – 2003. – P.286-291.

45. Lee D., Zolotov V., Blaauw D. Static Timing Analysis using Backward Signal Propagation. Proc. Design Automation Conference. – 2004. – P. 664-669.

46. Geng Bai, Sudhakar Bobba, Ibrahim N. Hajj. Static Timing Analysis Including Power Supply Noise Effect on Propagation Delay in VLSI Circuits. Proceedings of the Design Automation Conference. – 2001. P. 295-300.

47. Dongkun Shin, Jihong Kim, Seongsoo Lee. Low-Energy Intra-Task Voltage Scheduling Using Static Timing Analysis. Proceedings of the Design Automation Conference. – 2001. – P. 438-443.

48. Schubert T.(Intel Corporation). High Level Formal Verification of Next-Generation Microprocessors. Proceedings of the Design Automation Conference. – 2003. – P. 1-6.

49. Vittal A., Lia H., Brewer F., Marek-Sadowska M. Clock skew optimization for ground bounce control. ICCAD Digest of Technical Papers. – 1996. – P. 395-399.

50. Mustafa Badaroglu, Kris Tiri, Stephane Donnay, Piet Wambaca, Ingrid Verbauwhede, Georges Gielen, Hugo De Man. Clock Tree Optimization in Synchronous CMOS Digital Circuits for Substrate Noise Reduction Using Folding of Supply Current Transients. Proceedings of the Design Automation Conference. – 2002. – P. 399-404.

51. Dunlop A. E.and Kernighan B. W. A procedure for placement of standard-cell VLSI circuits. IEEE Trans. CAD. Jan. 1985. – P. 92-98.

52. Ke Zhong, Shantanu Dutt. Algorithms for Simultaneous Satisfaction of Multiple Constraints and Objective Optimization in a Placement Flow with Application to Congestion Control. Proceedings of the Design Automation Conference. – 2002. – P. 854-859.

53. Morton P., Dai W. Crosstalk. Crosstalk Noise Estimation for use in an Efficient Noise Management Strategy. Technical Report UCSC-CRL-01-09, UCSC. – 2001. – 7 p.

54. Giorgio Casinovi. An Algorithm for Frequency-Domain Noise Analysis in Nonlinear Systems. Proceedings of the Design Automation Conference. – 2002. – P. 514-517.

55. Abramovici M., Breuer M.A. and Friedman A.D. Digital systems testing and testable design. Computer Science Press. – 1998. – 652 p.

56. Mo F., Brayton R. K. (University of California at Berkeley). A Timing-Driven Module-Based Chip Design Flow. Proc. Design Automation Conference. – – 2004. – P. 67-70.

57. Lipsett R. VHDL: Hardware Description and Design. Kluwer Academic Publishers. – 1989. – 488 p.

58. IEEE Standard Verilog HardwareDescription Language. The Institute of Electrical and Electronics Engineers. New York. – 2001. – 791 p.

59. Bergeron Janick. Writing testbenches: functional verification of HDL models.– Boston: Kluwer Academic Publishers. – 2001. – 354 c.

60. Henftling R., Zinn A., Bauer M., Zambaldi M., Ecker W. (Infineon Technologies AG.) Re-Use-Centric Architecture for a Fully Accelerated Testbench Environmen. Proceedings of the Design Automation Conference. – 2003. – P. 372-37.

61. Ur S., Yadin Y. Micro-Architecture Coverage Directed Generation of Test Programs. Proceedings of the Design Automation Conference.–1999.– P. 175-180.

62. Asaf S., Marcus E., Ziv A. Defining Coverage Views to Improve Functional Coverage Analysis. Proc. Design Automation Conference. – 2004. – P.41-44.

63. Schubert K.-D.(IBM Corporation). Improvements in Functional Simulation Addressing Challenges in Large, Distributed Industry Projects. Proceedings of the Design Automation Conference. – 2003. – P.11-14.

64. Jagesh Sanghavi, Albert Wang. Estimation of Speed, Area, and Power of Parameterizable, Soft IP. Proceedings of the Design Automation Conference. – 2001. – P. 31-34.

65. Reyneri L. M. , Cucinotta F., Serra A., Lavagno L. A Hardware/Software Co-design Flow and IP Library Based of Simulink TM. Proceedings of the Design Automation Conference. – 2001. – P. 593-598.

66. Epson white paper. FPGA to ASIC Conversion. Copyright. 2003. Epson Electronics. – 22 p.

67. Rashinkar P., Paterson P. and Singh L. System-On-Chip Verification Methodology and Techniques. Kluwer Academic Publishers. – 2001. – 432 p.

68. Rowen C., Leibson S. (Tensilica, Inc.). Flexible Architectures for Engineering Successful SOCs. Proceedings of the Design Automation Conference. – 2004. – P.692-697.

69. Marick B. The craft of Software Testing, Subsystem Testing Including Object-Based and Object-Oriented Testing. Prentice Hall. – 1985. – 420 p.

70. Milan Saini, Technical Marketing Manager, Xilinx, Inc. White paper. Seamless FPGA Speeds Verification for Xilinx Virtex-4 and Virtex-II Pro FPGAs with Embedded IBM PowerPC Processors. – 5 p.

71. Harris D., Stokes D., Klein R. Executing an RTOS on Simulated Hardware using Co-verification. Proceedings of Embedded Systems Conference. – 2000. – P. 344-349.

72. Dipert B. Silicon Segmentation. Design Feature Magazine. September 2003. – P. 57-66.

73. Frank Ghenassia. Transaction Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems.– Published by Springer.– 2005.– 282 p.

74. David C. Black, Jack Donovan. SystemC from the ground up.– New York, Boston, London, Moscow: Kluwer Academic Publishers.– 2004.– 263 p.

75. Chevallaz C., Mareau N., and Gonier A. Advanced Methods for SoC Concurrent Engineering.– In Proc. of Design, Automation and Test in Europe Conference (DATE'02).– 2002.– P. 59-63.

76. Nicolescu G., Yoo S., and Jerraya A. Mixed-Level Co-simulation for Fine Gradual Refinement of Communication in SoC Design.– In Proc. of Design, Automation and Test in Europe Conference (DATE).– 2001.– P. 754-759.

77. Bhasker J. A SystemC Primer. Cadence Design Systems.– Star Galaxy Publishing.– 2002.– 283 p.

78. IEEE Std 1800-2005. – IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language.–Sponsor: Design Automation Standards Committee of the IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group.– Published by The Institute of Electrical and Electronics Engineers, Inc.– 3 Park Avenue. New York, NY 10016-5997, USA.– 22 November 2005.– 664 p.

79. IEEE Std 1364-2001. (Revision of IEEE Std 1364-1995).– IEEE Standard Verilog Hardware Description Language.– IEEE Computer Society Sponsored by the Design Automation Standards Committee.– Published by The Institute of Electrical and Electronics Engineers, Inc.– 3 Park Avenue, New York, NY 10016-5997, USA.– 28 September 2001.– 791 p.

80. Janick Bergeron, Eduard Cerny, Alan Hunter, Andrew Nightingale. Verification Methodology Manual for SystemVerilog.– New York: Springer.– 2005.– 338 p.

81. IEEE Standard VHDL Language Reference Manual.– New York: IEEE Std 1076-1993 Erschienen August 1994, IEEE, Taschenbuch ISBN: 1559373768, 1994.– 186 p.

82. IEEE Std 1076-2002 (Revision of IEEE Std 1076, 2000 Edition).– 1076 IEEE Standard VHDL Language Reference Manual.– IEEE Computer Society Sponsored by the Design Automation Standards Committee.– Published by The Institute of Electrical and Electronics Engineers, Inc.– 3 Park Avenue. New York, NY 10016-5997, USA.– 17 May 2002.– 309 p.

83. Ashenden, Peter J. The designer’s guide to VHDL.– San Francisco, Calis.: California: Morgan Kaufmann Publishers, Inc, 1996.– 688 p.

84. Charles H. Roth, Jr. Digital Systems Design Using VHDL.– Boston: PWS Publishing Company, 20 Parkl Plaza, Noston, MA 02116-4324 1998.– 470 p.

85. Bhasker J. A VHDL Synthesis Primer.– Allentown: Star Galaxy Publishing, 1998.– 296 p.

86. Pong P. Chu. RTL hardware design using VHDL. Coding for Efficiency, Portability, and Scalability.– A Wiley-Interscience publication.– 2006.– 696 p.

87. Bhasker J. Verilog HDL, Third Edition.-Star Galaxy Publishing.–2005.–568 p.

88. IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164). New York: The Institute of Electrical and Electronics Engineers– (Std_logic_1164).– ISBN 1-55937-299-0.1993.– 24 p.

89. Бибило П.Н.Синтез логических схем с использованием языка VHDL.– М.: Соломон-Р, 2002.– 384 с.

90. Грушвицкий Р. И., Мурсаев А. Х., Угрюмов Е.П. Проектирование систем на микросхемах программируемой логики.– СПб.: БХВ-Петербург, 2002.– 608 с.

91. Семенец В.В, Хаханова И.В., Хаханов В.И. Проектирование цифровых систем с использованием языка VHDL.– Харьков, 2003.– 510 с.

92. Хаханов В.И., Хаханова И.В. VHDL + Verilog = Синтез за минуты.– Харьков: СМИТ.– 2007.– 264 с.

93. Суворова Е. А., Шейнин Ю. Е. Проектирование цифровых систем на VHDL.– СПб.: БХВ-Петербург, 2003.– 576 с.

94. Поляков А.К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры.– М.: Солон-пресс.– 2003.– 365 с.

95. Сергиенко А.М. VHDL для проектирования вычислительных устройств.– Киев: ЧП “Корнейчук”.– 2003.– 320 с.

96. Donald E. Thomas. Philip R. Moorby. The Verilog Haraware Description Language.– New York, Boston, Dordrecht, London, Moscow: Kluwer Academic Publishers.– 2001.– 404 p.

97. IEEE Std 1364-1995. IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language.– The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street, New York, NY 10017-2394, USA.– 1995.– 400 p.

98. IEEE Std 1364-2001. IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language.– The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street, New York, NY 10017-2394, USA.– 2000.– 403 p.

99. Bergeron, Janick. Writing testbenches: functional verification of HDL models.– Boston: Kluwer Academic Publishers, 2001.– 354 p.

100. Bhasker, J. Verilog HDL Synthesis. A Practical Primer.– Allentown: Star Galaxy Publishing.– 1998.– 215 p.

101. Karen Parnell, Nick Mehta. Programmable Logic Design Quick Start Hand Book.– Xilinx.– January, 2002.– 201 p. (http://www.xilinx.com).

102. Samir Palnitkar. Verilog HDL. A guide tp digital design and synthesis.– SunSoft Press.– 1996.– 396 p.

103. Starter's Guide to Verilog 2001 by Michael D.Ciletti.– IEEE International Conference on Acoustics. Speech and Signal Processing.– Atlanta.–1996.– 428 p.

104. Douglas Densmore, Roberto Passerone, Alberto Sangiovanni-Vincentelli. A Platform-Based taxonomy for ESL design, Design & Test of computers, September-October 2006.– P. 359-373.

105. A. Sangiovanni-Vincentelli, G. Martin. Defining Plarform-Based Design and Software Design Metodology for Embedded Systems.– IEEE Design & Test.– Vol.18.– N. 6.– Nov.- Dec. 2001.– P. 23-33.

106. Chang H., Surviving the SoC Revolution: A Guide to Platform-Based Design.– Kluwer Academic Publishers.– 1999.– 235 p.

107. Gajski D.D., Kuhn R.H. Guest Editors Introduction New VLSI Tools.– Computer.– Vol. 16. N. 12.– Dec.1983.– P. 11-14.

108. Davare A. JPEG Encoding on the Intel MXP5800: A Platform-Based Design Case Study.– Proc. 3rd Workshop Embedded Systems for Real-Time Multimedia (ESTIMedia 05).– IEEE CS Press, 2005.– P. 89-94.

109. Steven W.Smith. The scientist and engineer's guide to digital signal processing.– California Technical Publishing.– 1999.– 630 p.

110. David S. Taubman, Michael W.Marcellin. JPEG2000: image compression fundamentals, standards and practice.– Boston/Dordrecht./London: Kluwer Academic Pulishers.– 2002.– 774 p.

3. ДЕДУКТИВНО-ПАРАЛЕЛЬНЕ МОДЕЛЮВАННЯ НЕСПРАВНОСТЕЙ ЦИФРОВИХ СИСТЕМ

Пропонується швидкодіючий дедуктивно-паралельний метод обратного моделювання несправностей, що використовує процедуру суперпозиції розв’язків, орієнтований на обробку цифрових проектів великої розмірності вентильного і реєстрового рівнів опису. Представляються структури даних і алгоритми для реалізації методу в складі автоматичної системи генерації тестів, орієнтованої на побудову алгоритмів пошуку дефектів шляхом аналізу таблиць несправностей.

3.1. Вступ

Актуальність дослідження визначається необхідністю значного підвищення швидкодії засобів моделювання та генерації тестів для структурно- і функціонально-складних цифрових систем, імплементованих у кристали програмованої логіки. Існуючі автоматичні системи тестування відомих фірм: Cadence, Mentor Graphics, Synopsys, Logic Vision здатні обробляти пристрої на кристалах розмірністю порядку 100 тис. вентилів за кілька годин (часові витрати відносяться до робочих станцій з частотою і оперативною пам'яттю в 0,5 ГГц і 0,5 Гбайт відповідно). Тим не менш, дані засоби синтезу тестів і моделювання несправностей можуть бути неприйнятними за часом обробки чипів, які налічують кілька мільйонів вентилів. Тому ринку електронних технологій необхідні нові підходи, що дозволяють на порядок підвищити швидкодію аналізу цифрових пристроїв на стадії їх проектування з метою побудови тестів перевірки несправностей.

Один з них, що вирішує проблему створення швидкодіючого методу моделювання одиночних константних дефектів для оцінки якості тестів перевірки несправностей для проектованих цифрових виробів на основі програмованої логіки, пропонується в даній роботі. Об'єкт тестування - цифрова система, що імплементується в кристали програмованої логіки, специфікація якої представлена на мові VHDL. Мета - розробка швидкодіючого методу моделювання одиничних константних несправностей для оцінки якості синтезованих тестів цифрових систем, імплементованих в ПЛІС, що містять міліони вентилів. Завдання: 1. Створення узагальненої моделі дедуктивно-паралельного аналізу цифрової схеми. 2. Розробка внутрішньої моделі і структур даних для аналізу цифрового пристрою. 3. Розробка алгоритмів структурно-функціонального аналізу цифрових систем з метою пошуку множини збіжних розгалужень (ЗР) і визначення схемних деревовидних структур (ДС). 4. Розробка методу моделювання несправностей, заснованого на роздільній обробці ЗР і ДС. 5. Алгоритмічна реалізація моделювання на основі реконфігурування моделі пристрою в процесі аналізу несправностей та застосування процедур зворотного простежування з метою істотного

зменшення часу для оцінки якості тестів. Основу ОДП-методу (Backtraced Deductive-Parallel) - зворотне дедуктивно-паралельне моделювання несправностей - складають: методи підвищення швидкодії аналізу несправностей [1-3], дедуктивна модель транспортування несправностей [4,5], паралельний метод обробки списків дефектів функціонального елемента [4 , 6] і алгоритм зворотного простежування примітивів [7] при обробці цифрового пристрою.

3.2. Узагальнена модель дедуктивно-паралельного аналізу дефектів

Модель дедуктивно-паралельного синхронного аналізу несправностей дискретного об'єкта дозволяє за одну ітерацію обробки схеми обчислювати всі дефекти, що перевіряються на двійковому тест-векторі. Така модель основана на вирішенні наступного рівняння [3,4]:

$$L = T \oplus F, \tag{3.1}$$

де $F = (F_{m+1}, F_{m+2}, ..., F_i, ...F_n)(i = \overline{m+1, n})$ – сукупність функцій справної поведінки пристрою; m – число його входів; $Y_i = F_i(X_{i1}, ..., X_{ij}, ..., X_{in_i})$– n_i-входовий i-й елемент схеми, що реалізує F_i для визначення стану лінії (виходу) Y_i на тест-векторі T_t; тут X_{ij} – j-й вхід i-го елементу; тест $T = (T_1, T_2, ..., T_t, ..., T_k)$ – упорядкована сукупність двійкових векторів, довизначена у процесі справного моделювання на множині вхідних, внутрішніх та вихідних ліній, об'єдиннана у матрицю

$$T = [T_{ti}] = \begin{bmatrix} T_{11}, T_{12}, ..., T_{1i}, ..., T_{1n} \\ ..... \\ T_{t1}, T_{t2}, ..., T_{ti}, ..., T_{tn} \\ ..... \\ T_{k1}, T_{k2}, ..., T_{ki}, ..., T_{kn} \end{bmatrix}, \tag{3.2}$$

невхідна координата якої визначається моделюванням функції $T_{ti} = Y_i = F_i(X_{i1}, ..., X_{ij}, ..., X_{in_i})$ на тест-векторі T_t; $L = (L_1, L_2, ..., L_t, ..., L_k)$ – множина дедуктивних схем або моделей, що визначаються виразом (3.1), де $L_t = (L_{t1}, L_{t2}, ..., L_{ti}, ..., L_{tn})$;

$$L_{ti} = T_t \oplus F_i \tag{3.3}$$

– дедуктивна функція (ДФ) паралельного моделювання несправностей на тест-векторі T_t, яка відповідає справному елементу F_i, що дає можливість обчислювати список вхідних несправностей, транспортуємих на вихід елемента F_i [8].

Поняття синхроності введеної моделі (3.1) визначається умовою: $\Delta t = (t_{j+1} - t_j) >> \tau >> \tau_i$, коли інтервал часу меіж зміною вхідних наборів $(t_{j+1} - t_j)$, що подаються на схему, набагато більше максимальної затримки

схеми τ та элементу τ_i. Це дозволяє виключити час як несуттєвий параметр [8], що використовується у технологіях моделювання та синтезу тестів.

У загальному випадку, коли функція пристрою подана таблицею істинності, застосування формули (3.1) дозволяє отримати для заданого тест-вектора таблицю транспортування несправностей, за якою можна записати ДФ моделювання дефектів. Приклади отримання таких функцій зображені у наступному вигляді (перший доданок - тест-вектор, другий і результат - таблиці істинності і транспортування дефектів):

X_1	X_2	Y_1
0	1	0

$\oplus$

X_1	X_2	Y_1
0	0	0
0	1	0
1	0	1
1	1	1

$=$

X_1	X_2	L_1
0	1	0
0	0	0
1	1	1
1	0	1

$L_1 = X_1X_2 \vee X_1\overline{X}_2;$

X_1	X_2	Y_2
1	1	1

$\oplus$

X_1	X_2	Y_2
0	0	0
0	1	0
1	0	0
1	1	1

$=$

X_1	X_2	L_2
1	1	1
1	0	1
0	1	1
0	0	0

$L_2 = X_1X_2 \vee X_1\overline{X}_2 \vee \overline{X}_1X_2.$

Тут дедуктивні функції L_1, L_2 записані у вигляді диз'юнктивної нормальної форми за конституентами одиниці таблиць транспортування дефектів.

З урахуванням розбиття тесту на складові вектори рівняння (3.1) отримання ДФ для $T_t \in T$ набуває натупного вигляду: $L_t = T_t \oplus F$. Якщо функціональний опис цифрового пристрою подано компонентами (примітивами), що формують стани всіх ліній схеми, то як формули перетворення справної моделі примітиву F_i на тест-векторі T_t в дедуктивну функцію L_{ti} виступає наступний выраз:

$$L_{ti} = T_t \oplus F_i = f_{ti}[(X_{i1} \oplus T_{t1}),(X_{i2} \oplus T_{t2}),\ldots, (X_{ij} \oplus T_{tj}),\ldots,(X_{in_i} \oplus T_{tn_i})] \oplus T_{ti}, \tag{3.4}$$

який є основою дедуктивного анализу цифрових проектів [3, 6].

З метою опису алгоритму аналізу дефектів на основі виразу (3.4) вводяться наступні визначення.

Визначення 3.1. Вектор перевіряючих на тест-векторі $T_t \in T$ нулевих (одиничних) дефектів $S^0 = (S_1^0,\ldots,S_i^0,\ldots,S_n^0)$ $(S^1 = (S_1^1,\ldots,S_i^1,\ldots,S_n^1))$ є упорядкована у відповідності з нумерацією ліній схеми множина одиночних константных несправностей, де одиничне значенне координати вектору

$S_i^0 = 1; (S_i^1 = 1)$ свідчить про перевірку дефекту $\equiv 0 (\equiv 1)$ лінії з номером і на поточному вхідному наборі $T_t \in T$ і неперевірці цієї несправності у протилежному випадку, коли $S_i^0 = 0; (S_i^1 = 0)$.

Визначення 3.2. Вектор перевірених на тесті Т нулевих (одиничних) дефектів $D^0 = (D_1^0, ..., D_i^0, ..., D_n^0)$ $(D^1 = (D_1^1, ..., D_i^1, ..., D_n^1))$ є упорядкована у відповідності з нумерацією ліній схеми множина одиночних константних несправностей, де одиничне значення координати вектору $D_i^0 = 1; (D_i^1 = 1)$ свідчать про перевірку дефекту $\equiv 0 (\equiv 1)$ лінії з номером і на хоча б одному наборі $T_t \in T$ та неперевірці цієї несправності у протилежному випадку, коли $D_i^0 = 0; (D_i^1 = 0)$.

Визначення 3.3. Матриця перевіряємих на тест-векторі $T_t \in T$ дефектів $M = [M_{ij}]$, розмірністю $n \times n$, є форма завдання списку одиночних константних несправностей, де її координати первісно ініціюються відповідно до виразу

$$[M_{ij}]\Big|_{(i,j=\overline{1,n})} = \begin{cases} 0 \leftarrow (i \neq j); \\ 1 \leftarrow (i = j). \end{cases} \tag{3.5}$$

У процесі моделювання тест-вектору нульові координати матриці можуть довизначатися одиницями $M_{ij} = 1$, що відповідає перевірці несправностей, інверсних до стану координат тест-вектору $T_{ti} \in T_t$.

Визначення 3.4. Лінія називається невхідною $\overline{X}$, якщо вона є вихідною Y або внутрішньою Z, тобто не відноситься до зовнішніх входів X схеми. Невхідна лінія є вихідною Y, якщо вона з'єднана з вихідним контактом схеми.

З урахуванням введених визначень практична реалізація виразу (3.4) оформляється у наступний алгоритм дедуктивно-паралельного аналізу дефектів.

1. Визначення начального значення індексу для оброблюваного тест-вектору t=0.

Ініціалізація векторів перевірених дефектів:

$$\forall i (D_i^0 = 0; D_i^1 = 0).$$

2. Визначення номеру чергового вхідного набору t=t+1 для $T_t \in T$. Якщо вхідних наборів немає (t > k) – кінец моделювання.

3. Справне моделюровання всіх примітивів $F_i(i=\overline{1,n})$ схеми на вхідному наборі $T_t^X \in T_t$ з метою довизначення невхідних координат вектору $T_t^{\overline{X}} \in T_t$:

$$T_t^{\overline{X}} = f(T_t^X, F). \tag{3.6}$$

Ідентичність вектору справного моделювання ліній у двох сусідніх ітераціях $T_t^r = T_t^{r-1}$ є умовою переходу до наступного пункту.

Примітка. Для моделювання послідовносних схем та організації событійності використовується аналіз пари сусідніх векторів (T_{t-1}, T_t). Примітив $F_i(i=\overline{1,n})$ моделюється, якщо на лініях виконується умова $[T_{t-1}^X(F_i) \neq T_t^X(F_i)]$ – наявність змін на входах розглядуваного элементу.

4. Ініціалізація матриці перевіряємих на тест-векторі дефектів $M=[M_{ij}]$ у відповідності до выразу (3.5). Ініціалізація векторів переверіяємих дефектів $\forall i(S_i^0 = 0;\ S_i^1 = 0)$. Реконфігурування всіх примітивів $F_i(i=\overline{1,n})$ на основі застосування формули (3.4) для поточного вектора справного стану $T_t \in T$ з метою отримання дедуктивної схеми $L_t \leftarrow \forall i(L_{ti} = T_t \oplus F_i)$.

5. Параллельное моделирование неисправностей с помощью полученных дедуктивных функций $L_{ti} \in L_t$ путем выполнения регистровых операций над строками матрицы проверяемых дефектов M в целях доопределения координат, соответствующих невходным линиям схемы.

6. Формування векторів перевіряємих несправностей шляхом застосування формул:

$$S^0 = (\bigvee_{\forall i \in Y} M_i) \wedge T_t;\ S^1 = (\bigvee_{\forall i \in Y} M_i) \wedge \overline{T}_t \tag{3.7}$$

до всіх рядків матриці, що відповідають вихідним спостерігаємим лініям схеми.

7. За ідентичністю списків несправностей у двох сусідніх ітераціях $(S^0 \vee S^1)^{r-1} = (S^0 \vee S^1)^r$ (r – індекс ітерації) визначається якість тест-вектору $T_t \in T$ за формулою

$$Q(T_t) = \frac{1}{2n}[\sum_{i=1}^{n}(S_i^0 + S_i^1)] \tag{3.8}$$

та здійснюється перехід до наступного пункту, інакше, якщо спостерігається зникнення перевіряємих несправностей в ітерації r за порівнянням з r-1:

$$\exists i[(S_i^0 \vee S_i^1 = 1)^{r-1} \,\&\, (S_i^0 \vee S_i^1 = 0)^r],$$

виконується виключення таких дефектів з процесу моделювання за правилом

$$(S_i^0 = S_i^1 = 0) \leftarrow \forall i[(S_i^0 \vee S_i^1 = 1)^{r-1} \,\&\, (S_i^0 \vee S_i^1 = 0)^r]. \qquad (3.9)$$

Перехід до п. 5.

8. Формування векторів перевірених несправностей у відповідності до виразу

$$D^0 = D^0 \vee S^0, \quad D^1 = D^1 \vee S^1 \qquad (3.10)$$

та обчислення якості тесту за формулою

$$Q(T) = \frac{1}{2n}[\sum_{i=1}^{n}(D_i^0 + D_i^1)]. \qquad (3.11)$$

Перехід до п. 2.

Запропонована алгоритмічна реалізація орієнтована як на табличний опис примітивів довільної складності RTL-рівня, так і на вентильному зображенні цифрових систем. Швидкодія алгоритму практично інваріантна компілятивним та інтерпретативним моделям цифрових пристроїв, проте чисто інтерпретативна реалізація є більш технологічною з позиції програмування.

Приклад 3.1. Отримати дедуктивні функції паралельного моделювання несправностей на вичерпному тесті для базису логічних елементів And, Or, Not. З урахуванням виразу (3.4) виконуються наступні очевидні перетворення для функції And:

$$L[T = (00,01,10,11), F = (X_1 \wedge X_2)] =$$
$$= L\{(\bar{x}_1\bar{x}_2 \vee \bar{x}_1x_2 \vee x_1\bar{x}_2 \vee x_1x_2) \wedge [(X_1 \oplus T_{t1} \wedge X_2 \oplus T_{t2}) \oplus T_{t3})]\} =$$
$$= (\bar{x}_1\bar{x}_2)\{[(X_1 \oplus 0) \wedge (X_2 \oplus 0)] \oplus 0\} \vee (\bar{x}_1x_2)\{[(X_1 \oplus 0) \wedge (X_2 \oplus 1)] \oplus 0\} \vee$$
$$\vee (x_1\bar{x}_2)\{[(X_1 \oplus 1) \wedge (X_2 \oplus 0)] \oplus 0\} \vee (x_1x_2)\{[(X_1 \oplus 1) \wedge (X_2 \oplus 1)] \oplus 1\} =$$
$$= (\bar{x}_1\bar{x}_2)(X_1 \wedge X_2) \vee (\bar{x}_1x_2)(X_1 \wedge \bar{X}_2) \vee (x_1\bar{x}_2)(\bar{X}_1 \wedge X_2) \vee (x_1x_2)(X_1 \vee X_2).$$

Аналогічно виконуються обчислення для функції Or:

$$L[T=(00,01,10,11),F=(X_1 \vee X_2)]=$$
$$=L\{(\bar{x}_1\bar{x}_2 \vee \bar{x}_1x_2 \vee x_1\bar{x}_2 \vee x_1x_2) \wedge [(X_1 \oplus T_{t1} \vee X_2 \oplus T_{t2}) \oplus T_{t3})]\}=$$
$$=(\bar{x}_1\bar{x}_2)\{[(X_1 \oplus 0) \vee (X_2 \oplus 0)] \oplus 0\} \vee (\bar{x}_1x_2)\{[(X_1 \oplus 0) \vee (X_2 \oplus 1)] \oplus 1\} \vee$$
$$\vee (x_1\bar{x}_2)\{[(X_1 \oplus 1) \vee (X_2 \oplus 0)] \oplus 1\} \vee (x_1x_2)\{[(X_1 \oplus 1) \vee (X_2 \oplus 1)] \oplus 1\}=$$
$$=(\bar{x}_1\bar{x}_2)(X_1 \vee X_2) \vee (\bar{x}_1x_2)(\bar{X}_1 \wedge X_2) \vee (x_1\bar{x}_2)(X_1 \wedge \bar{X}_2) \vee (x_1x_2)(X_1 \wedge X_2).$$

Тут $T_t=(T_{t1},T_{t2},T_{t3}),(t=\overline{1,4})$ – тест-вектор, що має 3 координати, де остання з них визначає стан виходу двовходового элементу And (Or). У наступному перетворенні $T_t=(T_{t1},T_{t2}),(t=\overline{1,2})$ – тест-вектор, що має 2 координати, де друга – стан виходу інвертора:

$$L[T=(0,1),F=\bar{X}_1]=L\{(\bar{x}_1 \vee x_1)[(\overline{X_1 \oplus T_{t1}}) \oplus T_{t2}]\}=$$
$$=\bar{x}_1[(\overline{X_1 \oplus 0}) \oplus 1] \vee x_1[(\overline{X_1 \oplus 1}) \oplus 0]=\bar{x}_1\bar{\bar{X}}_1 \vee x_1\bar{\bar{X}}_1=\bar{x}_1X_1 \vee x_1X_1.$$

Останній вираз ілюструє инвариантність інверсії до вхідного набору для транспортування дефектів. Вона трансформується у повторювач. Тому дана функція не фігурує на виходах дедуктивних елементів. Спільна апаратурна реалізація ДФ для залишившихся двохвходових елементів And, Or на вичерпному тесті подана універсальним функціональним примітивом (рис. 3.1) дедуктивно-паралельного аналізу несправностей.

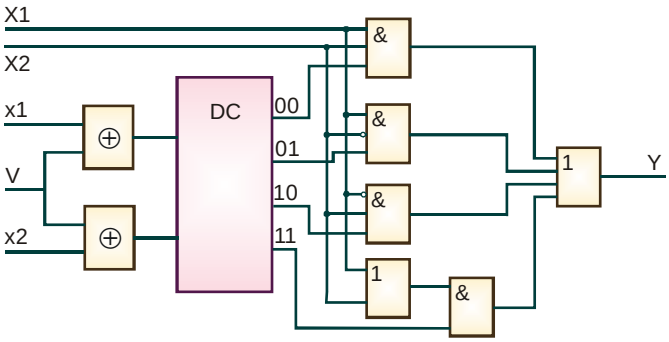


Рис. 3.1. Симулятор несправних примітивів

В симуляторі подані булеві (x1, x2) та регистрові (X1, X2) для кодування несправностей входи, змінна вибору типу справної функції (AND, OR), вихідна регістрова змінна Y. Стани двійкових входів x1, x2 і змінна вибору елемента визначають одну з чотирьох дедуктивних функцій для отримання вектора Y несправнестей, що перевіряються. Для ілюстрації паралельного моделювання вхідних 4-розрядних векторів несправностей з метою отримання на виході Y множини дефектів, що перевіряються, для логічних елементів 2And, 2Or використовується наступна таблиця:

And / Or	x_1x_2	X_1	X_2	Y
0	00	0111	1011	0011
1	00	0111	1011	1111
0	11	0101	0110	0111
1	11	1101	0111	0101

Застосування такого симулятора дає можливість трансформувати вентильну модель F справної поведінки схеми в дедуктивну L, яка інваріантна у сенсі універсальності тестовим наборам і не передбачає в процесі моделювання використовувати модель F. Тому симулятор, як апаратурна модель ДФ, орієнтований на створення вбудованих засобів дедуктівно-паралельного моделювання, що підвищують швидкодію аналізу в 10 - 1000 разів у порівнянні з програмною реалізацією. Але при цьому співвідношення обсягів вентильних (після синтезу) моделей справного моделювання та аналізу несправностей становить 1:10. Підхід апаратурного аналізу несправностей спрямований на розширення функціональних можливостей вбудованих засобів справного моделювання (HESTM - Hardware Embedded Simulator) фірми Aldec [9].

3.3. Внутрішня модель анализуа цифрового пристрою

Відповідно до п. 3 алгоритму дедуктивно-паралельного аналізу необхідно мати внутрішню модель справної поведінки схеми у вигляді відповідних структур даних, що розміщуються в оперативній пам'яті і орієнтованих на обробку елемента цифрового пристрою (рис. 3.2).

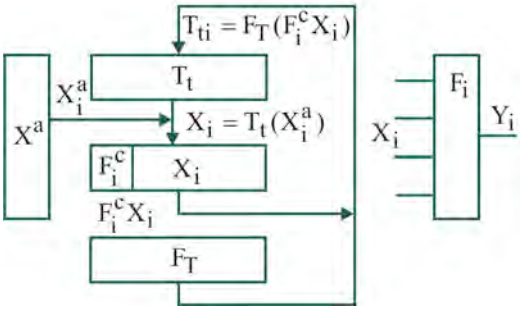


Рис. 3.2. Структурна модель аналізу елементу

Перш за все, йдеться про процедуру визначення стану координати тест-вектора $T_{ti} \in T$:

$$T_{ti} = F_i(X_{i1},..., X_{ij},..., X_{in_i}), \qquad (3.12)$$

що відповідає виходу Y_i логічного элементу F_i (F_i^c – його двійковий код або ідентифікатор) за його вхідним значенням, поданим вектором $X_i = (X_{i1},..., X_{ij},..., X_{in_i})$, за умови, що F_T є таблиця істиності сукупності булевих функцій для опису елементів схеми, вхідний набор для якої

формується конкатенацією слов $(F_i^c * X_i)$. З урахуванням даного факту вираз (3.12) трансформується до вигляду

$$T_{ti} = F_T(F_i^c X_i). \tag{3.13}$$

Інакше, щоби визначити стан координати $T_{ti} \in T$, необхідно сформувати двійковий вектор значень вхідних змінних $X_i = T_t(X_i^a); X_{ij} = T_t(X_{ij}^a)$ елемента F_i, використовуючи їх адреси $X_i^a = (X_{i1}^a,..., X_{ij}^a,..., X_{in_i}^a)$ для витягання станів з вектору T_t (див. рис.3.2). Потім следує виконати конкатенацію отриманого вектору X_i з двійковим кодом типу функції F_i^c з метою отримання вхідного слова $(F_i^c * X_i)$ для узагальненої таблиці істиності F_T, де у стовпці Y_i, який відповідає значенню функції F_i, знаходиться стан координати $T_{ti} \in T$.

Модель дедуктивно-паралельного аналізу несправностей, крім структури, що бере участь в справному моделюванні, містить два додаткових модулі M і L, як показано на рис. 3.3.

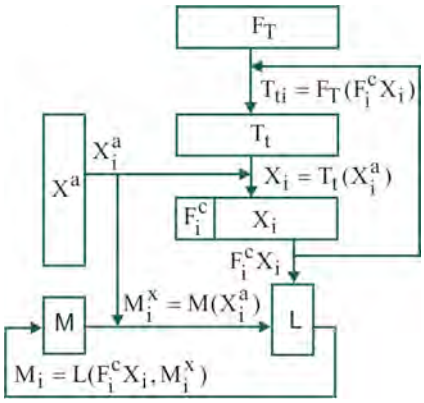


Рис. 3.3. Модель дедуктивно-паралельного аналізу

Аналітичний вираз для обчислення векторів перевіряючих несправностей, об'єднаних в матрицю M, за допомогою набору дедуктивних функцій L, отриманих з F за (3.4), має вигляд:

$$M_i = L(F_i^c X_i, M_i^x). \tag{3.14}$$

Тут моделюється підмножина векторів несправностей $M_i^x = M(X_i^a) = \{M(X_{i1}^a),..., M(X_{ij}^a),..., M(X_{in_i}^a)\}$, що відповідають адресам

вхідних змінних $X_i^a \in X^a$ і-го элемента на дедуктивній функції $L(F_i^c X_i) \in L$, що ідентифікується конкатенуємою адресою $(F_i^c X_i)$ та реалізується компілятивно для паралельного виконання регістрових операцій над векторами з множини M_i^x.

Таким чином, для визначення стану вихідного вектора несправностей M_i необхідно знайти адресу дедуктивної функції L_i, використовуючи отриману для справного моделювання конкатенацію двійкових слов $F_i^c X_i$. Вхідні змінні для елемента L_i є регістровими, теоретична розмірність яких дорівнює числу ліній в цифровому пристрої. Далі здійснюється послідовність виконання (n-1) регістрових операцій над вхідними векторами M_i^x. Результат у вигляді рядка M_i заноситься до матриці М. Вхідна змінна X_{ij} може мати знак інверсії. Тоді перед виконанням бінарної операції здійснюється інверсія вмісту регістрової змінної:

$$M(\overline{X}_{ij}) = \overline{M}(X_{ij}^a).$$

Як приклад розглянемо узагальнену таблицю істинності для визначення стану справної поведінки і вибору двійкової адреси дедуктивного функціонального елементу аналізу несправностей:

I	F_i	x_1x_2	Y_i	L_{ti}
∧	00	00	0	00
	00	01	0	01
	00	10	0	10
	00	11	1	11
∨	01	00	0	11
	01	01	1	10
	01	10	1	01
	01	11	1	00

I	F_i	x_1x_2	Y_i	L_{ti}
$\overline{\wedge}$	10	00	1	00
	10	01	1	01
	10	10	1	10
	10	11	0	11
$\overline{\vee}$	11	00	1	11
	11	01	0	10
	11	10	0	01
	11	11	0	00

Тут стовпчик F_i – код справної функції, I – умовне позначення, (x_1, x_2) – двійкові вхідні набори таблиці істинності кожної з чотирьох функцій, Y_i – стовпчик стану виходу справної функції, L_{ti} – код адреси однієї з чотирьох компілятивних дедуктивних функцій, визначених у відповідності з (3.3):

$$L = \begin{cases} 00 \to X_1 \wedge X_2; \\ 01 \to X_1 \wedge \overline{X_2}; \\ 10 \to \overline{X_1} \wedge X_2; \\ 11 \to X_1 \vee X_2. \end{cases}$$

Обчислювальна складність обробки цифрової схеми, що складається з n двовходових вентилів, визначається виразом

$$Q = [(2K + A) + A + (2n\tau)/W] = [2(K + A) + (2n\tau)/W] \times n,$$

де K - час, що витрачається на конкатенацію бітів для отримання адреси стану виходу примітиву; A - час вибірки вмісту комірки (біта) за її адресою; τ - час виконання регістрової операції (AND, OR, NOT); W - розрядність регістра.

Якщо врахувати, що перший доданок 2(K+A) є незрівнянно малим порівняно з другим, то обчислювальна складність обробки цифрового пристрою пропорційна квадрату числа вентилів:

$$Q = (2n^2\tau)/W. \qquad (3.15)$$

3.4. Дедуктивный метод структурного анализа графа схемы

Розглянемо деяку поточну вершину $V_j \in V$ в орієнтованому графі G ліній схеми. По відношенню до суміжних вершин-попередників, які є прообразами $f^{-1}(V_j)$, згадана вершина $V_j \in V$ є образом (рис. 3.4), де $f^{-1}(...)$ – оператор взяття прообразу.

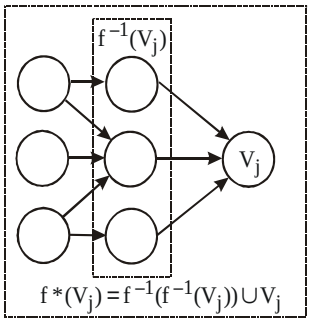


Рис. 3.4. Суміжність як відповідність прообразів образу

Для ілюстрації можливості застосування дедуктивного методу до структурного аналізу виконаємо перетворення графа в Or-модель за правилами: 1) вершини, що містять тільки вихідні дуги, стають зовнішніми входами схеми; 2) решта вершини зображені Or-елементами, а точніше їх виходами; 3) дуги графа відповідають функціональним зв'язкам між суміжними вершинами.

Таким чином, вершина графа перетворюється в лінію Or-структури, де число входів у кожному елементі дорівнює кількості суміжних вершин-

попередників. Наприклад, граф, що має 9 вершин (рис. 3.5), перетворюється у схему з 6 елементів з трьома зовнішніми входами.

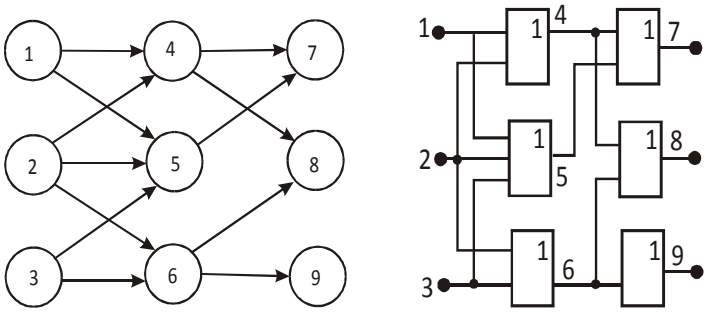


Рис. 3.5. Граф зі збіжними розгалуженнями

Визначення 3.5. Кожна вершина безконтурного графа має єдине об'єднання образу і суперпозиції прообразів:

$$f^*(V_j) = f^{-1}(f^{-1}(...(f^{-1}(V_j)))) \cup V_j, \qquad (3.16)$$

яке називається екстраобразом. Инакше, екстраобраз є множина всех попередників для V_j, включаючи і вершину V_j.

Твердження 3.1. Всі вершини (лінії), що входять до контуру, мають однакові екстраобрази.

Це є наслідком таких фактів: 1) всі лінії, що належать контуру, досяжні за відношенням один до одного; 2) кожна лінія, що входить до контуру, має як попередника будь-яку вершину, від якої існує шлях до однієї з контурних ліній. Звідси випливає, що всі вершини $V_j \in V$, що належать контуру, мають однакові екстраобрази [10].

Збіжному розгалуженню можуть передувати лінії, що не є такими, але проявляються на вершині-сходженні як ЗР Прикладом таких ліній є вершини, що складають шлях без розгалужень, який закінчується на лінії ЗР.

Твердження 3.2. Для того щоб усунути всіх попередників збіжного розгалуження, що входять до множини V_j кандидатів у ЗР для вершины V_j, але не є ЗР, необхідно і достатньо відрахувати з множини V_j об’єднання перетинів всіх пар сполучень $C^2_{n_j}$ екстраобразів $f^*(f^{-1}(V_j))$ для безпосередніх попередників вершини V_j, де n_j – число попередників.

Процедура дедуктивного аналізу. Якщо граф - орієнтований, без контурів глобальних зворотних зв'язків, то стратегія пошуку збіжних розгалужень зводиться до одноразового проходу всіх його вершин на основі процедури дедуктивної обробки $V_j \in V$, що включає операції:

$$1)\ V^j = \bigcup_{C^2_{n_j}} [f^*(f_p^{-1}(V_j)) \bigcap_{\substack{p=i \\ q=i+1,n_j \\ i=1,n_j-1}} f^*(f_q^{-1}(V_j))];$$

$$2)\ V^j = V^j \setminus \bigcup_{C^2_{m_j}} [f^*(V_p^j) \bigcap_{\substack{p=i \\ q=i+1,m_j \\ i=1,m_j-1}} f^*(V_q^j)]; \qquad (3.17)$$

$$3)\ V_j = [\bigcup_{i=1}^{n_j} f^*(f_i^{-1}(V_j)) \cup V_j] \setminus V^j,$$

де $f^*(f_i^{-1}(V_j))$ – екстраобрази прообразів $f^{-1}(V_j)$, число яких для вершини V_j дорівнює n_j.

Перше рівняння призначено для визначення збіжних розгалужень шляхом виконання операції об'єднання перетинань усіх пар екстраобразів для прообразів аналізованої вершини, число яких дорівнює кількості сполучень - $C^2_{n_j}$; друге - для вилучення зі списку ЗР вершин, які не є такими відповідно до твердження 3.2. Тут $f^*(V_i^j)$ - екстраобраз вершини, включеної до списку кандидатів у ЗР при аналізі $V_j \in V$, число яких дорівнює m_j. Третє рівняння призначене для формування екстраобраза оброблюваної вершини.

Приклад 3.2. Для графа (див. рис. 3.5) послідовна обробка всіх вершин за правилами (3.17) подана результатами обчислень:

$$V^1 = \varnothing;\ V_1 = \{1\};\ V^2 = \varnothing; V_2 = \{2\};\ V^3 = \varnothing; V_3 = \{3\};$$

$$V^4 = (V_1 \cap V_2) = \varnothing; V_4 = \{1,2,4\};$$

$$V^5 = (V_1 \cap V_2) \cup (V_1 \cap V_3) \cup (V_2 \cap V_3) = \varnothing; V_5 = \{1,2,3,5\};$$

$$V^6 = (V_2 \cap V_3) = \varnothing; V_6 = \{2,3,6\};$$

$$V^7 = (V_4 \cap V_5) = \{1,2\}; V_7 = \{1,2,3,4,5,7\} \setminus \{1,2\} = \{3,4,5,7\};$$

$$V^8 = (V_4 \cap V_6) = \{2\}; V_8 = \{1,2,3,4,6,8\} \setminus \{2\} = \{1,3,4,6,8\};$$

$$V^9 = \varnothing; V_9 = \{2,3,6,9\}.$$

Об’єднання всіх підмножин

$$V^{RC} = \bigcup_{j=1}^{n} V^j$$

дає повну множину збіжних розгалужень V^{RC} графа цифрової структури. Для даного прикладу

$$V^{RC} = V^1 \cup V^2 \cup V^3 \cup V^4 \cup V^5 \cup V^6 \cup V^7 \cup V^8 \cup V^9 =$$
$$= \varnothing \cup \varnothing \cup \varnothing \cup \varnothing \cup \varnothing \cup \varnothing \cup \{1,2\} \cup \{2\} \cup \varnothing = 1,2\}.$$

Враховуючи той факт, що лінії, вже визначені як збіжні розгалуження, є надлишковими для подальшого аналізу структури, їх можна вилучати зі списків при обробці кожної вершини. Однак множину ЗР, яка визначається в процесі обробки, необхідно накопичувати у складі окремої підмножини V^{RC}. З урахуванням сказаного виникає необхідність модифікації системи рівнянь (3.17). В цьому випадку формули для процедури дедуктивного аналізу вершин графа цифрової схеми набувають вигляду:

$$1)\ V^j = \bigcup_{C^2_{n_j}} [f^*(f_p^{-1}(V_j)) \bigcap_{\substack{p=\overline{i} \\ q=\overline{i+1,n_j}}\ i=\overline{1,n_j-1}} f^*(f_q^{-1}(V_j))];$$

$$2)\ V^j = V^j \setminus \bigcup_{C^2_{m_j}} [f^*(V_p^j) \bigcap_{\substack{p=\overline{i} \\ q=\overline{i+1,m_j}}\ i=\overline{1,m_j-1}} f^*(V_q^j)]; \qquad (3.18)$$

$$3)\ V^{RC} = V^{RC} \bigcup V^j;$$

$$4)\ V_j = [\bigcup_{i=1}^{n_j} f^*(f_i^{-1}(V_j)) \cup V_j] \setminus V^{RC}.$$

Відмінність процедури (3.18) від (3.17) полягає в накопиченні множини ЗР і її відніманні зі списків попередників для кожної вершини. Вираз (3.18) слід використовувати для зменшення потужності прообразів оброблюваної вершини, яке буде тим суттєвішими, ніж вище коефіцієнти розгалуження у ліній попередників, вже віднесених до списку V^{RC}.

3.5. Метод паралельного дедуктивного моделювання графових структур цифрових систем

Як ідентифікатор V_j постає вже вектор $V_j \in V$, що має в j-й позиції значення 1. Тоді сукупність векторів, які позначають всю множину вершин в графі, подана одиничною матрицею:

$$V = \left\| V_{ij} \right\|_{(i,j=\overline{1,n})};\ V_{ij} = \begin{cases} 1 \leftarrow i = j; \\ 0 \leftarrow i \neq j. \end{cases}$$

Враховуючи ізоморфізм булевої алгебри та алгебри множин, виконуємо перетворення виразу (3.17) у формули алгебри логіки:

$$1)\ V^j = \bigvee_{C^2_{n_j}} [f^*(f_p^{-1}(V_j)) \bigwedge_{i=\overline{1,n_j-1}}^{\substack{p=i \\ q=\overline{i+1,n_j}}} f^*(f_q^{-1}(V_j))];$$

$$2)\ V^j = V^j \wedge \neg \{ \bigvee_{C^2_{m_j}} [f^*(V_p^j) \bigwedge_{i=\overline{1,m_j-1}}^{\substack{p=i \\ q=\overline{i+1,m_j}}} f^*(V_q^j)]\}; \qquad (3.19)$$

$$3)\ V_j = [\bigvee_{i=1}^{n_j} f^*(f_i^{-1}(V_j)) \vee V_j] \wedge \overline{V}^j,$$

де операції кон’юнкції та диз’юнкції виконуються над відповідними векторами матриці V.

Аналогічні перетворення над (3.18) дають результат:

$$1)\ V^j = \bigvee_{C^2_{n_j}} [f^*(f_p^{-1}(V_j)) \bigwedge_{i=\overline{1,n_j-1}}^{\substack{p=i \\ q=\overline{i+1,n_j}}} f^*(f_q^{-1}(V_j))];$$

$$2)\ V^j = V^j \wedge \neg \{ \bigvee_{C^2_{m_j}} [f^*(V_p^j) \bigwedge_{i=\overline{1,m_j-1}}^{\substack{p=i \\ q=\overline{i+1,m_j}}} f^*(V_q^j)]\}; \qquad (3.20)$$

$$3)\ V^{RC} = V^{RC} \vee V^j;$$

$$4)\ V_j = [\bigvee_{i=1}^{n_j} f^*(f_i^{-1}(V_j)) \vee V_j] \wedge \overline{V}^{RC}.$$

Переваги використання формул (3.19) і (3.20) в програмному препроцесорі моделювання полягають в паралельності виконання операцій над вершинами графа. Це дозволяє в десятки і сотні разів підвищити швидкодію процедури аналізу для вершин, що мають велику кількість попередників, за рахунок збільшення пам'яті, необхідної для зберігання матриці ідентифікаторів вершин, і вилучення операцій над елементами множини. При цьому час, що витрачається на пошук ЗР при роботі препроцесора, практично не відрізняється від часу моделювання дефектів на вхідному наборі дедуктивним або паралельним методом.

Приклад 3.3. Ілюструє застосування формул (3.20) для знаходження множини збіжних розгалужень графа (див. рис. 3.5).

1. Початкове завдання матриці V^0 має наступний вигляд:

V^0	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	1

Вектор збіжних розгалужень ініціалізується нулями:

$$V^{RC} = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}.$$

2. Послідовна обробка вершин, заданих рядками матриці, у відповідності до структури графа формує результуючу матрицю V^1:

V^1	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0
4	1	1	0	1	0	0	0	0	0
5	1	1	1	0	1	0	0	0	0
6	0	1	1	0	0	1	0	0	0
7	0	0	1	1	1	0	1	0	0
8	1	0	1	1	0	1	0	1	0
9	0	1	1	0	0	1	0	0	1

3. Накопичення ЗР у процесі обробки вершин приводить до остаточного результату у вигляді вектору VRC:

$$V^{RC} = \begin{vmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix},$$

где значение 1 свидетельствует о принадлежности линии к подмножеству сходящихся разветвлений.

3.6. Структурний аналіз цифрових послідовносних схем

Особливість структурного аналізу послідовносних схем пов'язана з отриманням ідентичних списків екстраобразів для кожної вершини, що входить до контуру, тут і далі, локальної (формуючої елемент пам'яті) зворотного зв'язку. На прикладі схеми тригера (рис. 3.6), що має три обратні зв'язки, розглянемо процедуру ідентифікації ЗР.

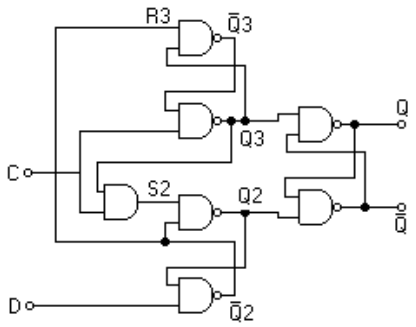


Рис. 3.6. Приклад тригерної структури

Графова модель тригерної структури, що зображена на рис. 3.7, містить 9 ліній, серед яких – 3 збіжні розгалуження ($C, Q3, Q2$) та 6 ліній, охоплених зворотніми зв’язками ($Q3, \overline{Q}3, Q2, \overline{Q}2, Q, \overline{Q}$).

Відповідно з твердженням 3.1 критерій пошукуЗР за наявністю однакових екстраобразів для більш, ніж однієї вершини графа на тлі ліній, охоплених контурами зворотних зв'язків, тут не працює.

Проблема полягає в тому, що контурні лінії можуть бути збіжними розгалуженнями. Наступне визначення дає можливість ідентифікувати ЗР для цифрових схем з контурами.

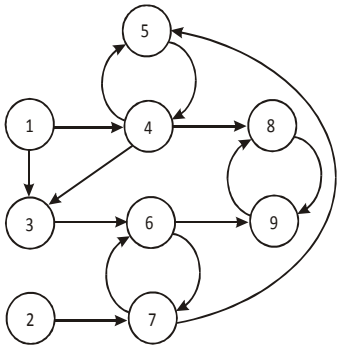


Рис. 3.7. Граф тригерної структури: 1 – C, 2 – D, 3 – S2, 4 – Q3, 5 – $\overline{Q}3$, 6 – Q2, 7 – $\overline{Q}2$, 8 – Q, 9 – $\overline{Q}$

Визначення 3.6. Лінія, що належить контуру зворотного зв'язку, є ЗР, якщо вона має більше однієї вихідної дуги, а також сходження на лінії, що не належить даному контуру.

Для структури, зображеної на рис. 3.8, збжними розгалуженнями є лінії (3,4), що входять до контуру і мають сходження на лініях 5 і 6, які не належать до контуру, що утворені лініями 3 і 4.

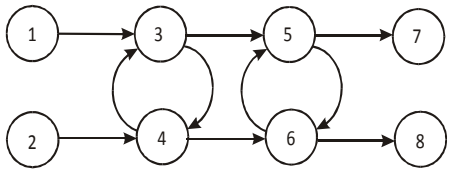


Рис. 3.8. Граф з контурами

Що стосується раніше розглянутого графа (див. рис. 3.7), то тут як ЗР виступають вершини, які задовольняють вимогам визначення 6.

Якщо в схемі присутні глобальні зворотні зв'язки, то перед виконанням процедури пошуку ЗР необхідно зробити їх псевдорозрив, наприклад, за алгоритмом, наведеним у [8], а потім застосувати описані вище моделі ідентифікації ЗР.

3.7. ОДП-метод моделювання несправностей цифрових систем

Запропонована інтерпретативно-компілятивна модель дедуктивно-паралельного аналізу несправностей і справної поведінки є базовою для ОДП-методу і гарантує знаходження рішення у вигляді множини всіх дефектів, що перевіряються на тест-векторі за час, пропорційний квадрату числа ліній n^2. Для зниження обчислювальної складності розв’язку даної задачі пропонується стратегія моделювання вхідного набору, що зображена на рис. 3.9.

Основна ідея підвищення швидкодії моделювання несправностей пов'язана з перетворенням ЗР в псевдовиходи з метою подальшого застосування процедури суперпозиції для деревовидних структур та їх необробки у випадку фіксації неперевірюваності ліній збіжних розгалужень.

Програмна реалізація ОДП-методу моделювання несправностей цифрового пристрою складається з двох основних компонентів: препроцесора попереднього структурного аналізу і власне алгоритму моделюювання тест-вектора.

Препроцесор структурного аналізу моделі схеми здійснює пошук ЗР в схемі цифрового пристрою. Обчислювальна складність даної процедури $Q_r = n^2$, але вона є разовою і виконується на стадії попереднього аналізу, тому практично не впливає на швидкодію моделювання вхідних наборів в цілому. Тут здійснюється визначення двійкового вектора ліній збіжних розгалужень (за допомогою одного з розглянутих вище методів структурного аналізу): $R = (R_1, R_2, ..., R_i, ..., R_n)$,

$$R_i = \begin{cases} 1 \leftarrow R_i \in R^{RC}; \\ 0 \leftarrow R_i \notin R^{RC}, \end{cases}$$

відносно якого реалізується процедура дедуктивно-паралельного аналізу несправностей.

Рис. 3.9. Стратегія ОДП-метода моделювання

Алгоритм моделювання тест-вектору орієнтирован на формування списку перевіряємих дефектів та складається з наступних пунктів.

1. Фаза справного моделювання цифрової схеми. Призначена для визначення реакції всіх невхідних ліній пристрою на вхідний набір $T_t \in T = [T_{ti}]$. У відповідності до визначення 3.4 тест-рядок матриці справної поведінки T може бути зображена як $T_t = (T_t^X, T_t^Z, T_t^Y)$. Аналогічно задається вектор перевіряємих несправностей $S = (S^X, S^Z, S^Y)$, який визначається кожного разу знову для чергового рядку T_t, що аналізується.

2. Моделювання несправностей ліній збіжних розгалужень на тест-векторі та модифікація схемної структури шляхом перетворення ЗР у псевдовиходи цифрового пристрою. Для цього виконується ініціалізація вектора перевіряємих несправностей схеми $S = (S_i^X = 0, S_i^Z = 0, S_i^Y = 1)$. Одиничне значення координати вектора $S_i = 1$ є індикатором перевірки одиночної несправності, що є інверсною двійковому справному стану лінії T_{ti}.

Потім, у відповідності до (3.5), виконується генерування вихідних дефектів ліній збіжних розгалужень схеми, оформлених у матрицю $M = [M_{ij}]$, але вже розмірністю $r \times n$, де r – кількість ЗР.

Моделювання дефектів ліній збіжних розгалужень $R^1 \subseteq R = (R^1 = \{R_1^1, R_0^1\}, R^0)$ (R^1, R_1^1, R_0^1, R^0 – ЗР; ЗР, що перевіряються та не перевіряються; лінії, що не відносяться до ЗР відповідно) здійснюється

паралельним методом обробки матриці M на дедуктивній моделі пристрою, що відповідає тест-вектору T_t. Використання методу орієнтовано на обробку дефектів ЗР, число яких у реальних проектах складає не больш 20%. Тут також можна використовувати дедуктивний метод моделювання ЗР, оскільки граничний список несправностей для схеми обмежений числом r.

3. Декомпозиція моделі схеми на основі результатів аналізу дефектів збіжних розгалужень. Виконується обчислення ліній підграфів схеми, моделювання несправностей яких на тест-векторі не повинно проводитися внаслідок існування формального доказу їх неперевіряємості. Іншими словами, з процесу моделювання виключаються екстраобрази - деревовидні підграфи з кореневими вершинами (збіжними розгалуженнями), несправності яких не перевіряються:

$$R = R \setminus f^*(R_0^1).$$

Отримані структури R є деревоподібними фрагментами цифрового пристрою, коректними для виконання суперпозиції рішень - векторів примітивів, що перевіряються на тест-векторі несправностей.

4. Моделювання несправностей залишившихся ліній, що доповнюють збіжні розгалуження, які перевіряються та не перевіряються: $R_1^1 \cup R_0^1$, і підграфи з неперевірюваними кореневими вершинами $f^*(R_0^1)$ до повної множини $R = R \setminus [R_1^1 \cup R_0^1 \cup f^*(R_0^1)]$. При цьому аналіз власних дефектів кожного примітиву виконується шляхом застосування дедуктивно-паралельного алгоритму до матриці дефектів, що перевіряються, але не схеми, а розглянутого елемента. Такий аналіз можна провести і дедуктивно, використовуючи власні вхідні списки перевіряємих несправностей кожного логічного елемента.

5. Виконання процедури суперпозиції векторів перевіряємих несправностей примітивів на скоригованій моделі цифрового пристрою. Згадана процедура зводиться до диз'юнкції вектора перевірюваних вхідних дефектів і-го примітиву

$$S^i = (S_1^i, S_2^i, \ldots, S_j^i, \ldots, S_{n_j}^i)$$

і вектора S перевірюваних несправностей схеми за умови, що на лінії, яка відповідає виходу і-го елемента, мається одиничне значення $S_i = 1$:

$$S(X_{ij}^a) = S(X_{ij}^a) \bigvee_{j=1}^{n_i - 1} S_j^i \leftarrow S_i = 1, \qquad (3.21)$$

де $X_i^a = (X_{i1}^a,...,X_{ij}^a,...,X_{i(n_i-1)}^a)$ – вектор номерів вхідних ліній і-го примітиву.

У масштабі всієї схеми суперпозиція зводиться до послідовного об'єднання векторів перевірюваних дефектів для примітивів, вихідні лінії яких є попередниками для зовнішніх виходів і перевіряються на вхідному наборі збіжних розгалужень - псевдовиходів. Інакше, моделювання дефектів виконується тільки щодо вихідних спостережуваних ліній схеми, доповнених розгалуженнями, несправності яких виявляються на тест-векторі T_t: $R^Y = R^Y \cup R_1^1$, де R^Y – вихідні лінії схеми, що доповнюються розгалуженнями R_1^1 з перевірюваними несправностями.

Приклад 3.4. Ілюструє процедуру зворотної суперпозиції для деревовидної схеми, що зображено на рис. 3.10, де у дужках наведені справні стани ліній.

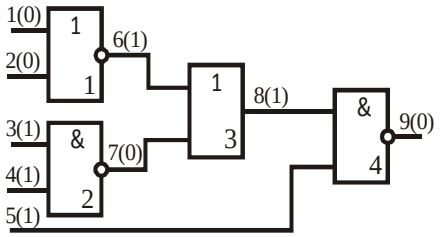


Рис. 3.10. Деревовидна схема

Вектори перевірюваних на виходах неправностей примітивів подані першими чотирма рядками (координати задають: 0 (1) - перевірка константи нуля (одиниці), точка - порожня множина перевірюваних дефектів на лінії) наступної таблиці:

1	2	3	4	5	6	7	8	9	$S = S \cup S^i \leftarrow S_i \cap S_i^i \neq \varnothing$
.	.	.	.	0	.	.	0	1	S^4
.	.	.	.	.	0	.	0	.	S^3
.	.	0	0	.	.	1	.	.	S^2
1	1	.	.	.	0	.	.	.	S^1
.	.	.	.	0	0	.	0	1	$S = S^4 \cup S^3 \leftarrow S_8^4 \cap S_8^3 = 0$
.	.	.	.	0	0	.	0	1	$S = S \leftarrow S_7 \cap S_7^2 = \varnothing$
1	1	.	.	0	0	.	0	1	$S = S \cup S^1 \leftarrow S^6 \cap S_6^1 = 0$

Рядки обробляються у відповідності з виразом:

$$S = S \cup S^i \leftarrow S_i \cap S_i^i \neq \varnothing, \tag{3.22}$$

який є спрощеним записом рівності (3.21), де S (S_i) - тризначний вектор перевірюваних дефектів схеми (примітиву). Суперпозиція виконується, якщо перетин двох векторів за лінією зв'язку двох відповідних примітивів не дорівнює пустої множині. При цьому число операцій суперпозиції - (n-1), де

n - кількість елементів у схемі. Результат у вигляді вектора перевірюваних на наборі 00111 дефектів поданий останнім рядком згаданої таблиці.

3.8. Оцінка швидкодії ОДП-метода

Як приклад реконфігурації виступає схема, наведена на рис. 3.11, де лінії 15, 17, 19 є ЗР, які збігаються на елементах з виходами 22, 24, 28. В результаті застосування алгоритму їх пошуку схема модифкується в чотири деревовидні структури, що визначаються підграфами з кореневими вершинами, які є виходами або псевдовиходамі схеми (рис. 3.12).

Моделювання несправностей такої схеми за допомогою зворотньої суперпозиції вимагає вже лінійних витрат пам'яті і часу у функції від числа ліній і квадратичних витрат для обробки збіжних розгалужень: $Q = (r^2 / W) + 2n + n(1 - r/n)$, де r^2 / W - час моделювання несправностей r збіжних розгалужень; $2n = n_r + n_p$, $n_r = n$ - час реконфігурування примітивів схеми на вхідному наборі; $n_p = n$ - час пошуку підграфів ліній, що відповідають неперевіряємим збіжним розгалуженням; $n(1 - 3r)$ - час виконання процедури суперпозиції на множині ліній схеми без збіжних розгалужень і попередників для неперевіряємих ЗР.

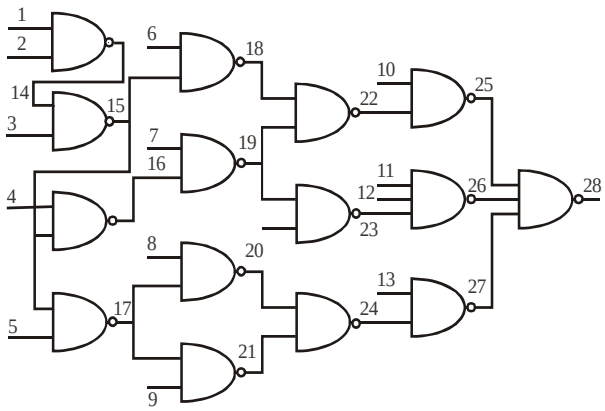


Рис. 3.11. Схема з розгалуженнями

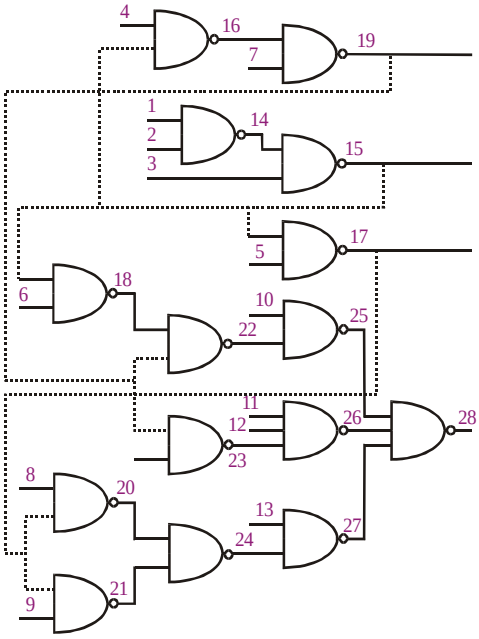


Рис. 3.12. Деревовидні фрагменти схеми

Таким чином, виграш у швидкодії запропонованого ОДП-методу тим більше, чим менше збіжних розгалужень в схемі цифрового пристрою.

3.9. Топологічний ОДП-метод моделювання

Проведемо модифікацію процедури суперпозиції в бік зворотнього простежування дефектів на топології схеми. Використанню процедури зворотньої суперпозиції або зворотнього простежування у загальному випадку перешкоджує неможливість виконання одновимірної активізації, яка може бути проілюстрована двома варіантами некоректності, що зображені такими прикладами.

Приклад 3.5. Виконати аналіз схем (рис. 3.13) з метою визначення множини дефектів, що перевіряються, на заданих тест-векторах за допомогою процедури зворотньої суперпозиції.

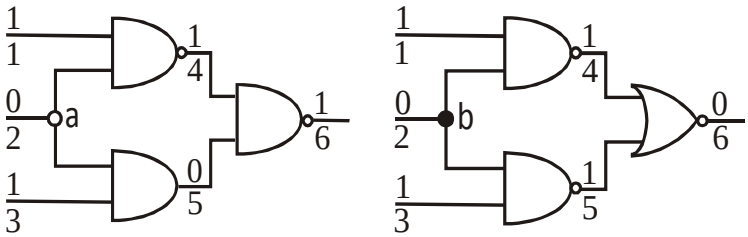


Рис. 3.13. Хибна перевірка та неперевірка дефектів

Для обох схем активізація несправностей на лініях a і b дає некоректні результати їх перевірки.

У лівій схемі константна несправність 21 ложно перевіряється на одновимірному шляху 2-5-6 за допомогою процедури зворотного простежування. Однак при цьому не враховується той факт, що даний дефект змінює стан лінії 4 з 1 на 0, що створює умови заборони транспортування несправності на вихід схеми. Тому несправність на лінії a - 21 є ложно перевірюваною на наборі (101) при розгляді її транспортування за одновимірним шляхом активізації.

Права схема є прикладом альтернативної ситуації - до виходу 6 немає одновимірного шляху транспортування дефектів з зовнішніх входів і, зокрема, від лінії b. Тим не менш, несправність на згаданій лінії - 21 перевіряється, оскільки її присутність створює кратний дефект на лініях 4 і 5, який змінює стан вихідної лінії схеми.

В обох випадках має місце суттєва некоректність, яка не дозволяє використовувати тільки ідею одновимірності при відстеженні шляху транспортування несправностей від виходів до входів схеми.

Проте враховуючи, що некоректність пов'язана виключно з одновимірною активізацією несправностей збіжних розгалужень, необхідно спочатку виконати тільки їх обробку, а потім виключити з розгляду, зробивши структуру схеми деревовидною і придатною для одновимірного зворотнього простежування. Природно, для цього на стадії попереднього аналізу (блок Р, рис. 3.14) слід виконати додаткову процедуру визначення всіх збіжних розгалужень.

Оскільки далі буде розглядатися аналіз несправностей у прив'язці до топології схеми цифрового пристрою, то природним видається назвати алгоритм аналізу топологічним. Таким чином, топологічний із зворотним простежуванням дедуктивно-паралельний алгоритм (ТОДП) моделювання цифрових систем (див. рис. 3.14) можна зобразити кроками:

1. Аналіз справної поведінки цифрового пристрою на заданому вхідному наборі.
2. Перетворення схеми на поточному тест-векторі в дедуктивну модель.
3. Моделювання несправностей ЗР за дедуктивною моделлю схеми.
4. Зворотне простежування перевірюваних несправностей примітивів за ДС дедуктивної моделі.

Пункт 3 орієнтований на обробку тільки збіжних розгалужень, кількість яких значно менше за інші лінії; пункт 4 - на моделювання несправностей ліній, що відносяться до деревоподібних підграфів. Теоретичним обгрунтуванням застосування останнього пункту є доведення наступних теорем і формулювання наслідків з них.

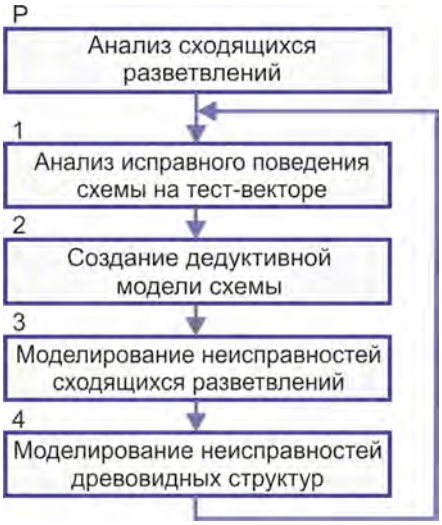


Рис. 3.14. Алгоритм топологічного моделювання

Лемма 3.1. Збіжні розгалуження $R_i^1 \in R^1$ у комбінаційній схемі спричиняють появу кратних дефектів на входах примітивів.

Д о к а з. Нехай комбінаційна схема не має збіжних розгалужень. В цьому випадку вона подана деревовидною структурою. Тоді перетин підграфів-попередників (екстраобразов) для будь-яких входів примітивного елемента дорівнює пустой множині

$$f^*(X_i) \cap f^*(X_j) = \varnothing. \tag{3.23}$$

Це справедливо для примітивів першого рівня, де входи елементів є входами схеми, занумерувані різними ідентифікаторами ліній. Якщо примітив знаходиться всередині схеми, то в силу деревоподібності (3.23) будь-які два входи не матимуть хоча б одного загального попередника. Інакше порушиться умова деревоподібної графової структури. Таким чином, два будь-яких входи примітиву деревовидної схеми, не маючи загальних попередників, не матимуть і загальних несправностей на конкретному двійковому вхідному наборі, які можуть бути транспортовані через розглянутий елемент.

Теорема 3.1. *Для деревовидної структури дедуктивної схеми інверсний вхід примітиву I:* $\overline{X}_i \in L$ *забороняє транспортування всіх несправностей, що належать лініям-попередникам.*

Д о к а з. По-перше, інверсія на вході може мати місце на примітиві, має дві і більше вхідні лінії, по-друге, вона означає вирахування списку несправностей, що належить даному входу, враховуючи ізоморфізм $\overline{X}_i X_j \approx X_j \setminus X_i$.

Але відповідно до леми 3.1, перетин двох списків ліній-попередників, що відносяться до входів одного примітиву, одно пустої множині (3.23). От-

же, віднімання списку ліній-попередників ніколи не зменшить потужність будь-якого іншого списку, що відноситься до неінверсного входу розглянутого елемента. Таким чином, для деревоподібних структур комбінаційних схем завжди буде виконуватися умова:

$$(X_i) \setminus f^*(X_j) = f^*(X_i) \leftarrow f^*(X_i) \cap f^*(X_j) = \varnothing.$$

Теорема 3.2. *Якщо у деревовидній структурі дедуктивної схеми L існує примітив I, що має більш одного неінверсного входу* $X_i X_j$*, то такий елемент забороняє транспортування всіх несправностей, що належать лініям-попередникам всіх його входів.*

Д о к а з. Наявність двох і більше неінверсних, прямих входів у дедуктивному елементі означає перетин списків ліній, які є попередниками для розглянутих входів. Але оскільки, відповідно до леми 3.1, входи одного примітиву ДС не мають спільних попередників, то звідси випливає, що перетин згаданих списків буде завжди дорівнювати пустой множині:

$$f^*(X_i) \cap f^*(X_j) = \varnothing \leftarrow X_i X_j.$$

Якщо ж дедуктивний елемент має і входи з інверсією, то згідно до теореми 1 віднімання з отриманої порожньої множини попередників будь-якого непорожнього, що належить входу з інверсією, дає також порожній результат.

Отже, дедуктивний елемент And, що має більш одного неінверсного входу, не пропустить через себе несправності ліній попередників.

Слідство: 1) Інверсний вхід дедуктивного елемента And забороняє активізацію всіх несправностей ліній-попередників, які стосуються даного входу. 2) Якщо дедуктивний примітив має більше одного неінверсного входу, то всі його входи слід визначити інверсними. 3) Дедуктивний елемент And пропустить несправності ліній-попередників тільки за неінверсним входом, який повинен бути єдиним. 4) Дедуктивний елемент Or не може мати інверсних входів. 5) Збіжне розгалуження відзначається інверсією (кружком на лінії розгалуження), якщо його несправність не перевіряється на тест-векторі. 6) Інверсія на лінії, вхідної або вихідної, є умовою розриву активізації несправностей і припинення зворотнього простежування дефектів по розглянутій гілки деревовидної структури. 7) Перевірювану на тест-векторі лінію збіжного розгалуження (на топологічному рисунку схеми (рис. 3.15) далі зазначається жирним кружечком) при виконанні процедури зворотнього простежування слід розглядати як спостережуваний вихід схеми. 8) Інтерпретація результату топологічного моделювання: несправності ліній, не відзначені знаками інверсій на дедуктивної моделі схеми, перевіряються.

Як ілюстрації основних кроків топологічного моделювання нижче пропонуються наступні два приклади.

Приклад 3.6. Нехай дано цифровий пристрій (рис. 3.15, перша схема, при розгляді зверху вниз), що містить 3 збіжних розгалуження.

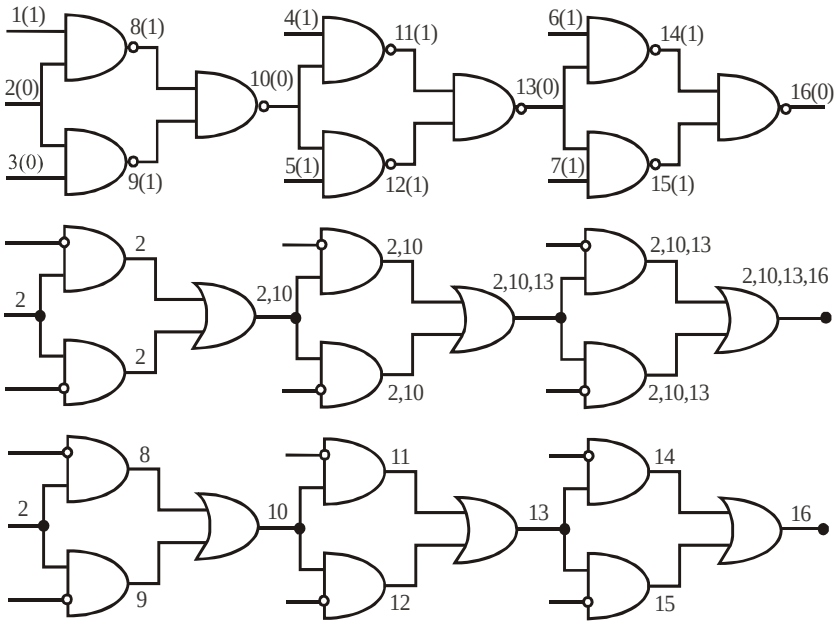


Рис. 3.15. Моделювання схеми з розгалуженнями

Пристрій має 16 ліній, у тому числі 7 входів, на них подається тест-вектор 1011111, для якого слід визначити дефекти константного типу, що перевіряються. Стани ліній після справного моделювання подані в дужках. Результат процесу перетворення справної моделі пристрою в дедуктивну та моделювання несправностей ліній ЗР зафіксований на другій схемі. Тут визначено, що всі несправності розгалужень (2, 10, 13), інверсні по відношенню до справного стану цих ліній, є такими, що перевіряються. Факт перевірки відзначений на схемі додатково чорними кружечками.

Процедура зворотнього простежування несправностей з метою визначення списку перевірюваних дефектів полягає в побудові максимальних деревовидних підграфів, обмежених на топології прозорими кружочками. При цьому чорні кружечки є перевірюванія лінії розгалужень, які слід розглядати на третій схемі (див. рис. 3.15) як спостережувані виходи. Тут же позначені всі несправності, що перевіряються на тест-векторі: {2, 8, 9, 10, 11, 12, 13, 14, 15, 16}. Знак дефекту визначається інверсією по відношенню до стану справної поведінки лінії. Для моделювання даного цифрового пристрою істинним є твердження: несправності ліній, не відзначені прозорими кружечками (знаками інверсії), перевіряються.

Приклад 3.7. Визначити список перевірюваних на тест-векторі 10111 несправностей для цифрового пристрою, що зображено на рис. 3.16, який містить 2 збіжних розгалуження.

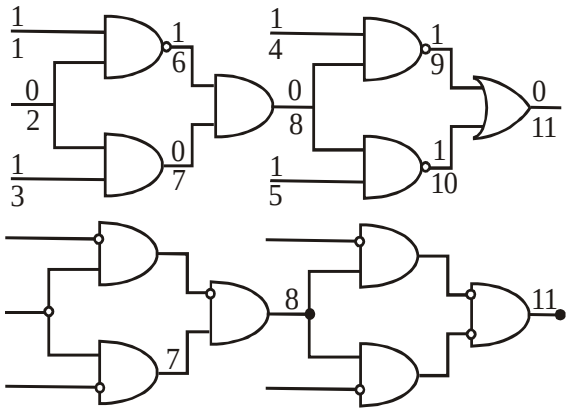


Рис. 3.16. Схема з двома розгалуженнями

Тут інтерес становить неперевірка несправностей ліній 9 і 10 поряд з фактом можливості перевірки ліній 7 і 8, завдяки наявності збіжних розгалужень 8, моделювання несправності якого показало, що вона транспортується на зовнішній вихід 11.

Приклад 3.8. Визначити список перевірюваних на тест-векторі 10111 несправностей для цифрового пристрою, зображеного на рис. 3.17.

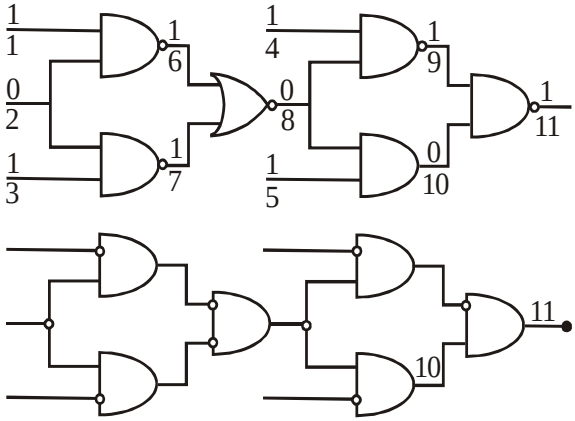


Рис. 3.17. Схема з розгалуженням, що не перевіряється

Результат топологічного моделювання показав, що неперевірка збіжного розгалуження 8 забороняє транспортування всіх дефектів, які відносяться до ліній-попередників згаданого розгалуження. Тому перевіряються зафіксовані

несправності тільки для ліній 10 і 11. Інакше перевіряються дефекти тих ліній, які не відмічені знаками інверсій.

На рис. 3.18 представлена структура еволюційного розвитку методів моделювання несправностей. В основу покладені дедуктивний (Д) і паралельний (П) алгоритми. Далі був розроблений універсальний, але відносно повільний, дедуктивно-паралельний метод (ДП). Потім був реалізований ОДП-метод, орієнтований на швидку обробку моделей цифрових систем. Останній був модифікований до ТОДП-методу, який орієнтований на вентильний рівень подання цифрових систем. При цьому в описі пристрою попередньо здійснюється пошук множини збіжних розгалужень і виділення деревоподібних структур як доповнення до ЗР. Для моделювання збіжних розгалужень використовується універсальний дедуктивно-паралельний алгоритм, для аналізу древовідних структур - ОДП- і ТОДП-методи. Такий розподіл функцій обробки цифрових схем великої розмірності дозволяє як мінімум на порядок підвищити швидкодію моделювання несправностей у порівнянні з базовими методами (дедуктивний і паралельний).

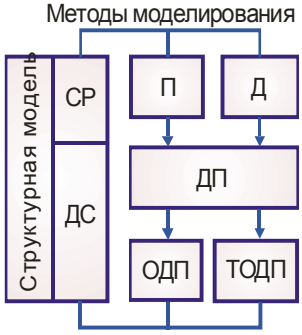


Рис. 3.18. Еволюція методів моделювання

Запропонований метод моделювання несправностей орієнтований на обробку цифрових пристроїв на основі ПЛІС, що містять мільйони вентилей. Тестові експерименти програмної реалізації методу на сотнях цифрових комбінаційних і послідовних схем дали добрі результати за швидкодією у порівнянні з традиційними алгоритмами паралельного і дедуктивного моделювання. Окремі приклади аналізу швидкодії розробленого методу (обробка тест-прикладів на 1000 вхідних послідовностей, IBM PC 500 МГц, 256 Мбайт) та існуючих базових показані на рис. 3.19. Прискорення моделювання становить не менше десяти разів. На рис. 3.20 зображено результати аналізу швидкодії для трьох реалізованих методів моделювання цифрових схем на одному і тому ж комп'ютері при обробці 1000 векторів. Показано перевагу ОДП- і ТОДП-методів перед дедуктивно-паралельним. Виграш у швидкодії більш істотний для схем великої розмірності. Число збіжних розгалужень в тест-схемах у середньому становить 20% від загальної кількості ліній.

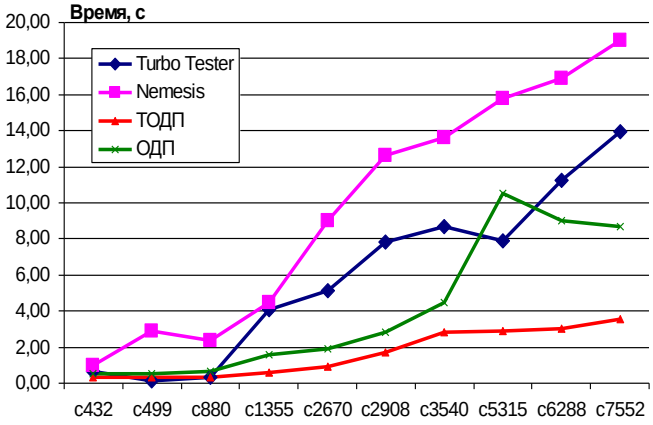


Рис. 3.19. Аналіз швидкодії систем моделювання

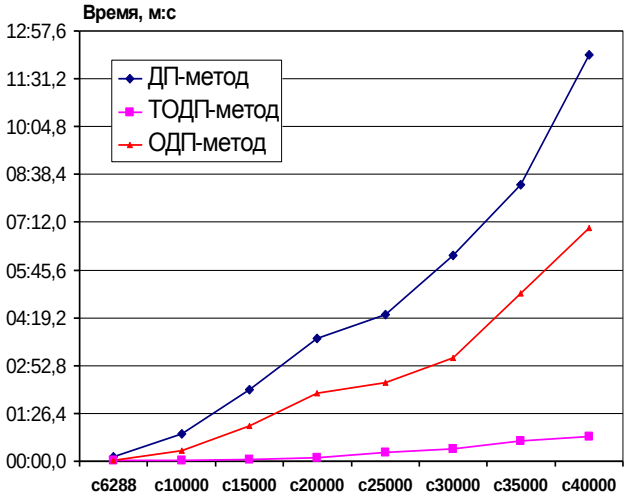


Рис. 3.20. Аналіз швидкодії методів моделювання

Таким чином, основним результатом є удосконалення дедуктивно-паралельного методу моделювання несправностей цифрових систем, а саме:

1) Створення узагальненої моделі процесу дедуктивно-паралельного аналізу цифрової схеми на основі процедури зворотньої суперпозиції, що має лінійну обчислювальну складність від числа ліній схеми.

2) Розробка дедуктивних алгоритмів структурно-функціонального аналізу цифрових систем з метою визначення множини збіжних розгалужень і реконфігурації структури для реалізації процедури суперпозиції.

3) Створення внутрішньої інтерпретативно-компілятивної моделі цифрового пристрою для ефективного справного аналізу логічних елементів і їх несправностей одиночного константного типу.

4) Розробка топологічного алгоритму моделювання несправностей за деревовидною структурою цифрової системи, що має лінійну обчислювальну складність у залежності від числа ліній схеми.

3.10. Дедуктивний аналіз дефектів

Розглядається дедуктивно-паралельний метод аналізу несправностей [1,2,4,5,7,8,11-15,17,19,20,23-26,28,29-30,31-40], орієнтований на обробку цифрових проектів великої розмірності вентильного і реєстрового рівнів опису з метою отримання таблиці несправностей і оцінки якості покриття тестом дефектів заданого класу. Об'єкт тестування подано у формі структур, таблиць, булевих рівнянь, кубічних покрить і реалізують складну цифрову систему, імплементовану у кристали SoC. Пропонований метод моделювання несправностей сполучає переваги дедуктивного визначення списків несправностей, ефективного з позиції математики, і виконання паралельних процедур, орієнтованих на високошвидкісну обробку цифрових пристроїв вентильного, системного і реєстрового рівнів опису.

Мета - створення швидкодіючого методу моделювання одиночних константних несправностей для оцінки якості синтезованих тестів цифрових систем, імплементованих у кристали, що містять мільйони вентилів.

Основа (Backtraced Deductive-Parallel) дедуктивно-паралельного моделювання несправностей - методи підвищення швидкодії аналізу несправностей [2,7], дедуктивна модель транспортування несправностей [6,8,31], паралельний метод обробки списків дефектів функціонального елемента [6] і алгоритм зворотного простежування примітивів [7] при обробоці цифрового пристрою.

Ідея дедуктивного методу моделювання несправностей з позиції теорії множин полягає в обчисленні вихідної підмножини дефектів, що перевіряються, за одну ітерацію аналізу примітиву. Найбільша загальна дедуктивна модель аналізу можливих дефектів будь-якого пристрою (багатовиходового примітиву) використовує формули визначення списку несправностей за виходами його таблиці функціонування (кубічного покриття C, таблиці переходів-виходів):

$$
\begin{aligned}
&L_r = \{ \bigcup_{Z^{\varnothing}} [(\bigcap_{X^{\varnothing}} L_j) \setminus (\bigcup_{X^{\overline{\varnothing}}} L_j)] \} \cup \{ \equiv \overline{T}_r^{z} \}, \\
&X^{\varnothing} = \forall j (T_j \cap C_{ij}^{x} = \varnothing); \\
&X^{\overline{\varnothing}} = \forall j [(T_j \cap C_{ij}^{x} \neq \varnothing) \,\&\, (C_{ij}^{x} \neq X)]; \\
&Z^{\varnothing} = \forall i (T_r \cap C_{ir}^{z} = \varnothing); \\
&i = \overline{1,n};\ j = \overline{1,m};\ r = \overline{1,k};
\end{aligned}
\tag{3.24}
$$

де n – число строк (кубов); m – кількість входних ліній; k – число вихідних ліній у пристрої (примітиві).

Умови на входах примітиву і його функціональність формують множину дедуктивних функцій (ДФ), призначених для тестування несправностей.

Тому на практиці очевидним є прагнення до спрощення функціональності, що призводить до технологічних і алгорітмічних нескладних процедур побудови моделей для аналізу дефектів. Тим не менш, універсальні алгоритми аналізу таблиць бувають більш переважними у випадках, коли цифрова система в явному вигляді подається на кристалі у табличній формі, як це має місце в PLD. Особливо важною є дана обставина у зв'язку з моделюванням великих проектів на кристалах, що нараховують мільйони вентилів. Тут, перш за все, необхідно укрупнювати компоненти системи шляхом їх композиції до макроблоків, а потім обробляти їх у вигляді примітивів, заданих узагальненими таблицями істинності. Для кожного тест-вектора функціональність примітиву модифікується від таблиці справної поведінки до дедуктивної (теж таблиці) з метою виконання моделювання дефектів, що перевіряються. Проблема полягає у виборі стратегії побудови моделі, що більш краще зробити: 1) модифікувати кожен раз таблицю істинності (переходів) чи 2) зробити всі модифікації або дедуктивні моделі заздалегідь, а потім їх викликати з пам'яті для обробки чергового вектора? Для кожного тест-вектора або вхідної умови пристрою створюється власна дедуктивна модель схеми.

Приклад 3.9. Задано кубічне покриття функції

$$C(\bar{x}_2\bar{x}_3 \vee x_1x_2x_3) = \begin{Bmatrix} x_1 & x_2 & x_3 & y \\ X & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ X & 0 & 1 & 0 \\ X & 1 & 0 & 0 \\ 0 & 1 & X & 0 \end{Bmatrix}.$$

Вхідні списки несправностей визначені підмножинами:

$$S_1 = \{1,2,4,5\};$$
$$S_2 = \{1,2,3,6\};$$
$$S_3 = \{1,3,4,7\}.$$

Визначити вихідний список несправностей для тест-вектора 1111.

Розв’язок зводиться до виконання перетинів між вихідними координатами вектора T і кубами покриття, що задовільнюють умовам необхідності $\forall i(T_r \cap C_{ir}^z) = \varnothing$. Далі аналіз кубів по (3.24) дає результат:

$$S^1(C_1 = X001) = \varnothing;$$
$$S^2(C_2 = 1111) = \varnothing;$$
$$S^3(C_3 = X010) = S_2 \setminus S_3 = \{2,6\};$$
$$S^4(C_4 = X100) = S_3 \setminus S_2 = \{4,7\};$$
$$S^5(C_5 = 01X0) = S_1 \setminus S_2 = \{4,5\};$$
$$S(T = 1111) = S^3 \cup S^4 \cup S^5 = \{2,4,5,6,7\},$$

де S^i – частковий список, що формується кубом C_i; $S(t)$ – список дефектів, що перевіряється тест-вектором T і отримується об'єднанням $S = \bigcup_{i=1}^{n} S^i$.

Для тест-вектора 0100 результат моделювання несправностей має вид:

$$S^1(C_1 = X001) = S_2 \setminus S_3 = \{2,6\};$$
$$S^2(C_1 = 1111) = S_1 \cap S_3 \setminus S_2 = \{4\};$$
$$S^3(C_3 = X010) = \varnothing;$$
$$S^4(C_4 = X100) = \varnothing;$$
$$S^5(C_5 = 01X0) = \varnothing;$$
$$S(T = 0100) = S^1 \cup S^2 = \{2,4,6\}.$$

Дедукція є умовивід у системі математичних доказів від загального до приватного. У разі її застосування до аналізу дефектів цифрових систем мається на увазі знаходження таких алгеброгічних закономірностей, які дозволяють використовувати один раз отримані, складні аналітичні моделі, багаторазово використовувані для обробки цифрових систем з метою моделювання несправностей. При цьому кожен дефект повинен бути спочатку описаний за допомогою таблиці істинності, булева рівняння, графа переходів. Фактично модель дедуктивного аналізу несправностей довільної цифрової функціональності дозволяє за одну ітерацію (декілька - для послідовних схем з глобальними зворотними зв'язками) обробки схеми обчислювати всі дефекти, що перевіряються на тест-векторі. Математична модель дедуктивного аналізу дефектів цифрових систем може бути подана матричним рівнянням [2,5,16,18,19, 21,22]:

$$(T_1, T_2, \ldots, T_t, \ldots, T_n) \oplus \begin{bmatrix} C_{11}, C_{12}, \ldots, C_{1i}, \ldots, C_{1n} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ C_{t1}, C_{t2}, \ldots, C_{ti}, \ldots, C_{tn} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ C_{k1}, C_{k2}, \ldots, C_{ki}, \ldots, C_{kn} \end{bmatrix} = \begin{bmatrix} L_{11}, L_{12}, \ldots, L_{1i}, \ldots, L_{1n} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ L_{t1}, L_{t2}, \ldots, L_{ti}, \ldots, L_{tn} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ L_{k1}, L_{k2}, \ldots, L_{ki}, \ldots, L_{kn} \end{bmatrix}, \quad (3.25)$$

де C – кубічне покриття справної поведінки пристрою, що має n ліній; $T = (T_1, T_2, ..., T_t, ..., T_n)$ – тест-вектор для перевірки дефектів, спотворюючих роботу функціональності C, довизначений у процесі справного моделювання на множині вхідних, внутрішніх та вихідних ліній; координата матриці дефектів визначається на основі виконання логічної операції XOR: $L_{ti} = T_t \oplus C_{ti}$; матриця $L = |L_{ti}|$ - є дедуктивна функція (ДФ) моделювання несправностей на тест-векторі T, що відповідає справному елементу з покриттям C, яка дає можливість обчислювати список вхідних несправностей, що транспортуються на виходи елемента [6].

У загальному випадку функція цифрового пристрою або його компонента перед-ставляется таблицею істинності (переходів), а застосування дедуктивної формули (3.25) дозволяє отримати для заданого тест-вектора T таблицю перевірки несправностей, за якою можна записати аналітичну формулу моделювання дефектів.

Приклади отримання таких функцій - аналітичний запис таблиці неисправностей - подані нижче (тест-вектор, таблиця істинності, таблиця перевірки дефектів):

| F – deductive | | |

яке ізоморфно виразу (3.25). Таким чином, формули (3.25) і (3.26) покривають всі цифрові системи, описи яких подані як на високому (системний, регістровий), так і на низькому (вентильному) рівні за допомогою мов опису апаратури.

3.11. Синтез дедуктивних компонентів для функцій SoC

Вентильний рівень опису схеми характеризується логічними елементами, функціонування яких задається таблицями істинності, кубічними покриттями або логічними рівняннями. В даному випадку технологічно розглядати процедури синтезу на основі використання аналітичної форми. При цьому двовходовий логічний елемент трансформується в чотиривходовий, де два додаткових входи (a, b) є регістровими і служать для транспортування списків несправностей. При цьому булеві входи (x, y), по суті, є керуючими для виконання операцій над зовнішніми списками дефектів. Нехай є ло-гічний елемент And, для якого дедуктивна функція подана вигляді карти Карно [27]:

$L = f(x, y, a, b) =$

(x, y) \ (a, b)	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	0	1	1	1
10	0	1	0	0

(3.27)

Мінімізація примітиву, заданого в (3.27), приводить до трьох варіантів дедуктивної функції з різною обчислювальною складеістю (кількість змінних і термів) за Квайном (19, 15 і 17):

$$
\begin{aligned}
&1)\ L = f(x, y, a, b) = (\bar{x}\,\bar{y} \wedge ab) \vee (y \wedge a\bar{b}) \vee (x \wedge \bar{a}b) \vee (xy \wedge b) \vee (xy \wedge a) = \\
&\qquad = (\bar{x}\,\bar{y} \wedge ab) \vee (x \wedge \bar{a}b) \vee (y \wedge a\bar{b}) \vee [(xy \wedge (a \vee b)]; \\
&2)\ L = f(x, y, a, b) = (\bar{x}\,\bar{y} \wedge ab) \vee (y \wedge a\bar{b}) \vee (x \wedge \bar{a}b) \vee (xy \wedge a) = \\
&\qquad = (\bar{x}\,\bar{y} \wedge ab) \vee (x \wedge \bar{a}b) \vee [(ya \wedge (x \vee \bar{b})]; \\
&3)\ L = f(x, y, a, b) = (\bar{x}\,\bar{y} \wedge ab) \vee (y \wedge a\bar{b}) \vee (x \wedge \bar{a}b) \vee (xy \wedge b) \vee (xy \wedge a) = \\
&\qquad = (\bar{x}\,\bar{y} \wedge ab) \vee [(ya \wedge (x \vee \bar{b})] \vee [(xb \wedge (y \vee \bar{a})].
\end{aligned}
\tag{3.28}
$$

Вибір кращої з них призводить до реалізації формули під номером 2. Аналогічні перетворення, застосовні до елементів Or, Not, приводять до синтезу булевих рівнянь і схемних структур. Синтез дедуктивної функції елемента Or представлений наступними перетвореннями:

$L = f(x, y, a, b) =$

(x, y) \ (a, b)	00	01	11	10
00	0	1	1	1
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$$1)\ L = f(x, y, a, b) = (\bar{x}\,\bar{y} \wedge a) \vee (\bar{y} \wedge a\bar{b}) \vee (\bar{x} \wedge \bar{a}b) \vee (xy \wedge ab) =$$
$$= [\bar{y}a \wedge (\bar{x} \vee \bar{b})] \vee (\bar{x} \wedge \bar{a}b) \vee (xy \wedge ab);$$
$$2)\ L = f(x, y, a, b) = (\bar{x}\,\bar{y} \wedge a) \vee (\bar{x}\,\bar{y} \wedge b) \vee (\bar{y} \wedge a\bar{b}) \vee (\bar{x} \wedge \bar{a}b) \vee (xy \wedge ab) =$$
$$= [\bar{x}\,\bar{y}(a \vee b)] \vee (\bar{y} \wedge a\bar{b}) \vee (\bar{x} \wedge \bar{a}b) \vee (xy \wedge ab);$$
$$3)\ L = f(x, y, a, b) = (\bar{x}\,\bar{y} \wedge a) \vee (\bar{y} \wedge a\bar{b}) \vee (\bar{x} \wedge \bar{a}b) \vee (xy \wedge ab). \tag{3.29}$$

За аналогією виконується синтез дедуктивної функції для элемента Xor.

$L = f(x, y, a, b) =$

(x, y) \ (a, b)	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

(3.30)

$$L = f(x, y, a, b) = \bar{a}b \vee a\bar{b}. \tag{3.31}$$

Результати апаратної реалізації мінімальних з точки зору оцінки за Квайном дедуктивних функцій трьох згаданих елементів імплементується у наступні схеми, що забражені на рис. 3.21.

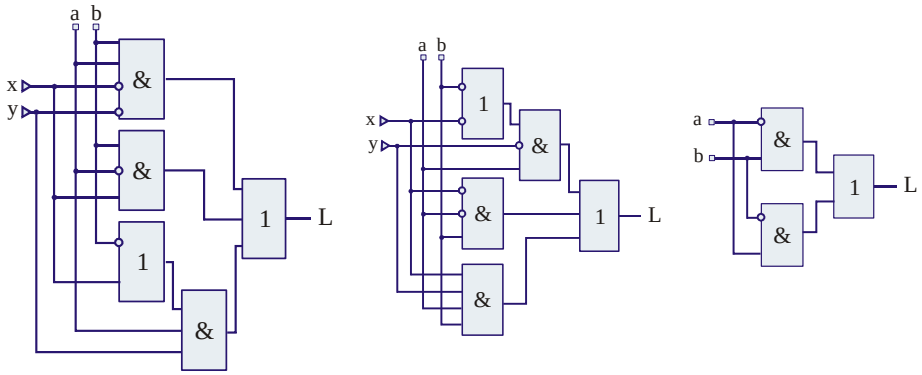


Рис. 3.21. Дедуктивні примітиви логічних елементів (and, or, xor)

Регістровий рівень опису компонентів цифрової системи відрізняється функціональної складністю, що впливає на розмірність таблиць істинності або кубічних покриттів. Тут розглядаються такі функціональності як: тригери, засувки, лічильники, мультиплексори, регістри, шинні структури. Аналогічні перетворення, спрямовані на синтез дедуктивної функції за допомогою таблиці істинності тригера (три булевих і три реєстрових входи) $Q = DC \vee \overline{C}\overline{D}Q(t-1)$, дають результат:

$L = f(T, X) =$

(T) \ (X)	000	001	011	010	110	111	101	100
000	0	1	1	0	1	1	0	0
001	0	1	1	0	0	0	1	1
011	0	1	1	0	1	1	0	0
010	0	1	1	0	0	0	1	1
110	0	0	1	1	1	0	0	1
111	0	0	1	1	0	1	1	0
101	0	0	1	1	1	0	0	1
100	0	0	1	1	0	1	1	0

(3.32)

$$L = f(c, d, q, C, D, Q) = (\bar{c} \wedge \bar{C}\bar{D}Q) \vee (\bar{C}DQ) \vee (c \wedge \bar{C}D\bar{Q}) \vee$$
$$\vee (\bar{c}\bar{d}\bar{q} \wedge CD) \vee (\bar{c}\bar{d}q \wedge C\bar{D}) \vee (\bar{c}dq \wedge CD) \vee (\bar{c}d\bar{q} \wedge C\bar{D}) \vee \qquad (3.33)$$
$$\vee (cd\bar{q} \wedge C\bar{Q}) \vee (cdq \wedge CQ) \vee (c\bar{d}q \wedge C\bar{Q}) \vee (c\bar{d}\bar{q} \wedge CQ).$$

Апаратна реалізація (оцінка за Квайном - 62) тригера зображено на рис. 3.22. Його дедуктивна функція має більш ніж 10-кратну апаратну надмірність у порівнянні зі справною функціональністю. Тим не менш, таке подання дає можливість вигравати у швидкодії дедуктивного моделювання несправностей у сотні разів.

Що стосується аналізу компонентів системного рівня, то в загальному випадку таблиця істинності (переходів-виходів) є частково або неповністю визначеною. Це означає, що алфавіт опису координат таблиці містить три символи, принаймні (0,1,X). Для даного випадку необхідна модифікація процедури дедуктивного аналізу несправностей, яка в трійковому алфавіті виконання операцій має вигляд $(i = 1, n)$:

$$L_r = \bigvee_{\forall i(T_r \oplus C_{ir}^{z} = 1)}^{r=1,k} [(\bigwedge_{\forall j(T_j \oplus C_{ij}^{x} = 1)}^{j=1,m} L_j) \setminus (\bigvee_{\forall j[(T_j \oplus C_{ij}^{x} = 0) \& (C_{ij}^{x} \neq X)]}^{j=1,m} L_j), \qquad (3.34)$$

де n, m, k – число рядків (кубів), вхідних і вихідних ліній у примітиві; L_r – список несправностей для выхода r у формі дефектів, що транспортуються через примітив або цифрову систему, від зовнішніх входів.

Основні операції у трійковому алфавиті мають вигляд:

$$Xor = [0 \oplus 0 = 0; 0 \oplus 1 = 1; 1 \oplus 0 = 1; 1 \oplus 1 = 0; 0 \oplus X = X; 1 \oplus X = X; X \oplus X = X];$$

$$Or = [0 \vee 0 = 0; 0 \vee 1 = 1; 1 \vee 0 = 1; 1 \vee 1 = 1; 0 \vee X = X; 1 \vee X = 1; X \vee X = X];$$

$$And = [0 \wedge 0 = 0; 0 \wedge 1 = 0; 1 \wedge 0 = 0; 1 \wedge 1 = 1; 0 \wedge X = 0; 1 \wedge X = X; X \wedge X = X].$$

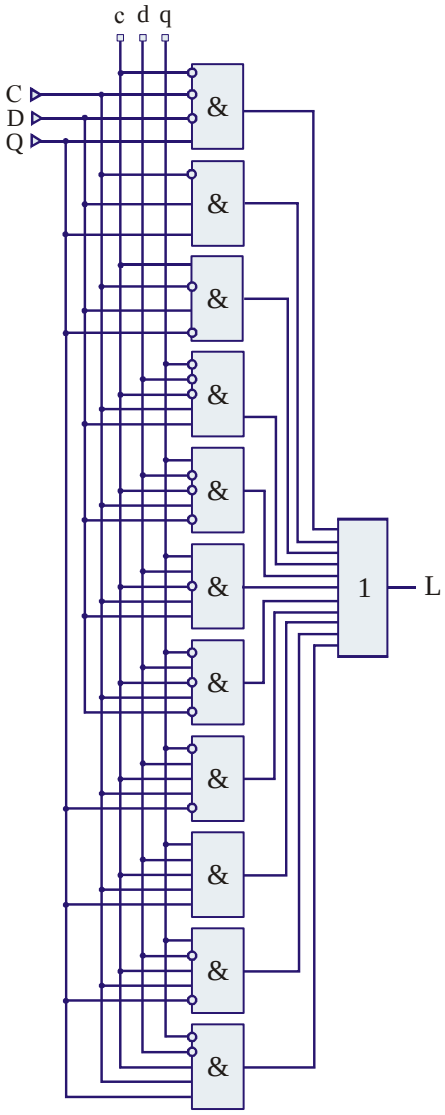


Рис. 3.22. Дедуктивна функція аналізу дефектів для D-тригера

З урахуванням введених визначень далі пропонується синтез дедуктивної функції для функціональності системного рівня, заданої у вигляді граф-схеми алгоритму на рис. 3.23.

Матриця переходів абстрактного автомата, що відповідає граф-схемі, наведеній на рис. 3.23, а також таблиця переходів-виходів структурного автомата з кодованими станами вхідних внутрішніх та вихідних змінних поуані в наступній таблиці:

$C =$

X	S_i	S_{i+1}	Y
–	S_0	S_1	$Y_0 : A = B + C$
X_1	S_1	S_2	$Y_1 : B = B + C$
$\overline{X_1}$	S_1	S_4	$Y_2 : A = \overline{A} + 1$
X_2	S_2	S_3	$Y_3 : C = \overline{C}$
$\overline{X_2}$	S_2	S_6	$Y_4 : C = A + B$
X_3	S_3	S_6	$Y_5 : A = \overline{C} + B$
$\overline{X_3}$	S_3	S_6	$Y_4 : C = A + B$
X_1	S_4	S_6	$Y_3 : C = \overline{C}$
$\overline{X_1}$	S_4	S_5	$Y_6 : B = \overline{B}$
X_3	S_5	S_6	–
$\overline{X_3}$	S_5	S_6	$Y_7 : A = A + \overline{B} + C$

$=$

X	S_i	S_{i+1}	Y
XXX	000	001	000
1XX	001	010	001
0XX	001	100	010
X1X	010	011	011
X0X	010	110	100
XX1	011	110	101
XX0	011	110	100
1XX	100	110	001
0XX	100	101	110
XX1	101	110	XXX
XX0	101	110	111

(3.35)

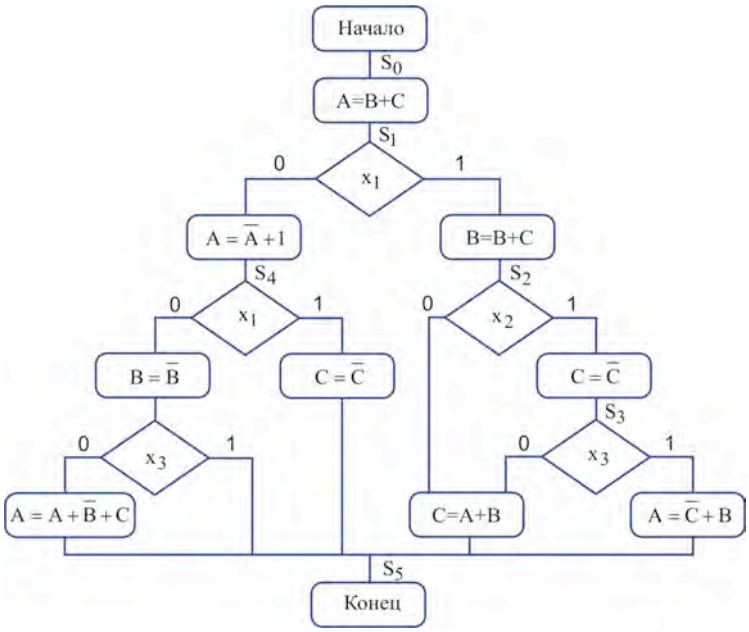


Рис. 3.23. Граф-схема функціональності

В даному випадку вхідними змінними вважаються вектори, що конкатенуються змінними ($X S_i$), вихідними лініями є - (S_i +1Y). Для побудови дедуктивної матриці, яка задає примітив моделювання всіх несправностей, що відповідають структурному автомату, необхідно побудувати таблицю істинності на множині рядків або кубів покриття. Вручну дана процедура важко реалізується. В комп'ютерному виконанні вона не має труднощів. Для одного вхідного вектора матриця дедуктивного аналізу дефектів, яка виходить в результаті виконання операції Xor між вхідною послідовністю і всіма координатами матриці справної поведінки, буде мати вигляд:

$T \oplus C = L \rightarrow (100100\ 110001) \oplus$

$\oplus$

X	S_i	S_{i+1}	Y
XXX	000	001	000
1XX	001	010	001
0XX	001	100	010
X1X	010	011	011
X0X	010	110	100
XX1	011	110	101
XX0	011	110	100
1XX	100	110	001
0XX	100	101	110
XX1	101	110	XXX
XX0	101	110	111

=

X	S_i	L_{i+1}	L_Y
XXX	100	111	001
0XX	101	100	000
1XX	101	010	011
X1X	110	101	010
X0X	110	000	101
XX1	111	000	100
XX0	111	000	101
0XX	000	000	000
1XX	000	011	111
XX1	001	000	XXX
XX0	001	000	110

(3.36)

Природно, що дана дедуктивна модель є структурою регістрового рівня, яку можна реалізовувати в кристалі FPGA, де для завдання функцій використовуються таблиці істинності безпосередньо. Проте, можлива схемна реалізація дедуктивних функцій (S_i + 1Y), записанних ув ДНФ за констітуентами одиниць у відповідному стовпці:

$$L^1_{i+1} = S^1_i\overline{S^2_i}\,\overline{S^3_i} \vee \overline{X_1}S^1_i\overline{S^2_i}S^3_i \vee X_2S^1_iS^2_i\overline{S^3_i};$$
$$L^2_{i+1} = S^1_i\overline{S^2_i}\,\overline{S^3_i} \vee X_1S^1_i\overline{S^2_i}S^3_i \vee X_1\overline{S^1_i}\,\overline{S^2_i}\,\overline{S^3_i}; \qquad (3.37)$$
$$L^3_{i+1} = S^1_i\overline{S^2_i}\,\overline{S^3_i} \vee X_2S^1_iS^2_i\overline{S^3_i} \vee X_1\overline{S^1_i}\,\overline{S^2_i}\,\overline{S^3_i}.$$

$$L^1_Y = \overline{X_2}S^1_iS^2_i\overline{S^3_i} \vee X_3S^1_iS^2_iS^3_i \vee \overline{X_3}S^1_iS^2_iS^3_i \vee X_1\overline{S^1_i}\,\overline{S^2_i}\,\overline{S^3_i} \vee \overline{X_3}\,\overline{S^1_i}\,\overline{S^2_i}\,\overline{S^3_i};$$
$$L^2_Y = X_1S^1_i\overline{S^2_i}S^3_i \vee X_2S^1_iS^2_i\overline{S^3_i} \vee X_1\overline{S^1_i}\,\overline{S^2_i}\,\overline{S^3_i} \vee \overline{X_3}\,\overline{S^1_i}\,\overline{S^2_i}S^3_i; \qquad (3.38)$$
$$L^3_Y = S^1_i\overline{S^2_i}\,\overline{S^3_i} \vee X_1S^1_i\overline{S^2_i}S^3_i \vee \overline{X_2}S^1_iS^2_i\overline{S^3_i} \vee \overline{X_3}S^1_iS^2_iS^3_i \vee X_1\overline{S^1_i}\,\overline{S^2_i}\,\overline{S^3_i}.$$

Вирази (3.37) і (3.38) визначають умови формування списків несправності за шостима виходами на тест-векторі (100100 110001). Навіть на одному векторі виходить досить складна цифрова схема, подана на рис. 3.24, апаратурні витрати якої за Квайном мають оцінку 42. Складніший результат у вигляді схеми має функція виходів, яка реалізується за допомогою 84 входів і 17 логічних елементів.

Таким чином, реалізація дедуктивної функції граф-схеми алгоритму (див. рис. 3.24) на одному вхідному наборі має обчислювальну складність, що дорівнює 84 + 42 = 126. Якщо таку комбінаційну схему мультиплікуювати на 2^{12} наборах, то в гіршому випадку апаратурні витрати призведуть до структури, яка визначається оцінкою:

$$Q = Q^t \times 2^{2\times(|X|+|S_i|)} = 126 \times 2^{12} = 516\ 096. \qquad (3.39)$$

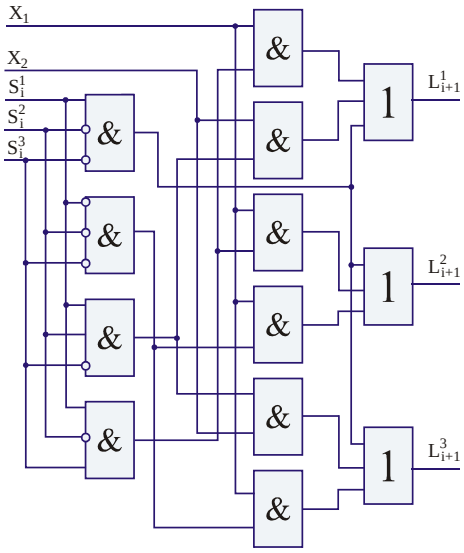


Рис. 3.24. Дедуктивна схема аналізу дефектів

Природно, що півмільйона вентилів є неприйнятною кількістю для моделювання несправностей, нехай навіть зі швидкістю, що перевищує у сотні разів програмний аналог. Виходом, в даному випадку, може служити гібридне рішення - програмно-апаратний комплекс моделювання несправностей, гнучкий за відношенням до тест-векторів. В цьому випадку програмно-орієнтована модель дедуктивного аналізу генерується у реальному масштабі часу, як функція від fault-free поведінки і тестових перевіряючих послідовностей $L = f(T, C)$. В даному випадку автоматна модель процесу аналізу дефектів, розгорнута ув часі (X, Z, Y - множина вхідних, внутрішніх та вихідних змінних відповідно), буде мати наступний вигляд:

$$M = \langle L, T, C, X, Z, Y \rangle,$$

$$\begin{cases} L_z^t = f(T_x^t, T_z^{t-1}, C); \\ L_y^t = f(T_x^t, T_z^{t-1}, C). \end{cases} \qquad (3.40)$$

Таким чином, технологія навіть апаратного вбудованого моделювання повертається в поле програмно-орієнтованих рішень. Справді, ринок електронних технологій у найближчі роки буде мати тенденцію до flexible reusable software solutions. Такий напрям має під собою підстави: 1. Реалізація системи на кристалі стає все більш програмно-орієнтованою, оскільки через 5 років пам'ять на кристалі буде складати 94% від його площі. 2. Для управління обчислювальними процесами, пов'язаними з моделюванням, необхідно мати на кристалі мікропроцесор, виконаний за гнучкою технологією і вбудований у пам'ять, або за жорсткою технологією - виконаний на кристалі.

3.12. Структурні моделі примітивів симулятора

У загальному випадку отримання дедуктивних примітивів для паралельного моделювання несправностей пов'язано з синтезом функцій на вичерпном тесті. Складність дедуктивних примітивів залежить від рівня представлення функціональностей. Найбільш простими є структури вентильного рівня у вигляді базису логічних елементів And, Or, Not.

За допомогою основного виразу (3.26) синтезу дедуктивних функцій, що що транспортують дефекти через логічний елемент, виконується побудова всіх базових компонентів (And, Or, Not) [14-17]:

$$
\begin{aligned}
&L_{And}[T=(00,01,10,11), F=(X_1 \wedge X_2)] = L\{(\bar{x}_1\bar{x}_2 \vee \bar{x}_1 x_2 \vee x_1\bar{x}_2 \vee x_1 x_2) \wedge \\
&\wedge [(X_1 \oplus T_{t1} \wedge X_2 \oplus T_{t2}) \oplus T_{t3})]\} = (\bar{x}_1\bar{x}_2)\{[(X_1 \oplus 0) \wedge (X_2 \oplus 0)] \oplus 0\} \vee \\
&\vee (\bar{x}_1 x_2)\{[(X_1 \oplus 0) \wedge (X_2 \oplus 1)] \oplus 0\} \vee (x_1\bar{x}_2)\{[(X_1 \oplus 1) \wedge (X_2 \oplus 0)] \oplus 0\} \vee \\
&\vee (x_1 x_2)\{[(X_1 \oplus 1) \wedge (X_2 \oplus 1)] \oplus 1\} = (\bar{x}_1\bar{x}_2)(X_1 \wedge X_2) \vee (\bar{x}_1 x_2)(X_1 \wedge \bar{X}_2) \vee \\
&\vee (x_1\bar{x}_2)(\bar{X}_1 \wedge X_2) \vee (x_1 x_2)(X_1 \vee X_2); \\
&L_{Or}[T=(00,01,10,11), F=(X_1 \vee X_2)] = (\bar{x}_1\bar{x}_2)(X_1 \vee X_2) \vee (\bar{x}_1 x_2)(\bar{X}_1 \wedge X_2) \vee \\
&\vee (x_1\bar{x}_2)(X_1 \wedge \bar{X}_2) \vee (x_1 x_2)(X_1 \wedge X_2); \\
&L_{Not}[T=(0,1), F=\bar{X}_1] = L\{(\bar{x}_1 \vee x_1)[(\overline{X_1 \oplus T_{t1}}) \oplus T_{t2}]\} = \bar{x}_1 \wedge \\
&\wedge [(\overline{X_1 \oplus 0}) \oplus 1] \vee x_1[(\overline{X_1 \oplus 1}) \oplus 0] = \bar{x}_1\bar{\bar{X}}_1 \vee x_1\bar{\bar{X}}_1 = \bar{x}_1 X_1 \vee x_1 X_1.
\end{aligned}
\tag{3.41}
$$

У даних рівняннях $T_t = (T_{t1}, T_{t2}, T_{t3}), (t = \overline{1,4})$ – тест-вектор, що має 3 координати, де остання из них визначає стан виходів елементів And, Or. Що стосується інвертора, то тут тест-вектор має 2 координати: $T_t = (T_{t1}, T_{t2}), (t = \overline{1,2})$, де остання координата є стан виходу елемента. Рівняння для інвертора ілюструє несуттєвість операції інверсії на виході елемента для транспортування дефектів. Тому дана функція (Not) не присутня на виходах дедуктивних примітивів. Апаратна реалізація дедуктивних функцій [14-15] для двовходових елементів (And, Or) на вичерпному тесті подана на рис.3.25 схемою дедуктивно-паралельного аналізу дефектів.

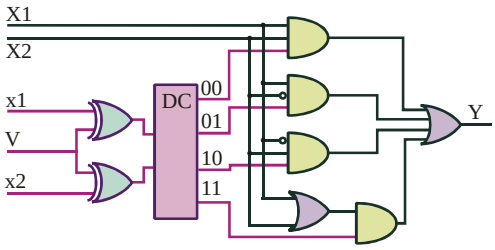


Рис. 3.25. Симулятор несправностей

В симуляторі подані булеві (x1, x2) і реєстрові (X1, X2) змінні, сигнал V вибору типу справної функції: V = 0 (And), V = 1 (Or), вихідна регістрова змінна Y. Стани двійкових входів x1, x2 і V формують одну з чотирьох де-

дуктивних функцій для отримання вектора несправностей Y, що перевіряються. Імплементація дедуктивної моделі в HDL-коді подана лістингом 3.1.

Лістинг 3.1. VHDL-модель секвенсора

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity Fub1 is
        port( i0, i1 : in STD_LOGIC;
                o00, o01, o10 , o11 : out STD_LOGIC);
end Fub1;
architecture Fub1 of Fub1 is
begin
        o00 <= not i0 and not i1;
        o01 <= not i0 and i1;
        o10 <= i0 and not i1;
        o11 <= i0 and i1;
end Fub1;

library IEEE;
use IEEE.std_logic_1164.all;
entity sequencer is
  port( V ,  X1_s, X2_s , x1, x2 : in STD_LOGIC;
      Y : out STD_LOGIC);
end sequencer;
architecture sequencer of sequencer is
component Fub1
        port( i0, i1 : in STD_LOGIC;
                o00, o01, o10 , o11 : out STD_LOGIC);
end component;
signal a0, a1, a2, a3, a4 : STD_LOGIC;
signal o00, o01, o10, o11 : STD_LOGIC;
signal x3, x4 : STD_LOGIC;
begin
        U1 : Fub1  port map(i0 => x3, i1 => x4, 00 => o00, o01 => o01, o10 => o10, o11 => o11 );

        a0 <= o00 and X2_s and X1_s;
        a1 <= not(X2_s) and o01 and X1_s;
        a2 <= not(X1_s) and X2_s and o10;
        a3 <= X2_s or X1_s;
        a4 <= o11 and a3;
        Y <= a4 or a2 or a1 or a0;
        x3 <= V xor x1;
        x4 <= x2 xor V;
end sequencer;
```

Робота симулятора демонструється в таблиці паралельного моделювання 8-розрядних вхідних векторів несправностей з метою отримання на Y-виході вектора перевіряюваних дефектів для логічних елементів And, Or:

(V, x1, x2) =	000	100	011	111	010	110
X1(RG)	01110001	01110001	10110110	00111011	00101010	10111001
X2(RG)	01111000	01111000	10110101	00110100	10111001	00101010
Y(RG)	01110000	01111001	10110111	00110000	10010001	10010001

Застосування такого симулятора дає можливість трансформувати вентильну модель F справної поведінки схеми в дедуктивну L, яка інваріантна у сенсі універсальності тестовим наборам і не передбачає в процесі моделювання використовувати модель F. Тому симулятор, як апаратна модель ДФ, орієнтований на створення вбудованих засобів дедуктівно-паралельного моделювання, що підвищують швидкодію аналізу в 10 - 1000 разів у порівнянні з програмною реалізацією. Але при цьому співвідношення обсягів післясинтезних моделей справного моделювання та аналізу несправностей становить 1:16. Апаратний аналіз несправностей спрямований на розширення функціональних можливостей вбудованих засобів справного моделювання (HESTM - Hardware Embedded Simulator) фірми Aldec (www.aldec.com). Обчислювальна складність обробки проекту, що складається з n вентилів, дорівнює $Q = (2n^2\tau)/W$, де τ - час виконання регістрової операції (And, Or, Not); W - розрядність регістра.

Для апаратної реалізації дедуктивно-паралельного моделювання на основі запропонованого симулятора може бути використана обчислювальна структура, що наведена на рис. 3.26 [13,38,39].

Особливість схемної реалізації полягає в спільному виконанні двох операцій: однобітових - для емуляції функцій логічних елементів And, Or і паралельної - для обробки багаторозрядних векторів несправностей шляхом виконання операцій логічного множення, заперечення і складання.

Функціональне призначення основних блоків (пам'ять і процесор): 1. $M = [M_{ij}]$ - квадратична матриця моделювання несправностей, де i, j = 1, q; q - загальне число ліній в оброблюваній схемі. 2. Вектори збереження станів справного моделювання, визначені в моменти часу t-1 і t, необхідні для формування дедуктивних функцій примітивів. 3. Модуль пам'яті для зберігання схемного опису у вигляді структури логічних елементів. 4. Буферні регістри, розмірністю q, для зберігання операндів і виконання регістрових паралельних операцій над векторами несправностей, що лічені з матриці M. 5. Блок справного моделювання для визначення двійкового стану виходу чергового обраблювального логічного елемента. 6. Дедуктивно-паралельний симулятор, що обробляє за один такт дві регістрові змінні X1, X2 з метою визначення вектора дефектів, що транспортуються на вихід логічного елементу Y.

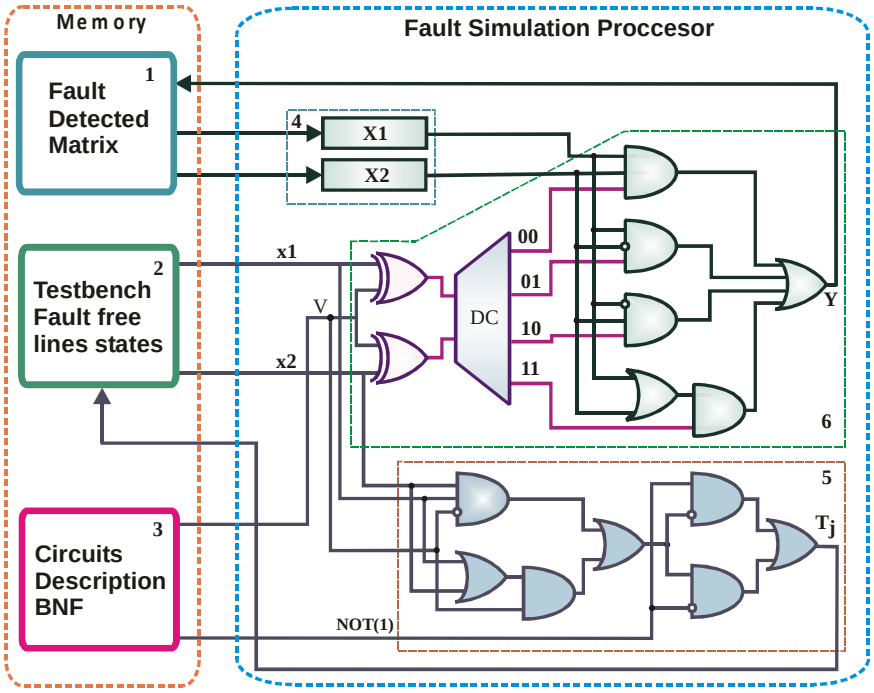


Рис. 3.26. HFS-структура апаратного моделювання

Перевага запропонованої структури моделювання несправностей. 1. Суттєве зменшення кількості модельованих дефектів, визначаються тільки числом збіжних розгалужень, яке становить до 20% від загального числа ліній. 2. Зниження обсягу пам'яті, необхідної для зберігання матриці модельованих несправностей. 3. Простота реалізації Hardware Fault Simulator (HFS) в апаратному виконанні, що дозволяє на порядок збільшити швидкодію моделювання несправностей. 4. Використання HFS як першої фази дедуктивно-топологічного метода, який грунтується на результаті обробки збіжних розгалужень для швидкодіючого аналізу деревоподібних структур.

Маршрут моделювання цифрових систем на кристалах [14] з попереднім розбиттям моделі пристрою на дві структурні частини (збіжні розгалуження і деревовидні підграфи), що наведено на рис.3.27.

Підсумки запропонованої технології моделювання з попереднім розбиттям схеми на збіжні розгалуження і деревовидні підграфи. Дедуктивно-паралельний аналіз дефектів на основі зворотного простежуювання несправностей вимагає практично лінійних витрат пам'яті і часу, що залежать від числа ліній схеми. Витрати часу для обробки збіжних розгалужень мають квадратичну залежність від їх числа:

$$Q = (r^2 / W) + n_r + n_p + (n - r - r^0).$$

Тут (r^2/W) – час моделювання несправностей r збіжних розгалужень, число яких визначається як $r = 0.2 \times n$; $n_r = n$ – час реконфігурування примітивів схеми на входному наборі; $n_p = n$ – час пошуку підграфів ліній, що відповідають неперевірюваним збіжним розгалуженням; $(n - r - r^0) = n - 0.2 \times n - 0.4 \times n = 0.4 \times n$ – час виконання суперпозиції розв'язків на множині ліній схеми без збіжних розгалужень та попередників для неперевірюваних збіжних розгалужень. Враховуючи значення параметрів функції від числа ліній, можна отримати оцінку швидкодії дедуктивно-паралельного методу [6,2,7]:

$$Q = [(0.2 \times n)^2 / W] + n + n + (n - 0.2 \times n - 0.4 \times n) = [(0.2 \times n)^2 / W] + 2.4 \times n).$$

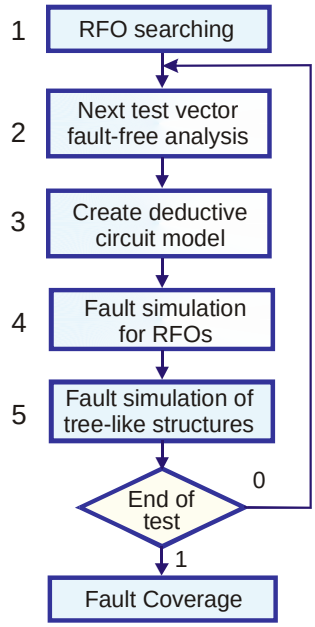


Рис. 3.27. Модель процесу дедуктивно-паралельного моделювання

Таким чином, вигриш у швидкодії запропонованого методу тим більше, чим менше процент збіжних розгалужень у схемі цифрового пристрою [15,16,28]. Для порівняння: паралельний алгоритм має обчислювальну складність C_p, що визначається функціональною залежністю від числа нееквівалентних несправностей (b), довжини комп'ютерного слова (W), кількості еквівалентних вентилів (G): $C_p = (b^2/W) \times G^3$. Дедуктивний алгоритм має відміни у формулі оцінки швидкодії: $C_d = b^2 \times Q \times G^2 \big|_{Q=G} = b^2 G^3$, де Q – середнє число

активізованих несправностями вентилів. Дедуктивно-паралельний метод без розбиття схеми має швидкодію, що визначається виразом: $C_{dp} = G^2 + (b^2 / W) \times G^2$. Перший доданок задає час справного моделювання, другий – час аналізу несправностей цифрового пристрою, лінії якого не ранжовані. Для комбінаційної ранжованої схеми швидкодія методу має оцінку $C^r_{dp} = G + (b^2 / W) \times G$. Швидкодія дедуктивно-паралельного методу выще паралельного і дедуктивного $(C^r_{dp} << \{C_p, C_d\})$, завдяки розподілу фаз справного та несправного моделювань.

Запропонована технологія програмно-апаратного дедуктивно-паралельного моделювання несправностей орієнтована на створення моделей дедуктивних примітивів вентильного, регістрового та системного рівнів з метою тестування цифрових систем на кристалах, що містять мільйони вентилів. Представлена структурна модель апаратного симулятора і пристрої моделювання в цілому, які орієнтовані на суттєве підвищення швидкодії засобів моделювання цифрових виробів великої розмірності шляхом розподілу функцій справного аналізу та обчислення списків перевірюваних дефектів на вхідних наборах.

На основі описаної вище технології створена система SIGETEST - (SImulation, GEneration of TEST) [14,15,39,40] швидкодіюча система моделювання несправностей і генерації тестів, що використовує моделі проектованих цифрових систем інтерпретативно-компілятивного типу. Об'єктом моделювання може виступати будь-яка цифрова структура у вигляді булевих рівнянь, що реалізуються в кристалах CPLD, FPGA, ASIC. Система обробляє складні цифрові проекти, які нараховують сотні тисяч логічних вентилів на стадії після синтезу (gate level description). Система має інтегроване середовище, що реалізовує графічний інтерфейс високого рівня. Введення проектів здійснюється у вигляді опису. Підтримувані операції: AND, OR, NOT, XOR. Підтримуються також шинні структури. Компілятор перетворює схемний опис до вигляду внутрішніх структур даних, зручних для моделювання. Ядро моделювання включає алгоритми справного і несправного моделювання: Parallel, Backtraced Quasi Exact, Deductive-Parallel і Backtraced-Deductive-Parallel. Генератор тестів включає набір алгоритмів (псевдовипадкових, детермінованих, алгоритмічних) для синтезу тестових послідовностей. Результатом роботи програми є Testbench в форматі VHDL. Система надає також інформацію про процеси справного і несправного моделювання, якість покриття несправностей, статистику моделювання. Результати моделювання можна переглянути за допомогою вікна Fault Coverage, що представляє собою багатозначну таблицю несправностей.

SIGETEST має засоби для управління та моніторингу процесу синтезу тестів. Моделювання можна обмежити за часом або задати число тестових наборів, які необхідно промоделювати. Мається можливість обмежити знизу

відсоток покриття несправностей генерованими наборами. В процесі моделювання система надає інформацію про моделювання у відсотках від загального числа векторів або наперед заданого часового інтервалу. Вона орієнтована на інтеграцію з сучасними засобами синтезу і моделювання, такими як Aldec Active-HDL, Riviera, Synopsis Design Compiler.

3.13. Висновки

1. Запропоновано узагальнену модель дедуктивно-паралельного аналізу цифрових систем, орієнтовану на аналіз дефектів вентильного регістрового та функціонального рівнів опису об'єктів.

2. Подана внутрішня модель і структури даних для аналізу цифрового пристрою з метою моделювання несправностей, заснована на паралельному аналізі справної поведінки і всіх можливих дефектів заданого класу.

3. Описано алгоритми структурно-функціонального аналізу цифрових систем з метою пошуку множини збіжних розгалужень (ЗР) і визначення схемних деревовидних структур (ДС), що дозволяє істотно, в десятки разів, підвищити швидкодію моделювання несправностей.

4. Наведено метод моделювання несправностей, заснований на окремій обробці ЗР і ДС, орієнтований на аналіз цифрових систем великої розмірності і маючий істотно менше число ітерацій для визначення несправностей в ациклічних структурах.

5. Запропоновано алгоритмічну реалізацію методу дедуктивно-паралельного моделювання на основі реконфігурування моделі пристрою у процесі аналізу несправностей та застосування процедур зворотнього простеження з метою істотного зменшення часу, необхідного для побудови таблиці несправностей та оцінки якості або повноти тестів.

3.14. Література

1. Wang X., Hill F.G., Mi Zh. A sequential circuit fault simulation by surrogate fault propagation // Proc. 1989 IEEE International test conference, IEEE Computer society.– 1989.– P. 9-18.

2. Nishida T., Miyamoto S., Kozawa T., Satoh K. RFSIM: Reduced fault simulator // IEEE Transactions on computer-aided design.– 1987.– Vol. CAD-6, No 3.– P. 392-402.

3. Hahanov V.I., Babich A.V., Hyduke S.M. Test Generation and Fault Simulation Methods on the Basis of Cubic Algebra for Digital Devices // Proceedings of the Euromicro Symposium on Digital Systems Design DSD2001.– Warsaw, Poland.– September, 4-6, 2001.– P. 228-235.

4. Хаханов В.И., Хак Х.М. Джахирул, Масуд М.Д. Мехеди. Модели анализа неисправностей цифровых систем на основе FPGA, CPLD // Технология и конструирование в электронной аппаратуре.– 2001.– № 2.– С. 3-11.

5. Levendel Y.H., Menon P.R. Comparison of fault simulation methods – Treatment of unknown signal values // Journal of digital systems.– 1980.– Vol. 4.– P. 443-459.

6. Abramovici M., Breuer M.A. and Friedman A.D. Digital systems testing and testable design.– Computer Science Press.– 1998.– 652 p.

7. Убар Р.Р. Анализ диагностических тестов для комбинационных цифровых схем методом обратного прослеживания неисправностей // Автоматика и телемеханика.– 1977.– №8.– С.168-176.

8. Автоматизированное проектирование цифровых устройств / С.С.Бадулин, Ю.М.Барнаулов и др./ Под ред. С.С. Бадулина.– М.: Радио и связь.– 1981.– 240 с.

9. Active-HDL User's Guide. Second Edition. Copyright. Aldec Inc.– 1999.– 213p.

10. Горбатов В.А. Основы дискретной математики.– М.: Высш. шк.– 1986.– 311 с.

11. Ghosh Sumit. Behavioral-Level Fault Simulation // IEEE Design and Test of Computers.–1988.– P. 31-42.

12. Hahanov V., Kteaman H., Ghribi W., Fomina E. HEDEFS – Hardware embedded deductive fault simulation // Proceedings of the 3rd IFAC Workshop.– Rydzyna, Poland.– 2006.– P. 25-29.

13. Hahanov V., Hahanova I., Obrizan V. High-performance deductive fault simulation method // Proceedings of the 10 IEEE European test symposium.– Tallinn, Estonia.–2006.– P. 91-96.

14. Hahanov V.I., Hahanova I.V., Khan S.U., Obrizan V.I. Topological fault simulation method // Proceedings of the 11th International Conference Mixdes Design of Integrated Circuits and Systems.– Szczecin.– 2004.– P. 211-214.

15. Hahanov V.I., Babich A.V., Hyduke S.M. Test Generation and Fault Simulation Methods on the Basis of Cubic Algebra for Digital Devices // Proceedings of the Euromicro Symposium on Digital Systems Design DSD 2001.– Warsaw, Poland.– 2001.– P. 228-235.

16. Hansen M.C., Yalcin H., Haye, J.P. Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering // Design & Test of Computers, IEEE 16.3.– 1999.– P. 72 – 80.

17. Harris Ian G. Fault Models and Test Generation for Hardware-Software Covalidation // IEEE Design and Test of Computers.– 2003.– P. 40-47.

18. Jain Sunil, Stroud Charles. Built-in Self Testing of Embedded Memories // IEEE Design and Test of Computers.– 1986.– P. 27-37.

19. Khoche Ajay, Sherlekar S.D., Venkatesh G., Venkateswaran R. A Behavioral Fault Simulator for Ideal // IEEE Design and Test of Computers.–1992.– P. 14-21.

20. Menon P.R., Chappell S.G. Deductive Fault Simulation with Functional Blocks // IEEE Transactions on Computers.– 1978.– P. 689-695.

21. Niggemeyer Dirk, Rudnick Elizabeth M. Automatic Generation of Diagnostic Memory Tests Based on Fault Decomposition and Output Tracing // IEEE Transactions on Computers.– 2004.– P. 1134-1146.

22. Novak Ondrey, Gramatova Elena, Ubar Raimund. Handbook of testing electronic systems.– Czech Technical University Publishing House.–2005.–402 p.

23. Ozguner F. Deductive Fault Simulation of Internal Faults of Inverter-Free Circuits and Programmable Logic Arrays // IEEE Transactions on Computers.– 1986.– P. 70-73.

24. Pomeranz Irith, Reddy Sudhakar M. On Fault Simulation for Synchronous Sequential Circuits // IEEE Transactions on Computers.– 1995.– P. 335-340.

25. Pomeranz Irith, Reddy Sudhakar M. Aliasing Computation Using Fault Simulation with Fault Dropping // IEEE Transactions on Computers.– 1995.– P. 139-144.

26. Pomeranz Irith, Reddy Sudhakar M. On Maximizing the Fault Coverage for a Given Test Length Limit in a Synchronous Sequential Circuit // IEEE Transactions on Computers.– 2004.– P. 1121-1133.

27. Rossen K. Discrete Mathematics and its Applications.– McGraw Hill.– 2003.– 824 p.

28. Silberman G.M., Spillinger I. Functional Fault Simulation as a Guide for Biased-Random Test Pattern Generation // IEEE Transactions on Computers.– 1991.– P. 66-79.

29. Waicukauski John, Lindbloom Eric, Rosen Barry, Iyengar Vijay. Transition Fault Simulation // IEEE Design and Test of Computers.– 1987.– P. 32-38.

30. Walczak K. Deductive Fault Simulation for Sequential Module Circuits // IEEE Transactions on Computers.– 1988.– P. 237-239.

31. Автоматизация диагностирования электронных устройств/ Ю.В.Малышенко и др./ Под ред.В.П.Чипулиса.– М.: Энергоатомиздат.– 1986.– 216 с.

32. Бондаренко М.Ф., Кривуля Г.Ф., Рябцев В.Г., Фрадков С.А., Хаханов В.И. Проектирование и диагностика компьютерных систем и сетей.– К.: НМЦ ВО.– 2000.– 306 с.

33. Пархоменко П.П., Согомонян Е.С. Основы технической диагностики (Оптимизация алгоритмов диагностирования, аппаратурные средства) / Под ред. П.П. Пархоменко.– М.: Энергия.– 1981.– 320 с.

34. Парфентий А.Н., Хаханов В.И., Литвинова Е.И. Модели инфраструктуры сервисного обслуживания цифровых систем на кристаллах // АСУ и приборы автоматики.– 2007.– №138.– С. 83-99.

35. Семенец В.В., Хаханова И.В., Хаханов В.И. Проектирование цифровых систем с использованием языка VHDL.– Харьков: ХНУРЭ.– 2003.– 492 с.

36. Хаханов В.И., Хаханова И.В. VHDL + Verilog = Синтез за минуты.– Харьков: СМИТ.– 2007.– 264 с.

37. Хаханов В.И. Техническая диагностика элементов и узлов персональных компьютеров.– К.: ІЗМН.– 1997.– 308 с.

38. Хаханов В.И., Сысенко И.Ю., Хак Х.М. Джахирул, Масуд М.Д. Мехеди. Кубическое моделирование неисправностей цифровых проектов на основе FPGA, CPLD // Радиоэлектроника, информатика, управление.– 2001.– №1.– С. 123-129.

39. Хаханов В.И., Сысенко И.Ю., Колесников К.В. Дедуктивно-параллельный метод моделирования неисправностей на реконфигурируемых моделях цифровых систем // Радиоэлектроника и информатика.– 2002.– №1.– С. 98-105.

40. Хаханов В.И., Колесников К.В., Хаханова А.В. BDP-метод моделирования неисправностей для синтеза тестов цифровых проектов // Радиоэлектроника и информатика.– 2002.– №2.– С. 60-66.

4. ІНФРАСТРУКТУРА ДІАГНОСТИЧНОГО ОБСЛУГОВУВАННЯ SOC

Запропоновано моделі та методи створення інфраструктури сервісного обслуговування (I-IP) функціональностей SoC, яка відрізняється мінімальним набором процесів вбудованого діагностування у реальному масштабі часу. I-IP дає можливість здійснювати сервіси: моделювання дефектів функціональних блоків для оцінки якості тестів і побудови таблиці несправностей; діагностування із заданою глибиною пошуку дефектів в SoC; відновлення працездатності вбудованої пам'яті на основі використання запасних компонентів. Подано швидкодіючий дедуктивно-паралельний метод аналізу несправностей для побудови таблиць несправностей і оцінки якості тестів. Описано алгеброгічні методи вбудованого діагностування дефектів і відновлення працездатності шляхом синтезу ДНФ, що формуює всі розв’язки щодо встановлення діагнозу функціональностей SoC у реальному масштабі часу.

4.1. Інфраструктура I-IP

Обчислювальна і апаратна складність сучасних цифрових систем на кристалах (System-on-Chip - SoC) характеризується мільйонами еквівалентлентних вентилів і вимагає створення і впровадження нових високорівневих технологій проектування - Electronic System Level (ESL) Design, моде-линга - Transaction Level Modeling (TLM) [6,11,12] і вбудованого сервісного обслуговування - Infrastructure Intellectual Property (I-IP) [5,9-12,39,49,50,53]. Це означає, що пошук швидкодіючих методів і засобів тестування [1,6,11,12,19,29,34,35,54,57,61,62] наводить всіх дослідників до необхідності підвищення рівня абстракції моделей створюваних функціональностей - Functional Intellectual Property (F-IP), що вбудовуються в кристал. Ринок програмних продуктів EDA вже пропонує інструменти для автоматизиції процесів тестування, моделювання та верифікації [8,13-17,20,24-26,28,30-33,36,42-44,54-67] пристроїв системного рівня, починаючи з компіляторів HDL-мов (C ++, SystemC, SystemVerilog, UML, SDL) [1,6,11,34,61] і закінчуючи графічними оболонками (Simulink, LabView, Xilinx EDK). Дані кошти дозволяють створювати проекти з існуючих бібліотечних компонентів шляхом використання ESL-меппінга і створення TLM-інтерфейсів [11]. Ринкова привабливість імплементації цифрової системи в кристал FPGA визначається: застосуванням порівняно дешевих чіпів замість універсальних процесорів, малої споживаної потужності, габаритними розмірами, якісним і надійним виконанням основних функцій, завдяки I-IP-інфраструктурі [49], вбудованої в SoC, що є актуальним у століття мобільних обчислювальних пристроїв.

Проблема моделювання несправностей, діагностування [2-4, 7, 22, 27, 40, 41, 47, 48, 54-59] компонентів цифрової системи і ремонту пам'яті

[18,21,23,37,38,45,46 , 51,52] пов'язана з тенденцією на постійне зменшення площі кристала, що відводиться для оригінальної і стандартизованої логіки з одночасним збільшенням вбудованої пам'яті. Як показано на рис. 4.1, збільшення питомої ваги пам'яті на кристалі призводить до її повного домінування для зберігання даних і програм, яке до 2014 року досягне 94% [9,10,49,50,53]. Це забезпечить не тільки високу швидкодію виконання функціональності, але і гнучкість, що притаманна програмному продукту у частині корекції помилок проектування. Особливістю елементів пам'яті є той факт, що в процесі їх виготовлення та експлуатації окремі осередки під впливом несправностей можуть виходити зі стану працездатності. Дана обставина не завжди призводить матрицю пам'яті до критичного стану, коли відновлення працездатності неможливо. Тому далі розглядається такий технічний стан пам'яті, коли сумарна кількість дефектних осередків не перевищує резервних можливостей виробу, призначених для ремонту.

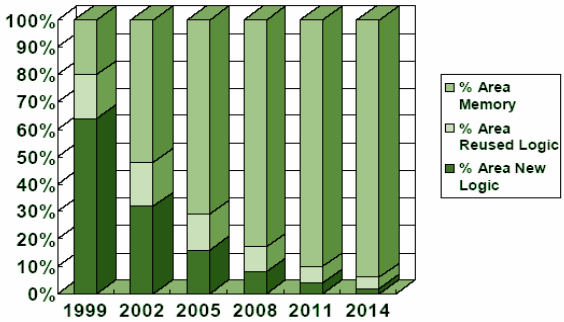


Рис. 4.1. Питома вага пам'яті на кристалі SoC

Мета дослідження - розробка технології вбудованого сервісного обслуживания функціональності цифрової системи на кристалі, призначеної для моделювання несправностей, діагностування компонентів SoC та ремонту вбудованої матричної пам'яті в реальному масштабі часу.

Завдання: 1) Стан ринку технологій сервісного обслуговування SoC [5,9-12,39,49,50,53]. 2) Дедуктивно-паралельне моделювання несправностей [14-17,20,28,30-33,36,42-44,54-67]. 3) Алгебрологічний (AL) метод вбудованого сервісного обслуговування на основі матриці покриття [58]. 4) Застосування AL-методу для діагностування компонентів SoC. 5) Адаптація AL-методу для ремонту вбудованої пам'яті. 6) Практичні результати дослідження.

Сучасні технології проектування цифрових систем на кристалах пропонують, поряд зі створенням функціональних блоків F-IP, розробку сервісних модулів I-IP, орієнтованих на комплексне вирішення проблеми якості проекту і підвищення виходу придатної продукції (Yield) в пропроцесі виробництва, яке визначається впровадженням в кристал наступних сервісів [9-12,39,49,50,53]:

1) Спостереження за станом внутрішніх і вихідних ліній в процесі функціонування, верифікації і тестування штатних блоків на основі використання стандарту граничного сканування IEEE 1500 [12].

2) Тестування функціональних модулів шляхом подачі перевіряючих наборів від різних тестових генераторів, орієнтованих на перевірку дефектів або справної поведінки.

3) Діагностування відмов і дефектів шляхом аналізу інформації, отриманої на стадії тестування та використання спеціальних методів вбудованого пошуку несправностей на основі стандарту IEEE 1500 [12].

4) Відновлення працездатності функціональних модулів і пам'яті після фіксації негативного результату тестування і визначення місця і виду дефекту при виконанні фази діагностування.

5) Вимірювання основних характеристик і параметрів функціонування виробу на основі вбудованих засобів, що дозволяють виробляти часові і вольт-амперні виміри.

6) Надійність і відмовостійкість функціонування виробу в процесі експлуатації, яка досягається диверсифікацією функціональних блоків, їх дублюванням і відновленням працездатності SoC у реальному масштабі часу.

На рис. 4.2 представлена урізана структура [49-50,53], орієнтована на виконання завдань: 1 Тестування функціональностей на основі генеруємих вхідних послідовностей (Automated Test Pattern Generator) і аналіз вихідних реакцій. 2. Моделювання (Fault Simulator) несправностей [13-17] з метою забезпечення діагностування та ремонту на основі таблиці несправностей (Fault Detection Table). 3. Діагностування дефектів із заданою глибиною шляхом використання мультизонда стандарту IEEE 1500. 4. Вбудований ремонт матричної пам'яті на основі використання запасних компонентів (spare) [18,21,37,38,51,52]. Перші два пункти далі розглядаються концептуально, а другі два - формально становлять сутність запропонованого дослідження.

Модуль синтезу тестів (див. рис. 4.2) для перевірки функціональностей і одиночних несправностей складається з набору генераторів вхідних послідовностей, які забезпечують створення наступних тестів [1,6,11,12,19,29,34,35,54,57 , 61,62]: 1. PRTG - псевдовипадковий генератор вхідних стимулів з рівномірним законом розподілу нульових і одиничних сигналів за вхідними змінними. 2. SATG - тестовий генератор шістнадцяткових кодів на основі сигнатурного аналізу. 3. SPTG - алгоритмічний генератор тестових векторів, які активізують одномірні логічні шляхи, орієнтовані на перевірку заданих одиночних несправностей. 4. ADTG - тестовий генератор, призначений для перевірки суматорних схем АЛУ. 5. BSTG - тестовий генератор для шинних структур прийому і передачі даних. 6. METG - генератор тестів, орієнтований на перевірку матричної пам'яті. 7.

DFTG - синтезатор тестів для автоматів, заданих у вигляді граф-схем алгоритмів. 8. RCTG - тестовий генератор для послідовних лічильно-реєстрових структур і тригерних схем.

Модуль ATPG аналізує структурно-функціональну модель блоку, що підлягає тестуванню, і призначає підмножину таких синтезаторів тестів, які забезпечують задану якість покриття дефектів (Fc) і функціональних режимів (Pc):

$$F^c\left(\bigcup_{i=1}^{n_{min}} T_i\right) \geq F^c_{min};\ P^c\left(\bigcup_{i=1}^{n_{min}} T_i\right) \geq P^c_{min},$$
$$T = \{T_1^{PR}, T_2^{SA}, T_3^{SP}, T_4^{AD}, T_5^{BS}, T_6^{ME}, T_7^{DF}, T_8^{RC}\}. \quad (4.1)$$

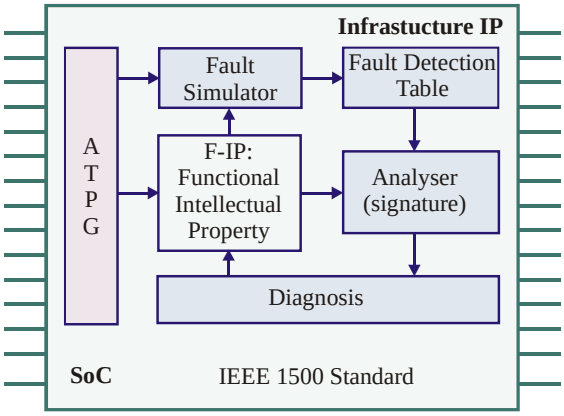


Рис. 4.2. Інфраструктура сервисів SoC DSP

Узагальнена структура синтезу Testbench [6], зображена на рис. 4.3, включає також генератор HDL-коду, який призначений для тестування та верифікації функціональностей на стадії розробки проекту.

Кількість тестових генераторів на стадії проектування SoC може бути істотно більше, ніж підмножина, яка далі вбудовується в кристал. Тому на стадії моделювання та верифікації виконується аналіз покриваючих властивостей кожного тест-генератора з метою пошуку їх мінімальної сукупної конфігурації, яка задовольняє виразу (4.1). Важливо відзначити, що в найближчі 5 років ідеологія синтезу тестів для цифрових систем на кристалах буде запозичувати кращі традиції, які від ESL-, TLM-проектування [11].

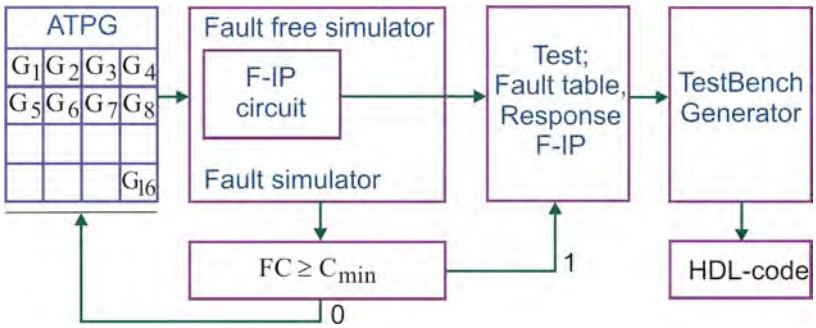


Рис. 4.3. Структура процесу синтезу Testbench для F-IP

Це означає: 1) Використання бібліотек тестів (Testbench) провідних компаній планети для тестування і верифікації стандартизованих функціональностей, позначених як F-IP. 2) Застосування стандартних рішень сервісного обслуговування I-IP для вбудованого тестування компонентов системи на кристалі. 3) Створення власних бібліотек тестів для функциональностей, що розробляються. 4) Впровадження нової технології синтезу тестів для цифрової системи, заснованої на дискретному мепінгу [11] (рис. 4.4) покриття функціональностей і дефектів вихідної специфікації за допомогою мінімальної сукупності Testbench, з бібліотеки тестів. 5) Застосування вбудованих засобів тестопридатності, таких як IEEE boundary scan - засоби граничного сканування і шість компонентів I-IP для підвищення технологічності процедур синтезу тестів.

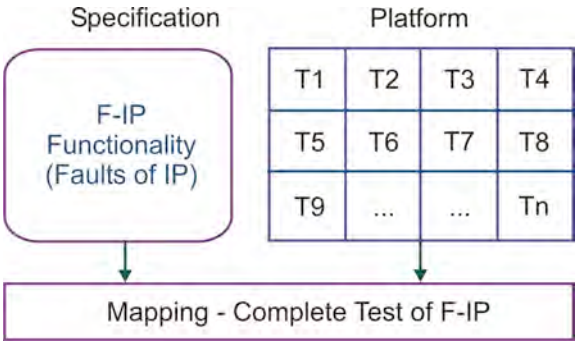


Рис. 4.4. Mapping моделі синтезу тестов для F-IP

4.2. Алгебрологічний метод діагностування дефектів

Основна роль відводиться технології граничного сканування [12], яка у теперішній час, що впроваджена в кристал, покликана полегшити вирішення практично всіх завдань сервісного обслуговування функціональних модулів системи на кристалі. Контролер доступу до внутрішніх ліній і портів регістра граничного сканування використовує осередок або розряд регістра. У сукупності число таких осередків, що забезпечують в даному випадку

моніторинг, має дорівнювати кількості проблемних спостережуваних ліній проекту, які необхідні для точного встановлення діагнозу.

Структура системи сервісного обслуговування I-IP для діагностування дефектів у функціональних блоках F-IP зображено на рис. 4.5.

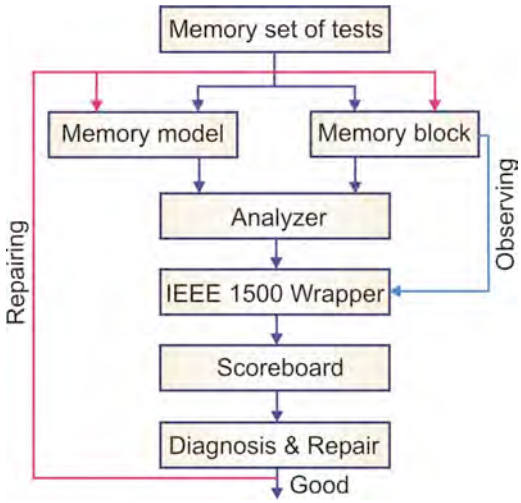


Рис. 4.5. Модель процесу діагностування блоків пам'яті

Модуль (Analyser) аналізує вихідні реакції моделі Memory Model і реального пристрою Memory Block на тестові вектори, що надходять від генератора вхідних послідовностей. IEEE 1500 Wrapper є мультізондом, призначеним для встановлення точного діагнозу. Scoreboard виконує функцію аналізу результатів діагностування з метою подальшого ремонту компонентів SoC. В даному випадку встановлюється діагноз з використанням вектора експериментальної перевірки і ТН ($M = [M_{tr}], t = \overline{1,p}; r = \overline{1,q+n}$ розмірністю $p \times n$, де p - число тест-векторів, n - кількість розрядів IEEE 1500 Wrapper boundary scan регістра). Результат являє собою множину несправних ліній і елементів на поточному вхідному наборі. Для організації обчислювальних процесів, що призводять до точного діагнозу, надзвичайно важлива метрика або форма подання вихідної інформації.

Цікаве рішення задачі діагностування може бути отримано шляхом застосування булевої алгебри і таблиці несправностей M, що уявляє собою декартів добуток тесту T на множину заданих дефектів F, у сукупності з вектором експериментальної перевірки V, де ворзв'язок задачі покриття дає максимально точний результат у вигляді ДНФ, а кожен терм є можливий варіант наявності в пристрої дефектів. Отже, модель процесу діагностування подана компонентами:

$$
\begin{aligned}
&A = \langle T, F, M, V \rangle, \\
&T = (T_1, T_2, \ldots, T_i, \ldots, T_n);\ F = (F_1, F_2, \ldots, F_j, \ldots, F_m); \\
&M = \left| M_{ij} \right|, i = \overline{1,n};\ j = \overline{1,m};\ V = (V_1, V_2, \ldots, V_i, \ldots, V_n); \\
&V_i = R(T_i) \oplus R^*(T_i);\ \{V_i, T_i, M_{ij}, F_j\} \in \{0,1\}.
\end{aligned} \tag{4.2}
$$

Значення координати вектора V є результат виконання операції Xor над узагальненою модельною та фактичною реакціями виходів пристрою.
Рішення завдання діагностування зводиться до аналізу таблиці несправнестей, отриманої в результаті моделювання дефектів, шляхом подальшго запису логічного добутку диз'юнкцій, записаних за одиничним значенням рядків таблиці несправностей:

$$
F = \bigwedge_{\forall V_i = 1}^{i=\overline{1,n}} \Big(\bigvee_{\forall M_{ij} = 1}^{j=\overline{1,m}} F_j \Big). \tag{4.3}
$$

Кон'юнктивна нормальна форма (КНФ), отримана з ТН, трансформується до діз'юнктівної нормальної форми (ДНФ) за допомогою еквівалентних перетворень (логічний добуток, мінімізація і поглинання) [4]. В результаті виходить булева функція, де терми - логічні добутки є повна множина рішень у вигляді сполучення дефектів (що дають за виходами SoC або її компоненту - вектор експериментальної перевірки V):

$$
F = \bigwedge_{\forall V_i = 1}^{i=\overline{1,n}} \Big(\bigvee_{\forall M_{ij} = 1}^{j=\overline{1,m}} F_j \Big) = \left| \begin{bmatrix} a \vee ab = b \\ a \vee a = a \end{bmatrix} \right. = \bigvee_{i=1}^{2^m} \Big(\bigwedge_{j=1}^{m} k_j F_j \Big), k_j = \{0,1\}. \tag{4.4}
$$

Функція (4.4) у загальному випадку формує діагноз у вигляді деякої підмножини комбінацій (кратних) дефектів, які далі потребують уточнення шляхом застосування додаткового зондування внутрішніх точок за допомогою регістра граничного сканування. Кількість одиниць у векторі експериментальної перевірки V формує число діз'юнктівних термів КНФ (4.4). Кожен терм - порядкоий запис дефектів (через логічну операцію АБО), що впливають на виходи функціональності. Подання таблиці у вигляді аналітичного запису - кон'юнктівної нормальної форми дає потенційну можливість істотно скоротити обсяг діагностичної інформації для пошуку дефектів. Подальше перетворення КНФ до ДНФ на основі тотожностей алгебри логіки дозволяє істотно зменшити булеву функцію.

Алгебрологічний метод розглядається на прикладі наступної таблиці несправностей М1 і поданий у вигляді п'яти пунктів алгоритму.

$M_1 =$

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	V
T_1	1			1			1
T_2		1			1		1
T_3			1	1		1	1
T_4	1		1				1
T_5		1			1	1	0

1. Визначення всіх рядків таблиці несправностей, відповідних нульовим значенням вектора експериментальної перевірки з метою обнулення всіх одиничних координат знайдених рядків. В даному випадку - це один рядок T5.

2. Знаходження всіх стовпців, які мають нульові значення координат рядків з нульовим станом ВЕП. Обнулення одиничних значень знайдених стовпців. В даному випадку: F2, F5, F6.

3. Видалення з таблиці несправностей рядків і стовпців, що мають тільки нульові значення координат (знайдені в пунктах 1 і 2).

$M_1 =$

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	V
T_1	1			1			1
T_2		0			0		1
T_3			1	1		0	1
T_4	1		1				1
T_5		0			0	0	0

=

T_i / F_j	F_1	F_3	F_4	V
T_1	1		1	1
T_3		1	1	1
T_4	1	1		1

4. Побудова КНФ за одиничним значенням ВЕП:

$$F = (F_1 \vee F_4) \wedge (F_3 \vee F_4) \wedge (F_1 \vee F_3) = (F_1F_3 \vee F_3F_4 \vee F_1F_4 \vee F_4F_4) \wedge (F_1 \vee F_3) =$$
$$= F_1F_1F_3 \vee F_1F_3F_4 \vee F_1F_1F_4 \vee F_1F_4F_4 \vee F_1F_3F_3 \vee F_3F_3F_4 \vee F_1F_3F_4 \vee F_3F_4F_4 =$$
$$= F_1F_3 \vee F_1F_3F_4 \vee F_1F_4 \vee F_3F_4 \vee F_1F_3F_4 \vee F_3F_4 = F_1F_3 \vee F_1F_4 \vee F_3F_4.$$

5. Перетворення КНФ до ДНФ з подальшою мінімізацією функції. В даному випадку це призводить до отримання шуканого результату у вигляді сполучення несправностей:

$$F = F_1F_3 \vee F_1F_4 \vee F_3F_4.$$

Запропонований алгоритм орієнтований на аналіз таблиці несправностей з метою зменшення її обсягу і наступних обчислень, пов'язаних з побудовою ДНФ, яка формує всі розв’язки щодо встановлення діагнозу функціональностей SoC. Подальше уточнення діагнозу можливе тільки з застосуванням мультизонду на основі регістра граничного сканування даних [15].

Приклад 4.1. Виконати алгебрологічний аналіз таблиці несправностей $M = T \times F$, що містить 10 дефектів. Тест, довжиною 11 вхідних наборів, перевіряє всі несправності. Вектор експериментальної перевірки цифрового

пристрою V = (10001001001), отриманий при виконанні діагностичного експерименту, фіксує неспівпадання виходів пристрою порівняно із золотим еталоном на чотирьох (1,5,8 і 11) тестових наборах.

$M = T \times F =$

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	V
T_1				1						1	1
T_2		1					1				0
T_3			1			1			1		0
T_4	1										0
T_5					1			1			1
T_6	1	1									0
T_7			1								0
T_8				1							1
T_9					1	1					0
T_{10}							1				0
T_{11}								1	1	1	1

В соответствии с числом единиц в векторе экспериментальной проверки V формируется количество дизъюнктивных термов КНФ, равное 4. Каждый терм есть построчная запись дефектов через логическую операцию ИЛИ, оказывающих влияние на искажение выходных сигналов функциональности. Далее осуществляется преобразование КНФ к ДНФ на основе правил алгебры логики, что дает возможность получить результат:

$$\begin{aligned} F &= (F_4 \vee F_{10})(F_5 \vee F_8)(F_4)(F_8 \vee F_9 \vee F_{10}) = (F_5 \vee F_8)(F_4)(F_8 \vee F_9 \vee F_{10}) = \\ &= (F_4F_5 \vee F_4F_8)(F_8 \vee F_9 \vee F_{10}) = (F_4F_5F_8 \vee F_4F_5F_9 \vee F_4F_5F_{10} \vee F_4F_8F_8 \vee \\ &\vee F_4F_8F_9 \vee F_4F_8F_{10}) = (F_4F_5F_8 \vee F_4F_5F_9 \vee F_4F_5F_{10} \vee F_4F_8 \vee F_4F_8F_9 \vee F_4F_8F_{10}) = \\ &= (F_4F_5F_9 \vee F_4F_5F_{10} \vee F_4F_8). \end{aligned} \tag{4.5}$$

Функція, що зображена у вигляді

$$F = (F_4F_5F_9 \vee F_4F_5F_{10} \vee F_4F_8), \tag{4.6}$$

містить во всех термах дефект F4, що означає його обов'язкову присутність у функціональності SoC. Якщо прийняти гіпотезу про існування одиночного або мінімального числа кратних дефектів, то кращим є рішення, яке визначається третім термом - в схемі існує два дефекти, які формують на виходах вектор експериментальної перевірки, що дорівнює V = (10001001001).

4.3. Моделювання для уточнення діагнозу

Подана в (4.4) діз'юнктівна форма є основною моделлю для пошуку несправностей. Вона не завжди однозначно визначає дефект функціональності, тому потребує процедур, уточнюючих діагноз. Всі рядки $M = T \times F$, які відзначені нульовими значеннями вектора експериментальної перевірки, слід об'єднати в диз'юнкцію несправностей (4.4), які не можуть бути присутніми в схемі. Отримання такої форми з розглянутої ТН дає можливість визначити всі дефекти, які не можуть бути присутніми в схемі:

$$\bar{F} = (F_2 \vee F_7) \vee (F_3 \vee F_6 \vee F_9) \vee (F_1) \vee (F_1 \vee F_2) \vee (F_3) \vee (F_5 \vee F_6) \vee (F_7) = \\ = (F_2 \vee F_7 \vee F_3 \vee F_6 \vee F_9 \vee F_1 \vee F_5) = (F_1 \vee F_2 \vee F_3 \vee F_5 \vee F_6 \vee F_7 \vee F_9). \quad (4.7)$$

Аналіз виразів, поданих формулами (4.6) і (4.7), призводить до інтересних висновків: 1. Дефекти, які не можуть бути присутніми в схемі, визначаються в термах ДНФ, отриманих за нульовими рядкам відносно вектора експериментальної перевірки. 2. Дефекти, які містяться в ДНФ (4.7), повинні бути видалені з функції (4.6). 3. Виняток дефекту F5 призводить до руйнування двох термів $F_4F_5F_9 \vee F_4F_5F_{10}$, оскільки без несправності F5 кожен з дефектів окремо не зможе сформувати заданий вектор експериментальної перевірки. 4. Таким чином, можна зробити єдиний висновок - в схемі присутня дворазова помилка, обумовлена термом $F = (F_4F_8)$. 5. Обчислювальна складність отримання точної і повної множини розв’язків визначається виразом $Q = 2^{m+1}(2m+1)$, m - число дефектів.

Позначивши відсутність конкретної несправності як $F_i = 0$, можна сфор-рмувати вхідні умови для ДНФ з метою подальшого моделювання функції при наступних початкових умовах:

$$(F_1, F_2, F_3, F_5, F_6, F_7, F_9) = (0000000).$$

Тогда результат моделювання функції $F = (F_4F_5F_9 \vee F_4F_5F_{10} \vee F_4F_8)$ дорівнює

$$F = (F_4 0 F_9 \vee F_4 0 F_{10} \vee F_4F_8) = F_4F_8.$$

Дійсно, якщо несправності $(F_1, F_2, F_3, F_5, F_6, F_7, F_9)$, що теоретично перевіряються на тестових наборах, дають негативний результат - всі вони не спотворюють стани виходів, то значить - вони відсутні в схемі. Обгрунтування даного факту підтверджується наступними доказами.

Лема 4.1. Повна множина всіх можливих поєднань дефектів, перевіряємих тестом T, визначається як ДНФ, отримана перетворенням кон'юнктівної форми

$$F = \bigwedge_{\forall V_i = 1}^{i=\overline{1,n}} \left(\bigvee_{\forall M_{ij} = 1}^{j=\overline{1,m}} F_j \right) = \bigvee_{i=1}^{2^m} \left(\bigwedge_{j=1}^{m} k_j F_j \right),$$

кожний терм якої записаний за одиничним значенням рядку таблиці несправностей [54,56,59,62] $M = T \times F$, що має стан ВЕП $V_i = 1$.

Вихідна інформація, записана відповідно до одиничних значень ВЕП, являє собою повну модель несправної поведінки реального об'єкту, яка формує вектор експериментальної перевірки з фіксованим числом одиниць (рядків таблиці несправностей), рівним k. Кожен рядок формує диз'юнкцію де-

фектів, записану за функцією АБО. Число таких диз'юнкцій дорівнює k, які логічно перемножуються, утворюючи повну і несуперечливу множину подій - дефектів, одночасно присутніх в схемі. Шляхом перемноження елементарних диз'юнкцій з подальшим спрощенням виразів, використовуючи аксіоми ($a \vee ab = b$; $a \vee a = a$), отримуємо ДНФ, яка утворює всі можливі сполучення, записані у вигляді елементарних кон'юнкцій. Зважаючи на тотожність виконаних перетворень отримана функція за логікою еквівалентна вихідної КНФ, а по суті вона є технологічна форма запису всіх рішень - поєднань дефектів, що мають місце в схемі.

Лема 4.2. Всі перевірювані у рядках таблиці несправностей $M = T \times F$ дефекти, відзначені нульовими значеннями вектора експериментальної перевірки $V_i = 0$, в реальному об'єкті відсутні.

Дійсно, таблиця несправностей $M = T \times F$ має два типи рядків: одиничні та нульові відносно фактичного значення вектора експериментальної перевірки: $[M_p(0110) \rightarrow V_p = 1] \& [M_q(0101) \rightarrow V_q = 0]$.

Рядок p виявляє наявність у схемі двох дефектів $F_2 \vee F_3$. Рядок q свідчить про теоретичну перевірку несправностей $F_2 \vee F_4$, якби вектор був дорівнений 1: $V_q = 1$. Але фактично сигнал $V_q = 0$ ідентифікує несуттєвість дефектів $F_2 \vee F_4$ для спотворення виходів схеми. Інакше, дані дефекти відсутні в тестованому виробі. Підставив у функцію $F = F_2 \vee F_3$ нульові сигнали $F_2 \vee F_4 = 0$, отримаємо результат: $F = F_2 \vee F_3 \big|_{F_2 = F_4 = 0} = F_3$. Аналогічно, всі дефекти, які визначені в рядках, що мають нульове значення ВЕП, будуть відсутні в схемі. Але якщо це так, то їх слід виключити з ДНФ, записаної за одиничними значеннями ВЕП. Отже, маючи терми ДНФ і множину дефектів, які не можуть існувати в схемі для заданого ВЕП, можна виконати процедуру підстановки нульових сигналів в змінні елементарних кон'юнкцій функції ДНФ. Але з урахуванням того факту, що $0 \wedge a \wedge b \wedge c ... = 0$, результат підстановки і наступних перетворень з метою отримання мінімальної функції буде містити тільки ті терми, які не мають змінних - дефектів з нульовим значенням сигналів. Це означає, що з ДНФ будуть виключені всі дефекти, які стосуються нульових (щодо ВЕП) рядків таблиці несправностей.

Теорема 4.1. *Мінімальна множина всіх можливих поєднань дефектів, що визначаються за таблицею несправностей* $M = T \times F$, *обчислюється шляхом моделювання ДНФ множині початкових умов*

$$F = \bigvee_{i=1}^{2^m} \left(\bigwedge_{j=1}^{m} k_j F_j \right) \Big|_{(\forall F_q = 0) \leftarrow (\exists M_{pq} = 1) \& (V_p = 0)},$$

що віизначені нульовими значеннями всіх перевірюваних дефектів, які відповідають нульовим сигналам вектора експериментальної перевірки.

Відповідно до леми 4.1 повне безліч всіх можливих поєднань дефектів, що перевіряються тестом, визначається у вигляді ДНФ:

$$F = \bigvee_{i=1}^{2^m} \left(\bigwedge_{j=1}^{m} k_j F_j \right),$$

яка формує всі розв'язки, що задовільнюють одиничним значенням вектора експериментальної перевірки $V_q = 1$. Воно може бути зменшено за рахунок виключення тих дефектів, які теоретично перевіряються тестом, але фактично на тестових наборах вони не спотворюють станів виходів, що означає їх відсутність в реальній схемі. Отже, такі дефекти можна виключити з термів ДНФ, що утворюють повну множину всіх можливих поєднань. Механізм такого винятку, відповідно до леми 4.2, полягає в підстановці нульових значень змінних в терми ДНФ з наступним моделюванням функції. Якщо терм має 0-компонент деякої змінної F_i, згідно алгебрі логіки, весь терм обертається на 0, що означає його виключення з ДНФ. Таким чином, в результаті мінімізації на основі урахування умов леми 4.2 залишається мінімальна ДНФ, що містить найменше число можливих поєднань дефектів (одиночних і кратних), яке неможливо скоротити без використання додаткової діагностичної інформації, що надходить від мультизонда на основі boundary scan регистру.

Практично корисними є твердження для уточнення діагнозу.

Твердження 4.1. Якщо F_j присутній у всіх термах ДНФ, то даний дефект існує в схемі - його не слід тестувати. В іншому випадку, якщо припустити, що перевірка буде дорівнювати нулю, то всі терми обертаються на нуль, а це суперечить умові існування ненульових значень вектора експериментальної перевірки V.

Твердження 4.2. У схемі присутній тільки одне поєднання дефектів, визначене одним термом ДНФ. Якщо знайдено одне підтверджене рішення у вигляді терма ДНФ, то решту термів слід виключити з розглядання шляхом їх обернення на нуль.

Тому задача мінімізації точок контролю зводиться до виконання двох альтернативних стратегій: 1) розгляд змінних в термах мінімальної довжини для підтвердження всіх дефектів в логічному добутку шляхом їх зондування; 2) перевірка таких змінних, які обертають в нуль максимальне число термів ДНФ.

Таким чином, запропонований алгеброгічний метод діагностування використовує булево зчислення як базового апарату для розв'язання задачі

покриття шляхом отримання діз'юнктівної форми, яка далі мінімізується шляхом виключення термів, що мають змінні дефектів, які відносяться до рядків, з нульовими значеннями ВЕП. Перевага алгебро-логічного методу полягає в отриманні термів ДНФ, які формують всі можливі покриття (кратні несправності) одиничних координат вектора експериментальної перевірки. Обчислювальна складність методу має експоненційну залежність від числа дефектів: $Q = 2^n$. Для незначного фактичного числа дефектів в цифровій системі обчислювальна складність дозволяє здійснювати пошук несправностей в реальному масштабі часу.

4.4. Структурно-логічний метод діагностування дефектів

Таблиця несправностей, яка виходить в результаті виконання процедури моделювання дефектів, не має інформації щодо зовнішніх виходів, які отримали викривлення на тестових наборах. Даний факт може істотно зменшити глибину діагностування цифрового вироби. З іншого боку, додаткове використання згаданої ін-формації в сукупності з таблицею несправностей позитивно впливає на результат діагнозу і зменшує потужність дефектних компонентів.

Нехай мається ТН з виходами Y_1, Y_2, Y_3, які несуть інформацію про викревлення на тестових векторах T_1, T_4, T_8, T_{11}:

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	Y_1	Y_2	Y_3	V
T_1				1						1	0	0	1	1
T_2		1					1				0	0	0	0
T_3			1			1			1		0	0	0	0
T_4	1										0	0	0	0
T_5					1			1			1	1	1	1
T_6	1	1									0	0	0	0
T_7			1								0	0	0	0
T_8				1							0	1	0	1
T_9					1	1					0	0	0	0
T_{10}							1				0	0	0	0
T_{11}								1	1	1	1	0	0	1
$\vec{Y}_1$	1			1	1			1		1	1			
$\vec{Y}_2$	1	1		1			1	1	1			1		
$\vec{Y}_3$	1		1		1	1			1	1			1	

(4.8)

Вектор експериментальної перевірки V був отриманий шляхом виконання операції АБО над вихідними результатами діагностичного експерименту $V = \bigvee_{j=1}^{n} Y_j$, що полягає у подачі тесту на входи виробу і наступним порівнянням еталонної R та експериментальної R* реакцій за кожним виходом Y_j схеми: $Y_j = R(T_i) \oplus R^*(T_i)$. Стосовно до таблиці несправностей

(4.8) була виконана операція $V = Y_1 \vee Y_2 \vee Y_3$, яка дозволила отримати ВЕП, істотно скоротивши обсяг інформації, але потенційно втративши при цьому глибину діагностування. Необхідно мати можливість для уточнення діагнозу шляхом врахування схемної структури і точного аналізу станів виходів виробу. Для цього вводиться додаткова інформація у вигляді векторної форми матриці досяжностей всіх спостережуваних виходів з боку компонентів схеми, таких як лінії і елементи, що схильні дефектам:

$$Y = \left|\vec{Y}_j\right|, i = \overline{1,n}; j = \overline{1,m},$$

n – число виходів або векторів матриці досяжностей; m – розмірність кожного вектора, що дорівнює числу дефектів схеми.

Для розглянутого прикладу матриця досяжностей трьох виходів (скалярів) схеми Y_1, Y_2, Y_3 з боку потенційних несправних компонентів подана трьома останніми рядками виразу (4.8). Тут виходи розглядаються як незалежні один від одного, хоча, в загальному випадку, це не завжди має місце.

Процедура діагностування з урахуванням структури модифікується на стадії складання КНФ. До кожного рядку таблиці несправностей, де $M_i(Y_j = 1)$, застосовується операція маскування відповідним рядком $\vec{Y}_j$ матриці досяжностей:

$$M^1 = M^1 \vee [\vec{Y}_j \bigwedge_{\forall Y_j = 1} M_i(Y_j)]. \qquad (4.9)$$

Якщо всі виходи схеми мають одиничні значення результатів експерименту $\forall Y_j = 1$, то поточний рядок таблиці несправностей маскується всіма векторами матриці досяжностей, з об'єднанням результату за функцією АБО, що означає - процедуру маскування виконувати не слід взагалі.

Стосовно до запропонованого вище прикладу виконується процедура маскування (4.9) над рядками 1,5,8,11 ТН (4.8) за допомогою векторів $\vec{Y}_1, \vec{Y}_2, \vec{Y}_3$, записаних в нижній частині таблиці несправностей. Це призводить до зменшення числа одиничних координат в ТН:

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	Y_1	Y_2	Y_3	V
T_1										1	0	0	1	1
T_2		1					1				0	0	0	0
T_3			1			1			1		0	0	0	0
T_4	1										0	0	0	0
T_5					1			1			1	1	1	1
T_6	1	1									0	0	0	0
T_7			1								0	0	0	0
T_8				1							0	1	0	1
T_9					1	1					0	0	0	0
T_{10}							1				0	0	0	0
T_{11}								1		1	1	1	0	1

Тут з таблиці несправностей видалені одиничні координати $\{(T_1, F_4), (T_{11}, F_9)\}$. Дана обставина призводить до більш простої КНФ, що в цілому робить процедуру отримання ДНФ менш трудомісткою. Процес отримання ДНФ дефектів об'єкта має наступний вигляд:

$$F = (F_{10})(F_5 \vee F_8)(F_4)(F_8 \vee F_{10}) = (F_4)(F_{10})(F_5 \vee F_8)(F_8 \vee F_{10}) =$$
$$= F_4F_{10}F_5F_8 \vee F_4F_{10}F_5F_{10} \vee F_4F_{10}F_8F_8 \vee F_4F_{10}F_8F_{10} =$$
$$= F_4F_5F_8F_{10} \vee F_4F_5F_{10} \vee F_4F_8F_{10} \vee F_4F_8F_{10} =$$
$$= F_4F_5F_{10} \vee F_4F_8F_{10} = F_4F_{10}(F_5 \vee F_8).$$

Порівнюючи результати з раніше отриманими розв’язками $F = (F_4F_5F_9 \vee F_4F_5F_{10} \vee F_4F_8)$, можна констатувати факт відсутності розв’язку, оскільки він не покриває одиничні координати ВЕП (1,5,8,11). Отримана підмножина з двох поєднань дефектів свідчить про обов'язкову наявність несправних компонентів 4 і 10, яке доповнюється одним з двох дефектів: 5 або 8. Для уточнення діагнозу необхідно виконати процедуру зондування однією з ліній 5 або 8, що призведе до отримання рішення у вигляді єдиного терму, що визначає підмножину дефектів, реально існуючих у схемі.

4.5. Векторно-логічний метод діагностування за таблицею несправностей

Обробка ТН для отримання діагнозу виконується за сценарієм, заснованим на використанні векторних операцій кон'юнкції, диз'юнкції і заперечення над рядками таблиці несправностей. Кон'юнкція узагальненого вектора, що відповідає одиничним координатам ВЕП з інверсним узагальненим вектором щодо відносно координат ВЕП:

$$F = M^1 \wedge \overline{M}^0 = \left(\bigvee_{V_i = 1} M_i \right) \wedge \left(\overline{\bigvee_{V_i = 0} M_i} \right). \qquad (4.10)$$

Діагностування одиночних дефектів відрізняється виконанням на першому кроці сценарію операції кон'юнкції (замість диз'юнкції) усіх векторів, що відповідають одиничним координатам ВЕП:

$$F = M^1 \wedge \overline{M}^0 = \left(\bigwedge_{V_i=1} M_i \right) \wedge \left(\overline{\bigvee_{V_i=0} M_i} \right). \qquad (4.11)$$

Приклад 4.2. Виконати діагностування кратних дефектів у схемі векторно-логічним методом, для якої задані таблиця несправностей і вектор експериментальної перевірки:

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	V
T_1				1						1	1
T_2		1					1				0
T_3			1			1			1		0
T_4	1										0
T_5					1			1			1
T_6	1	1									0
T_7			1								0
T_8				1							1
T_9					1	1					0
T_{10}							1				0
T_{11}								1	1	1	1
M^1				1	1			1	1	1	1
M^0	1	1	1		1	1	1		1		0
$\overline{M}^0$	0	0	0	1	0	0	0	1	0	1	1
F	0	0	0	1	0	0	0	1	0	1	1

(4.12)

Обробка таблиці несправностей відповідно до формули (4.10) дає результат, поданий у чотирьох нижніх рядках (4.12). Останній рядок таблиці несправностей фіксує факт наявності в схемі дефектів, зображених у векторній або теоретико-множинній формі $F = (0001000101) = \{F_4, F_8, F_{10}\}$.

Для перетворення отриманого розв’язку до вигляду ДНФ використовується структура ТН і множина дефектів, зафіксованих в останньому рядку таблиці. Синтез діз'юнктівной форми дає наступний результат:

$$F = (F_4 \vee F_{10})(F_8)(F_4)(F_4 \vee F_{10}) = F_4F_8F_4F_4 \vee F_{10}F_8F_4F_4 \vee F_4F_8F_4F_{10} \vee F_{10}F_8F_4F_{10} =$$
$$= F_4F_8 \vee F_{10}F_8F_4 \vee F_4F_8F_{10} \vee F_{10}F_8F_4 = F_4F_8.$$

Він цікавий тим, що завдяки запису дефектів у вигляді термів ДНФ, що покривають всі одиничні координати ВЕП, з'явилася можливість виключити дефект $F_{10} \in F$ зі списку несправностей. Аналогічний результат був отриманий раніше, при розгляді алгебрологічного методу диагностування несправностей.

Перевага векторно-логічного методу полягає в технологічності аналізу таблиці несправностей, обчислювальна складність якого має мультиплікативну залежність від числа дефектів і потужності тесту:

$Q = n \times m$. Метод слід використовувати при перевазі в таблиці несправностей одиничних координат. Недолік полягає в неможливості надання всіх поєднань дефектів, що формують терми для покриття одиничних координат вектора експериментальної перевірки.

Приклад 4.3. Виконати векторно-логічне діагностування дефектів ліній схеми, поданої на рис. 4.6, з урахуванням схемної структури.

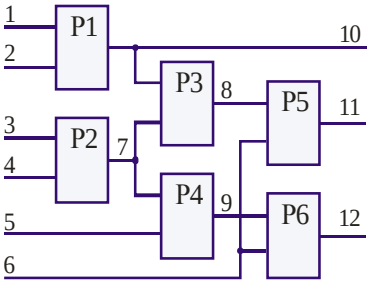


Рис. 4.6. Приклад схеми для діагностування

Наведеній схемній структурі відповідає таблиця несправностей (перші 5 рядків) і вектор експериментальної перевірки V:

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	Y_{10}	Y_{11}	Y_{12}	V
T_1	1	1		1	1	1	1		1	1	1	1	1	0	0	1
T_2		1	1		1			1	1	1	1	1	0	1		1
T_3		1					1		1	1	1	1	0	0	0	0
T_4									1	1	1	1	0	0	0	0
T_5	1	1	1	1			1	1		1	1	1	0	0	1	1
$\vec{Y}_{10}$	1	1								1						
$\vec{Y}_{11}$	1	1	1	1		1	1	1			1					
$\vec{Y}_{12}$			1	1	1	1	1		1			1				
M^1	1	1	1	1	1	1	1	1	1	1	1	1				
M^0		1					1		1	1	1	1				
$\overline{M}^0$	1		1	1	1	1		1								
F	1		1	1	1	1		1								
$M^1(\vec{Y})$	1	1	1	1			1	1	1	1	1	1				
$F(\vec{Y})$	1		1	1				1								

(4.13)

Використання векторно-логічного методу з урахуванням виразу (4.10) дає можливість без урахування структури схеми отримати результат: $F = (101111010000) = \{F_1, F_3, F_4, F_5, F_6, F_8\}$. Синтез диз'юнктивної форми за таблицею несправностей, але маскованої отриманими дефектами $F = \{F_1, F_3, F_4, F_5, F_6, F_8\}$, надає розв'язок:

$$F = (F_1 \vee F_4 \vee F_5 \vee F_6)(F_3 \vee F_5 \vee F_8)(F_1 \vee F_3 \vee F_4 \vee F_8) =$$
$$= (F_1 \vee F_4 \vee F_5 \vee F_6)(F_1F_3 \vee F_1F_5 \vee F_1F_8 \vee F_3F_3 \vee F_3F_5 \vee F_3F_8 \vee$$
$$\vee F_4F_3 \vee F_4F_5 \vee F_4F_8 \vee F_3F_8 \vee F_5F_8 \vee F_8F_8) =$$
$$= F_1F_1F_3 \vee F_1F_1F_5 \vee F_1F_1F_8 \vee F_1F_3F_3 \vee F_1F_3F_5 \vee F_1F_3F_8 \vee$$
$$\vee F_1F_4F_3 \vee F_1F_4F_5 \vee F_1F_4F_8 \vee F_1F_3F_8 \vee F_1F_5F_8 \vee F_1F_8F_8 \vee$$
$$\vee F_1F_3F_4 \vee F_1F_4F_5 \vee F_1F_4F_8 \vee F_3F_3F_4 \vee F_3F_4F_5 \vee F_3F_4F_8 \vee$$
$$\vee F_3F_4F_4 \vee F_4F_4F_5 \vee F_4F_4F_8 \vee F_3F_4F_8 \vee F_4F_5F_8 \vee F_4F_8F_8 \vee$$
$$\vee F_1F_3F_5 \vee F_1F_5F_5 \vee F_1F_5F_8 \vee F_3F_3F_5 \vee F_3F_5F_5 \vee F_3F_5F_8 \vee$$
$$\vee F_3F_4F_5 \vee F_4F_5F_5 \vee F_4F_5F_8 \vee F_3F_5F_8 \vee F_5F_5F_8 \vee F_5F_8F_8 \vee$$
$$\vee F_1F_3F_6 \vee F_1F_5F_6 \vee F_1F_6F_8 \vee F_3F_3F_6 \vee F_3F_5F_6 \vee F_3F_6F_8 \vee$$
$$\vee F_3F_4F_6 \vee F_4F_5F_6 \vee F_4F_6F_8 \vee F_3F_6F_8 \vee F_5F_6F_8 \vee F_6F_8F_8 =$$
$$= F_1F_3 \vee F_1F_5 \vee F_1F_8 \vee F_3F_4 \vee F_3F_5 \vee F_3F_6 \vee F_4F_5 \vee F_4F_8 \vee F_5F_8 \vee F_6F_8.$$

У випадку додаткового використання структури схеми (див. рис.4.6) у вигляді рядків Y_{10}, Y_{11}, Y_{12} таблиці (4.13), що накладається на ТН, результат не может бути гірше. Обчислення вектору $M^1(\vec{Y})$ в (4.14) здійснюється шляхом накладання масок з векторів досяжностей $\{\vec{Y}_{10}, \vec{Y}_{11}, \vec{Y}_{12}\}$, що відповідають несправним на тест-векторах виходам схеми:

$$M^1(\vec{Y}) = [\vec{Y}_{10} \wedge M_1(Y_{10})] \vee [(\vec{Y}_{11}) \wedge M_2(Y_{11})] \vee [(\vec{Y}_{12}) \wedge M_5(Y_{12})]. \quad (4.14)$$

Підстановка конкретних векторів з таблиці несправностей у виразі (4.10) дає результат:

$$M^1(\vec{Y}) = [110000000100] \vee [011000010010] \vee [001100101001] =$$
$$= [111100111111] = \{F_1, F_2, F_3, F_4, F_7, F_8, F_9, F_{10}, F_{12}\}.$$

Наступні обчислення, пов’язані з виконанням векторних операцій $F(\vec{Y}) = M^1(\vec{Y}) \wedge \overline{M}^0$ на таблиціе несправностей, формують остаточне рішення:

$$F(\vec{Y}) = (111100111111) \wedge (101111010000) =$$
$$= (101100010000) = \{F_1, F_3, F_4, F_8\}.$$

Останній вектор (111100111111) створює маску для формування диз’юнктивної нормальної форми, яка має наступні терми:

$$F(\vec{Y}) = (F_1 \vee F_4)(F_3 \vee F_8)(F_1 \vee F_3 \vee F_4 \vee F_8) =$$
$$= (F_1 \vee F_4)(F_3 \vee F_8) = F_1F_3 \vee F_1F_8 \vee F_3F_4 \vee F_4F_8.$$

У загальному випадку урахування структури схеми у вигляді матриці досяжностей дозволяє отримати більш точний діагноз шляхом виключення таких дефектів, які не можуть впливати на несправні виходи.

Запропоновані методи діагностування: 1) алгебрологічний, 2) векторно-логічний та 3) структурно-логічний пропонують фахівцеві в області проектування і тестування цифрових систем на кристалах математичний апарат, який здатний здійснити діагноз дефектних компонентів, якщо мається попередньо побудована таблиця несправностей. При цьому теоретико-множинні розв'язки, ефективно отримані в другому методі, можна подавати усіма можливими поєднаннями дефектів у вигляді термів ДНФ, що характерно для першого методу. Другий метод ефективний при числі одиниць в матриці несправностей, що перевищує 10-20%. Третій метод передбачає наявність додаткової інформації у вигляді матриці досяжностей для всіх зовнішніх виходів, що дає можливість істотно зменшити потужність діагностованих дефектів або число термів, які задають всі можливі поєднання несправностей, які формують вектор експериментальної перевірки.

Слід зауважити, що всі три методи можуть генерувати результат у трьох формах: векторній, множинній і асоціативній у вигляді термів ДНФ. Фаза моделювання несправностей має самостійний інтерес з метою зменшення дефектної області цифрової системи. Алгебрологічний метод також може бути доповнений процедурою зменшення дефектів за рахунок відрахування всіх неіснуючих несправностей аналізованої схемної структури.

4.6. Програмний засіб «Defect Analyzer»

Програмний засіб «Defect Analyzer» призначений для моделювання процесів тестування і діагностування цифрових систем на кристалах з використанням таблиці несправностей. «Defect Analyzer» є додатком, який функціонує під управлінням операційної системи Windows XP, розроблений з використанням мови программування C++ і середовища програмування Borland C ++ Builder 2006, містить 659 операторів, 604 рядка.

У програмному додатку реалізовані алгебрологічний метод діагностування схемної структури, послідовний і паралельний методи вибору контрольних точок, в яких проводиться інтерактивна процедура зондування. Як вхідні дані використовується таблиця покриття, побудована в процесі тестування схеми на наявність дефектів. Таблиця відображає рівні сигналів, що подаються на вхід, а також виявляє несправності в схемі на поточному тестовому наборі, на що вказує значення результуючого вектора V (рис. 4.7).

З метою спрощення вихідної таблиці покриття несправностей можна використовувати інформацію про нульові елементи тестових наборів і значеннях відповідних сигналів на виходах схемної структури для кожного тестового набору, а також виконувати попереднє приведення КНФ до простішого вигляду.

Процедура зондування виконується покроково у певних контрольних точках, що відображається для кожного з методів у вигляді змінюваної таблиці покриття та спрощеної ДНФ (рис. 4.8).

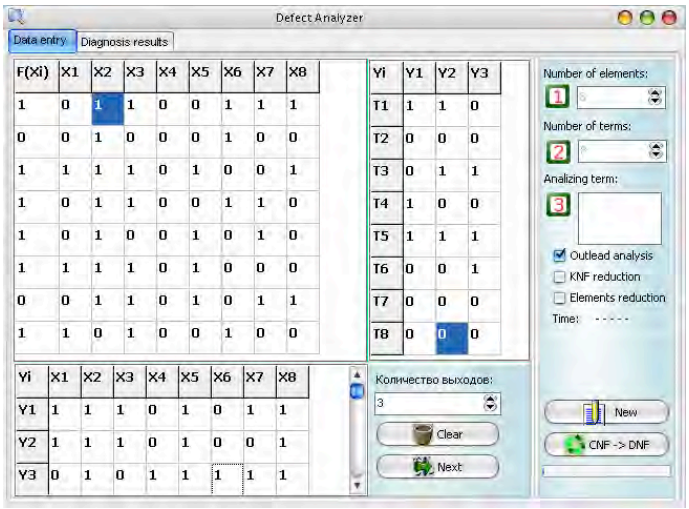


Рис. 4.7. Головне вікно програмного засобу

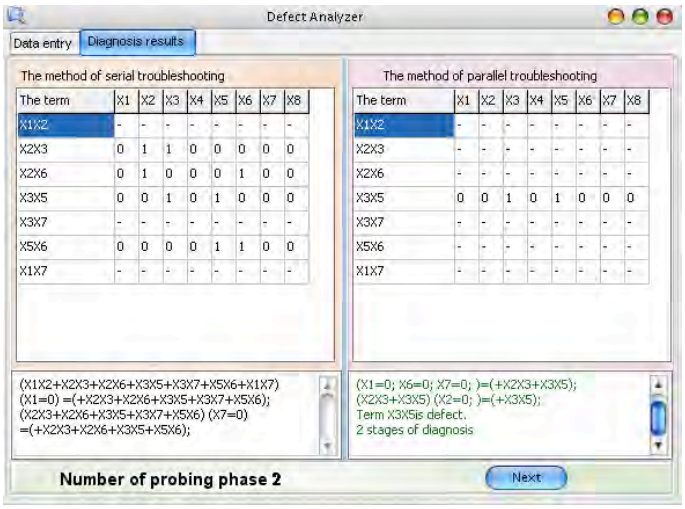


Рис. 4.8. Покрокове виконання процедури зондування

Обсяг даних, який дозволяє обробляти програмний додаток, обмежений кількістю термів неспрощеної ДНФ, що відображаються у таблиці. Кількість рядків таблиці (термів) не повинно перевищувати 32766. Число термів ДНФ залежить від вихідних даних і може бути обчислено за наступною формулою:

$$N = \prod_{\forall i(V_i=1)}^{i=\overline{1,n}} (N_i) \ ,$$

де N_i – кількість одиничних елементів в тестовому наборі.

При встановленні діагнозу розглядаються вхідні послідовності, для яких значення вектора експериментальної перевірки дорівнює одиниці.

Часові витрати процесу діагностування пов'язані з перетворенням КНФ в ДНФ і подальшим її спрощенням. Обчислювальна складність перетворень залежить від результатів експерименту і розмірності схеми (таблиці несправностей):

$$Q = \prod_{i=1}^{n}(N_i!) + \frac{n!}{2(n-2)!}nm,$$

де n – кількість тестових наборів; m – кількість елементів, що беруть участь у експерименті; N_i – кількість одиничних элементів у i-му входному наборі.

Нижче наведено графік залежності часу, витраченого на перетворення КНФ в ДНФ, від кількості тестових наборів і елементів (рис. 4.9). При цьому вважається, що число одиничних елементів в рядку ТН, що відповідає тестовому набору, становить не більше 25%, кількість вхідних тестів, при яких значення виходів еталонної моделі та тестованій схемі різні, становить не більше 50%.

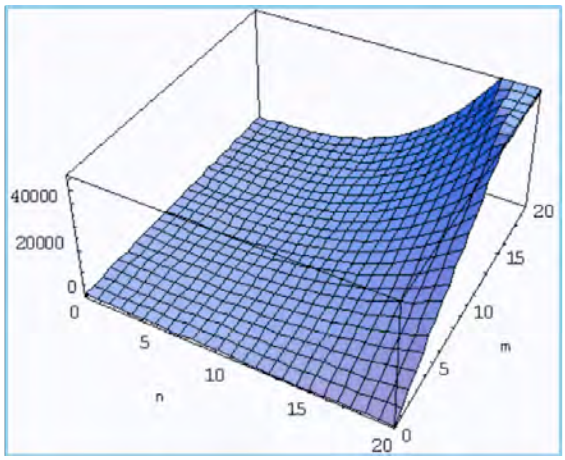


Рис. 4.9. Залежність часу перетворення КНФ у ДНФ

Отримання інформації про нульові елементи, спрощення таблиці покриття несправностей шляхом аналізу станів виходів схемної структури, приведення вихідної КНФ до простішого вигляду до її перетворення в ДНФ дозволяють значно зменшити час діагностування.

Приклад 4.4. Дана схемна структура, що зображена на рис. 4.10.

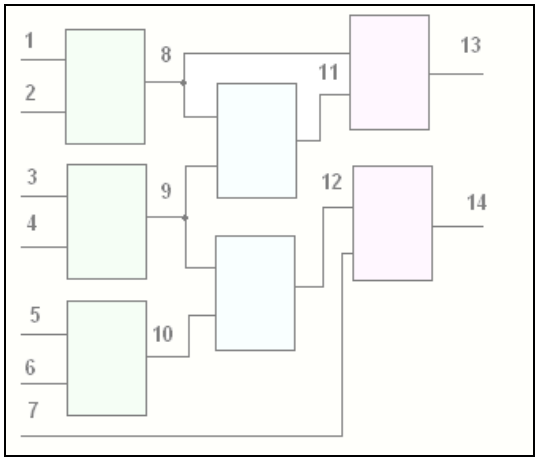


Рис. 4.10. Приклад схемної структури

Фрагменту схеми відповідає таблиця несправностей (перші 5 рядків) і вектор експериментальної перевірки V, наведені нижче.

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	V	Y_1	Y_2
T_1		1		1			1								0		
T_2	1					1		1		1	1		1	1	1	1	
T_3			1				1								0		
T_4	1	1	1	1			1	1	1		1			1	1		1
T_5	1		1		1		1		1	1		1	1		1	1	1
Y_1	1	1	1	1				1	1		1		1				
Y_2			1	1	1	1	1		1	1		1		1			

Виконання процедури маскування рядків 1,2,4,5 ТН за допомогою векторів $\vec{Y}_1, \vec{Y}_2$ у нижній частині таблиці несправностей призводить до зменшення кількості одиничних координат у ТН:

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	V
T_1		1		1			1								0
T_2	1					0		1		0	1		1		1
T_3			1				1								0
T_4	0	0	1	1			1	0	1		0			1	1
T_5	1		1		1		1		1	1		1	1		1

Затем определяют номера столбцов, в которых нулевым значениям ВЭП соответствуют единичные значения координат строк таблицы, и исключают их из рассмотрения (для данного примера: столбцы F2, F3, F4, F7):

T_i / F_j	F_1	F_5	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
T_2	1		1			1		1	
T_4				1					1
T_5	1	1		1	1		1	1	

Виконується побудова КНФ за одиничним значенням ВЕП, що дає наступний результат:

$$F=(F_1 \vee F_8 \vee F_{11} \vee F_{13})(F_9 \vee F_{14})(F_1 \vee F_5 \vee F_9 \vee F_{10} \vee F_{12} \vee F_{13}) =$$

$$= F_1F_9 \vee F_1F_{14} \vee F_8F_9 \vee F_5F_8F_{14} \vee F_8F_{10}F_{14} \vee F_8F_{12}F_{14} \vee F_9F_{11} \vee$$

$$\vee F_5F_{11}F_{14} \vee F_{10}F_{11}F_{14} \vee F_{11}F_{12}F_{14} \vee F_9F_{13} \vee F_{13}F_{14}.$$

Висновок: у схемі існує кратний дефект F_8F_9.

Послідовне зондування в контрольних точках виконується так, як описано нижче:

F_i^1	F_1	F_5	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
$W(F_i)$	2	2	4	4	2	4	2	2	8

У розглянуваном прикладі мінімальна вага елементів F_i дорівнює двом. Для дведення існування кратного дефекту F_8F_9 як першої контрольної точки слід взяти F_1:

$$F_1F_9 \vee F_1F_{14} \vee F_8F_9 \vee F_5F_8F_{14} \vee F_8F_{10}F_{14} \vee F_8F_{12}F_{14} \vee F_9F_{11} \vee F_5F_{11}F_{14} \vee$$

$$\vee F_{10}F_{11}F_{14} \vee F_{11}F_{12}F_{14} \vee F_9F_{13} \vee F_{13}F_{14}(F_1 = 0) =$$

$$= F_8F_9 \vee F_5F_8F_{14} \vee F_8F_{10}F_{14} \vee F_8F_{12}F_{14} \vee F_9F_{11} \vee F_5F_{11}F_{14} \vee F_{10}F_{11}F_{14} \vee$$

$$\vee F_{11}F_{12}F_{14} \vee F_9F_{13} \vee F_{13}F_{14};$$

F_i	F_1	F_5	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
$W(F_i)$	0	2	4	3	2	4	2	2	7

$$F_8F_9 \vee F_5F_8F_{14} \vee F_8F_{10}F_{14} \vee F_8F_{12}F_{14} \vee F_9F_{11} \vee F_5F_{11}F_{14} \vee F_{10}F_{11}F_{14} \vee$$

$$\vee F_{11}F_{12}F_{14} \vee F_9F_{13} \vee F_{13}F_{14}(F_5=0) =$$

$$= F_8F_9 \vee F_8F_{10}F_{14} \vee F_8F_{12}F_{14} \vee F_9F_{11} \vee F_{10}F_{11}F_{14} \vee F_{11}F_{12}F_{14} \vee F_9F_{13} \vee F_{13}F_{14};$$

F_i^1	F_1	F_5	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
$W(F_i)$	0	0	3	3	2	3	2	2	5

$$F_8F_9 \vee F_8F_{10}F_{14} \vee F_8F_{12}F_{14} \vee F_9F_{11} \vee F_{10}F_{11}F_{14} \vee F_{11}F_{12}F_{14} \vee F_9F_{13} \vee F_{13}F_{14}(F_{13}=0) =$$

$$= F_8F_9 \vee F_8F_{10}F_{14} \vee F_8F_{12}F_{14} \vee F_9F_{11} \vee F_{10}F_{11}F_{14} \vee F_{11}F_{12}F_{14};$$

F_i^1	F_1	F_5	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
$W(F_i)$	0	0	3	2	2	3	2	0	4

$$F_8F_9 \vee F_8F_{10}F_{14} \vee F_8F_{12}F_{14} \vee F_9F_{11} \vee F_{10}F_{11}F_{14} \vee F_{11}F_{12}F_{14}{}_{(F_{12}=0)} =$$

$$= F_8F_9 \vee F_8F_{10}F_{14} \vee F_9F_{11} \vee F_{10}F_{11}F_{14};$$

F_i^1	F_1	F_5	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
$W(F_i)$	0	0	2	2	2	2	0	0	2

;

$$F_8F_9 \vee F_8F_{10}F_{14} \vee F_9F_{11} \vee F_{10}F_{11}F_{14}{}_{(F_{14}=0)} = F_8F_9 \vee F_9F_{11}.$$

Обрав як останню контрольну точку F_{11}, отримаємо терм F_8F_9.

Паралельне зондування декількох термів:

	F_1	F_5	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}
F_1F_{14}	1								1
F_8F_9			1	1					
$F_5F_8F_{14}$		1	1						1
$F_8F_{10}F_{14}$			1		1				1
$F_8F_{12}F_{14}$			1				1		1
F_9F_{11}				1		1			
$F_5F_{11}F_{14}$		1				1			1
$F_{10}F_{11}F_{14}$					1	1			1
$F_{11}F_{12}F_{14}$						1	1		1
F_9F_{13}				1				1	
$F_{13}F_{14}$								1	1
F_1F_9	1			1					

У розглядуваному прикладі як контрольні точки можна обрати одночасно $F_1, F_5, F_{10}, F_{12}, F_{13}$, у результаті чого таблиця покриття набуває вигляду:

	F_8	F_9	F_{11}
F_8F_9	1	1	
F_9F_{11}		1	1

Обрав як останню контрольну точку F_{11}, отримаємо терм спрощеної ДНФ F_8F_9, що підтверджує існування у схемі відповідного кратного дефекту.

Нижче наведена таблиця, яка містить дані про кількість елементів і термів для декількох експериментів, а також число необхідних етапів зондування.

Для порівняння ефективності різних методів пошуку дефектів далі наведено графік (рис. 4.11) залежності кількості необхідних точок зондування від величини оцінки схеми за Квайном Q (за результатами розглянутих прикладів).

Таблиця 4.1

Характеристики досліджуваної ДНФ дефектів

Номер	Число элементов	Количество термов	Число зондирований	
			Последов.	Параллель.
1	10	5	4	2
2	9	12	7	2
3	12	9	8	8
4	6	4	2	2
5	10	6	5	3

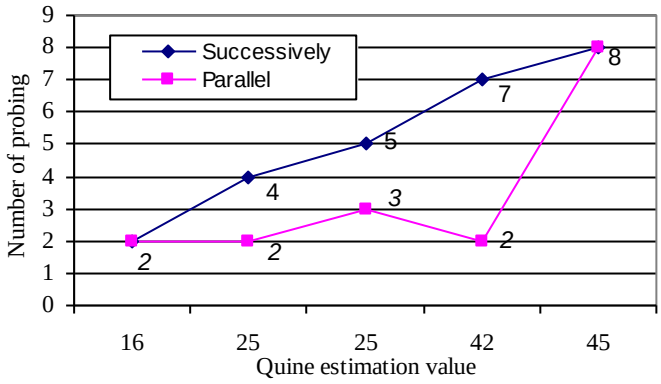


Рис.4.11. Залежність кількості точок зондування від Q

З графіка видно, що використання інтерактивного методу зондування паралельно в декількох точках призводить до меншої кількості кроків процедури аналізу ДНФ дефектів. Отже, на пошук дефекту в схемі йде менший проміжок часу, що зменшує матеріальні витрати при проектуванні цифрових систем на кристалі.

Розглянуті методи забезпечують перевірку схеми на існування дефекту. Метод паралельного зондування дозволяє вибирати кілька контрольних точок, що забезпечує більш швидкий пошук дефектного терма. Однак при деяких вихідних ДНФ паралельне зондування не має переваг над послідовним. Це обумовлено відсутністю на будь-якому з кроків перевірки (винятком може бути останній етап зондування) різних елементів з однаковими ваговими коеффіцієнтами.

Наукова новизна одержаних результатів. Запропоновано новий матричний метод діагностування дефектів функціональностей в SoC, який характеризуется використанням діз'юнктівної нормальної форми і таблиці покриття несправностей, що дає можливість отримувати повні та мінімальні

сполучення кратних несправностей на основі застосування процедур мультизондування.

Практичне значення отриманих результатів. Запропоновані методи послідовного і паралельного моделювання процесу тестування схем на кристалі на наявність несправностей реалізовані програмно. Методи орієнтовані на вбудоване сервісне обслуговування функціональність поданими IP модулями.

4.7. Алгебро-логічний метод ремонту пам'яті

Представлено точний метод діагностування та ремонту елементів пам'яті за допомогою резервних рядків і стовпців, який дозволяє покрити множену дефектних комірок мінімально можливою кількістю надлишкових компонентів. Метод орієнтований на імплементацію в інфраструктуру сервісного обслуговування функціональності цифрової системи на кристалі. Запропоновано структуровані рішення для реалізації методу діагностування та ремонту дефектних комірок матриці пам'яті [37,38,40,41,45,46,51,52].

В процесі виробництва та експлуатації будь-яких видів пам'яті необхідні гарантії її відповідності технічним умовам. Для цього передбачено виконання трьох процедур: 1) Тестування пам'яті, що полягає у подачі тестових впливів, орієнтованих на виявлення певних класів дефектів [23,37,38,56]. 2) У разі виникнення несправності необхідна додаткова процедура діагностування, яка дозволяє визначити місце, причину і вигляд дефекту. 3) Після визначення множини дефектів, які перешкоджають виконанню функцій пам'яті, необхідно активізувати процес відновлення працездатності - заміну дефектних елементів надлишковими резервними компонентами, що з самого початку знаходяться на силіконовому кристалі [9-12]. Таким чином, згадані дії орієнтовані на підвищення виходу придатних виробів без істотних додаткових часових і матеріальних витрат. Для відновлення працездатності необхідний спеціальний механізм ремонту пам'яті шляхом заміни дефектних компонентів на справні з резерву силіконового кристала.

Процедура тестування, як правило, здійснюється за допомогою BIST-блоку (Built-In Self Test), який представляє апаратний швидкодіючий генератор тестових наборів, а також аналізатор (сигнатурний) реакцій виходів пам'яті на тестові послідовності. Модуль аналізу відновлення працездатності (Repair Analysis) призначений для визначення можливості покриття дефектних елементів пам'яті існуючими у наявності резервними компонентами. Блок пам'яті поданий двома частинами: 1) функціональні комірки, які безпосередньо застосовуються для зберігання даних і програм при їх використанні у системі на кристалі; 2) резервні або запасні осередки, які призначені для відновлення працездатності пам'яті у випадку відмов функціональних осередків. Штатні та резервні комірки об'єднуються ув стовпці і рядки. При виявленні дефекту рядок (стовпець), що містить дефектний елемент, відключається від функціональної структури осередків пам'яті, а на її місце підключається рядок (стовпець) з резерву кристала. Оскільки кількість резервних компонентів обмежена, необхідний спеціальний механізм, що дозволяє ефективно розподіляти ресурси відновлення працездатності з метою

забезпечення покриття дефектних елементів пам'яті мінімально можливою кількістю надлишкових стовпців і рядків.

Описана вище процедура пошуку покриття дефектних комірок мінімаль-ною кількістю резервних рядків і стовпців може бути реалізована як вбудований модуль відновлення працездатності, так і зовнішній до кристалу. У другому випадку дані про помилки надходять ззовні, обробляються і передаються контролеру, забезпечують відновлення пам'яті. Це призводить до значних втрат часу. Тому перевага віддається on-chip реалізації модуля, коли дані про помилки передаються безпосередньо з BIST. Такий механізм носить назву BIRA [37,38,45,46,51,52] (Built-In Repair Analysis) - вбудований аналіз відновлення працездатності.

Ремонт пам'яті здійснюється за допомогою відключення дефектних елементів (рядків і стовпців матриці) шляхом електричного плавлення перемичок і підключення резервних. Процес пайки може бути електричним або лазерним [18]. Пристрій електричної пайки має менші габаритні розміри, ніж лазерної, і застосовується частіше. Пайка перемичок виконується за допомогою набору інструкцій, які можуть зберігатися в постійній пам’яти всередині чіпа (hard repair) або в оперативній пам'яті (soft repair) [37-38]. Soft repair має ряд переваг: при виникненні помилки нова виправлена інструкція може бути легко записана в пам'ять; забезпечується економне використання площі кристала і достатня надійність. Hard repair дозволяє застосовувати спрощений виробничий тест і забезпечує виявлення помилок, які в силу певних обставин не можуть бути зафіксовані при soft repair.

Структура процесів вбудованого аналізу та самовідновлення пам'яті - BISR (Built-In Self Repair) [40,45,46] зображена на рис. 4.12. Тут мається: 1 Активізація чіпа, заповнення регістра BISR нульовими значеннями. 2. Запуск контролера BIST, тестування пам'яті і накопичення інформації про дефектні осередки в регістрі BIRA. 3. Передача інформації про дефектні осередки в регістр BISR для подальшої перепайки. 4. Сканування BIRA регістрів для отримання інформації про дефекти. 5. Запуск контролера пайки в режимі запису і передача з BISR інструкцій відновлення. 6. Перезавантаження чіпа. Запис в регістр BISR інформації про пайку перемичок, заміна дефектних рядків і стовпців резервними компонентами. 7. Запуск контролера BIST з метою повторного тестування пам'яті і перевірки правильності результату відновлення.

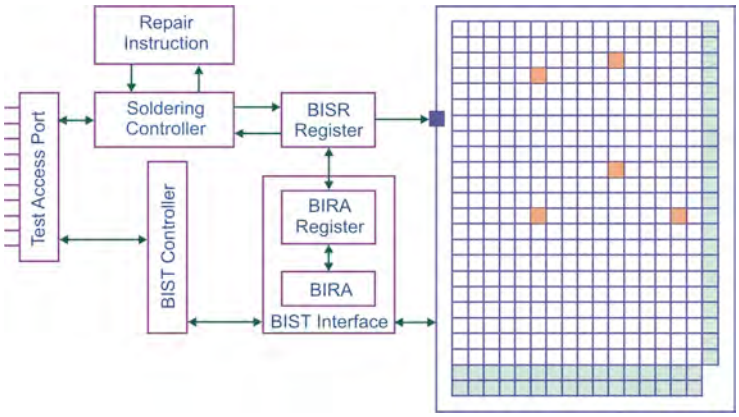


Рис. 4.12. Схема вбудованого аналізу та відновлення пам’яті

Ідея алгебрологічного методу зводиться до оптимального заміщення дефектних елементів матриці пам'яті шляхом вирішення задачі покриття дефектів резервом рядків і стовпців. Функція мети визначається як мінімізація резервних компонентів матриці пам'яті (S), необхідних для її ремонту в процесі функціонування SoC шляхом синтезу діз'юнктівної нормальної форми покриття дефектних елементів з подальшим вибором мінімального кон'юнктивного терма $X^t(R^t, C^t) \in Y$, що задовольняє обмеженням за числом резервних рядків і стовпців S^r_{max}, S^c_{max}, що входять до складу логічного добутку:

$$Z = \min_{t=\overline{1,n}}(|X^t|) \Big| |S^r| + |S^c| \le S_{max}; |S^r| \le S^r_{max}; |S^c| \le S^c_{max},$$

$$X^t \in Y = \{X^1, X^2, ..., X^t, ..., X^n\},\ X^t = (X^t_1 \& X^t_2 \&, ..., \& X^t_i \&, ..., \& X^t_{m_t}).$$

Тут кожен результуючий кон'юнктивний терм функції Y складений з ідентифікаторів рядків і стовпців $X^t = (R^t, C^t)$, що покривають всі дефекти в матриці пам'яті. Кращим рішенням є терм мінімальної довжини за Квайном, в якому містяться як рядки, так і стовпці, що покривають всі дефекти. Зокрема, рішення може не містити рядків (стовпців), коли для ремонту пам'яті достатньо лише наявних стовпців (рядків) з резерву матриці пам'яті. Модель процесу визначення мінімального числа резервних компонентів, що покривають всі виявлені дефекти в матриці пам'яті, зводиться до наступних пунктів:

1. Перетворення двовимірної моделі дефектів матриці пам'яті в таблицю покриття дефектів резервними рядками і стовпцями матриці. Для досягнення поставленої мети розглядається топологічна модель пам'яті у вигляді матриці, ідентифікуючої виявлені дефекти:

$$M = \left|M_{ij}\right|, M_{ij} = \begin{cases} 1 \leftarrow T \oplus f = 1; \\ 0 \leftarrow T \oplus f = 0. \end{cases} \tag{4.15}$$

Тут координата матриці відзначається 1, якщо функція справної поведінки f комірки і тестова реакція дають одиничне значення, що відповідає дефекту. Після фіксації всіх несправностей виконується побудова таблиці покриття дефектів $Y = \left|Y_{ij}\right|, i = \overline{1,n}; j = \overline{1,m}$, де стовпці відповідають множині встановлених дефектів m, а рядки є номери стовпців і рядків матриці пам'яті, які мають несправності:

$$Y = \left|Y_{ij}\right|, Y_{ij} = \begin{cases} 1 \leftarrow C_i(R_i) \cap F_j \neq \varnothing; \\ 0 \leftarrow C_i(R_i) \cap F_j = \varnothing. \end{cases} \tag{4.16}$$

Замість компонентів двовимірної метрики C і R використовується одновимірний вектор, конкатенований з двох послідовностей C і R, потужність якого дорівнює n = p + q:

$$\begin{aligned} X &= C * R = (C_1, C_2, ..., C_i, ..., C_p) * (R_1, R_2, ..., R_j, ..., R_q) = \\ &= X^c * X^r = (X_1, X_2, ..., X_i, ..., X_p, X_{p+1}, X_{p+2}, ..., X_{p+j}, ..., X_{p+q}). \end{aligned} \tag{4.17}$$

При цьому між елементами вихідних наборів (C, R) і результуючим вектором X існує взаємно-однозначна відповідність, що встановлена в першому стовпці матриці Y. Слід зауважити, що перетворення виконується лише для зручності розгляду і подальшої побудови диз'юнктивної нормальної форми в рамках одноманітності змінних, що формують булеву функцію. Якщо дану процедуру не виконувати, то функція буде визначена на двох типах змінних, що містять стовпці і рядки матриці пам'яті.

2. Побудова КНФ для аналітичного, повного і точного рішення задачі покриття. Після формування матриці покриття, що містить нульові і одиничні координати, виконується синтез аналітичної форми покриття шляхом запису КНФ за стовпцями. Тут число кон'юнктивних термів дорівнює кількості стовпців таблиці, а кожна диз'юнкція записується за одиничними значеннями аналізованого стовпця:

$$Y = \bigwedge_{j=1}^{m} (Y_{pj} \vee Y_{qj})_{\{Y_{pj}, Y_{qj}\}=1} = \bigwedge_{j=1}^{m} (X_{pj} \vee X_{qj}). \tag{4.18}$$

З останнього виразу видно, що кожен стовпець має тільки дві координати, які мають одиничне значення, а число логічних добутків дорівнює загальній кількості дефектів m, виявлених в матриці пам'яті.

3. Перетворення КНФ до ДНФ дає можливість побачити всі рішення задачі покриття. Для цього до кон'юнктівної нормальної форми необхідно застосувати операцію логічного множення і правила мінімізації (поглинання) для отримання діз'юнктівної нормальної форми:

$$Y = \bigvee_{j=1}^{w} (k_1^j X_1 \wedge k_2^j X_2 \wedge \ldots \wedge k_i^j X_i \wedge \ldots \wedge k_n^j X_n), k_i^j = \{0,1\}. \qquad (4.19)$$

Тут подано узагальнений запис ДНФ, де в межі число термів дорівнює $w = 2^n$, n - число рядків в множині (C, R) або кількість змінних X в матриці Y, на множині яких формуються всі рішення - покриття дефектів резервними компонентами; якщо k_i^j при X_i приймає значення нуля, то змінна X_i перетворюється на несуттєву.

4. Вибір мінімальних і точних рішень задачі покриття. Процедура пов’язана з визначенням кон'юнктивних термів мінімальної довжини у отриманій ДНФ. Подальше перетворення до рядків і стовпчиківах матриці пам'яті на основі використання раніше введеної відповідності дає можливість записати мінімальне покриття або їх сукупність у двовимірній метриці рядків і стовпців, що відповідає умовам (обмеженням) функції мети на кількість резервних компонентів.

Приклад 4.5. Виконати процес відновлення працездатності матриці пам'яті в частині визначення мінімального числа резервних компонентів, що покривають всі дефекти. Матриця пам'яті з дефектами і резервом [11] зображена на рис. 4.13.

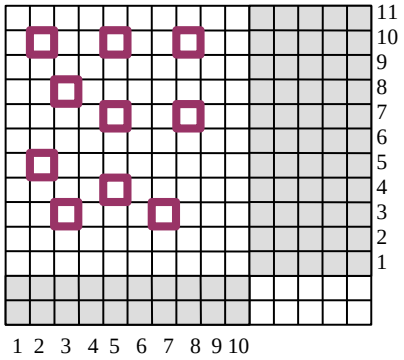


Рис. 4.13. Матриця пам’яті з дефектами і резервом

Матриця має обмеження на можливість діагностування та відновлення працездатності десяти дефектних комірок, які визначаються двома рядками і п'ятьма стовпцями. Відповідно до пункту 1 моделі процесу визначення мінімального числа резервних компонентів, що покривають всі виявлені дефекти в матриці пам'яті, будується таблиця покриття десяти дефектів $F = (F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10})$ одинадцятьма рядками:

$Y =$

X_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
$C_2 \to X_1$				1						1
$C_3 \to X_2$		1					1			
$C_5 \to X_3$			1			1			1	
$C_7 \to X_4$	1									
$C_8 \to X_5$					1			1		
$R_3 \to X_6$	1	1								
$R_4 \to X_7$			1							
$R_5 \to X_8$				1						
$R_7 \to X_9$					1	1				
$R_8 \to X_{10}$							1			
$R_{10} \to X_{11}$								1	1	1

, (4.20)

що подані у вигляді конкатенації столбців C та рядків R, які знаходяться у взаємно-однозначній відповідності з вектором змінних X:

$$C * R = (C_2, C_3, C_5, C_7, C_8) * (R_3, R_4, R_5, R_7, R_8, R_{10}) \approx$$
$$\approx X = (X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}). \quad (4.21)$$

У відповідності з таблицей покриття виконується побудова КНФ, терми якої записані за одиничним значенням столбців:

$$Y = (X_4 \vee X_6)(X_2 \vee X_6)(X_3 \vee X_7)(X_1 \vee X_8)(X_5 \vee X_9) \&$$
$$\& (X_3 \vee X_9)(X_2 \vee X_{10})(X_5 \vee X_{11})(X_3 \vee X_{11})(X_1 \vee X_{11}). \quad (4.22)$$

Наступні перетворення, пов'язані з отриманням діз'юнктівної нормальної форми, грунтуються на застосуванні тотожностей булевої алгебри, які дозволяють виконати: логічне перемножування всіх десяти співмножників і подальшу мінімізацію термів ДНФ шляхом застосування оператора співграней $(ab \vee a\bar{b} = a)$, аксіом поглинання, виключення однакових термів. Опустивши проміжні обчислення, остаточний результат можна привести до наступного вигляду:

$$\begin{aligned} Y = {} & X_1 X_2 X_3 X_4 X_5 \vee X_2 X_3 X_4 X_5 X_8 X_{11} \vee X_1 X_2 X_4 X_9 X_3 X_{11} \vee \\ & \vee X_1 X_3 X_2 X_4 X_9 X_{10} X_{11} \vee X_1 X_7 X_{10} X_{11} X_6 X_9 \vee X_6 X_9 X_7 X_8 X_{10} X_{11} \vee \\ & \vee X_2 X_4 X_9 X_3 X_8 X_{11} \vee X_1 X_2 X_4 X_9 X_7 X_{11} \vee X_2 X_4 X_9 X_7 X_8 X_{11} \vee \\ & \vee X_3 X_2 X_4 X_9 X_8 X_{10} X_{11} \vee X_1 X_2 X_4 X_9 X_7 X_{10} X_{11} \vee X_1 X_2 X_3 X_5 X_6 \vee \\ & \vee X_1 X_3 X_5 X_6 X_{10} \vee X_2 X_3 X_5 X_6 X_8 X_{11} \vee X_3 X_5 X_6 X_8 X_{10} X_{11} \vee \\ & \vee X_1 X_2 X_3 X_{11} X_6 X_9 \vee X_1 X_3 X_{10} X_{11} X_6 X_9 \vee X_2 X_3 X_8 X_{11} X_6 X_9 \vee \\ & \vee X_1 X_2 X_7 X_{11} X_6 X_9 \vee X_2 X_7 X_8 X_{11} X_6 X_9 \vee X_3 X_8 X_{10} X_{11} X_6 X_9. \end{aligned} \quad (4.23)$$

Вибір термів мінімальної довжини, що у даному випадку містять 5 змінних, формує множину мінімальних рішень:

$$Y = X_1 X_2 X_3 X_4 X_5 \vee X_1 X_2 X_3 X_5 X_6 \vee X_1 X_3 X_5 X_6 X_{10}. \quad (4.24)$$

Трансформування отриманої функції до покриття, яке містить позначення змінних у вигляді рядків і стовпців матриці пам'яті, дозволяє зобразити згадані терми у формі:

$$Y = C_2C_3C_5C_7C_8 \vee C_2C_3C_5C_8R_3 \vee C_2C_5C_8R_3R_8. \qquad (4.25)$$

Всі отримані мінімальні рішення задовольняють вимогам за обмеженнями на число резервних компонентів, визначених числами:

$$(\left|C^r\right| \le 5) \,\&\, (\left|R^r\right| \le 2).$$

Інші рішення, визначені в ДНФ, що не представляють інтересу, остільки вони мають неоптимальне покриття дефектних комірок, яке визначається числом змінних (рядки + стовпчики) в термах, більше п'яти. Подальша технологія вбудованого ремонту дефектних комірок полягає в електричному перепрограмуванні дешифратора адреси стовпчика чи рядка матриці пам'яті. Стосовно до пам'яті, зображеної на рис. 4.13, процедура запису або зчитування інформації при зверненні до будь-якої комірки стовпця 2 буде переадресовано до резервного стовпцю 11. Відповідно останньому отриманому рішенню у вигляді першого терма ДНФ функції Y будуть замінені та інші дефектні стовпці на справні з резерву пам'яті: 3 – на 12; 5 – на 13; 7 – на 14; 8 – на 15.

Обчислювальна складність алгебрологічного методу відновлення працездатності в частині рішення задачі покриття [35] визначається наступним виразом:

$$Q = 2^{|F|} + |C + R| \times 2^{|F|}, \qquad (4.26)$$

де $2^{|F|}$ - витрати, пов'язані з синтезом ДНФ шляхом логічного перемноження виключно двокомпонентних диз'юнкцій (координата дефекту визначається номером рядка і стовпчика), число яких дорівнює кількості дефектних комірок; де $|C + R| \times 2^{|F|}$ - верхня межа обчислювальних затрат, необхідних для мінімізації отриманої ДНФ на граничній множині змінних, що дорівнює сумарному числу рядків і стовпців $|C + R|$.

В гіршому випадку, коли координати всіх дефектних комірок за рядками і стовпчиками не корельовані, - унікальні, наприклад, діагональні дефекти, обчислювальна складність матричного методу стає залежною тільки від числа дефектних комірок, а її аналітичний запис трансформується до наступного вигляду:

$$Q = 2^{|F|} + |C + R| \times 2^{|F|}\Big|_{|C+R| \le 2\times|F|} = 2^{|F|} + 2 \times |F| \times 2^{|F|} = 2^{|F|} \times (1 + 2 \times |F|). \qquad (4.27)$$

Якщо замість потужності множини дефектів записати їх число, дорівняноє m, тоді попередній вираз подається в більш простій формі:

$$Q = 2^m \times (1 + 2 \times m) = 2^m (2m + 1). \qquad (4.28)$$

Згідно технології вбудованого сервісного обслуговування функциональних модулів цифрових систем на кристалах, матричний метод відновлення працездатності на основі вирішення задачі покриття імплементрується у кристал, як один з компонентів I-IP, націленого на підтримку працездатності матричної пам'яті і SoC в цілому .

4.8. Висновки

У частині діагностування логічних компонентів і пам'яті.

Запропоновано алгебро-логічний метод вбудованого діагностування дефектів у функціональних блоках SoC і дві його модифікації, що використовують аналіз ТН з метою зменшення обсягу і наступних обчислень, пов'язаних з побудовою ДНФ, яка формує всі розв’язки по встановленню діагнозу функціональностей SoC у реальному масштабі часу. Описані в [54] методи наближення та виключення містять дві процедури, орієнтовані на діагностування одиночних або кратних дефектів. Запропонований алгебро-логічний метод є універсальним за відношенням до числа дефектів, присутніх в схемі, і дозволяє в рамках однієї процедури поставити точний діагноз, який вказує на одиничну або кратну несправність, які в сукупності повинні покривати всі одиничні координати вектора експериментальної перевірки.

Запропоновано скорочена інфраструктура сервісного обслуговування функнальних SoC. Вона відрізняється мінімальним набором процесів вбудованого діагностування у реальному масштабі часу і дає можливість здійснювати сервіси: тестування штатних функцій на основі генеруються вхідних послідовностей і аналіз вихідних реакцій; діагностування із заданою глибиною пошуку дефектів шляхом використання мультизонду стандарту IEEE 1500; моделювання дефектів з метою забезпечення перших двох процедур на основі таблиці несправностей.

Запропоновано мепінг-модель процесу синтезу тестів, яка відрізняється використанням бібліотеки вбудованих тестових генераторів для функціональних SoC, що дає можливість істотно скоротити час побудови тестів, призначених для верифікації штатних режимів і перевірки дефектів.

В частині ремонту пам'яті.

Пам'ять SoC у найближчому майбутньому складатиме більше 90% обсягу кристала, орієнтованого на використання гнучких програмних засобів. Актуальною видається розробка не тільки моделей і методів швидкого і точного діагностування, а й створення технологій для здійснювання ремонту дефектних комірок вбудованими засобами сервісного обслуговування в

реальному часі і на всіх стадіях життєвого циклу виробу. Це дозволить суттєво зменшити кількість висновків чіпа, підвищити вихід придатної продукції, зменшити час виходу виробу на ринок - time-to-market, скоротити витрати на сервісне обслуговування, а також виключити зовнішні засоби діагностування та ремонту.

Алгебро-логічний метод відновлення працездатності пам'яті грунтується на рішенні задачі покриття дефектних комірок резервними елементами шляхом використання апарату булевої алгебри. Метод має квадратичну обчислювальну складність і може бути апаратурно реалізований як в програмному виконанні за межами кристала, так і всередині нього у вигляді додаткового сервісного модуля корекції дефектів, що дозволяє автоматично виконувати відновлення працездатності елементів пам'яті в процесі функціонування.

Класична задача покриття оперує двома одновимірними векторами (X, F), коли оператор покриття P дозволяє знайти мінімальну підмножину компонентів X, що покривають своєю сукупною функціональністю всі елементи з F: $X_{min} = P(X,F) \leftarrow X \cap F = X_{min}$. Формулювання проблеми покриття властивостей одновимірного вектора F двовимірною матрицею $M = (C \times R)$ потребує приведення обох компонентів до єдиної метриці - такій системі координат, яка є спільним знаменником для обох структур. Природно, що такою метрикою для матриці $M = (C \times R)$ і вектора F є одновимірна структура. Тому в даному випадку апріорі необхідно здійснити перетворення двовимірної структури (матриці дефектів пам'яті) $M = (C \times R)$ до одновимірної шляхом виконання операції конкатенації $X = (C * R)$ з метою подальшого вирішення класичної задачі покриття за допомогою формальних дій, що визначаються оператором $X_{min} = P(X,F)$.

Запропоновано метод оптимального відновлення працездатності дефектів пам'яті, який відрізняється від аналогів застосуванням алгебро-логічної технології покриття несправностей двовимірною топологією матриці пам'яті, що дозволяє отримувати мінімальні та повні рішення для подальшого ремонту в реальному масштабі часу, заснованого на використанні резервних компонентів у вигляді рядків і стовпців пам'яті.

Практична значущість методу полягає в його імплементації в інфраструктуру сервісного обслуговування функціональних блоків цифрової системи на кристалі. Це дозволяє істотно (на 5-10%) підвищити відсоток виходу придатної продукції на ринок електронних технологій шляхом відновлення працездатності дефектних кристалів пам'яті в процессі виробництва та експлуатації, а також збільшити життєвий цикл пам'яті шляхом її ремонту в реальному масштабі часу.

Вбудований ремонт орієнтований на всі компоненти системи, що мають адресу: пам'ять, мультипроцесори, матричні процесори. Якщо необхідно ремонтувати інші структури, їх слід перепроектувати з урахуванням

Адресуємий компонентів. Адресується і регулярність компонентів перетворює систему в надійну, робастності, ремонтопридатність і живучу.

Алгебро-логічне зображення задачі покриття має привабливість, спрямовану на оптимальне вирішення всіх задач синтезу та аналізу складних систем, де існує проблема меппінга (покриття): 1) специфікації - набором бібліотечних компонентів; 2) дефектів - тестовими послідовностями; 3) функціональностей - Testbench; 4) несправних елементів - резервними; 5) станів об'єкту - лініями спостереження.

З метою зменшення розмірності проблеми меппінга необхідно структурувати вихідну модель шляхом введення ієрархії, що характерно і повсімісно використовується в автоматизації проектування систем (ESL-, TLM-технології).

Привабливо виглядає апріорне завдання таблиці несправностей у вигляді булевої функції з точки зору компактності, яка на конкретному векторі експериментальної перевірки трансформується в компактний запис, що визначає терми ДНФ, як всі можливі рішення несправних компонентів, що підлягають ремонту.

Ервант Зоріан (EWDTS '2007, Yerevan): «В даний час основна проблема ремонту системи на кристалі полягає в розробці технологій і методів вбудованого відновлення працездатності логіки, яка займає не більше 10% від площі кристала».

Подальші дослідження орієнтовані на розробку ремонтопридатної структури системи і апаратного модуля BIRA для вбудованого відновлення працездатності будь-яких компонентів при виникненні дефектів на стадії виробництва і експлуатації.

4.9. Література

1. Abramovici M., Breuer M.A. and Friedman A.D. Digital System Testing and Testable Design.– Computer Science Press.– 1998.– 652 p.

2. Aitken Robert C. Modeling the Unmodelable: Algorithmic Fault Diagnosis // IEEE Design and Test of Computers.– 1997.– P.98-103.

3. Armstrong D.B. A Deductive Method for Simulating Faults in Logic Circuits // IEEE Transactions on Computers.– 1972.– P.464-471.

4. Bayraktaroglu Ismet, Orailoglu Alex. The Construction of Optimal Deterministic Partitionings in Scan-Based BIST Fault Diagnosis: Mathematical Foundations and Cost-Effective Implementations // IEEE Transactions on Computers.– 2005.– P.61-75.

5. Benso Alfredo, Carlo Stefano Di, Prinetto Paolo, Zorian Yervant. A Hierarchical Infrastructure for SoC Test Management // IEEE Design and Test of Computers.– 2003.– P. 32-39.

6. Bergeron J. Writing testbenches: functional verification of HDL models.– Springer.– 2003.

7. Bernardi Paolo, Veiras Bolzani Leticia Maria, Rebaudengo Maurizio, Sonza Reorda Matteo, Vargas Fabian Luis, Violante Massimo. A New Hybrid Fault Detection Technique for Systems-on-a-Chip // IEEE Transactions on Computers.– 2006.– P. 185-198.

8. Cha Hungse, Rudnick Elizabeth M., Patel Janak H., Iyer Ravishankar K., Choi Gwan S. A Gate-Level Simulation Environment for Alpha-Particle-Induced Transient Faults // IEEE Transactions on Computers.– 1996.– P. 1248-1256.

9. Chandramouli R. Infrastructure IP design for repair in nanometer technologies // IEEE Design and Test of Computers.– 2005.– P. 17.

10. Clark C. J., Ricchetti Mike. Infrastructure IP for Configuration and Test of Boards and Systems // IEEE Design and Test of Computers.– 2003.– P. 78-87.

11. Douglas Densmore, Roberto Passerone, Alberto Sangiovanni-Vincentelli. A Platform-Based taxonomy for ESL Design // Design&Test of computers.– September-October, 2006.– P. 359-373.

12. Francisco DaSilva, Yervant Zorian, Lee Whetsel, Karim Arabi, Rohit Kapur. Overview of the IEEE P1500 Standard // ITC International Test Conference.– 2003.– P.988–997.

13. Ghosh Sumit. Behavioral-Level Fault Simulation // IEEE Design and Test of Computers.– 1988.– P. 31-42.

14. Hahanov V., Kteaman H., Ghribi W., Fomina E. HEDEFS – Hardware embedded deductive fault simulation // Proceedings of the 3rd IFAC Workshop, Rydzyna, Poland.– 2006.– P. 25-29.

15. Hahanov V., Hahanova I., Obrizan V. High-performance deductive fault simulation method // Proceedings of the 10 IEEE European test symposium.–Tallinn, Estonia.– 2006.– P. 91-96.

16. Hahanov V.I., Hahanova I.V., Khan S.U., Obrizan V.I. Topological fault simulation method // Proceedings of the 11th International Conference Mixdes Design of Integrated Circuits and Systems.– Szczecin.– 2004.– P. 211-214.

17. Hahanov V.I., Babich A.V., Hyduke S.M. Test Generation and Fault Simulation Methods on the Basis of Cubic Algebra for Digital Devices // Proceedings of the Euromicro Symposium on Digital Systems Design DSD 2001.– Warsaw, Poland.– 2001.– P. 228-235.

18. Hamdioui S., Gaydadjiev G. N., Van de Goor A. J.. (2004) The State-of-the-art and Future Trends in Testing Embedded Memories // Records IEEE Interna-

tional Workshop on Memory Technology, Design, and Testing, San Jose, CA, August.– P. 54-59.

19. Hansen, M.C.; Yalcin, H.; Hayes, J.P. Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering // Design & Test of Computers, IEEE 16.3.– 1999.– P. 72 – 80.

20. Harris Ian G. Fault Models and Test Generation for Hardware-Software Co-validation // IEEE Design and Test of Computers.– 2003.– P. 40-47.

21. Huang Rei-Fu, Chen Chao-Hsun, Wu Cheng-Wen. Economic Aspects of Memory Built-in Self-Repair // IEEE Design and Test of Computers.– 2007.– P. 164-172.

22. I. Tomoo, M. Satoshi, Fujiwara Hideo. Universal Fault Diagnosis for Lookup Table FPGAs // IEEE Design and Test of Computers.– 1998.– P. 39-44.

23. Jain Sunil, Stroud Charles. Built-in Self Testing of Embedded Memories // IEEE Design and Test of Computers.– 1986.– P. 27-37.

24. Khoche Ajay, Sherlekar S.D., Venkatesh G., Venkateswaran R. A Behavioral Fault Simulator for Ideal // Design and Test of Computers.– 1992.– P. 14-21.

25. Levendel Y.H., Menon P.R. Comparison of fault simulation methods – Treatment of unknown signal values // Journal of digital systems. 4.– 1980.– P. 443-459.

26. Menon P.R., Chappell S.G. Deductive Fault Simulation with Functional Blocks // IEEE Transactions on Computers.– 1978.– P. 689-695.

27. Niggemeyer Dirk, Rudnick Elizabeth M. Automatic Generation of Diagnostic Memory Tests Based on Fault Decomposition and Output Tracing // IEEE Transactions on Computers.– 2004.– P. 1134-1146.

28. Nishida T., Miyamoto S., Kozawa T., Satoh K. RFSIM: Reduced fault simulator // IEEE Transactions on computer-aided design. CAD-6.3.– 1987.– P. 392-402.

29. Novak Ondrey, Gramatova Elena, Ubar Raimund. Handbook of testing electronic systems.– Czech University Publishing House.– 2005.– 402 p.

30. Ozguner F. Deductive Fault Simulation of Internal Faults of Inverter-Free Circuits and Programmable Logic Arrays // IEEE Transactions on Computers.– 1986.– P. 70-73.

31. Pomeranz Irith, Reddy Sudhakar M. On Fault Simulation for Synchronous Sequential Circuits // IEEE Transactions on Computers.– 1995.– P. 335-340.

32. Pomeranz Irith, Reddy Sudhakar M. Aliasing Computation Using Fault Simulation with Fault Dropping // IEEE Transactions on Computers.– 1995.– P. 139-144.

33. Pomeranz Irith, Reddy Sudhakar M. On Maximizing the Fault Coverage for a Given Test Length Limit in a Synchronous Sequential Circuit // IEEE Transactions on Computers.– 2004.– P. 1121-1133.

34. Rashinkar P., Paterson P., Singh L. System-on-chip Verification: Methodology and Techniques.– Kluwer Academic Publishers.– 2002.– 324 p.

35. Rossen K. Discrete Mathematics and its Applications.– McGraw Hill.– 2003.– 824 p.

36. Silberman G.M., Spillinger I. Functional Fault Simulation as a Guide for Biased-Random Test Pattern Generation // IEEE Transactions on Computers.– 1991.– P. 66-79.

37. Shoukourian S., Vardanian V., Zorian Y. SoC Yield Optimization via an Embedded-Memory Test and Repair Infrastructure // IEEE Design and Test of Computers.– 2004.– P. 200-207.

38. Shoukourian Samvel, Vardanian Valery, Zorian Yervant. SoC Yield Optimization via an Embedded-Memory Test and Repair Infrastructure // IEEE Design and Test of Computers.– 2004.– P. 200-207.

39. Tabatabaei Sassan, Ivanov Andre. Embedded Timing Analysis: A SoC Infrastructure // IEEE Design and Test of Computers.– 2002.– P. 24-36.

40. Treuer Robert, Agarwal Vinod K. Built-In Self-Diagnosis for Repairable Embedded RAMs // IEEE Design and Test of Computers.– 1993.– P. 24-33.

41. Venkataraman Srikanth, Scott Brady Drummonds. Poirot: Applications of a Logic Fault Diagnosis Tool // IEEE Design and Test of Computers.– 2001.– P. 19-30.

42. Waicukauski John, Lindbloom Eric, Rosen Barry, Iyengar Vijay. Transition Fault Simulation // IEEE Design and Test of Computers.– 1987.– P. 32-38.

43. Walczak K. Deductive Fault Simulation for Sequential Module Circuits // IEEE Transactions on Computers.– 1988.– P. 237-239.

44. Wang X., Hill F.G., Mi Zh. A sequential circuit fault simulation by surrogate fault propagation // Proc. IEEE International test conference, IEEE Computer society.– 1989.– P. 9-18.

45. Youngs Lynn, Paramanandam Siva. Mapping and Repairing Embedded-Memory Defects. IEEE Design and Test of Computers. – 1997. – P. 18-24.

46. Youngs L., Paramanandam S. Mapping and Repairing Embedded-Memory Defects // IEEE Design and Test of Computers.– 1997.– P. 18-24.

47. Zhao Jun, Meyer Fred J., Lombardi Fabrizio. Analyzing and Diagnosing Interconnect Faults in Bus-Structured Systems // IEEE Design and Test of Computers.– 2002.– P. 54-64.

48. Zhong Y., Dropsho S.G., Shen X., Studer A., Ding C. Miss Rate Prediction Across Program Inputs and Cache Configurations // IEEE Transactions on Computers.– 2007.– P. 328-343.

49. Zorian Yervant. What is Infrustructure IP? // IEEE Design & Test of Computers.– 2002.– P. 5-7.

50. Zorian Yervant, Gizopoulos Dmytris. Gest editors' introduction: Design for Yield and reliability // IEEE Design & Test of Computers. – 2004. – P. 177-182.

51. Zorian Y., Shoukourian S. Embedded-Memory Test and Repair: Infrastructure IP for SoC Yield // IEEE Design and Test of Computers.– 2003.– P. 58-66.

52. Zorian Yervant, Shoukourian Samvel. Embedded-Memory Test and Repair: Infrastructure IP for SoC Yield // IEEE Design and Test of Computers.–2003.– P. 58-66.

53. Zorian Yervant. Guest Editor's Introduction: Advances in Infrastructure IP // IEEE Design and Test of Computers.– 2003.– 49 p.

54. Автоматизация диагностирования электронных устройств/ Ю.В.Малышенко и др./ Под ред.В.П.Чипулиса. – М.: Энергоатомиздат.– 1986.– 216 с.

55. Автоматизированное проектирование цифровых устройств / С.С.Бадулин, Ю.М.Барнаулов и др./ Под ред. С.С. Бадулина. – М.: Радио и связь.– 1981.– 240 с.

56. Бондаренко М.Ф., Кривуля Г.Ф., Рябцев В.Г., Фрадков С.А., Хаханов В.И. Проектирование и диагностика компьютерных систем и сетей.– К.: НМЦ ВО.– 2000.– 306 с.

57. Пархоменко П.П., Согомонян Е.С. Основы технической диагностики (Оптимизация алгоритмов диагностирования, аппаратурные средства) / Под ред. П.П. Пархоменко.– М.: Энергия.– 1981.– 320 с.

58. Парфентий А.Н., Хаханов В.И., Литвинова Е.И. Модели инфраструктуры сервисного обслуживания цифровых систем на кристаллах // АСУ и приборы автоматики.– 2007.– №138.– С. 83-99.

59. Семенец В.В., Хаханова И.В., Хаханов В.И. Проектирование цифровых систем с использованием языка VHDL.– Харьков: ХНУРЭ.– 2003.– 492 с.

60. Убар Р.Р. Анализ диагностических тестов для комбинационных цифровых схем методом обратного прослеживания неисправностей // Автоматика и телемеханика.– 1977.– №8.– С. 168-176.

61. Хаханов В.И., Хаханова И.В. VHDL + Verilog = Синтез за минуты.– Харьков: СМИТ.– 2007.– 264 с.

62. Хаханов В.И. Техническая диагностика элементов и узлов персональных компьютеров.– К.: ІЗМН.– 1997.– 308 с.

63. Хаханов В.И., Хак Х.М. Джахирул, Масуд М.Д. Мехеди. Модели анализа неисправностей цифровых систем на основе FPGA, CPLD // Технология и конструирование в электронной аппаратуре.– 2001.– №2.– С. 3-11.

64. Хаханов В.И., Сысенко И.Ю., Хак Х.М. Джахирул, Масуд М.Д. Мехеди. Кубическое моделирование неисправностей цифровых проектов на основе FPGA, CPLD // Радиоэлектроника, информатика, управление.– 2001.– №1.– С. 123-129.

65. Хаханов В.И., Сысенко И.Ю., Колесников К.В. Дедуктивно-параллельный метод моделирования неисправностей на реконфигурируемых моделях цифровых систем // Радиоэлектроника и информатика.– 2002.– №1.– С. 98-105.

66. Хаханов В.И., Колесников К.В., Хаханова А.В. BDP-метод моделирования неисправностей для синтеза тестов цифровых проектов // Радиоэлектроника и информатика.– 2002.– №2.– С. 60-66.

67. Хаханов В.И. Убар Р.-Й.Р. Технологии проектирования систем на кристаллах. Моделирование неисправностей сверхбольших цифровых проектов // АСУ и приборы автоматики.– 2002.– №122.– С. 16-35.

5. ТЕХНОЛОГІЇ ТЕСТУВАННЯ SYSTEM-IN-PACKAGE

Розглядається проблема адаптації технологій тестування цифрових систем на кристалах (System on Chip - SoC) для нового конструктивного покоління цифрових систем - System-in-Package (SiP), що дозволяє ефективно і компактно імплементувати в кристали надскладні спеціалізовані обчислювальні і радіочастотні пристрої для ринку електронних технологій. Разом з тим пакет кристалів формує спектр нових завдань сервісного обслуговування SiP-функциональностей в реальному масштабі часу, який істотно відрізняється від процесів вбудованого діагностування SoC [1-3]. У зв'язку з цим пропонується алгебрологічний метод діагностування і відновлення працездатності функціональних логічних блоків FPGA, заснований на використанні таблиць несправностей і їх аналізі в реальному масштабі часу.

Конструктивна технологія System-in-Package (SiP) як система в пакеті криталів стає все більш привабливою на ринку мікроелектроніки для мініатюрної і енергозберігаючої реалізації високопродуктивних програмноапаратних додатків, що включають спеціалізовані обчислювачі, бездротові приймачі, DSP-перетворювачі, персональні сервери.

Еволюція SiP протягом останніх 10 років на ринку EDA призвела до появи істотних переваг (на тлі універсальних і енерговитратних комп'ютерів), таких як: підвищення продуктивності цифрових систем, мініатюризація обсягу, зниження ваги виробу, зменшення затримок розповсюдження сигналів, енергоспоживання і вартості пристрою, що позитивно було оцінено всіма компаніями світу, які входять в сімейство, що формує індекс NASDAQ.

Описані вище переваги досягаються за рахунок наступних найбільш суттєвих і не завжди позитивних властивостей. 1) Апаратне ускладнення цифрової системи в пакеті, що нараховує вже мільярди вентилів, до 700 мільйонів транзисторів на одному кристалі. 2) Висока вартість проектування та виробництва (Design and Manufacturing Cost). 3) Низька технологічна привабливість існуючих на ринку моделей (Unavailable Simulation Models), методів аналізу та прийняття рішень (Decision Making Process). 4) Відсутність гарантії якості (Bare Die Quality) використовуваних силіконових пластин. 5) Низький рівень виходу придатних виробів SiP (Low Assembly Yields). 6) Технологічна складність процесів відновлення працездатності (Complex Repair Process). 7) Обмежений ринок засобів проектування (Limited CAD Tools). 8) Наявність проблем в управлінні відводом тепла (Thermal Management).

Нове конструктивне виконання цифрових систем в пакеті кристалів додає розробникам апаратури технологічні проблеми тестування (SiP Test Challenges) пластин (Wafer / Die Test), підкладки (Substrate Test) і функціональних модулів (Module Test). Крім того, існує багато питань, пов'язаних зі з'єднанням

силіконових кристалів (silicon die) в цифрову електронну систему. В першу чергу тут актуальними є такі завдання: 1) Механічний захист нутрощів силіконових кристалів. 2) Зручності та переваги використання, збірки і тестування (handling, assembly, test). 3) Розсіювання тепла шляхом його відводу від гарячих компонентів (removal of wasted heat).

Однією з найбільш актуальних проблем на ринку електронних технологій в галузі тестування, діагностування та ремонту цифрових систем, оформлених у вигляді SoC або SiP, є відновлення працездатності логіки користувача, яка в 2014 році становитиме лише 6% від розмірності проекту, розміщеного на кристалі. Тим не менш, сервісне обслуговування функціональних блоків, пам'яті та ремонту компонентів, що відмовили, в реальному масштабі часу залишається відкритою темою для дослідників.

Спочатку цифрові системи в пакетах кристалів розроблялися з метою їх використання в екстремальних умовах, коли працездатність компонентів необхідно відновлювати в процесі функціонування в реальному масштабі часу, дистанційно, автономно, без використання зовнішнього устаткування і втручання людини. Тому для забезпечення якості та надійності системи створюється інфраструктура сервісного обслуговування (Infrastructure Intellectual Property - IIP), що включає технології тестування, діагностування та відновлення працездатності SiP.

Мета - розробка технології відновлення працездатності логічних компонентів (Reused and New Logic) цифрової системи на кристалах SoC & SiP в реальному масштабі часу, на основі використання надлишкових площ кристалів програмованої логіки.

Задачі: 1) Аналіз конструктивно-технологічних особливостей SiP. 2) Ана-ліз методів тестування, діагностування та ремонту компонентів SiP. 3) Технології діагностування логічних блоків цифрових систем в пакетах на основі FPGA (Field Programmable Gate Array). 4) Алгебрологічний метод відновлення працездатності логічних блоків цифрової системи на основі FPGA. 5) Метод обходу матриці логічних блоків для покриття дефектних компонентів FPGA ремонтними клітинами. 6) Практичні результати і майбутні напрямки дослідження.

5.1. Вступ до проблеми

За останні роки з'явилися десятки робіт, які розглядають питання, пов'язані з проблемою тестування, діагностування та ремонту цифрових систем на кристалах і в пакетах (SoC, SiP). Особливе місце тут обіймає задача тестування і ремонту логічних компонентів цифрових систем через технологічну складність відновлення працездатності логічних елементів при виникненні в них дефектів. Існуючі рішення, запропоновані в сучасних публікаціях, зводяться до наступної класифікації:

1. Дублювання логічних елементів або областей кристала, що приводить до подвоєння апаратної реалізації функціональності. При фіксації несправного елемента чи області здійснюється перемикання на справний компонент за до-

помогою мультиплексора [4]. Запропоновані Xilinx моделі FPGA (Field Programmable Gate Array) можуть бути застосовані також при ремонті компонентів FPGA від компанії Altera. Основна одиниця виміру при ремонті - стовпець або рядок.

2. Використання генетичних алгоритмів для діагностування і відновлення працездатності на основі автономної реконфігурації кристала FPGA без використання зовнішніх пристроїв керування [5]. Надійність діагностування дефектів дорівнює 99%, час ремонту - 36 мілісекунд замість 660 секунд, необхідних для стандартного конфігурування проекту.

3. Відновлення працездатності кристалів FPGA, не критичне до часу, шляхом заміни локальних CLB на надлишкові запасні компоненти запропоновано в [6,7]. Доступний рівень об'єднання CLB, який підлягає заміні, для критично важливих додатків становить близько тисячі геологічних елементів.

Технології відновлення працездатності логічної частини цифрових систем, імплементованих в кристали програмованої логіки, основани на існуванні або внесенні надмірності LUT-компонентів FPGA після виконання процедур Place and Route. Фізичні дефекти кристала, що у процесі його виготовлення або експлуатації, проявляються як логічні або часові і призводять до неправильного функционирования цифрового виробу. Дефекти прив'язуються не тільки до вентилів або LUT-компонентам, але і до конкретного місця на кристалі. Ідея ремонту цифрової структури зводиться до виключення дефектного місця при повторном виконанні процедур Place and Route після встановлення діагнозу. При цьому можливі дві технології ремонту: 1) Заборона дефектної області шляхом написання керуючих скриптів для досить тривалої процедури Place and Route, що не завжди прийнятно для цифрових систем, що працюють в реальному масштабі часу. Однак такий підхід прийнятний і орієнтований на виключення дефектних областей будь-якої кратності, що мають місце на кристалі. Заборона таких областей при повторному виконанні процедури Place and Route призводить до відновлення працездатності. 2) Для цифрових систем реального часу виконання процедури Place and Route, спрямованої на відновлення працездатності, може привести до катастрофічних наслідків. Потрібен технологічний підхід, здатний відновити функціональність цифрової системи за мілісекунди, необхідно для перепрограмування FPGA шляхом подачі нового бітстріма, що виключає дефектні області з формування функціональності. Такий підхід заснований на попередній підготовці всіх можливих бітстримів, ізолюючих майбутні дефектні області шляхом їх потрапляння в надлишкову нефункціональну область кристала. Чим більше резервна область, тим менше число бітстрімів - проектних варіантів, прив'язаних до топології, яке необхідно згенерувати апріорі. Що стосується кратних дефектів, що не покриваються однієї резервної областю, тут слід сегментувати цифровий проект, розбивши його попередньо на непересічні частини, які мають власні карти Place and Route. В даному випадку можна ремонтувати цифрову систему, яка має для n розподілених на пластині дефектів n резервних сегментів. При цьому загальна площа кристала являє збій n+m однакових частин.

5.2. Технології відновлення працездатності SiP

Кристали FPGA широко використовуються для швидкого прототипування з метою зменьшения вартості проектування складних цифрових систем [7]. Модульна структура FPGA дозволяє виконувати дистанційне переспрограмування для заміни дефектних логічних ресурсів (блоків) справними резервними елементами. Зазначена властивість дає можливість суттєво підвищити відмовостійкість апаратних додатків. Особливо це важливо для систем, експлуатованих у критичних умовах (космічний простір, екологічно небезпечне для людини середовище). Природно, що промислово вироблювані FPGA можуть бути піддані повному тестуванню ще до реалізації етапу їх програмування. Після імплементації в кристал заданої функціональності виконується оперативний контроль FPGA і автономне тестування компонентів, завдяки наявності спеціальних вбудованих засобів.

Систематичні (константні) і перемежовувані (нестійкі) несправності можуть бути виявлені і локалізовані шляхом використання вбудованих методів тестування. Застосування певної частини ресурсів FPGA як резервних дозволяє замінити дефектну область, в якій виявлена систематична несправність. Заміна відбувається шляхом перепрограмування кристала при збереженні його функціональності. Схема, за якою резервні ресурси розподіляються всередині FPGA і, отже, алгоритм реконфігурування, залежать від типу кристала. Використання процесу часткового конфігурування дозволяє істотно зменшити середній час відновлення працездатності і довжину бітового потоку (bitstream), який застосовується для реконфігурування FPGA. Для вибору ефективної стратегії розподілу резервних ресурсів велике значення має структура міжз’єднань кристала.

В опублікованих роботах [4, 8-13] описані різні технології відновлення працездатності FPGA, однак порівняльний аналіз наведених методів і метрики оцінки їх ефективності тут практично відсутній. Тому далі розглядається об'єкт дослідження, представлений у вигляді FPGA, його структура та особливості застосування, а також існуючі методи діагностування та ремонту, що використовують специфіку перепрограмування для усунення дефектів.

Кристал FPGA є масив комплексних логічних блоків (CLB - Complex Logic Block), кожен з яких зображує собою апаратну реалізацію від двох до восьми таблиць істинності на 4 входи, кожна з яких з'єднана з тригером за допомогою ресурсів трасування, що зображують собою програмовані матриці перемикачів (PSM). З метою забезпечення відмовостійкості і надійності FPGA виконується резервування функціональної частини CLB шляхом апріорного формування підмножини запасних блоків (spares), потужність якого залежить від структури міжз’єднань кристала, що впливають на процедури ремонту блоків FPGA.

Перший тип міжз’єднань орієнтований на матричну топологію CLB, коли оптимальна структура функціональності зображена з'єднанням сусідніх по

вертикалі і горизонталі логічних блоків. Якщо дефектний і резервний блоки не є сусідніми в «манхеттенській» системі координат, то для відновлення працездатності першого CLB необхідно виконати досить складну процедуру трасування, використовуючи матриці перемикачів. Дана структура ідеально підходить для забезпечення гнучкості при виборі CLB при розподілі резервних ресурсів. Недоліки: 1) Велика кількість блоків, через які проходить сигнал, призводить до зниження продуктивності системи через збільшення довжини логічного шляху, який залежить від числа використовуваних проміжних перемикальних матриць. 2) Процес перепрограмування міжз’єднань або трасування є досить складним завданням, коли заміна дефектного логічного блоку вимагає значного часу (десятки хвилин) для пошуку квазіоптімального шляху.

Другий тип міжз’єднань зображено на рис. 5.1. Тут нижній рівень ієрархії зв'язків дозволяє з'єднати сусідні блоки CLB. Групи блоків використовують міжз’єднання вищого рівня ієрархії, що дозволяє зменшити кількість проміжних перемикальних матриць. Дана структура дає можливість ефективно використовувати мозаїчну (tile) та ієрархічну стратегії розподілу резервних ресурсів [7].

Третій тип міжз’єднань заснований на сегментації блоків CLB за допомогою програмованих матриць перемикачів, розбивають логічні компоненти на підмножини, як показано на рис. 5.2. Тут маються 4 сегменти по 9 логічних блоків у кожному. Така структура не вимагає виконання процедури перетрассіровкі, якщо резервний і дефектний блоки розташовані на одній лінії, вертикальної або горизонтальної, в межах одного сегмента. У цьому випадку може бути застосований «мозаїчний» підхід [7], що дозволяє використовувати резервний CLB для відновлення працездатності сусіднього по горизонталі або вертикалі дефектного блоку в межах одного сегмента.

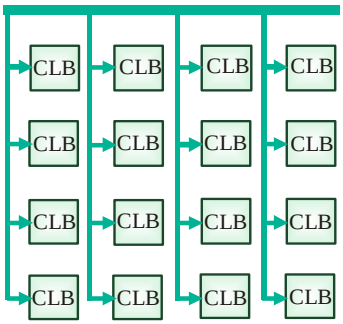


Рис. 5.1. Ієрархічна структура FPGA

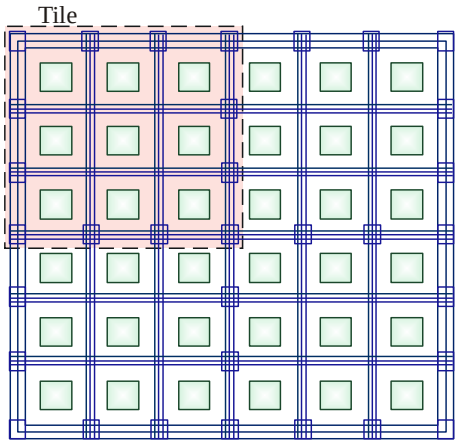


Рис. 5.2. Часткова сегментація зв’зків у структурі FPGA

Цікавим видається також рішення задачі ремонту логічних компонентів цифрових систем на кристалах, пов'язаної з вибором стратегії відновлення, яка зводиться до створення протоколу перепрограмування FPGA. Тут слід розглянути два методи часткового реконфігурування. Перший дозволяє визначити ресурси Atmel AT40K FPGA, які повинні бути перепрограмовані, використовуючи пару керуючих регістрів для вибору координати конкретного блоку у вигляді номера рядка і стовпчика реконфігурованого ресурсу. Після програмування нова конфігурація ресурсів може бути завантажена в керуючі регістри. Дана стратегія характеризується незначним часом виправлення помилки і високим ступенем деталізації в процесі реконфігурування кристала. Другий метод, який використовується компанією Xilinx, полягає в розподілі на стовпці перепрограмованих ресурсів, що може забезпечувати ступінь деталізації (пористість - granularity) не гірше, ніж у попередньому випадку.

Для усунення постійної у часі несправності в кристалі FPGA використовується модель процесу відновлення працездатності, яка зображена на рис. 5.3.

Рис. 5.3. Процедура відновлення працездатності FPGA

Крок 1. Виявлення несправностей в FPGA. До початку реконфігурування виконується процес їх діагностування. Тестування несправностей відбувається з використанням самоперевірюваних схем (self-checking circuits). Додаток, що імплементірується в FPGA, розділяється на частини, до кожної з них додається надмірність, в результаті чого функціональний модуль стає самоперевірюваною схемою щодо дефектів, які можуть виникнути всередині модуля. Ступінь деталізації, або глибина діагностування несправностей, визначається кількістю CLB, підозрюваних в наявності дефектів. Їх пошук здійснюється в процесі безперервного і послідовного циклічного тестування структурних компонентів FPGA, кожен з яких являє собою сукупність блоків, об'єднаних в область самотестування (self-testing areas - STARs). У кожен конкретний момент виконується перевірка тільки однієї області, яка виводиться зі штатного режиму шляхом її заміни резервом, в той час як решта FPGA продовжує функціонувати у нормальному режимі.

Після завершення процедури тестування однієї області здійснюється реконфігурування FPGA з метою функціонального відключення наступної області, яка підлягає тестуванню, і повернення перевіреного структурного компонента до штатного режиму. Описана процедура дозволяє автоматично усувати перемежовувані (transient) несправності за допомогою конфігураційної пам'яті FPGA, яка зберігає інформацію про тестировану область. Оскільки кристал постійно реконфігурується для тестування чергового компонента функціональності FPGA, процедура діагностування має високий ступінь деталізації або глибину пошуку дефектів, яка визначається не більше ніж 10 CLB. Недоліком методу є залежність часу діагностування дефекту від тривалості циклу тестування функціональних компонентів FPGA, яка задається множиною самоперевіряємих областей FPGA, отриманою на стадії тестопридатного проектування. Крім того, наявність затримки між двома подіями - виникненням і виявленням дефекту - може призводити до

невірного діагнозу про працездатність системи у певний момент часу, що призводить до небажаних наслідків для критичних систем реального часу.

Крок 2. Дозволяє розрізнити перемежовуються і систематичні (константні) несправності. Коли пристрій діагностування фіксує факт наявності дефекту, виконується оновлення конфігураційної пам'яті FPGA. Потім виконується ініціалізація таймера, контролюючого MTBF (Mean Time Between Failures - середнє напрацювання на відмову), для розрізнення перемежованих і систематичних несправностей. Якщо дві помилки виявлені в одному місці протягом інтервалу часу меншого, ніж MTBF, фіксується факт наявності систематичної несправності з подальшим виконанням кроків 3 і 4.

Крок 3. У разі виявлення систематичної несправності виконується її діагностування зі ступенем деталізації кращої, ніж при розподілі схеми на самотестовані модулі. Для пошуку дефектного CLB може бути використаний метод, запропонований в [14].

Крок 4. Ремонт дефектного модуля CLB. Механізм відновлення працездатності безпосередньо залежить від архітектури FPGA. При цьому використовуються різні технології у залежності від можливостей часткової або динамічної реконфігурації FPGA, а також від структури бітового струму для перепрограмування кристала і міжз’єднань: 1) ієрархічна модель передбачає формування двох рівнів надмірності. На нижньому рівні виконується розподіл FPGA на клітини (tile) - підматріці блоків CLB, в кожній з них створюються резервні блоки. На верхньому рівні дефектні клітини можуть бути замінені резервними клітинами, як показано на рис. 5.4, а. 2) Оптимальна модель припускає, що резервні CLB можуть бути використані для відновлення працездатності будь-якого дефектного модуля цифрової системи на кристалі FPGA. При цьому немає необхідності виконувати часовитратні процедури перетрассіровки (рис. 5.4, б). Даний підхід інваріантний до структури FPGA, але потребує суттєвого часу відновлення працездатності і додаткового обсягу пам'яті для зберігання попередньо скомпільованого бітового потоку (bitstream), що перепрограмує кристал. Використання bitstream є обов'язковим, оскільки процедура перепрограмування може зачіпати всі ресурси FPGA. Отже, повинен бути реалізований повний алгоритм place-and-route. Дана процедура є часовитратною і не може бути виконана в режимі on-line. Вона реалізується під час компіляції проекту з використанням методів зменшення розмірності попередньо скомпільованого бітового потоку. 3) Модель з надмірністю оперує матрицею блоків CLB, розділеною на стовпці, серед яких є один або кілька надлишкових, використовуваних для відновлення працездатності цифрової системи (рис. 5.4, в). Якщо несправність виявлена у функціональному стовпці, він маркується як дефектний і замінюється резервним. Модель процесу дозволяє виконувати часткове реконфігурірования структури FPGA шляхом розподілу bitstream, як це використовується в кристалах компанії Xilinx. Процедура реконфігурування виконується в реальному масштабі часу. Також є можливість спростити, а значить - прискорити її ре-

алізацію за допомогою укрупнення деталізації функціональних модулів. Недолік: при виявленні дефектного CLB інші справні блоки, що належать стовпцю, також маркуються як дефектні і виводяться з функціонування. 4) Мозаїчна модель. Тут структура FPGA ділиться на клітини, що містять резервні CLB, кожний з яких може відновити один дефектний блок в розглянутій клітці (рис. 5.4, г).

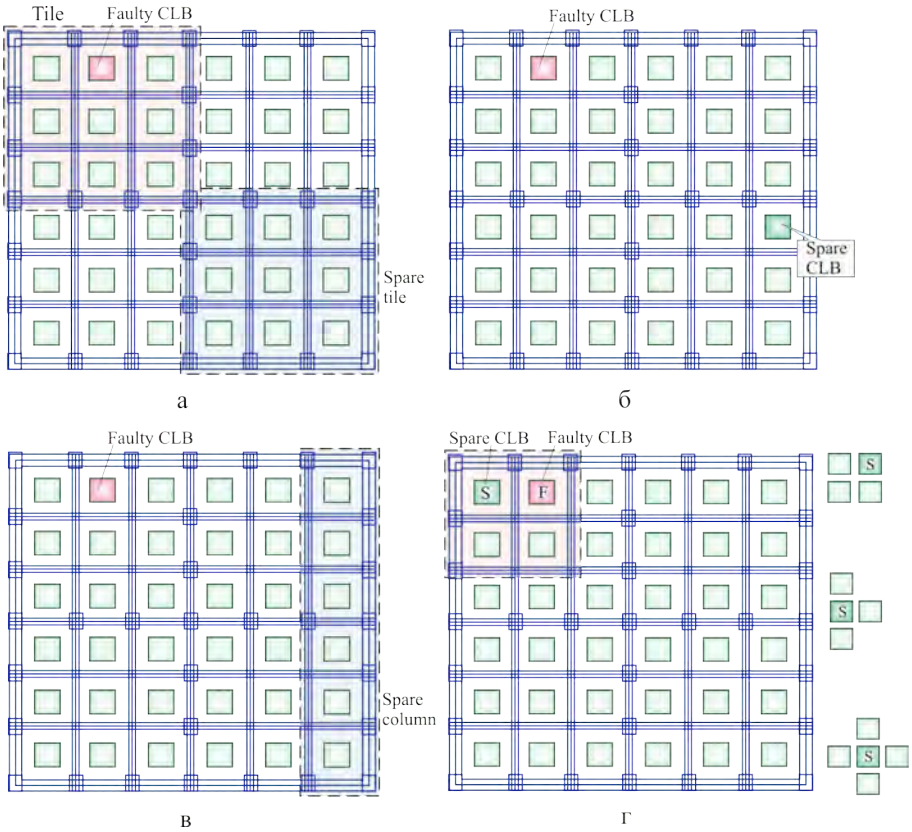


Рис. 5.4. Моделі FPGA для відновлення працездатності

В процесі діагностування визначається місце розташування дефектної області з глибиною деталізації вище, ніж розмір клітини (осередку). Це дає можливість замінити дефектний блок CLB резервним елементом клітини. Реконфигурація клітини забезпечує вихідну функціональність на новій карті. Міжз’єднання за периметром клітини з іншою частиною FPGA повинні бути незмінними в процесі реконфігурації. Такий підхід дозволяє зменшити час відновлення працездатності цифрової системи після виявлення дефекту. Запасні конфігурації генеруються на стадії проектування і зберігаються в пам'яті. Кожна клітина формується як набір блоків CLB і міжз’єднань FPGA

за допомогою опису інтерфейсу, який визначає і забезпечує зв'язки з іншими клітинами кристала. Використання інтерфейсу клітин дозволяє не залучати інші компоненти в процесі реконфігурування, зменшуючи активну пам'ять. Модель дає можливість відновлювати не тільки CLB, а й локальні дефектні міжз'єднання. Помилки глобальних міжз'єднань вимагають інших підходів, оскільки вони перетинають периметр клітин, роблячи їх залежними один від одного. Структура клітини залежить від міжз'єднань FPGA. Як правило, один з декількох CLB, що належать клітині, є резервним елементом. Решта блоків використовується в штатному режимі. При виявленні несправності клітина реконфігурується шляхом вилучення з функціонування дефектного блоку. На рис. 5.4, г показані структури клітини, форма яких відрізняється від квадрата. В кристалах Atmel FPGA використовуються діагональні міжз'єднання блоків, для яких модель, зображена на рис. 5.4, г, є більш кращою [15].

Ієрархічна модель являє собою найбільш загальний випадок відновлення працездатності. Всі інші, описані вище, можуть розглядадтися як її окремі випадки. Оптимальна структура застосовна тільки на нижньому рівні ієрархії, у той час як модель з надмірністю має резервні ресурси тільки на верхньому рівні для заміни дефектного стовпця блоків. Мозаїчна модель має надмірність на нижньому рівні ієрархії, тому один з двох дефектних блоків, розташованих в одній клітці, є невідновлювальним. Для усунення даного недоліку слід застосовувати додаткові резервні клітини.

При використанні дворівневої архітектури цифрової системи в кристалі FPGA ймовірність її справного функціонування при наявності не більше ніж q дефектів, рівного кількості резервних елементів, визначається виразом [7]:

$$P_{nf}(t) = \sum_{i=0}^{q} \binom{m}{i} P_{tile}(t)^{m-i} (1 - P_{tile}(t))^{i},$$

де m – загальне число клітин в FPGA або блоків CLB в клітці; q - кількість резервних клітин або блоків у кожній з них.

Для порівняння чотирьох моделей відновлення працездатності SiP в [7] використовувалася матрична (квадратна) структура FPGA (число рядків і стовпців одно 100) з резервом CLB, рівним 25%. 1) Оптимальна модель: кількість CLB, застосовуваних у процесі функціонування системи, так само 7500, резервних елементів - 2500. 2) Мозаїчне відновлення: кожна клітина (tile) включає 4 CLB, 3 з яких використовуються в процесі функціонування і один є резервним, кількість клітин дорівнює 2500. 3) Модель з надмірністю: кількість стовпців дорівнює 100, кожен стовпець складається з 100 CLB, 75 з яких використовується в роботі і 25 - резервні. 4) Ієрархічна модель: кількість клітин дорівнює 2000, кожна з них містить 5 CLB, 4 з яких - функціональні та один - резервний. Використовується 1875 робочих клітин і 125 резервних. Загальна кількість резервних блоків CLB дорівнює 2500.

Для малих значень інтенсивності відмов надійність відновлення з надлишковістю перевершує надійність ремонту на основі мозаїчної моделі. При збільшенні інтенсивності відмов надійність першого підходу різко падає внаслідок обмеженої кількості резервних елементів, а надійність другого - плавно зменшується завдяки меншій надлишковості блоків CLB для відновлення працездатності. Таким чином, існуючі структури відновлення працездатності можна розділити на дві групи: 1) Моделі, орієнтовані на заміну одиничного елемента (оптимальна і мозаїчна). 2) Моделі, орієнтовані на заміну групи дефектних елементів (ієрархічна і модель з надмірністю).

Реалізація розглянутих у зазначеній вище публікації моделей відновлення працездатності цифрових систем заснована на виконанні часовитратної процедури Place and Route або використанні керуючих регістрів для оперативного реконфігурування структури FPGA в реальному масштабі часу з метою виключення з процесу функціонування дефектних блоків.

5.3. Технології діагностування блоків FPGA

Основна роль в процесі діагностування несправних блоків і компонентів FPGA відводиться вбудованій у кристал інфраструктурі граничного сканування [18], яка орієнтована на вирішення практично всіх завдань сервісного обслуговування SiP-функціональностей, включаючи відновлення працездатності в реальному масштабі часу. Контролер доступу стандарту IEEE 1500 забезпечує моніторинг всіх проблемних внутрішніх ліній проекту за допомогою регістра граничного сканування, число розрядів якого має відповідати наперед заданій глибині діагностування, а, отже, кількості блоків LUT, клітин або кристалів цифрової системи в пакеті (SiP). Модель процесу сервісного обслуговування SiP-функціональності для діагностування дефектів у компонентах кристалів зображено на рис. 5.5.

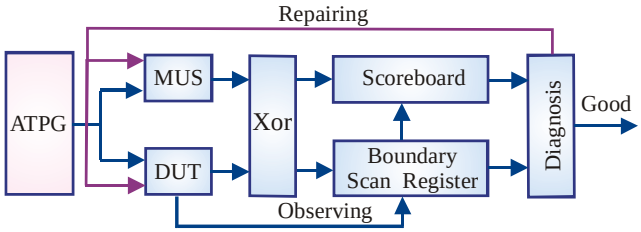


Рис. 5.5. Модель процессу діагностування F-IP

Тут модуль функції Xor аналізує вихідні реакції моделі MUS і реального пристрою DUT на тестові вектори, що надходять від вбудованого генератора вхідних послідовностей. Блок Boundary Scan Register є мультизондом, призначеним для моніторингу стану всіх блоків або клітин, далі - компонентів кристала. Модуль Scoreboard виконує функцію аналізу результатів моніторингу для діагностування і подальшого ремонту компонентів SoC. Результат діагностування SiP є множина несправних блоків, які підлягають

виключенню з функціонального режиму шляхом їх заміни на резервні компоненти.

Цікаве рішення задачі діагностування може бути отримано шляхом застосування булевої алгебри для аналізу таблиці несправностей (ТН) M, що є декартовим добутком тесту T на множину наперед заданих дефектів F. Вектор експериментальної перевірки V, рівний за довжиною розрізняним тестовим сегментам, дозволяє привести процедуру пошуку дефектів до задачі покриття, що дає максимально точний результат у вигляді ДНФ, де кожен терм є можливий варіант наявності в пристрої підмножини або поєднання дефектів SiP. Модель процесу діагностування подана у наступному вигляді:

$$\begin{aligned} &A = \langle T, F, M, V \rangle, \\ &T = (T_1, T_2, ..., T_i, ..., T_n);\ F = (F_1, F_2, ..., F_j, ..., F_m); \\ &M = \left| M_{ij} \right|, i = \overline{1,n};\ j = \overline{1,m};\ V = (V_1, V_2, ..., V_i, ..., V_n); \\ &V_i = R(T_i) \oplus R^*(T_i);\ \{V_i, T_i, M_{ij}, F_j\} \in \{0,1\}. \end{aligned} \tag{5.1}$$

Значення координати вектора V є результат виконання операції Xor над узагальненої модельною та фактичною реакцією виходів пристрою. Якщо хоча б на одному виході пристрою зафіксовано одиничне значення вектора V на тестовому сегменті з номером i, то узагальнений стан виходу дорівнює 1. В іншому випадку значення координати вектора V дорівнює 0.

5.3.1. Алгебро-логічний метод діагностування несправностей.

Рішення завдання діагностування зводиться до аналізу таблиці несправностей, отриманої в результаті моделювання дефектів компонентів SiP, шляхом запису логічного добутку диз'юнкцій рядків таблиці несправностей, записаних за одиничними значеннями вектора експериментальної перевірки:

$$F = \bigwedge_{\forall V_i = 1}^{i=\overline{1,n}} \left(\bigvee_{\forall M_{ij} = 1}^{j=\overline{1,m}} F_j \right). \tag{5.2}$$

Кон'юнктивна нормальна форма (КНФ), отримана з ТН, трансформується до діз'юнктівної нормальної форми (ДНФ) за допомогою еквівалентних перетворень (логічне множення, мінімізація і поглинання) [17]. В результаті виходить булева функція, де терми - логічні добутки представляють повну множину рішень у вигляді сполучення дефектів (що дають за виходами SiP або її компонентами двійкові координати вектора експериментальної перевірки V):

$$F = \bigwedge_{\forall V_i = 1}^{i=\overline{1,n}} \left(\bigvee_{\forall M_{ij} = 1}^{j=\overline{1,m}} F_j \right) = \left| \begin{bmatrix} a \vee ab = b \\ a \vee a = a \end{bmatrix} \right. = \bigvee_{i=1}^{2^m} \left(\bigwedge_{j=1}^{m} k_j F_j \right), k_j = \{0,1\}. \tag{5.3}$$

Функція (5.3) в загальному випадку формує діагноз у вигляді деякої підмножини сполучень (кратних) дефектів, які далі потребують уточнення шляхом застосування додаткового зондування внутрішніх точок за допомогою регістра граничного сканування. Кількість одиниць у векторі експериментальної перевірки V формує число діз'юнктівних термів КНФ (5.3). Кожен терм - порядковий запис дефектів (через логічну операцію АБО), що впливають на виходи функціональності. Подання таблиці у вигляді аналітичного запису - кон'юнктівної нормальної форми дає потенційну можливість істотно скоротити обсяг діагностичної інформації для пошуку дефектів. Подальше перетворення КНФ до ДНФ на основі тотожностей алгебри логіки дозволяє істотно зменшити булеву функцію.
Алгебрологічний метод розглядається на прикладі таблиці несправностей М1 і зображений у вигляді наступних пунктів алгоритму [21].

$M_1 =$

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	V
T_1	1			1			1
T_2		1			1		1
T_3			1	1		1	1
T_4	1		1				1
T_5		1			1	1	0

1. Визначення рядків таблиці несправностей цифрового пристрою, що відповідають нульовим значенням вектора експериментальної перевірки з метою обнулення одиничних координат знайдених рядків. В даному випадку - це один рядок T_5.

2. Знаходження всіх стовпців, які мають нульові значення координат рядків з нульовим станом ВЕП. Обнулення одиничних значень знайдених стовпців. В даному випадку: F_2, F_5, F_6.

3. Видалення з таблиці несправностей рядків і стовпців, що мають тільки нульові значення координат (знайдені в пунктах 1 і 2).

$M_1 =$

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	V
T_1	1			1			1
T_2		0			0		1
T_3			1	1		0	1
T_4	1		1				1
T_5		0			0	0	0

=

T_i / F_j	F_1	F_3	F_4	V
T_1	1		1	1
T_3		1	1	1
T_4	1	1		1

4. Побудова КНФ за одиничними значеннями ВЕП. Перетворення КНФ до ДНФ з подальшою мінімізацією функції:

$$F = (F_1 \vee F_4) \wedge (F_3 \vee F_4) \wedge (F_1 \vee F_3) = (F_1F_3 \vee F_3F_4 \vee F_1F_4 \vee F_4F_4) \wedge (F_1 \vee F_3) =$$
$$= F_1F_1F_3 \vee F_1F_3F_4 \vee F_1F_1F_4 \vee F_1F_4F_4 \vee F_1F_3F_3 \vee F_3F_3F_4 \vee F_1F_3F_4 \vee F_3F_4F_4 =$$
$$= F_1F_3 \vee F_1F_3F_4 \vee F_1F_4 \vee F_3F_4 \vee F_1F_3F_4 \vee F_3F_4 = F_1F_3 \vee F_1F_4 \vee F_3F_4.$$

Запропонований алгоритм орієнтований на аналіз таблиці несправностей з метою зменшення обсягу ТН і наступних обчислень, пов'язаних з побудовою ДНФ, яка формує всі рішення щодо встановлення діагнозу функціональностей SoC. Алгоритм орієнтований на обробку розріджених таблиць несправностей, коли число одиничних координат на тлі нульових значень становить не більше 20%. Подальше уточнення діагнозу можливо тільки із застосуванням мультізонда на основі регістра граничного сканування даних [18].

5.3.2. Векторно-логічний метод діагностування несправностей.

Обробка ТН для отримання діагнозу виконується за сценарієм, заснованим на використанні векторних операцій кон'юнкції, диз'юнкції і заперечення над стоками таблиці несправностей. Аналітична векторно-логічна модель діагностування кратних дефектів визначається кон'юнкцією двох компонентів, де перший є диз'юнкція векторів, відповідних одиничним координатам ВЕП, а другий - інверсія диз'юнкцій векторів, відповідних нульовим координатам ВЕП:

$$F = M^1 \wedge \overline{M}^0 = \left(\bigvee_{V_i=1} M_i \right) \wedge \left(\overline{\bigvee_{V_i=0} M_i} \right). \qquad (5.4)$$

Модель діагностування одиночних дефектів відрізняється виконанням на першому кроці сценарію операції кон'юнкції (замість диз'юнкції) усіх векторів, що відповідають одиничним координатам ВЕП:

$$F = M^1 \wedge \overline{M}^0 = \left(\bigwedge_{V_i=1} M_i \right) \wedge \left(\overline{\bigvee_{V_i=0} M_i} \right). \qquad (5.5)$$

Приклад 5.1. Виконати діагностування кратних дефектів в структурі SiP векторно-логічним методом, для якої задані таблиця несправностей і вектор експериментальної перевірки:

T_i / F_j	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	V
T_1				1						1	1
T_2		1					1				0
T_3			1			1			1		0
T_4	1										0
T_5					1			1			1
T_6	1	1									0
T_7			1								0
T_8				1							1
T_9					1	1					0
T_{10}							1				0
T_{11}								1	1	1	1
M^1				1	1			1	1	1	1
M^0	1	1	1		1	1	1		1		0
$\overline{M}^0$	0	0	0	1	0	0	0	1	0	1	1
F	0	0	0	1	0	0	0	1	0	1	1

(5.6)

Обробка таблиці несправностей відповідно до формули (5.4) дає результат, представлений у чотирьох нижніх рядках (5.6). Останній рядок таблиці несправностей фіксує факт наявності в схемі дефектів, поданих у векторній або теоретико-множинній формі $F = (0001000101) = \{F_4, F_8, F_{10}\}$.

Для перетворення отриманого рішення до ДНФ кратних дефектів використовується структура ТН, маскована вектором F, що поданий останнім рядком таблиці:

$$F = M^1 \wedge F = \bigwedge_{V_i = 1} (M_i \wedge F). \qquad (5.7)$$

Отримані вектори, число яких в межі дорівнює кількості одиниць ВЕП, логічно перемножуються. При цьому кожен вектор може бути компактно записаний у вигляді диз'юнкції значень одиничних координат. Для поданої вище ТН застосування формули (5.7) дає наступний результат:

$$F = (F_4 \vee F_{10})(F_8)(F_4)(F_4 \vee F_{10}) = F_4F_8F_4F_4 \vee F_{10}F_8F_4F_4 \vee F_4F_8F_4F_{10} \vee F_{10}F_8F_4F_{10} =$$
$$= F_4F_8 \vee F_{10}F_8F_4 \vee F_4F_8F_{10} \vee F_{10}F_8F_4 = F_4F_8.$$

Він цікавий тим, що завдяки запису дефектів у вигляді термів ДНФ, що покривають всі одиничні координати ВЕП, з'явилася можливість виключити дефект $F_{10} \in F$ зі списку несправностей.

Перевага векторно-логічного методу полягає в технологічності аналізу таблиці несправностей, обчислювальна складність якої має мультиплікативну залежність від числа дефектів і потужності тесту: $Q = n \times m$. Метод слід використовувати при перевазі в таблиці несправностей одиничних координат (> 20%), коли алгебро-логічний метод дає високу оцінку за Квайном складності ДНФ і КНФ. Недолік полягає в неможливості подання всіх поєднань дефектів, що формують терми для покриття одиничних координат вектора експериментальної перевірки.

Запропоновані методи діагностування: алгебро-логічний та векторно-логічний пропонують фахівцеві в області проектування і тестірування цифрових систем на кристалах математичний апарат, який здатний здійснити діагноз дефектних компонентів, якщо мається попередньо побудована таблиця несправностей. При цьому векторні рішення, ефективно отримані в другому методі, можна представляти усіма можливими поєднаннями дефектів у вигляді термів ДНФ, що характерно для першого методу. Другий метод ефективний при числі одиниць в матриці несправностей, що перевищує 10-20%.

5.4. Алгебро-логический метод ремонта блоков FPGA

Подано точний метод відновлення працездатності логічних блоків FPGA за допомогою резервних компонентів шляхом квазіоптимального розв'язання задачі покриття множини дефектних комірок мінімально можливою кількістю надлишкових компонентів. Метод орієнтований на імплементацію в інфраструктуру сервісного обслуговування функціональностей цифрової системи в пакеті. Функція мети визначається як мінімізація числа резервних компонентів S^t матриці блоків FPGA, необхідних для ремонту дефектних блоків у процесі функціонування SiP шляхом синтезу діз'юнктівної нормальної форми покриття несправностей з подальшим вибором мінімального кон'юнктивного терма $(S_1^t, S_2^t, ..., S_i^t, ..., S_{m_t}^t) \in S^t$, що задовільнює обмеженим за числом резервних компонентів $m_t \le p$, які входять до складу логічного добутку:

$$Z = \min_{t=\overline{1,n}} (|S^t|)|_{m_t \le p},\ S = (S^1 \vee S^2 \vee ... \vee S^t \vee ... \vee S^n);\ S^* = \{S_1, S_2, ..., S_j, ..., S_p\}.$$

Модель функціональності FPGA подається у вигляді матриці логічних блоків, що оперує рядками і стовпцями структури $M = |M_{ij}|$. В процесі проектування матриці надається певний резерв, складений із запасних рядків і стовпців, які можна переадресовувати в процесі реконфігурування структури пристрою у разі виявлення дефектів.

Модель процесу визначення мінімального числа резервних компонентів (рядків і стовпців), що покривають всі виявлені дефекти в матриці логічних блоків FPGA, зводиться до наступних пунктів:

1. Побудова таблиці покриття виявлених дефектних блоків FPGA резервними рядками і стовпцями матриці. Для досягнення поставленої мети розглядається топологічна модель результатів тестування функціональності FPGA у вигляді матриці, координати якої ідентифікують виявлені дефекти (справні і несправні блоки):

$$M = |M_{ij}|, M_{ij} = \begin{cases} 1 \leftarrow T \oplus f = 1; \\ 0 \leftarrow T \oplus f = 0. \end{cases} \tag{5.8}$$

Тут координата матриці дорівнює 1, якщо сума за модулем 2 функції справної поведінки f блоку і фактичної тестової реакції дають одиничне значення, що відповідає наявності в блоці дефекту. Після тестування FPGA - фіксації усіх несправних блоків виконується побудова таблиці покриття дефектів $Y = \left|Y_{ij}\right|, i = \overline{1,n}; j = \overline{1,m}$, де стовпці відповідають множині несправностей (одиничних координат), зафіксованих в матриці M ($\left|M\right| = m$), а рядки є номери стовпців і рядків матриці блоків FPGA, що покривають несправності, зазначені в стовпцях:

$$Y = \left|Y_{ij}\right|, Y_{ij} = \begin{cases} 1 \leftarrow C_i(R_i) \cap F_j \neq \varnothing; \\ 0 \leftarrow C_i(R_i) \cap F_j = \varnothing. \end{cases} \tag{5.9}$$

Оперуючи далі рядками і стовпцями, в яких містяться дефекти, можна знайти оптимальний розв’язок у вигляді покриття виявлених дефектів в метриці несправних рядків і стовпців. Далі виконується тривіальна процедура перепризначення дефектним рядкам і стовпчикам аналогічних справних компонентів з резерву кристала FPGA.

З міркування технологічності рішення задачі покриття замість компонентів двовимірної метрики, позначеної рядками і стовпцями (C, R), використовується одновимірний вектор, зконкатенований з двох послідовностей C і R, потужність якого дорівнює n = p + q:

$$\begin{aligned} X &= C * R = (C_1, C_2, ..., C_i, ..., C_p) * (R_1, R_2, ..., R_j, ..., R_q) = \\ &= X^c * X^r = (X_1, X_2, ..., X_i, ..., X_p, X_{p+1}, X_{p+2}, ..., X_{p+j}, ..., X_{p+q}). \end{aligned} \tag{5.10}$$

При цьому між елементами вихідних наборів (C, R) і результуючим вектором X існує взаємно-однозначна відповідність, що встановлена в першому стовпці матриці Y. Слід зауважити, що перетворення $X = C * R$ виконується лише для зручності розгляду і подальшої побудови диз'юнктивної нормальної форми в рамках однаковості змінних, що формують булеву функцію. Якщо дану процедуру не виконувати, то функція буде визначена на двох типах змінних, що містять стовпці і рядки матриці пам'яті.

2. Побудова КНФ шляхом аналітичного, повного і точного рішення задачі покриття. Після формування матриці покриття, що містить нульові і одиничні координати, виконується синтез аналітичної форми покриття шляхом запису КНФ за одиничними координатам стовпців матриці. Тут число кон'юнктивні термів дорівнює кількості стовпців таблиці, а розмірність диз'юнктивного терма дорівнює числу одиниць в розглянутому стовпці:

$$Y = \bigwedge_{j=1}^{m} (Y_{pj} \vee Y_{qj})_{\{Y_{pj}, Y_{qj}\}=1} = \bigwedge_{j=1}^{m} (X_{pj} \vee X_{qj}). \tag{5.11}$$

З останнього виразу видно, що кожен стовпець ідентифікує два варіанти покриття несправності - рядком і стовпцем. Тому стовпець має тільки дві координати з одиничних значень, а число логічних добутків дорівнює загальній кількості дефектів m, виявлених в матриці блоків FPGA.

3. Перетворення КНФ до ДНФ дає можливість побачити всі рішення задачі покриття. Для цього до кон'юнктівної нормальної форми необхідно застосувати операцію логічного множення і правила мінімізації (поглинання) для отримання діз'юнктівної нормальної форми:

$$Y = \bigvee_{j=1}^{w} (k_1^j X_1 \wedge k_2^j X_2 \wedge \ldots \wedge k_i^j X_i \wedge \ldots \wedge k_n^j X_n), k_i^j = \{0,1\}. \qquad (5.12)$$

Тут подано узагальнений запис ДНФ, де число термів в межі дорівнює $w = 2^n$, n – число рядків в множині (C,R) або кількість змінних X в матриці Y. Всі можливі рішення – покриття дефектів резервними компонентами – записуються на множині ідентифікаторів рядків таблиці покриття. Якщо k_i^j при X_i обертається на нуль, то X_i перетворюється в неістотну змінну.

4. Вибір мінімальних і точних рішень задачі покриття. Процедура пов’язана з визначенням кон'юнктивних термів мінімальної довжини за Квайном в отриманій ДНФ. Подальше перетворення до рядків і стовпчиків матриці пам'яті на основі використання раніше введеної відповідності дає можливість записати мінімальне покриття або їх сукупність в двовимірній метриці рядків і стовпців, що відповідає умовам (обмеженням) функції мети на кількість резервних компонентів.

5. Реалізація процедури перепризначення дефектним рядкам і стовпчикам аналогічних справних компонентів з резерву кристала FPGA.

Приклад 5.2. Виконати процес відновлення працездатності матриці блоків FPGA в частині визначення мінімального числа резервних компонентів, що покривають всі дефекти. Матриця кристала з дефектами і резервом, зафарбовані темним кольором [19,20], зображена на рис. 5.6.

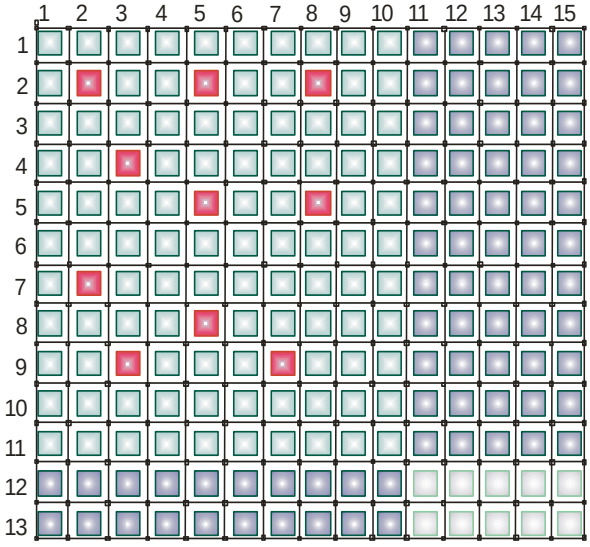


Рис. 5.6. Матриця блоків FPGA з дефектами і резервом

Матриця має резерв діагностування та відновлення працездатності дефектних комірок, який визначається двома рядками і п'ятьма стовпцями.

Відповідно до пункту 1 моделі процесу визначення мінімального числа резервних компонентів, що покривають всі виявлені дефекти в матриці пам'яті, будується таблиця покриття десяти дефектів

$$F = (F_{2,2}, F_{2,5}, F_{2,8}, F_{4,3}, F_{5,5}, F_{5,8}, F_{7,2}, F_{8,5}, F_{9,3}, F_{9,7})$$

одинадцятьма рядками таблиці (5.13).

$$Y = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
X_i / F_{i,j} & F_{2,2} & F_{2,5} & F_{2,8} & F_{4,3} & F_{5,5} & F_{5,8} & F_{7,2} & F_{8,5} & F_{9,3} & F_{9,7} \\
\hline
C_2 \to X_1 & 1 & & & & & & 1 & & & \\
\hline
C_3 \to X_2 & & & & 1 & & & & & 1 & \\
\hline
C_5 \to X_3 & & 1 & & & 1 & & & 1 & & \\
\hline
C_7 \to X_4 & & & & & & & & & & 1 \\
\hline
C_8 \to X_5 & & & 1 & & & 1 & & & & \\
\hline
R_2 \to X_6 & 1 & 1 & 1 & & & & & & & \\
\hline
R_4 \to X_7 & & & & 1 & & & & & & \\
\hline
R_5 \to X_8 & & & & & 1 & 1 & & & & \\
\hline
R_7 \to X_9 & & & & & & & 1 & & & \\
\hline
R_8 \to X_{10} & & & & & & & & 1 & & \\
\hline
R_9 \to X_{11} & & & & & & & & & 1 & 1 \\
\hline
\end{array} \qquad (5.13)$$

Потужність або число рядків у таблиці визначається конкатенацією стовпців C і рядків R, що знаходяться у взаємно-однозначній відповідності з вектором змінних X:

$$C*R=(C_2,C_3,C_5,C_7,C_8)*(R_2,R_4,R_5,R_7,R_8,R_9)\approx$$
$$\approx X=(X_1,X_2,X_3,X_4,X_5,X_6,X_7,X_8,X_9,X_{10},X_{11}). \quad (5.14)$$

За таблицею покриття виконується побудова КНФ шляхом запису термів за одиничними значеннями стовпців:

$$Y=(X_1 \vee X_6)(X_3 \vee X_6)(X_5 \vee X_6)(X_2 \vee X_7)(X_3 \vee X_8)\&$$
$$\&(X_5 \vee X_8)(X_1 \vee X_9)(X_3 \vee X_{10})(X_2 \vee X_{11})(X_4 \vee X_{11}). \quad (5.15)$$

Отримання діз'юнктівної нормальної форми грунтується на застосуванні тотожностей булевої алгебри, які дозволяють виконати логічне перемноження всіх десяти співмножників і подальшу мінімізацію термів ДНФ шляхом застосування оператора співграней $(ab \vee a\bar{b}=a)$, аксіом поглинання і вилучення однакових термів. Опустивши проміжні обчислення, остаточний результат приведемо до наступного вигляду:

$$Y=X_1X_2X_3X_4X_5 \vee X_2X_3X_4X_5X_6X_9 \vee X_1X_2X_3X_4X_6X_8 \vee$$
$$\vee X_2X_3X_4X_6X_8X_9 \vee X_1X_2X_4X_6X_8X_{10} \vee X_2X_4X_6X_8X_9X_{10} \vee$$
$$\vee X_1X_4X_6X_7X_8X_{10}X_{11} \vee X_1X_2X_3X_5X_{11} \vee X_2X_3X_5X_6X_9X_{11} \vee$$
$$\vee X_1X_2X_3X_6X_8X_{11} \vee X_2X_3X_6X_8X_9X_{11} \vee X_1X_2X_6X_8X_{10}X_{11} \vee \quad (5.16)$$
$$\vee X_2X_6X_8X_9X_{10}X_{11} \vee X_1X_3X_5X_7X_{11} \vee X_3X_5X_6X_7X_9X_{11} \vee$$
$$\vee X_1X_3X_6X_7X_8X_{11} \vee X_3X_6X_7X_8X_9X_{11} \vee X_1X_6X_7X_8X_{10}X_{11} \vee$$
$$\vee X_6X_7X_8X_9X_{10}X_{11}.$$

Вибір термів мінімальної довжини, що в даному випадку містять 5 змінних, формує множину мінімальних рішень:

$$Y=X_1X_2X_3X_4X_5 \vee X_1X_2X_3X_5X_{11} \vee X_1X_3X_5X_7X_{11} \quad (5.17)$$

Трансформування отриманої функції до покриття, яке містить позначення змінних у вигляді рядків і стовпців матриці блоків FPGA, дозволяє зображувати згадані терми у формі:

$$Y=C_2C_3C_5C_7C_8 \vee C_2C_3C_5C_8R_9 \vee C_2C_5C_8R_4R_9. \quad (5.18)$$

Всі отримані мінімальні рішення задовольняють вимогам по обмеження на число резервних компонентів, визначених числами:

$$(\left|C^r\right| \le 5)\&(\left|R^r\right| \le 2).$$

Подальша технологія вбудованого ремонту дефектних блоків FPGA полягає у електричному перепрограмуванні дешифратора адреси стовпця чи рядка матриці блоків FPGA. Стосовно до FPGA, що зображена на рис. 5.6, переадресація стовпців з несправними логічними блоками на резервні стовпці буде відбуватися, наприклад, відповідно до першого терма з (5.18), який визначає відношення:

Faulty column	2	3	5	7	8
Spare column	11	12	13	14	15

.

Обчислювальна складність алгебрологічного методу відновлення працездатності в частині рішення задачі покриття [17,20] визначається наступним виразом:

$$Q = 2^{|F|} + |C + R| \times 2^{|F|}, \tag{5.19}$$

де $2^{|F|}$ – витрати, пов'язані з синтезом ДНФ шляхом логічного перемноження виключно двокомпонентних диз'юнкцій (координата дефекту визначається номером рядка і стовпчика), число яких дорівнює кількості дефектних блоків; де $|C + R| \times 2^{|F|}$ - верхня межа обчислювальних затрат, необхідних для мінімізації отриманої ДНФ на граничній множині змінних, що дорівнює сумарному числу рядків і стовпців $|C + R|$.

В гіршому випадку, коли координати всіх дефектних блоків за рядками і стовпчиками не корельовані, - унікальні, наприклад, діагональні дефекти, обчислювальна складність матричного методу стає залежною тільки від числа дефектних комірок, а її аналітичний запис трансформується до наступного вигляду:

$$Q = 2^{|F|} + |C + R| \times 2^{|F|}\Big|_{|C+R| \le 2\times|F|} = 2^{|F|} + 2 \times |F| \times 2^{|F|} = 2^{|F|} \times (1 + 2 \times |F|). \tag{5.20}$$

Если вместо мощности множества дефектов записать их число, равное m, тогда предыдущее выражение представляется в более простой форме:

$$Q = 2^{m} \times (1 + 2 \times m) = 2^{m}(2m + 1). \tag{5.21}$$

Згідно технології вбудованого сервісного обслуговування функциональних логічних блоків FPGA, алгебрологічний метод відновлення працездатності логічних блоків на основі вирішення задачі покриття імплементується в кристал, в якості одного з компонентів інфраструктури сервісного обслуговування (I-IP), націленого на підтримку працездатності блоків FPGA і SoC в цілому.

5.5. Метод обходу матриці логічних блоків для покриття дефектних компонентів FPGA ремонтними клітинами

Топологія кристалу подана матрицею кліток $M = |M_{ij}| i = \overline{1,p}; j = \overline{1,q}$, що масштабована за горизонталлю і вертикаллю цілими числами ($p \times q$). Кожна клітка M_{ij} має n^2 логічних блоків. Матрица має довільне число дефектів, що дорівнює k. В вожній клітині може бути не більш ніж n^2 несправних логічних блоків. Приклад матриці клітин логічних блоків системи на кристалі с дефектами зображено на рис. 5.7, де розмірність клітини n дорівнює 3, а

розмірність матриці у масштабі числа клітин за рядками і стовпчиками дорівнює 5.

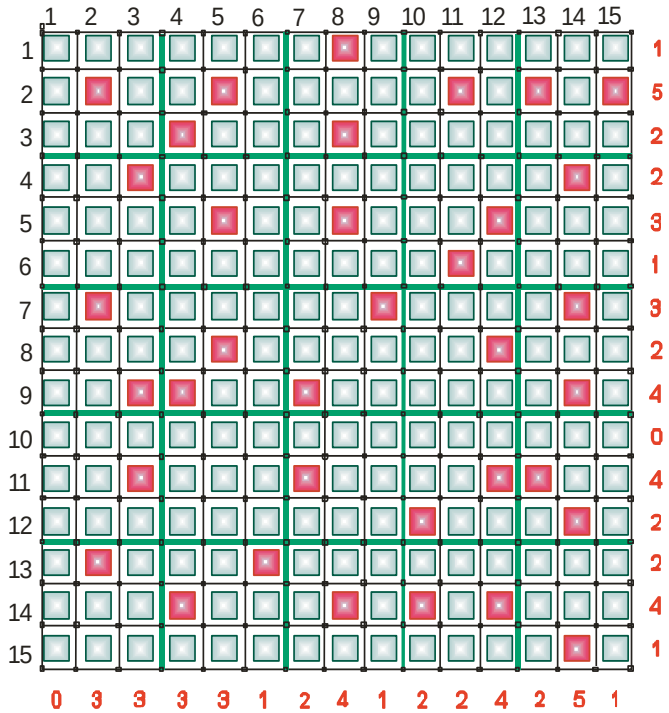


Рис. 5.7. Матриця блоків FPGA в масштабі клітин

Метод обходу матриці логічних блоків з метою відновлення працездатності компонентів FPGA, що приводить до квазіоптимального покриття всіх дефектних блоків мінімальним числом ремонтних клітин, подано наступними пунктами:

1. Визначення координат всіх дефектних блоків матриці $M = \left|M_{ij}\right|$, що дає топологію кристала.

2. Побудова двійкових матриць покриття дефектних блоків шляхом обходу клітин за рядкми і стовпчиками, розмірність яких відповідно визначається параметрами:

$$M_r = \left|M_{ij}^r\right|, i = \overline{1, p/n}; j = \overline{1, q};$$

$$M_c = \left|M_{ij}^c\right|, i = \overline{1, p}; j = \overline{1, q/n}.$$

Тут кожні n координат рядку (стовпца) замінюються однією зі значенням в неї, що визначається $f^r(f^c)$ функцією Or від n координат:

$$M_{ij}^{c} = f^{r}(f^{c}) = \begin{cases} 0 \leftarrow (000); \\ 1 \leftarrow (1XX) \vee (X1X) \vee (XX1), X = \{0,1\}. \end{cases}$$

Наприклад, процедура отримання матриці обходу рядків призводить до результату

$$M = \left|M_{ij}\right| = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \xrightarrow{f^{r}} M_{r} = \left|M_{ij}^{r}\right| = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Тут кожен стовпець стискається в дві координати за правилами логічної операції Or, оскільки параметр клітини n, тут і далі, дорівнює 3.

Аналогічно, процедура отримання матриці обходу стовпців дає результат

$$M = \left|M_{ij}\right| = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \xrightarrow{f^{c}} M_{c} = \left|M_{ij}^{c}\right| = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

3. Визначення критеріїв якості покриття дефектних блоків шляхом використання отриманих двійкових матриць на основі підрахунку числа одиничних координат, приведених до фактичного одиничного каркасу матриць. Критерій порядкового покриття дефектних блоків поданий наступним виразом:

$$Q_{r} = \sum_{i=1}^{p/n} \left[\frac{1}{H_{i}^{r} - L_{i}^{r} + 1} \sum_{j=1}^{q} M_{ij}^{r} \right].$$

Тут $H_{i}^{r}(L_{i}^{r})$ - максимальний (мінімальний) індекс j-координати для рядку матриці $\left|M_{ij}^{r}\right|$, після (до) якої в рядку можуть бути тільки нульові значення координат. Фактично $H_{i}^{r} - L_{i}^{r} + 1$ є інтервал розкиду одиниць в рядку матриці $\left|M_{ij}^{r}\right|$, до якого наводиться сума одиничних координат рядка. Наведені далі оцінки всіх рядків підсумовуються, що і є критерієм ефективності порядкового покриття дефектних блоків.

Критерій покриття дефектних блоків за стовпцями має наступний вигляд:

$$Q_{c} = \sum_{j=1}^{q/n} \left[\frac{1}{H_{j}^{c} - L_{j}^{c} + 1} \sum_{i=1}^{p} M_{ij}^{c} \right].$$

Тут $H_j^c(L_j^c)$ - максимальний (мінімальний) індекс і-координати для стовпця матриці $\left|M_{ij}^c\right|$, після (до) якої в стовпці можуть бути тільки нульові значення координат. Фактично $H_j^c - L_j^c + 1$ є інтервал розкиду одиниць в стовпці матриці $\left|M_{ij}^c\right|$, до якого наводиться сума одиничних координат стовпця. Далі наведені оцінки всіх стовпців сумуються, що і є критерієм ефективності покриття дефектних блоків за стовпцями.

4. Ухвалення рішення про вибір стратегії $S = \{S_r, S_c\}$ покриття дефектних логічних блоків резервними клітинами шляхом порівняння значень критеріїв структуризації Q_r, Q_c для рядків і стовпців:

$$S = \begin{cases} S_r \leftarrow Q_r < Q_c; \\ S_c \leftarrow Q_r \geq Q_c. \end{cases}$$

У першому випадку виконується стратегія покриття дефектних блоків шляхом послідовного обходу всіх клітинних рядків. У другому - осуществляється обхід клітинних стовпців.

5. Стратегія обходу клітин матриці за рядками. Виконується на модифікованій матриці $\left|M_{ij}^r\right|$. Кожний рядок матриці подано двійковим вектором $M_i^r = (M_{i1}^r, M_{i1}^r, \ldots, M_{ij}^r, \ldots, M_{iq}^r)$. Шаг 1. Обнуління лічильника нульових координат і лічильника числа ремонтних клітин: $j = 0,\ Q = 0$. Шаг 2. Виконується послідовнісне сканування комірок вектору $j = j + 1 \leftarrow M_{ij}^r = 0$ до першої зустрічної одиниці $M_{ij}^r = 1 \rightarrow (Q = Q + 1, j = j + n - 1)$. Від даної 1 відраховивається n комірок, які покриваються ремонтною клітиною. Число ремонтних клітин Q збільшується на 1. Шаг 3. Якщо виконується умова $j \geq q$ – кінець процедури обробтки рядку. Інакше – перехід до шагу 2. Описана процедура застосовується до всіх рядків модифікованої матриці $\left|M_{ij}^r\right|$($\left|M_{ij}^c\right|$), в результаті чого лічильник Q буде містити мінімальне число ремонтних кліток, необхідних для покриття всіх дефектних блоків. Аналогічно виконується стратегія обходу кліток матриці за стовпцями ($\left|M_{ij}^c\right|$). В даному випадку зміні в процедурі обходу підлягає індекс стовпця і замість j.

6. Визначення якості покриття отриманого варіанту розв’язку шляхом підрахунку числа дефектних блоків матриці, наведених до мінімального числа резервних клітин N, що покривають всі несправні блоки:

$$Q_{cr} = \frac{1}{N}\sum_{i=1}^{n} F_i .$$

7. Кінець виконання процедури пошуку квазіоптимального покриття дефектних блоків резервними клітинами.

Приклад 5.3. Для кристала, що зображений на рис. 5.7, у відповідності з пунктом 2 моделі процесу відновлення працездатності виконується побудова двох матриць M_r і M_c. Далі у відповідності з пунктом 3 вказаної вище моделі для M_r і M_c виконується підрахунок критерієв структуризації Q_r, Q_c:

$$M_r = \left|M_{ij}^r\right| =$$

0	1	0	1	1	0	0	1	0	0	1	0	1	0	1
0	0	1	0	1	0	0	1	0	0	1	1	0	1	0
0	1	1	1	1	0	1	0	1	0	0	1	0	1	0
0	0	1	0	0	0	1	0	0	1	0	1	1	1	0
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

$$M_c = \left|M_{ij}^c\right| =$$

0	0	1	0	0
1	1	0	1	1
0	1	1	0	0
1	0	0	0	1
0	1	1	1	0
0	0	0	1	0
1	0	1	0	1
0	1	0	1	0
1	1	1	0	1
0	0	0	0	0
1	0	1	1	1
0	0	0	1	1
1	1	0	0	0
0	1	1	1	0
0	0	0	0	1

$$Q_r = \sum_{i=1}^{p/n}\left[\frac{1}{H_i^r - L_i^r + 1}\sum_{j=1}^{q} M_{ij}^r\right] = \frac{7}{14} + \frac{6}{12} + \frac{8}{13} + \frac{6}{12} + \frac{7}{13} = 2{,}64;$$

$$Q_c = \sum_{j=1}^{q/n}\left[\frac{1}{H_j^c - L_j^c + 1}\sum_{i=1}^{p} M_{ij}^c\right] = \frac{6}{12} + \frac{7}{13} + \frac{7}{14} + \frac{7}{13} + \frac{7}{14} = 2{,}58.$$

Як видно із згаданих вище критеріїв, наведене до одиничного каркасу матриці число дефектних координат за стовпцями менше, ніж за рядками. Тому, з урахуванням пункту 4, вибирається стратегія вирішення задачі покриття шляхом обходу стовпців покриття: $S_c \leftarrow (Q_r = 2{,}64 \geq Q_c = 2{,}58)$. Структурна

привабливість стовпців вище, ніж рядків, оскільки наведене до матриці число дефектів тут менше.

Даний шлях дає фактичну якість покриття - число несправних логічних блоків матриці (на одну ремонтну клітку), наведене до необхідної кількості резервних компонентів, дорівнює

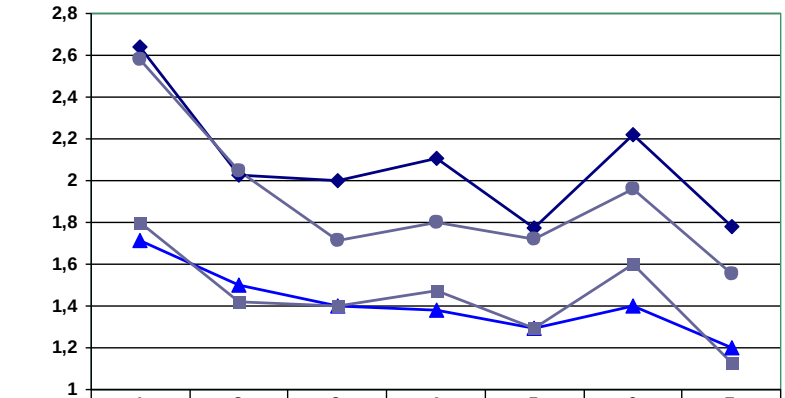
$$Q_c^* = \frac{1}{N}\sum_{i=1}^{n} F_i = 36/20 = 1{,}8.$$

Для порівняння – процедура обходу рядків дає більш низьку якість:

$$Q_r^* = \frac{1}{N}\sum_{i=1}^{n} F_i = 36/21 = 1{,}71.$$

Останнє покриття множини дефектів має на 1 клітину більше (21), ніж розв'язок, отриманий першим способом (20). Таким чином, вибір стратегії покриття на основі підрахунку та порівняння критеріїв підрахунку числа одиничних координат, приведених до фактичного одиничного каркасу матриць, підтверджує їх спроможність і подальшу оптимальність отриманого покриття.

На рис. 5.8 наведена статистика обробки різних видів матриць логічних блоків з дефектними компонентами. Вона свідчить про правомірність застосування запропонованого критерію, який у всіх випадках, крім останнього, вказує на оптимальну і ефективну стратегію вибору обходу матриці для отримання мінімального покриття ремонтними клітинами дефектних логічних блоків. В останньому варіанті критерій структуризації за рядками вище, ніж за стовпцями $Q_r = 1{,}78 > Q_c = 1{,}55$, що є підставою для виконання покриття дефектів за стовпцями. В даному випадку було отримано якість покриття, що дорівнює $Q_c^* = 1{,}125$, що відповідає 16 ремонтним клітинам, необхідним для відновлення працездатності всіх 18 дефектних блоків, тоді як оптимальне рішення є обхід матриці за стовпцями, де якість покриття визначається значенням $Q_c^* = 1{,}2$, що відповідає тільки 15 ремонтним клітинам.



	1	2	3	4	5	6	7
Qr	2,64	2,03	2	2,11	1,77	2,22	1,78
Qc	2,58	2,05	1,71	1,8	1,72	1,96	1,55
Q*r	1,71	1,5	1,4	1,38	1,29	1,4	1,2
Q*c	1,8	1,42	1,4	1,47	1,29	1,6	1,125

Рис. 5.8. Критерії структуризації і якість покриття для прикладів

5.6. Програмний засіб aGalls для покриття дефектних компонентів FPGA ремонтними клітинами

Програмний засіб «aGalls» є додатком, що працює під OS Windows. Написано на мові C ++ в середовищі програмування CodeGear RAD Studio 2007, містить 191 рядок і 388 операторів. Програмний засіб «aGalls» призначений для виконання процедури покриття дефектних компонентів FPGA ремонтними клітинами з використанням методу обходу матриці логічних блоків. Загальний вигляд інтерфейсу програмного продукту зображений на рис. 5.9.

Вихідна інформація подана у вигляді матриці клітин з дефектами $M = \left|M_{ij}\right| i = \overline{1,p}; j = \overline{1,q}$, яка має розмірність $p \times q$, де кожна клітина має n*n логічних блоків. В матрицю вноситься вихідна кількість дефектів k. Програма визначає координати всіх дефектів, будує двійкові матриці покриття несправних блоків шляхом обходу клітин за стовпчиками і рядками, обчислює критерії якості покриття дефектних блоків (рис. 5.10). Програма дає можливість переглянути зміни проміжних матриць після проведення процедури покриття дефектних блоків, як за горізонталлю, так і за вертикаллю. Крім того, як результат видається реальна кількість необхідних резервних клітин для покриття всіх дефектів, що дозволяє оцінити достовірність обраного розв’язку на основі обчислення критеріїв (рис. 5.11).

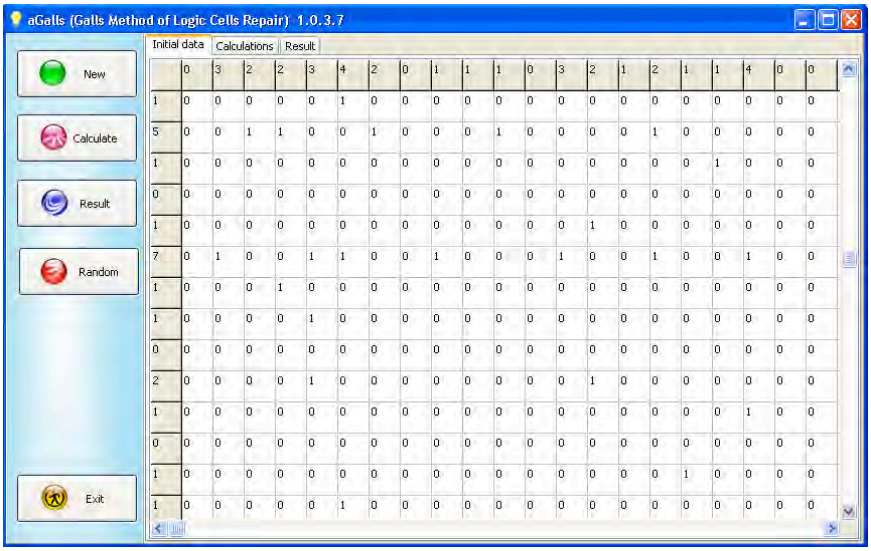


Рис. 5.9. Інтерфейс програмного засобу «aGalls»

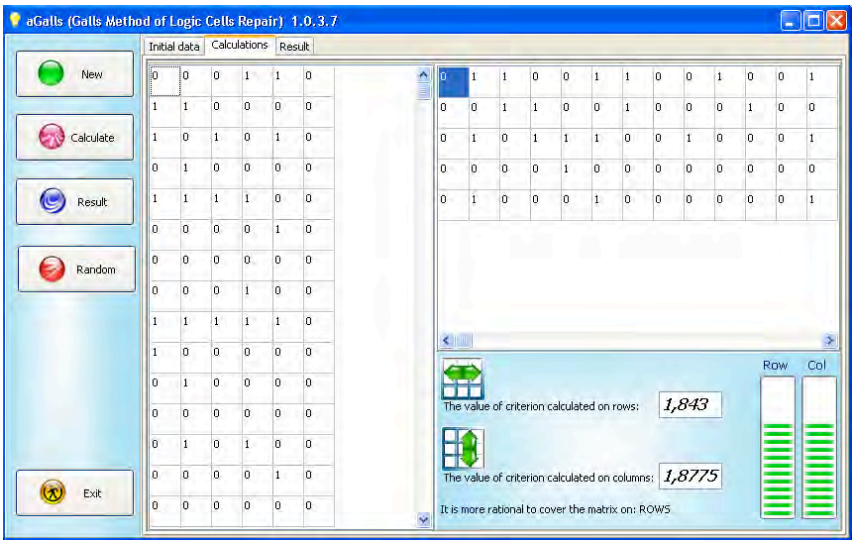


Рис. 5.10. Результати обчислень

На рис. 5.12 подано скорочений алгоритм розв’язку задачі покриття, який реалізовано у програмному засобі:

1. Початок розв’язання задачі покриття резервних блоків клітинами.

2. Визначення координат всіх дефектних блоків матриці.

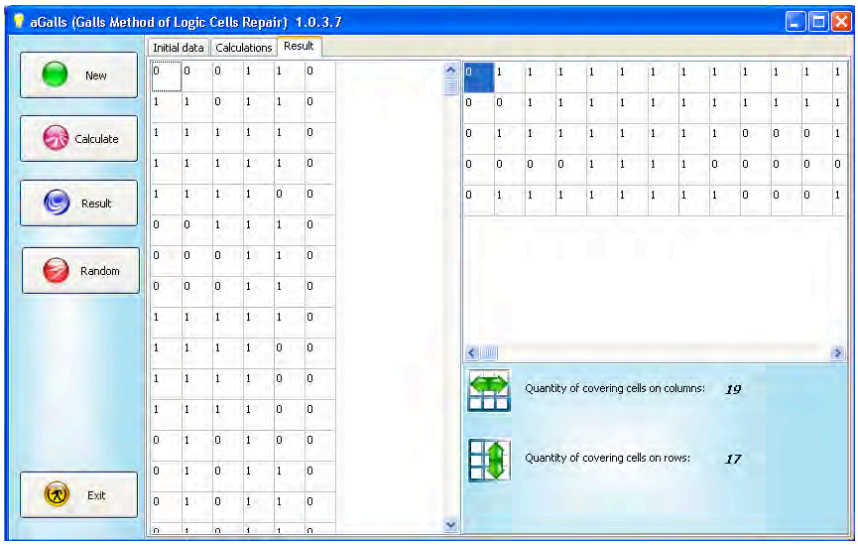


Рис.5.11. Результат покриття матриці з дефектами резервними клітинами

3. Побудова двійкових матриць покриття дефектних блоків для обходу клітин за рядками і стовпчиками.

4. Побудова покриття дефектів модифікованої матриці шляхом її обходу за рядками.

5. Виявлення першої та останньої одиниці.

6. Підрахунок кількості одиниць в рядку.

7. Прирощення номера оброблюваного рядка.

8. Обчислення значення критерію структуризації за рядками.

9. Виконання пунктів 4, 5, 6, 7 для стовпців.

10. Ухвалення рішення про вибір стратегії покриття дефектних блоків за напрямлінням, відповідним меншому значенню критерію.

11. Обнулення лічильника нульових координат і лічильника ремонтних клітин $j = 0,\ \ Q = 0$.

12. Послідовне сканування комірок вектору $j = j + 1 \leftarrow M_{ij}^{r} = 0$ до першої зустрічної одиниці $M_{ij}^{r} = 1 \rightarrow (Q = Q + 1, j = j + n - 1)$ за строками.

13. Від даної одиниці відраховується n осередків, які покриваються ремонтної кліткою.

14. Значення Q збільшується на 1.

15. Якщо $j \geq q$ - кінець процедури обробки рядка, інакше - крок 11.

16. Виконання пунктів 10, 11, 12, 13, 14 при обході за стовпцями.

17. Кінець виконання процедури пошуку квазіоптимального покриття дефектних блоків резервними клітинами.

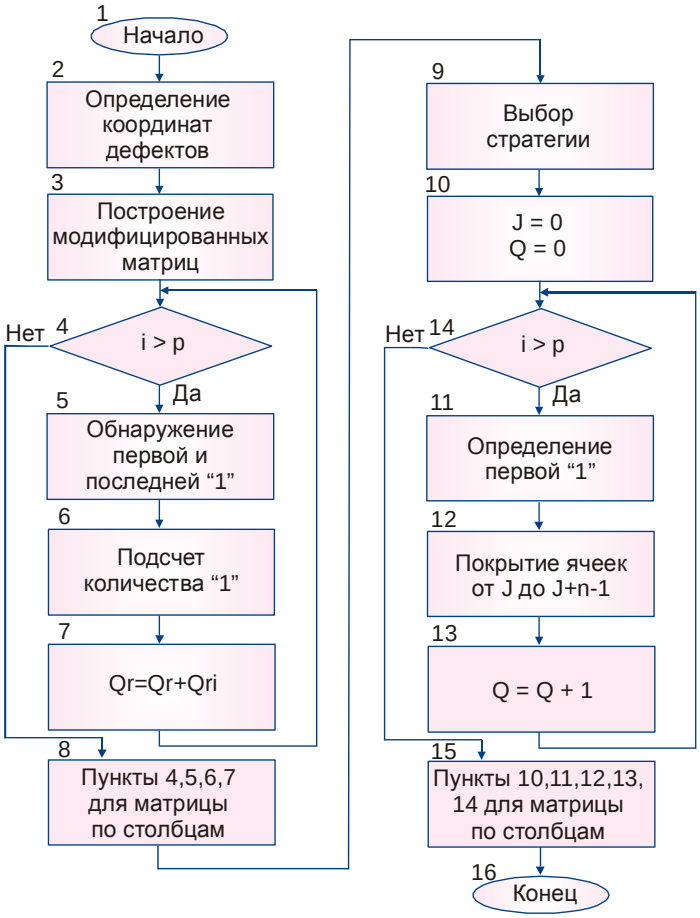


Рис.5.12. Модель розв’язку задачі оптимального покриття

Для перевірки достовірності критеріїв було проведено 200 експериментів для матриці M, розмірністю p*q з кількістю блоків в клітці за горизонталлю і вертикаллю n і кількістю дефектів k, де:

$$p = \overline{3,7};\ n = \overline{2,5};\ q = \overline{3,7};\ k = \overline{3,(n \times p \times q)}.$$

Як результат, за критеріями не був визначений найбільш раціональний спосіб ремонту у 29% випадків, відповідно позитивний результат - 71%. На рис. 5.13 зображені діаграми, що показують значення критеріїв структуризації та якості покриття при обході модифікованої матриці за рядками і за стовпцями для 10 прикладів. При цьому розмірність матриці була постійна: 4 клітини за

горизонталлю і 5 за вертикаллю, при цьому розмірність клітини була постійна і містила 3*3 блоків.

Тут же наведені: Qr і Qc - значення критеріїв структуризації при обході за рядками і стовпчиками; Q*r і Q*c - значення критеріїв якості покриття за рядками і стовпчиками. Обсяг даних, які обробляє програмний засіб, обмежений наступними числами (рис. 5.14): $n \in [2,50]$ - розмірність однієї клітини; $p \in [1,100]$ - кількість клітин за горизонталлю; $q \in [1,100]$ - за вертикаллю.

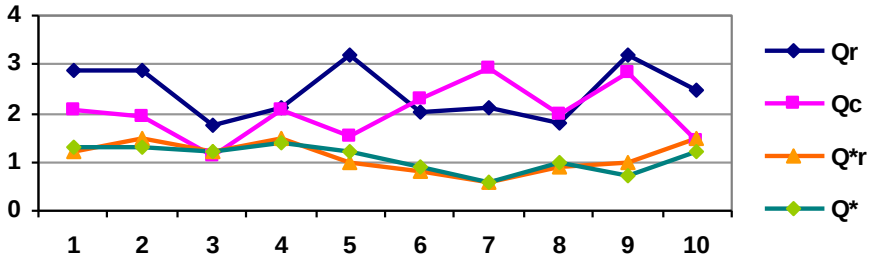


Рис.5.13. Критерії структуризації і якість покриття для примерів

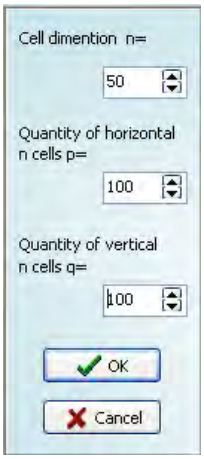


Рис. 5.14. Вихідні дані у програмному засобі

Для нормального функціонування програми необхідний процесор з тактовою частотою 1GHz і більше, 4Mb вільного дискового простору, операційна система Windows XP / 2000 або вище. Програмний продукт використовується для верифікації та тестування методу отримання квазіоптимального покриття дефектів мінімальною кількістю ремонтних клітин. Він може бути імплементований в кристал як вбудований компонент сервісного обслуговування, де вихідна інформація про топологію кристала повинна бути поміщена в пам'ять на стадії Place and Route.

5.7. Наукова новизна та практична значимість

На основі конструктивного аналізу опублікованих робіт існуючі технології відновлення працездатності можна розділити на дві групи. Перша орієнтована на заміну одиничного елемента (оптимальна і мозаїчна моделі). Друга - на заміну групи дефектних елементів (ієрархічна структура і модель з надмірністю). З іншого боку, відомі з публікацій методи можна класифікувати в три групи: 1) Дублювання логічних елементів або областей кристала, що приводить до подвоєння апаратної реалізації функціональності. При фіксації несправного елемента чи області здійснюється перемикання на справний компонент за допомогою мультиплексора [4]. Запропоновані Xilinx моделі FPGA (Field Programmable Gate Array) застосовні також при ремонті компонентів FPGA від компанії Altera. Основна одиниця виміру при ремонті - стовпець або рядок. 2) Використання генетичних алгоритмів для діагностування і відновлення працездатності на основі автономної реконфігурації кристала FPGA без застосування зовнішніх пристроїв керування [5]. Надійність діагностування дефектів дорівнює 99%, час ремонту - 36 мілісекунд замість 660 секунд, необхідних для стандартного конфігурування проекту. 3) Відновлення працездатності кристалів FPGA, не критичне до часу, шляхом заміни локальних CLB на надлишкові запасні компоненти запропоновано в [6,7]. Доступний рівень об'єднання CLB, який підлягає заміні, для критично важливих додатків становить близько тисячі логічних елементів.

Запропоновані методи діагностування: алгебро-логічний та векторно-логічний пропонують фахівцеві в області проектування і тестування цифрових систем на кристалах математичний апарат, який здатний здійснити діагноз дефектних компонентів на основі використання попередньо побудованої таблиці несправностей. При цьому векторні розв’язки, ефективно отримані векторно-логічним методом, можна зображувати усіма можливими поєднаннями дефектів у вигляді термів ДНФ, що характерно для алгебро-логічного методу. Другий метод ефективний при числі одиниць в матриці несправностей, що перевищує 10-20%.

Алгебро-логічний метод відновлення працездатності функціональних логічних блоків FPGA на основі розв’язання задачі покриття орієнтований на імплементацію в кристал, в якості одного з компонентів інфраструктури сервісного обслуговування (I-IP), націленого на підтримку працездатності блоків FPGA і SiP в цілому. Метод дозволяє знайти точне і оптимальне рішення, пов'язане з мінімальним числом резервних блоків, необхідних для відновлення працездатності логічних блоків кристала FPDA, за наявністю дефектів будь-якої кратності.

Метод обходу матриці логічних блоків призначений для відновлення працездатності компонентів FPGA шляхом отримання рішення у вигляді квазіоптимального покриття всіх дефектних блоків мінімальним числом ремонтних клітин. Пропонується вибір однієї з двох стратегій обходу рядків або стовпців матриці логічних блоків на підставі критеріїв структуризації, що

визначають число несправних блоків, наведених до фактичного одиничного каркасу модифікованої матриці рядків або стовпців.

Технологічні рішення, представлені в оглядовій частині, а також запропоновані методи діагностування і відновлення працездатності цифрових систем на кристалах і в пакетах кристалів, в частині їх актуальності та перспективності для ринку електронних технологій, добре корелюються з аналітичними дослідженнями ринку електроніки на 2009 рік і формулюються у вигляді «Гарячої IT-десятки» від Gartner Research Group: 1) Віртуалізація. 2) «Хмарні обчислення» (Cloud computing). 3) Сервери майбутнього, що йдуть на зміну blade-серверів. 4) Веб-орієнтовані архітектури. 5) Змішані корпоративні додатки (mashups). 6) Спеціалізовані системи на кристалах. 7) Соціальні мережі та програмне забезпечення для них. 8) Об'єднані комунікації (unified communications). 9) Бізнес-аналітика. 10) «Зелені» IT. Джерело: http://www.gartner.com/.

Згадана вище топ-десятка досить добре корелюється з аналітичним дослідженням від компанії Computer Sciences Corporation (CSC), яке оформлено у сім тенденцій: 1) Нові ЗМІ. Internet став повноцінною структурою для створення та використання аудіо-, відео- та текстового контенту у масштабі планети. 2) Соціальне ПЗ. Соціальні мережі залучають мільйони користувачів, використовуючи спільність інтересів. 3) Розширена реальність. Поступово, але наполегливо проникає в наше життя. Віртуальна реальність, в якій образи користувачів подорожують за віртуальними світами, стає практично доцільною при пошуку підходящих товарів, послуг, виробів без їх попереднього придбання. 4) Прозорість інформації. Дозволить бачити себе і навколишній світ із заданим ступенем деталізації за допомогою сенсорів і internet-камер, розміщених як в офісі, так і по всій планеті. Зворотний бік медалі - як сховатися і зберегти особистий простір. 5) Інновації у бездротовому зв'язку. Дозволять запускати будь-який додаток на будь-якому пристрої у будь-якій точці планети. Тут слід очікувати появу конфліктів при розподілі частот між телекомунікаційними операторами, радіо і телебаченням, кабельними і супутниковими компаніями, провайдерами Internet послуг. Слід очікувати інтегрованого вирішення даного питання під пріоритетом бездротових технологій з мобільними Internet-сервісами. 6) Нові апаратні та про-програмні платформи. Підвищується рівень віртуалізації. Число додатків, що працюють на одній машині під управлінням різних операційних систем, зростає експоненціально. «Хмарні обчислення», коли користувач платить за застосування комп'ютерної інфраструктури та додатків провайдерам, які бережуть дані клієнтів на своїх серверах, істотно змінять всю структуру обчислень. Стають більш реальними перспективами нанотехнологій, молекулярних, квантових і оптичних обчислень. Замість кремнієвих чіпів будуть працювати більш легкі і дрібні елементи: атоми, ДНК, спини електронів і світло. 7) Розумний світ. Семантичні і мережні технології дозволять обчислювальним пристроям інтерпретувати інформацію за алгоритмами природного інтелекту, будь то

текст, мова, зображення або життєві ситуації. Комп'ютери вчитимуть, давати рекомендації, робити передбачення на основі інформації, отриманої від навколишнього середовища і конкретної людини. Розвиваються засоби семантічного пошуку в Internet з самонавчанням. Створюється справді розумний світ, де люди і машини будуть вміти міркувати і спілкуватися один з одним на основі поєднання семантичних і мережевих технологій. Це призведе до появи штучного інтелекту, а можливо сверхінтеллекту, читаючого думки людини. Джерело: http://www.pcweek.ru/.

Згадані вище тенденції у розвитку IT-індустрії надають конструктивно мислячим вченим шляхи ефективного застосування творчого потенціалу, який може бути затребуваний на ринку електронних технологій у вигляді нових математичних, технологічних та програмно-апаратних додатків. Інше застосування згаданих тенденцій - добра можливість верифікувати відносно них практично орієнтовані наукові розробки в частині їх валідності та ринковій привабливості для світової IT-індустрії.

5.8. Література

1. Kwang-Ting (Tim) Cheng. The Need for a SiP Design and Test Infrastructure // IEEE Design and Test of Computers.– May–June, 2006.– P. 181.

2. Fontanelli A. System-in-Package Technology: Opportunities and Challenges // Quality Electronic Design, 2008.– ISQED 2008, 9th International Symposium.– March, 2008.– P. 589 – 593.

3. Appello D., Bernardi P., Grosso M., Reorda M.S. System-in-package testing: problems and solutions // IEEE Design & Test of Computers.– Vol. 23, Issue 3.– May-June, 2006.– P. 203 – 211.

4. Subhasish M.; Huang W.-J., Saxena N.R., Yu S.-Y., McCluskey E.J. Reconfigurable architecture for autonomous self-repair // IEEE Design & Test of Computers.– Volume 21, Issue 3.– May-June, 2004.– P. 228-240.

5. Ross R., Hall R. A FPGA Simulation Using Asexual Genetic Algorithms for Integrated Self-Repair // Adaptive Hardware and Systems, 2006.– AHS 2006.– First NASA/ESA Conference on Volume.– Issue 15-18 June 2006.– P. 301 – 304.

6. Habermann S., Kothe R., Vierhaus H.T. Built-in self repair by reconfiguration of FPGAs // Proceedings of the 12th IEEE International Symposium on On-Line Testing.– 2006.– P. 187-188.

7. S. Pontarelli, M. Ottavi, V. Vankamamidi, A. Salsano, F. Lombardi. Reliability Evaluation of Repairable/Reconfigurable FPGAs // 21st IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'06) .– October, 2006.– P. 227-235.

8. Koal T., Vierhaus H. T. Basic Architecture for Logic Self Repair // 14th IEEE International On-Line Testing Symposium.– 2008.– P. 177–178.

9. Kothe R., Vierhaus H.T., Coym T., Vermeiren W., Straube B. Embedded Self Repair by Transistor and Gate Level Reconfiguration // IEEE Design and Diagnostics of Electronic Circuits and Systems.– 2006.– P. 208–213.

10. Mange D., Sipper M., Stauffer A., Tempesti G. Toward self-repairing and self-replicating hardware: the Embryonics approach // Evolvable Hardware, Proceedings of the Second NASA/DoD Workshop.– 2000.– P. 205-214.

11. Miclea L., Szilard E., Benso A. Intelligent agents and BIST/BISR – working together in distributed systems // Proceedings of the Test Conference.– 2002.– P. 940-946.

12. Rashad S. Oreifej, Carthik A. Sharma, Ronald F. DeMara. Expediting GA-Based Evolution Using Group Testing Techniques for Reconfigurable Hardware // Reconfigurable Computing and FPGA's, 2006. ReConFig 2006. IEEE International Conference.– Sept., 2006.– P. 1-8.

13. Anand D., Cowan B., Farnsworth O., Jakobsen P., Oakland S., Ouellette M.R., Wheater D.L. An on-chip self-repair calculation and fusing methodology // IEEE Design & Test of Computers.– Volume 20, Issue 5.– Sept.-Oct., 2003.– P. 67-75.

14. Wang S.-J., Tsai T.-M. Test and diagnosis of faulty logic blocks in FPGAs // IEEE Proceedings Computers and Digital Techniques.– Vol. 146, Issue 2.– March, 1999.– P.100 – 106.

15. Pontarelli S., Cardarilli G.C., Malvoni A,. Ottavi M., Re M., Salsano A. System-on-chip oriented faulttolerant sequential systems implementation methodology // IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems.– Oct., 2001.– P. 455 – 460.

16. Peter Rickert, William Krenik. Cell Phone Integration: SiP, SoC, and PoP // IEEE Design and Test of Computers.– May–June, 2006.– P. 188-195.

17. Rossen K. Discrete Mathematics and its Applications, McGraw Hill, 2003.– 824p.

18. IEEE 1500 Web Site. http://grouper.ieee.org/groups/1500/.

19. Парфентий А.Н., Хаханов В.И., Литвинова Е.И. Модели инфраструктуры сервисного обслуживания цифровых систем на кристаллах // АСУ и приборы автоматики.– 2007.– Вып. 138.– С.83-99.

20. Hahanov V., Hahanova A., Chumachenko S., Galagan S. Diagnosis and repair method of SoC memory // WSEAS transactions on circuits and systems.– 2008.– Vol.7.– P. 698-707.

21. Hahanov V., Obrizan V., Litvinova E., Ka Lok Man. Algebra-logical diagnosis model for SoC F-IP // WSEAS transactions on circuits and systems.– 2008.– Vol. 7.– P. 708-717.

6. ТЕСТУВАННЯ ПРОГРАМНО-АПАРАТНИХ КОМПОНЕНТІВ SOC

Пропонуються інноваційні технології тестопридатного проектування програмних і апаратних продуктів, орієнтовані на ефективну розробку тестів і верифікацію компонентів цифрових систем на кристалах шляхом використання графових моделей компонентів SoC.

Мета - адаптувати методи тестування і верифікації цифрових систем для перевірки працездатності програмних компонентів шляхом використання технологій тестопридатного проектування і графових моделей автоматного подання функціональностей.

Задачі: 1 Класифікація основних напрямів застосування технологій тестопридатного проектування цифрових систем на кристалах в задачах тестування та верифікації програмних продуктів. 2. Розробка універсальної моделі програмного і апаратного компонента у вигляді орієнтованого графа регістрових передач та управління, на якому можна вирішувати завдання тестопригодного проектування, синтезу та аналізу тестів. 3. Використання технологічної моделі ассерція для програмного продукту при тестуванні і верифікації цифрових систем на кристалах. 4. Розробка метрики оцінювання тестопридатності (спостереження та управління) програмних і апаратних продуктів, моделі яких зображені орієнтованими графами регістрових передач та управління.

6.1. Тестопридатність програмно-апаратних продуктів

Подано технології та моделі тестопридатного проектування програмних продуктів, засновані на модифікаціях технологічних і технічних рішень синтезу та аналізу цифрових систем на кристалах. Запропоновані універсальні моделі подання програмних та апаратних компонентів у вигляді орієнтованих графів реєстрових передач і управління обчислювальним процесом, які використовуються для тестопридатного аналізу і синтезу тестів програмно-апаратних модулів.

Ініціюючим ядром появи нових технологій тестування та верифікації в програмній та комп'ютерній інженерії слід вважати силіконовий кристал, який є основою для створення обчислювальних і/або комунікаційних пристроїв. Кристал розглядається як випробувальний полігон для апробації нових засобів і методів трасування, розміщення, синтезу та аналізу компонентів. Технологічні рішення, які витримали випробування часом в мікроелектроніці, далі захоплюються і адаптуються в макроелектроніці, що зображена комп'ютерними системами та мережами. Ось деякі історичні факти, пов'язані з наступністю розвитку технологічних інновацій.

1. Стандарты граничного сканування [4] на рівні плати та кристала призвели док появи механізму асерцій для тестування і верифікації програмних продуктів.

2. Засоби аналізу тестопридатності [1,2] (керованості і спостерігаємості) цифрових структур можна адаптувати для оцінки програмного коду з метою визначення критичних місць і наступного поліпшення програмного продукту щодо критерію тестованості.

3. Технології аналізу якості покриття [18] наперед заданих несправностей тестовими блоками слід використовувати для створення таблиці покриття дефектів програмних продуктів з метою оцінки достовірності тестів і встановлення діагнозу.

4. Графова модель реєстрових передач Тетта-Абрахама [15], С.Г. Шаршунова [16-17] використовуються при тестуванні програмних продуктів, які шляхом структурно-логічного аналізу приводяться до більш технологічної форми.

5. Розділення автомата на керуючу [1,7] і операційну частини також застосовується для спрощення процесу верифікації програмного коду на основі попереднього синтезу графів управління і передачі даних.

6. Крива життєвого циклу апаратного виробу [8,9] відображає також часові стадії зміни yield при створенні, тиражуванні і супроводженні програмного продукту.

7. Платформенно-орієнтований синтез [3] HDL-коду (platform-based electronic system-level design) з використанням існуючих наборів компонентів (chip set) під керуванням GUI ізоморфний технології об'єктно-орієнтованого програмування на основі застосування напрацьованих бібліотек. Застосування технології Electronic System Level (ESL) [19-20] в програмуванні дає можливість використовувати програмні компоненти готових функціональностей з базових бібліотек, розроблених для створення нових програмних продуктів. У цьому випадку основна процедура проектування полягає у виконанні мепінга, який орієнтований на покриття функцій специфікації існуючими компонентами, де новий код становить не більше 10% проекту.

8. Поняття Testbench [6,21], що використовується для тестування і верифікації апаратних проектів за допомогою HDL-компіляторів, з'являється і у програмних продуктах, реалізованих на рівні мов C ++ і вище.

9. Платформенно-орієнтований синтез [3] Testbench (platform-based testbench synthesis) з використанням існуючих бібліотек тестів (ALINT) для компонентів - стандартизованих функціональностей F-IP SoC під управлінням інтерфейсу GUI слід застосовувати для генерування тестів програмних модулів на основі напрацьованих бібліотек провідних компаній.

10. Стандартні рішення сервісного обслуговування F-IP в рамках I-IP [12] можна використовувати для вбудованого тестування компонентів програмної системи, в тому числі і для відновлення працездатності дефектного програмного модуля.

11. Забезпечення планарности (двовимірних) в структурі взаємопов'язаних функціональних компонентів (IP-cores) створюваного програмного продукту орієнтоване на використання мультиядерних архітектур для технологічного розпаралелювання обчислювальних процесів [22-23], що в умовах технологічної революції, запропонованої компанією Intel, є досить актуальним.

12. Створення адресного простору для функціональностей SoC, реалізованих як в апаратному, так і у програмному виконанні [9-11], надає цифрової системі чудову властивість самовідновлення працездатності програмних та апаратних компонентів засобами I-IP. Прикладом тому може служити мультипроцесорне виконання апаратних продуктів, яке є стійким до виникаючих дефектів. При цьому адресований компонент, що відмовив, може бути замінений іншим, справним, в процесі виконання функціональності. Властивість адресуємоості слід використовувати при створенні критичного програмного продукту, в якому наявність адресованих діверсних (мультиверсних) компонентів надає програмній системі здатність заміняти один одного при виникненні несправностей.

13. Вельми цікавою та ринково привабливою уявляється технологічна проблема автономного внутрикристального самотестування, самодіагностування і самовідновлення із застосуванням зовнішніх засобів (і без них) [9-11], якої сьогодні займаються всі провідні компанії. Для вирішення даної проблеми залучаються сучасні бездротові і Internet технології дистанційного сервісного обслуговування, що входять в моду. Зворотний бік медалі полягає у несанкціонованому доступі до вмісту кристала на відстані, що може привести до небажаних деструктивних наслідків і виведення з ладу цифрового виробу. Проте, специфіка цифрових систем на кристалах полягає у дивовижній можливості виправляти помилки на відстані, завдяки наявності зв'язку кристала із зовнішнім світом за допомогою Internet або бездротових технологій (wi-fi, wi-max, bluetooth), присутніх у силиконі. Дистанційна корекція програмних помилок стає можливою завдяки використанню пам'яті (що займає до 94% силікону) SoC для зберігання програм, куди можна, у разі виявлення некоректності, записати новий код, який не має помилок. Дистанційна корекція апаратних помилок стала можливою завдяки використанню програмованих логічних інтегральних схем, куди можна, у разі виявлення несправності, записати новий bit stream, який не має помилок, що фактично створює нову апаратуру шляхом перепрограмування кристала.

Зближення і взаємопроникнення технологій призводить до ізоморфних методів проектування, тестування і верифікації стосовно програмних і

апаратних комплексів, що за суттю є закономірним процесом асиміляції прогресивних концепцій. Тому сприяє факт, що найбільш важливі параметри життєвого циклу вироби, такі як time-to-market і yield стають сумірними за часом і виходу придатної продукції. Крива життєвого циклу апаратного виробу, зображена на рис. 6.1, з точністю до ізоморфізму відображає часові етапи програмного продукту, який проходить аналогічні стадії: проектування, збільшення обсягу випуску і виробництво з доопрацюванням і супроводженням виробів. В контексті життєвого циклу існують дві актуальні проблеми, які пов'язані з підняттям графіка (кривої) вверх за вісью ординат, а також з компресією згаданої кривої з часовою вісью, що означає зменшення параметру time-to-market. Тут підвищення yield відбувається на всіх стадіях: design - за рахунок усунення помилок розробки; production ramp up - за рахунок виправлення коду, імплементованого в пам'ять системи на кристалі; volume - за рахунок випуску service pack, коректуючих помилки шляхом поширення через Internet або супутники (satellites).

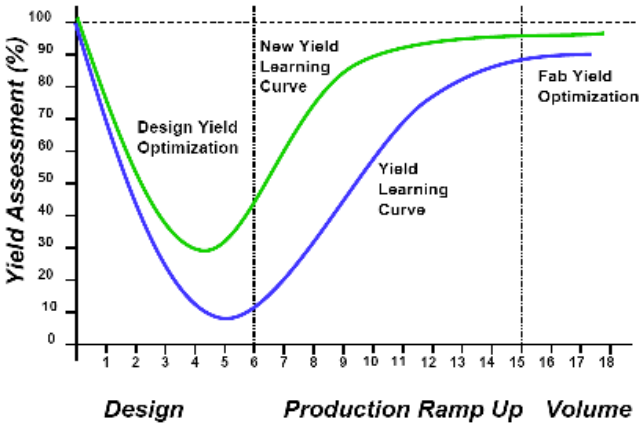


Рис. 6.1. Графік життєвого циклу програмно-апаратного комплексу

Таким чином, питання практичної імплементації ідеї в software або hardware визначається лише масштабністю проекту на ринку електронній продукції. На рис. 6.2 зображена картина перетворення software в hardware залежно від рівня масовості виробництва або потреби виробу на ринку. Кожен компонент, зображений у верхньому ряду, інтегрується з підмножиною різних типів конструктивів, об'єднаних в бібліотеки або chip sets.

Згідно [21,24,25] вартість верифікації програмно-апаратних продуктів на основі ASIC, IP-core, SoC становить 70% від загальних витрат проектування. Аналогічна оцінка, близько 80%, визначає розмірність Testbench-коду від загальної довжини формального опису проектованого виробу. Завдання, які вирішуються в процесі верифікації компонентів цифрової системи на кристалі, пов'язані з усуненням помилок проектування якомога на більш ранній його стадії з метою приведення коду прототипу до відповідності із його специфікацією. Пропуск помилки збільшує її вартість на порядок при переході від рівня проектування блоку до рівня кристала і далі до системи.

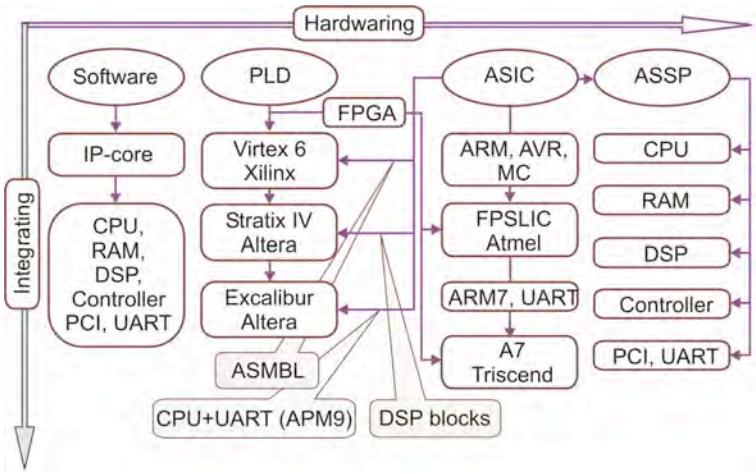


Рис. 6.2. Трансформування software в hardware

Введення в проект програмної надмірності - механізму асерцій [24,25] - дозволяє виконувати аналіз основних специфікованих умов в процесі моделювання проекту та діагностувати помилки в разі їх виявлення на ранніх стадіях проектування програми (C ++) або апаратури (HDL).

Мета - розробка ефективних моделей тестопридатного проектування програмних продуктів, імплементуємих в цифрову систему на кристалі, шляхом адаптації hardware design технологій і приведення програмних структур до існуючих стандартів та шаблонами тестування і верифікації.

Задачі: 1) Створити модель процесу верифікації та валідації software. 2) Розробити модель зображення програмного продукту, орієнтований на тестопридатний аналіз і синтез тестів для операційної та управляючої частин software.

6.2. Модель процесу верифікації проекту

Модель процесу верифікації проекту на системному рівні можна зобразити у вигляді узагальненого рівняння $T \oplus F = L$ або більш детально в компонентах:

$$\{T_w, T_t\} \oplus \{F_s, F_v\} = \{L_f, L_p, L_s\},$$

де $\{T_w, T_t\}$ – тестові впливи з очікуваними реакціями; $\{F_s, F_v\}$ – специфіцкована модель з доповненням для верифіикації проектованого виробу; $L = \{L_f, L_p, L_s\}$ – асерції для перевірки помилок управління та передачі даних.

Витрати на підготовку та проведення модельного експерименту визначаються формулою:

$$Q = \frac{Q(T_t) \times Q(F_s)}{Q(T_w) \times [Q(F_s) + Q(F_v)]},$$

де $Q(T_t) \times Q(F_s)$ – час машинного моделювання класичного теста перевірки дефектів або справної поведінки ув форматі Testbench на моделі специфікації; $Q(T_w) \times [Q(F_s) + Q(F_v)]$ – час аналізу тесту перевірки функціональностей на моделях специфікації та асерцій.

Для більш точного розуміння співвідношень між ключовими поняттями верифікація, валідація, асерція вводяться такі визначення [25,26]. Верифікація - процес аналізу системи або компонентів для визначення коректності перетворень вхідного опису на черговий стадії проектування. Валідація - процес визначення працездатності системи та її компонентів шляхом перевірки відповідності основним вимогам специфікації після виконання кожної стадії проектування. Сертифікація - письмова гарантія того, що система або її компоненти повністю задовольняють вимогам специфікації і прийнятні для використання за призначенням. Асерція - висловлювання системного рівня, визначальне коректність перетворень в процесі проектування щодо вхідного опису поточного етапу або вимог специфікації.

Слідуючи цьому визначенню, можна записати ассерцію у вигляді предиката $L_i = f(L_{i1}, L_{i2}, ..., L_{ij}, ..., L_{in}) = Y = \{0,1\}$, який на множині i_n змінних приймає значення {true, false}. Відповідно до згаданого предикату асерцію можна диференціювати на три типи $L = \{L_f, L_p, L_s\}$ - перевірки несправностей, функціональних шляхів і можливих станів проектованого виробу. Така класифікація дозволяє перевірити в програмному коді як механізм обчислень, так і керування обчислюваними процесами. Механізм ассерції є повна система висловлювань $L = (L_i, L_i, ..., L_i, ..., L_{ik})$ та засоби їх аналізу, призначені для верифікації та валідації процесу проектування, а також діагностування помилок.

Структурна модель процесу проектування в частині взаємодії компонентів валідації та верифікації зображено на рис. 6.3. Тут слід зазначити, що верифікація застосовується до компоненту (етапу) процесу проектування, а валідація - до системи (процесу) в цілому. Однак це поняття відносно, оскільки система може розглядатися як компонент для більш складного проектованого виробу, і навпаки, компонент за необхідної деталізації стає системою (структурою).

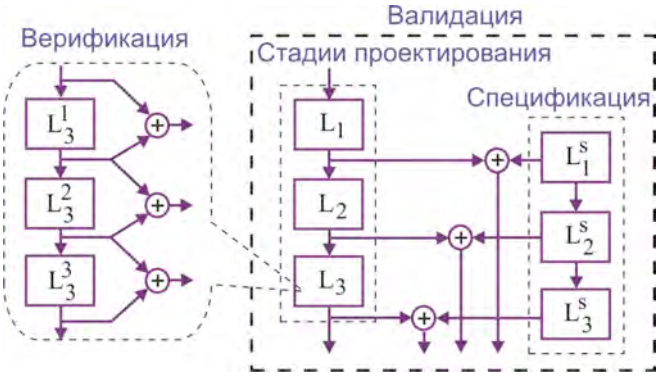


Рис. 6.3. Взаємодія процесів валідації та верифікації

Зокрема, процеси специфікації та валідації можуть мати одні й ті ж точки аналізу для побудови асерцій (рис. 6.4). Така регулярна структура характерна для верифікації системи, що з функціонально закінчених блоків або IP-cores, коли за результатом виконання чергового етапу проектування формуються ассерції відносно його вхідних умов і специфікації. Число точок спостереження для формування асерцій необхідно мінімізувати, але без шкоди якості верифікації. При цьому довжина тесту має тенденцію до зростання.

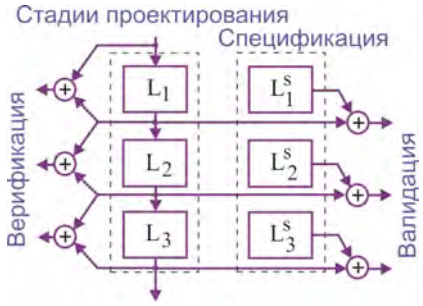


Рис. 6.4. Двовимірна взаємодія процесів валідації та верифікації

З урахуванням наведених визначень і пояснень модель макропроцесу веріфікації програмного проекту включає компоненти (рис. 6.5, зліва): 1) Test, написаний на одному з системних або HDL-мов. 2) Повну специфіковану структуру $F = (F_1, F_2, F_3, F_4, ..., F_n)$, оброблювану у середовищі моделювання. 3) Регістрова структура вхідних і вихідних змінних $R = (R_1^I, R_2^I, R_3^I, R_4^I, ..., R_4^I, R_1^O, R_2^O, R_3^O, R_4^O, ..., R_n^O)$, які використовуються для запису предикатів або асерцій, котрі ідентифікують технічні стани програмних модулів. 4) Сукупність асерцій (AC - Assertion Controller) для тестування проекту використовує регістрові змінні з R. Асерції зображені на спеціалізованій мові або мові створення проекту, розподілені в структурі функціональності. 5) Модулі діагностування і відновлення працездатності використовують інформацію, отриману від Assertion Controller. Вони вирішують

питання точного визначення місця і виду некоректності, а також виправлення коду або програмного модуля. Права частина рис. 6.5 ілюструє тестування апаратної частини проекту на основі використання стандарту IEEE 1500 граничного сканування (TAP - Test Access Port Controller). Очевидно, що технологія, представлена в лівій частині, є загальним випадком, придатним для тестування і верифікації апаратних і програмних ком-понентов цифрової системи на кристалі. Обидві технології є ізоморфними, тому моделі і методи тестування апаратури можуть і повинні бути адаптовані до вирішення задач верифікації програмних продуктів.

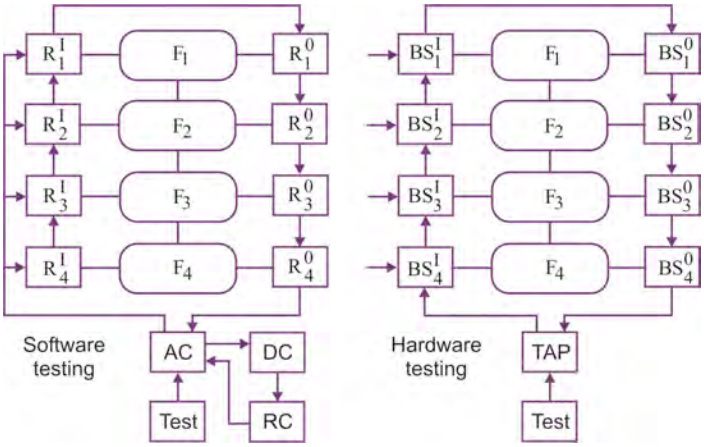


Рис. 6.5. Моделі тестування Hardware & Software

Запропонована модель верифікації проекту є технологічно замкненою щодо об'єкту діагностування (Testbench, формальна специфікація проекту), де як засоби діагностування виступає модуль аналізу асерцій і блок алгоритмів діагностування, працюючий під управлінням вектора Z експериментальної перевірки (ВЕП) вихідних сигналів:

$$Z = (Z_1, Z_2, ..., Z_j, ..., Z_n);$$

$$Z_j = \bigvee_{i=1}^{k} R_j^O, k = |T|;$$

$$R = (R_1^O, R_2^O, ..., R_j^O, ..., R_n^O).$$

Практично, якщо виконані умови тестопригодності [1], то ВЕП може однозначно ідентифікувати адресу (місце) і тип дефекту, виявленого на основі використання раніше визначеної або побудованої таблиці несправностей [13,14]. Відповідно до запропонованої структури уточнена модель діагностування проектованого виробу на основі механізму асерцій має вигляд: $M = \{T, F_S, F_V, Z, D\}$, де $P = f(T, F_S)$; $Z = g(T, F_S, F_V)$; $D = h(T, F_S, F_V)$ - функціональність проекту, умови для перевірки помилок

специфікованої моделі, модуль діагностування дефектів. Узагальнена модель процесу діагностування має наступний вигляд: $(T \oplus F_S \vee T \oplus F_V = 1) \rightarrow D = h(T, F_S, F_V)$, коли при фіксації ненульових сигналів асерція виконується діагностування дефектів і відновлення працездатності проекту.

Програмно-орієнтована структура процесу верифікації та діагностування проекту на основі аналізу асерцій включає архітектурні компоненти, взаємодія яких у часі показано на рис. 6.6.

Рис. 6.6. Програмна реалізація системи верифікації та діагностування

Тут система верифікації та діагностування містить наступні компоненти: 1) Модуль вводу асерцій, поданий транслятором мови опису ассерцій або проекту. 2) Внутрішня модель опису та обробки виразів у вигляді логіко-часових відносин, алгоритмів діагностування або директив верифікації. 3) Модуль аналізу асерцій для перевірки заданих формальних логіко-часових обмежень у процесі моделювання. 4) Блок синтезу квазіоптимальної моделі аналізу асерцій за наявними мовним конструкціями. 5) Структури даних, що забезпечують взаємодію та сумісність між множиною ассерцій і функціональністю в процесі компіляції та моделювання.

6.3. Тестування, діагностування та ремонт F-IP

Основною функцією IEEE 1500 Standard for Embedded Core Test (SECT) [4] є уніфікація інтерфейсу доступу для тестування внутрішнього модуля, що входить до складу цифрової системи на кристалі, і виключення впливу на нього з боку навколишнього середовища. Він дозволяє також тестувати середовище, зовнішнє за відношенням до модуля, шляхом ізоляції останнього і забезпечення його прозорості. Основна перевага всіх стандартів тестопригодного проектування полягає у забезпеченні сканування внутрішніх

змінних цифрової системи на платі і в кристалі, включаючи змінні функціональних блоків, розміщених на кристалі (рис. 6.7).

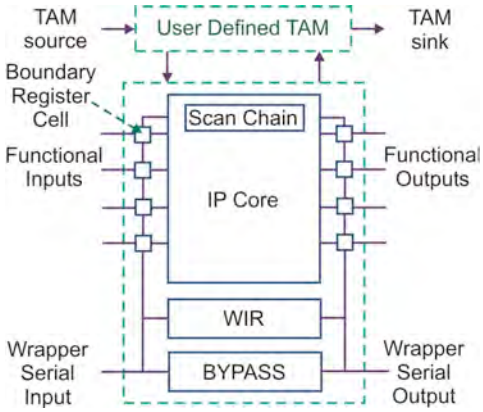


Рис. 6.7. Standard IEEE P1500 Wrapper Components

Кожен з компонентів системи підключається до TAP-контролеру і тестується як незалежна схема, яка повинна мати окремий тест, TAP-контроллер, регістр команд, boundary scan регістр і додаткову логіку для декодування команд. Використання стандарту дозволяє перевірити функціональність модуля, помилки міжз'єднань, визначати місце, причину і вид дефекту. Основним компонентом boundary scan регістра [4] є осередок сканування. Її слід розглядати як апаратурну надмірність для кожної лінії, яка підлягає спостереженню і управлінню. Тому число таких ліній в проекті має бути обмеженим.

Стандарт IEEE 1500 SECT слід розглядати як ефективний компонент інфраструктури сервісного обслуговування цифрових систем на кристалі. Його основне призначення - тестування всіх функціональностей F-IP, а також гальванічних зв'язків між ними. Наступним кроком в еволюції стандарту на шляху створення ремонтопридатності кристалів є розробка компонентів I-IP з сервісними функціями діагностування та відновлення працездатності SoC, які в сукупності з модулем тестування є ринково привабливими: $I = \{T, D, R\}$. Процедури діагностування та ремонту поки не регламентуються стандартами тестопридатного проектування зважаючи на складність і неоднозначність універсального рішення даної проблеми для різних типів обчислювальних пристроїв. Для нерегулярних або унікальних структур вирішення всіх трьох завдань грунтується на апріорній надмірності - диверсифікації функциональностей всіх компонентів, що входять до складу SoC. Тільки за таких умов можна говорити про внутрікристальний ремонт елементу, що відмовив. Що стосується регулярних структур, які мають надлишковість за визначенням, таких як мульти- або матричні процесори, то тут можна запропонувати структуру контролера, який буде поєднувати ви-

конання всіх трьох згаданих функцій за допомогою IEEE стандарту граничного сканування:

$$I = \{T, F, S, D, R\};$$

$$T = \{T^1, T^2, ..., T^i, ..., T^n\};$$

$$F = \{F^1, F^2, ..., F^i, ..., F^n\};$$

$$S = \{S^1, S^2, ..., S^i, ..., S^n\};$$

$$D = f(T, F, S) = F^D \in F;$$

$$R = g(D, F) = (F^R \subseteq F) \,\&\, (F^R \cup F^D = F).$$

Тут три компоненти (T, F, S) моделі є тести для функціональностей; компоненти, що представляють функції, а також осередки регістру граничного сканування для ідентифікації технічного стану функціональностей. Наступні два зображені функціями для виконання діагностування та ремонту SoC. Перша функція (D) визначає сукупність несправних компонентів, обчислюваних на основі вектора експериментальної перевірки S і тесту, який покриває всі дефекти функціоналів, що записисується у формі таблиці несправностей. Друга функція (R) формулює правила зменшення потужності компонентів шляхом виключення з адреси дефектних елементів і утворення нової працездатної підмножини F-IP SoC для його використання за призначенням.

Питання про те, де розміщувати тест і аналізатор перевірки функціональності, не є настільки проблематичним. Якщо мова йде про унікальні компоненти, то їх необхідно пов'язувати з сервісними I-IP компонентами без рознесення в просторі кристала. У разі регулярної матричної структури, то тут тест для всіх осередків однаковий, тому він повинен обслуговувати всі компоненти, також як і повинен існувати один аналізатор тестування, діагностування та ремонту на всю структуру.

Цікавим видається питання про перерозподіл обчислювальних ресурсів після фіксації дефектного осередку. Якщо є в системі додаткові запасні компоненти для ремонту, то задача зводиться до оптимального заміщення несправних елементів пам'яті резервними рядками і стовпцями. В іншому випадку існують інші моделі відновлення працездатності системи, які залежать від форми подання мультипроцессора. Лінійна або одновимірна форма адресації визначає сталість вхідних змінних n дешифратора і адресованих компонен-тів, пов'язаних між собою співвідношенням: $|A| = 2^n$. Матричне зображення мультипроцесора задає двовимірну адресацію компонентів, яка технологічно орієнтована на конвеєрну обробку даних. В обох випадках виконується дешифрування номера обчислювальної комірки за його адресою. Отже, з метою переадресації дефектного компонента на один зі справних необхідно модифікувати структуру дешифратора. Дана задача є суто технічною, і її

розв’язок зводиться до маскування адрес несправних елементів. Інший розв’язок пов'язаний з наявністю в структурі процесорів надлишкових резервних компонентів. В даному випадку постановка задачі може бути зведена до заміни одного або декількох дефектних процесорів справними елементами з існуючого резерву. Оптимальний розв’язок даної проблеми було розглянуто для випадку, коли в матриці пам'яті існує кілька несправних осередків [5]. Задача істотно ускладнюється, якщо функціональність цифрової системи вже є розпаралеленою на існуючій матриці процесорів

$$P = P_{ij}, |P| = m \times n,$$

що має дефектні елементи, і необхідно перерозподілити множину справних осередків $|P^*| \le |P|$ з метою квазіоптимального покриття функціональних підзадач підмножиною справних процесорних елементів.

Дуже актуальним видається створення формальної моделі програмного продукту, до якої можна застосувати технології від CAD і EDA з метою використання формальних методів синтезу тестів, оцінки їх якості покриття несправностей, визначення тестопридатності (спостереження та управління) для подальшої модернізації структури програмних компонентів. Для вирішення згаданих задач пропонується використовувати автоматну модель, подану двома основними компонентами (M^{OA}, M^{CA} – операційний і керуючий автомат) (рис. 6.8).

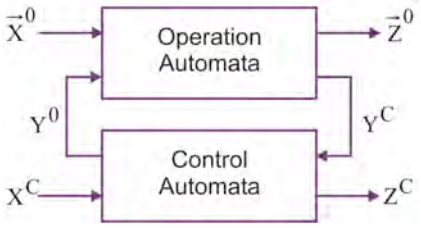


Рис. 6.8. Автоматна модель програмного модуля

Даному рисунку ставиться у відповідність наступна аналітична абстрактна структура:

$$\begin{cases} M = (M^{OA}, M^{CA}), \\ M^{OA} = \{\vec{X}^O, Y^C, Y^O, \vec{Z}^O\}; \\ M^{CA} = \{X^C, Y^O, Y^C, Z^C\}; \\ \vec{Z}^O = f^O(\vec{X}^O, Y^C); Y^O = g^O(\vec{X}^O, Y^C); \\ Z^C = f^C(X^C, Y^O); Y^C = g^C(X^C, Y^O). \end{cases}$$

Тут $\vec{X}^O, \vec{Z}^O$ – вектори або регистрові вхідні та вихідні змінні; Y^C, Y^O, Z^C – сигнали управління (ініціалізації) операцій, сповіщальні сигнали, сигнали моніторингу керуючого автомату відповідно; $f^O, g^O (f^C, g^C)$ – функції, що визначають відношення між інтерфейсними сигналами в операційному та керуючому автоматах.

Однак подана автоматна модель $M = (M^{OA}, M^{CA})$ не є технологічною для користувача при вирішенні практичних завдань тестопридатного проектування. Натомість пропонується процесорно (програмно-) орієнтована модифікація, що складається з двох графових компонентів з орієнтованими дугами:

$$\begin{cases} M = (M^{OR}, M^{CG}), \\ M^{OR} = \{R, I\}; \\ R = \{R_1, R_2, ..., R_i, ..., R_n\}, I = \{I_1, I_2, ..., I_j, ..., I_m\}; \\ R_i = f(R_{k,} I_j); \\ M^{CG} = \{S, E\}; \\ S = \{S_1, S_2, ..., S_i, ..., S_p\}, E = \{E_1, E_2, ..., E_j, ..., E_q\}; \\ S_i = f(S_{k,} E_j). \end{cases}$$

Тут M^{OR} – граф регістрових передач С.Г Шаршунова [16], що має множину вершин R, які задають всі компоненти пам’яті (регістри, тригери, лічильники, пам'ять, вхідні і вихідні шини), використовувані в програмі, і дуг, відзначених інструкціями I, які активізують передачу інформації між вершинами. Вираз $R_i = f(R_{k,} I_j)$ визначає функціональну залежність між суміжними вершинами $R_i \to R_{k,}$ які з’єднуються для (та шляхом) виконання операції $I_j \in I$. Компонент M^{CG} являє собою змістовний граф керуючого автомата, визначений на множині вершин S, які з'єднані орієнтованими дугами E, відзначеними умовами переходів. Вираз $S_i = f(S_{k,} E_j)$ визначає функціональну залежність між суміжними вершинами $S_i \to S_k$ керуючого графа, які поєднюються для виконання переходу $E_j \in E$.

Приклади завдання графів реєстрових передач і управління зображені на рис. 6.9 і 6.10 відповідно.

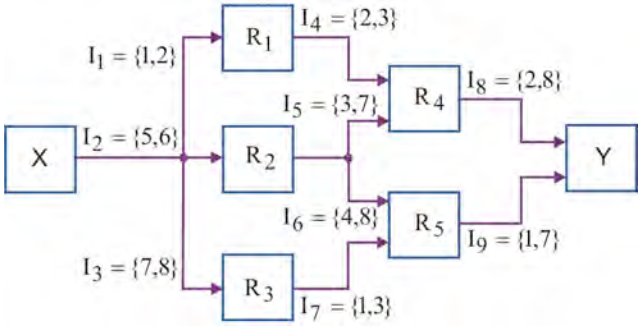


Рис. 6.9. Граф регістрових передач

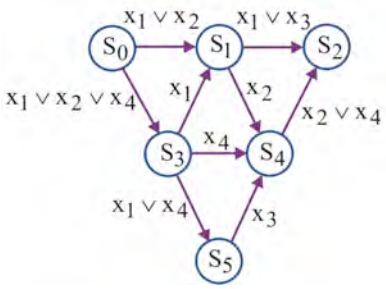


Рис. 6.10. Змітовний граф керуючого автомата

Переваги графових моделей полягають не тільки в структурному відображенні взаємодії функціоналів, але і в можливості застосовувати методи аналізу тестопридатності, оскільки орієнтовані графові моделі мають виражені напрямки інформаційних потоків, вхідні і вихідні вершини. Виходячи з досвіду оцінювання тестопридатності цифрових схем, можна запропонувати наступну метрику аналізу спостереження та управління запропонованих графових структур:

$$G = \{R, I\};$$
$$R = \{R_1, R_2, ..., R_n\}, I = \{I_1, I_2, ..., I_m\};$$
$$I_{ij} \in I_i \approx (R_p R_q);$$

$$C(R_q) = \frac{1}{k} \times \sum_{i=1}^{k} \left[\frac{1}{m} \times \left| \bigcup_j I_{ij} \in (R_p R_q) \right| \times C(R_p) \right];$$

$$O(R_p) = \frac{1}{k} \times \sum_{i=1}^{k} \left[\frac{1}{m} \times \left| \bigcup_j I_{ij} \in (R_p R_q) \right| \times O(R_q) \right];$$

$$C(R_x) = 1;\ O(R_y) = 1.$$

Тут модель програмного (апаратного) модуля подана графом $G = \{R, I\}$, що складається з вершин (регистрів) та дуг (інструкцій). Кожна дуга графа

відзначена не менш, ніж однією операцією $I_{ij} \in I_i \approx (R_p R_q)$, що формує множину команд, яка належить дузі $(R_p R_q)$. Критеріий керованості вершини $C(R_q)$ залежить від керваності попередньої вершини $C(R_p)$, а також від наведеної адитивної міцності множини команд

$$\frac{1}{k} \times \sum_{i=1}^{k} \left[\frac{1}{m} \times \left| \bigcup_j I_{ij} \in (R_p R_q) \right| \right] = \frac{1}{k} \times \sum_{i=1}^{k} \left[\frac{1}{m} \times d \right] = \frac{1}{k} \times \sum_{i=1}^{k} \left[\frac{d}{m} \right],$$

що активізують k дуг, які входять до аналізованої вершини $C(R_q)$. Тут кожна дуга графа містить d операцій (m - загальна кількість команд), що ініціюють передачу інформації в $(R_p R_q)$. За аналогією формується критерій оцінювання спостерігаємості $C(R_q)$, орієнтований на аналіз вершин наступників і дуг, що виходять з $C(R_q)$. Перевагою введених моделей і критеріїв оцінювання спостереження та управління є їх універсальність, орієнтована на виконання прямої та обратної імплікації на графі, а також їх інваріантність по відношенню до тестопридатності аналізу і синтезу тестів програмних і апаратних компонентів. Керованість $C(R_x) = 1$ вхідних і наблюдаемість $O(R_y) = 1$ вихідних вершин графа ініціюється одиничними значеннями. За мірою просування аналізу вершин до внутрішніх ліній значень згаданих оцінок можуть тільки зменшуватися.

Таким чином, вершини графа подані такими компонентами: вхідні змінні, вихідні змінні, реєстрові змінні, блок АЛУ, масиви пам'яті, зображені у форматі їх завдання у програмному (апаратному) модулі. Дуги ідентифікують сукупність операндів (команд), які здійснюють передачу (перетворення) інформації між вершинами. Повна модель пристрою, що подана графами регістрових передач та управління, охоплює всі оператори передачі та управління даними у програмному (апаратному) модулі, що необхідно для синтезу тестопридатного продукту. При цьому синтез тесту грунтується на розв’язуванні задачі покриття операторами Testbench всіх шляхів і вершин в графах регістрових передач та управління.

Інтегральна оцінка тестопридатності вершини в графі обчислюється за формулою: $T(R_i) = C(R_i) \times O(R_i)$.

Загальна тестопридатність графа програмного (апаратного) продукту визначається відповідно до виразу: $T_{total} = \frac{1}{n} \sum_{i=1}^{n} T(R_i)$.

Для прикладу, поданого графом регістрових передач (див. рис. 6.9), нижче наводиться розрахунок тестопридатності.

Показники керованості:

$$C(X) = 1;$$

$$C(R_1) = C(X) \times 1 \times \frac{d}{m} = 1 \times 1 \times \frac{2}{8} = \frac{2}{8} = 0{,}25;$$

$$C(R_2) = C(X) \times 1 \times \frac{d}{m} = 1 \times 1 \times \frac{2}{8} = \frac{2}{8} = 0{,}25;$$

$$C(R_3) = C(X) \times 1 \times \frac{d}{m} = 1 \times 1 \times \frac{2}{8} = \frac{2}{8} = 0{,}25;$$

$$C(R_4) = \left(\left[C(R_1) \times \frac{d}{m}\right] + \left[C(R_2) \times \frac{d}{m}\right]\right) \times \frac{1}{2} = \frac{1}{16} = 0{,}0625;$$

$$C(R_5) = \left(\left[C(R_2) \times \frac{d}{m}\right] + \left[C(R_3) \times \frac{d}{m}\right]\right) \times \frac{1}{2} = \frac{1}{16} = 0{,}0625;$$

$$C(Y) = \left(\left[C(R_4) \times \frac{d}{m}\right] + \left[C(R_5) \times \frac{d}{m}\right]\right) \times \frac{1}{2} = \frac{1}{64} = 0{,}015625.$$

Мінімальну керованість має вершина Y.

Обчислення спостерігаємості:

$$O(Y) = 1;$$

$$O(R_4) = O(Y) \times \frac{2}{8} = 1 \times \frac{2}{8} = 0{,}25;$$

$$O(R_5) = O(Y) \times \frac{2}{8} = 1 \times \frac{2}{8} = 0{,}25;$$

$$O(R_3) = O(R_5) \times \frac{2}{8} = \frac{1}{4} \times \frac{2}{8} = 0{,}0625;$$

$$O(R_1) = O(R_4) \times \frac{2}{8} = \frac{1}{4} \times \frac{2}{8} = 0{,}0625;$$

$$O(R_2) = \left[\left(O(R_4) \times \frac{2}{8}\right) + \left(O(R_5) \times \frac{2}{8}\right)\right] / 2 = \frac{1}{16} = 0{,}0625;$$

$$O(X) = \left[\left(O(R_1) \times \frac{2}{8}\right) + \left(O(R_2) \times \frac{2}{8}\right) + \left(O(R_3) \times \frac{2}{8}\right)\right] / 3 = \frac{1}{64} = 0{,}015625.$$

Мінімальне значення спостерігаємості має вершина X.

Обчислення тестопридатності:

$T(X) = 1 \times 0{,}015625 = 0{,}015625;$

$T(R_1) = 0{,}25 \times 0{,}0625 = 0{,}015625;$

$T(R_2) = 0{,}25 \times 0{,}0625 = 0{,}015625;$

$T(R_3) = 0{,}25 \times 0{,}0625 = 0{,}015625;$

$T(R_4) = 0{,}0625 \times 0{,}25 = 0{,}015625;$

$T(R_5) = 0{,}0625 \times 0{,}25 = 0{,}015625;$

$T(Y) = 0{,}015625 \times 1 = 0{,}015625.$

Загальна тестопридатність схеми $T_{total} = 0{,}015625.$

Обчислення показників тестопридатності для графа керуючого автомата (див. рис.6.10) здійснюється аналогічно.

Визначення керованості графа:

$C(S_0) = 1;$

$$C(S_1) = \left(\left[C(S_0) \times \frac{d}{m}\right] + \left[C(S_3) \times \frac{d}{m}\right]\right) \times \frac{1}{2} = \frac{11}{32} = 0{,}34375;$$

$$C(S_3) = C(S_0) \times 1 \times \frac{d}{m} = 1 \times 1 \times \frac{3}{4} = \frac{3}{4} = 0{,}75;$$

$$C(S_5) = C(S_3) \times 1 \times \frac{d}{m} = \frac{3}{4} \times 1 \times \frac{2}{4} = \frac{6}{16} = 0{,}375;$$

$$C(S_4) = \left(\left[C(S_1) \times \frac{d}{m}\right] + \left[C(S_3) \times \frac{d}{m}\right] + \left[C(S_5) \times \frac{d}{m}\right]\right) \times \frac{1}{3} = \frac{47}{384} = 0{,}1224;$$

$$C(S_2) = \left(\left[C(S_1) \times \frac{d}{m}\right] + \left[C(S_4) \times \frac{d}{m}\right]\right) \times \frac{1}{2} = \frac{179}{1536} = 0{,}11654.$$

Таким чином, мінімальну керованість має вершина S2.

Обчислення спостерігаємості:

$O(S_2) = 1;$

$$O(S_4) = O(S_2) \times 1 \times \frac{2}{4} = 1 \times 1 \times \frac{2}{4} = 0{,}5;$$

$$O(S_5) = O(S_4) \times 1 \times \frac{1}{4} = \frac{1}{2} \times 1 \times \frac{1}{4} = \frac{1}{8} = 0{,}125;$$

$$O(S_3) = \left[\left(O(S_1) \times \frac{1}{4}\right) + \left(O(S_4) \times \frac{1}{4}\right) + \left(O(S_5) \times \frac{2}{4}\right)\right] / 3 = \frac{7}{96} = 0{,}07292;$$

$$O(S_1) = \left[\left(O(S_2) \times \frac{2}{4}\right) + \left(O(S_4) \times \frac{1}{4}\right)\right] / 2 = \frac{1}{8} = 0{,}125;$$

$$O(S_0) = \left[\left(O(S_1) \times \frac{2}{4}\right) + \left(O(S_3) \times \frac{3}{4}\right)\right] / 2 = \frac{15}{256} = 0{,}05859375.$$

Мінімальне значення спостерігаємості має вершина S0.

Обчислення тестопридатності:

$T(S_0) = 1 \times 0{,}05859375 = 0{,}05859375;$

$T(S_1) = 0{,}34375 \times 0{,}125 = 0{,}04296875;$

$T(S_2) = 0{,}11654 \times 1 = 0{,}11654;$

$T(S_3) = 0{,}75 \times 0{,}07292 = 0{,}05469;$

$T(S_4) = 0{,}1224 \times 0{,}5 = 0{,}0612;$

$T(S_5) = 0{,}375 \times 0{,}125 = 0{,}046875.$

Вершина S1 має найгіршу тестопридатність.

Загальна тестопридатність схеми $T_{total} = 0{,}063478.$

Таким чином, запропонований метод оцінки тестопридатності має наступні переваги: 1) висока технологічність і універсальність відносно його використання для оцінки тестопридатності графа регістрових передач і управління; 2) можливість визначати вузькі місця в програмному або апаратному продукті з метою модифікації структури проекту; 3) вибір кращого проекту шляхом порівняння тестопридатності альтернативних варіантів.

6.4. Синтез і аналіз тестів на граф-моделі

Для вирішення задачі покриття операторами Testbench всіх шляхів і вершин в графах реєстрових передач та управління (див. рис. 6.10) будується матриця (таблиця), де сукупність рядків являє собою мінімальну множину шляхів, що покривають всі компоненти графа, поданого на рис. 6.9. Згадана множина дуг записується у формі ДНФ вершин [13], де кожен терм є унікальний одновимірний шлях, від вхідного порту до виходу, що покриває підмножина внутрішніх ліній:

$$P = X14Y \vee X24Y \vee X25Y \vee X35Y.$$

Кожній дузі ставиться у відповідність сукупність активизуючих інструкцій (об'єктів), записаних за допомогою операції диз'юнкції (кон'юнктівний терм). Це дає можливість записати тест мінімального покриття всіх вершин і дуг графа за допомогою ініціюючих команд:

$$P = [(1 \vee 2)(2 \vee 3)(2 \vee 8)] \vee [(5 \vee 6)(3 \vee 7)(2 \vee 8)] \vee$$
$$\vee [(5 \vee 6)(4 \vee 8)(1 \vee 7)] \vee [(7 \vee 8)(1 \vee 3)(1 \vee 7)].$$

Подальше перетворення ДНФ пов'язано з розкриттям дужок з метою получения повного тесту, який перевіряє всі транзакції в графі (покриття всіх вершин, дуг і інструкцій):

$$P = [(122) \vee (222) \vee (132) \vee (232) \vee (128) \vee (228) \vee (138) \vee (238)] \vee$$
$$\vee [(532) \vee (632) \vee (572) \vee (672) \vee (538) \vee (638) \vee (578) \vee (678)] \vee$$
$$\vee [(541) \vee (641) \vee (581) \vee (681) \vee (547) \vee (647) \vee (587) \vee (687)] \vee$$
$$\vee [(711) \vee (811) \vee (731) \vee (831) \vee (717) \vee (817) \vee (737) \vee (837)].$$

Якість тесту формується шляхом перевірки всіх вершин, дуг графа і активізації множини команд (об'єктів), задіяних у програмному модулі. Повний тест, поданий в ДНФ, орієнтований на перевірку транзакцій, реалізованих у вигляді послідовностей операцій (об'єктів). При цьому вважається, що апаратні компоненти (регістри, пам'ять, змінні), які використовуються в програмі, є працездатними. Будемо також вважати, що всі оператори в компіляторі мови є проацездатними. Тому таблиця несправностей орієнтована на перевірку послідовностей операцій (об'єктів) на дугах, що створюють шляхи активізації даних до точок спостереження, зображених вершинами графа, які складають програмний продукт. Нижче дана таблиця перевірки несправностей програмних об'єктів на повному тесті $P = X14Y \vee X24Y \vee X25Y \vee X35Y$, де тестові набори не розшифровуються (записані у загальному вигляді):

T_i / I_j	I_{11}	I_{12}	I_{25}	I_{26}	I_{37}	I_{38}	I_{42}	I_{43}	I_{53}	I_{57}	I_{64}	I_{68}	I_{71}	I_{73}	I_{82}	I_{88}	I_{91}	I_{97}	V
X14Y	1	1					1	1							1	1			1
X24Y			1	1					1	1					1	1			0
X25Y			1	1							1	1					1	1	1
X35Y					1	1							1	1					0
Faults	1	1					1	1			1	1					1	1	

Тому глибина діагностування для такого тесту при фіксації вектора експериментальної перевірки V = (1010) визначається великою кількістю підозрюваних дефектів: Faults = $\{I_{11}, I_{12}, I_{42}, I_{43}, I_{64}, I_{68}, I_{91}, I_{97}\}$.

Квазімінімальна множина термів ДНФ, які розрізняють всі одиночні дефекти програмних об'єктів графа регістрових передач, складають тест діагностування. Таким може бути наступна сукупність термів:

$$P = [(122) \vee (222) \vee (132) \vee (128)] \vee [(532) \vee (632) \vee (572) \vee (538)] \vee$$
$$\vee [(541) \vee (641) \vee (581) \vee (547)] \vee [(711) \vee (811) \vee (731) \vee (717)],$$

що формує таблицю несправностей, яка розрізняє дефекти всіх інструкцій, заданих в ДНФ:

T_i / I_j	I_{11}	I_{12}	I_{25}	I_{26}	I_{37}	I_{38}	I_{42}	I_{43}	I_{53}	I_{57}	I_{64}	I_{68}	I_{71}	I_{73}	I_{82}	I_{88}	I_{91}	I_{97}	V
122	1						1								1				1
222		1					1								1				0
132	1							1							1				1
128	1						1									1			1
532			1						1						1				0
632				1					1						1				0
572			1							1					1				1
538			1							1						1			1
541			1								1						1		0
641				1							1						1		0
581			1									1					1		0
547			1								1							1	0
711					1								1				1		0
811						1							1				1		0
731					1									1			1		0
717					1								1					1	0
Faults	1									1									

Для вектора експериментальної перевірки V = (1011001100000000) множина підозрюваних дефектів визначатиметься наступним виразом: $\text{Faults} = \{I_{11}, I_{57}\}$, що істотно менше, ніж отримано раніше для узагальненого тесту.

Таким чином, дуже важливо створити тест мінімальної довжини, але із заданою глибиною діагностування. Тестом може бути послідовність, яка ініціює обхід усіх шляхів у графі, або покриття всіх вершин (дуг) мінімальною безліччю одновимірних шляхів.

Сутність перетворення програмного продукту з метою діагностування в процесі виконання функцій полягає в створенні планарної графової структури, де дуги є послідовності операцій або фрагменти кодів, а вершини - точки спостереження інформації або асерції. Число точок контролю повинно бути достатнім для постановки діагнозу із заданою глибиною. В іншому випадку необхідно виконати аналіз тестопрідатності графа регістрових передач програмного продукту і визначити мінімальну додаткову кількість ліній спостереження у вигляді асерцій, які дозволять виявляти некоректно працюючі модулі з заданої (в межі до оператора) глибиною діагностування.

6.5. Технологія діагностування software модуля

При розробці програмних продуктів великого обсягу актуальною задачею є перевірка створюваного проекту на коректність виконуваних операцій. Складний програмний засіб містить величезну кількість розгалужень, і перевірку безпомилковеого виконання програми на кожній з логічних шляхів являє собою досить складну задачу. Далі на конкретному прикладі розглядається метод пошуку некоректних операторів (помилок або дефектів) у програмному засобі, заснований на уявленні алгоритму програми у вигляді графової структури вз метою подальшого створення тестів і діагностування

дефектів. Нехай необхідно верифікувати програмний засіб, що дозволяє обчислювати наступну суму функцій:

$$S = (x) + \omega(x),$$

$$x = \begin{cases} x+3; & x < 2; \\ 2x-3; & 2 \le x < 12; \\ -3x+7; & x \ge 12; \end{cases} \quad \omega(x) = \begin{cases} \sin(x+\pi/3), x < 2\pi/3; \\ \sin(\pi x)+2\ , x \ge 2\pi/3. \end{cases}$$

Один з можливих варіантів розв'язання задачі на мові С ++ поданий наступним лістингом:

Лістинг 6.1.

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{           const double Pi=3.14159;
            double F, w, f, x;
            cin>>x;
if (x<2) f=x+3;
else if ((x>=2) && (x<12)) f=2*x-3;
else f=-3*x+7;
if (x<2./3.*Pi)
w=sin(x+Pi/3);
else w=sin(Pi*x)+2;
F=f+w;
cout<<F<<endl;
            return 0;
}
```

Нехай помилка міститься в одному з операторів обчислювальної частини програми. Замість правильного оператора

```
else w=sin(Pi*x)+2;
```

записано наступний:

```
else w=sin(Pi*x) - 2;
```

Необхідно визначити некоректний оператор в програмному коді на основі технології тестування, що використовує граф-модель коду. Стадії діагностування програмного коду містять 4 процедури, зображені на рис. 6.11.

Рис. 6.11. Технологія тестування програмного кода

6.5.1. Побудова графа регістрових передач.

Дуги в графі являють собою сукупність фрагментів коду або окремих операцій (рис. 6.12), а вершинами є точки моніторингу інформації (регістри, змінні, пам'ять), які також використовуються для формування асерцій.

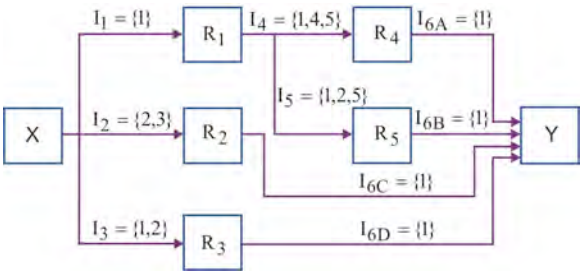


Рис. 6. 12. Граф регистровых передач

Число точок контролю в графі (регістри, змінні, пам'ять) повинно бути достатнім для постановки діагнозу із заданою глибиною. В іншому випадку необхідно виконати аналіз тестопридатності ГРП програмного продукту і визначити мінімальну додаткову кількість ліній спостереження для формування асерцій, які дозволять виявляти некоректно працюючі модулі із заданою глибиною діагностування. Кожна дуга (див. рис. 6.12) відзначена сукупністю арифметичних операцій: {1} - підсумовування; {2} - множення; {3} - віднімання; {4} - розподіл; {5} - знаходження тригонометричного синуса. Якщо в програмі зустрічається розгалуження, то число дуг, що виходять з вершини, дорівнює кількості суміжних вершин-стоків, яке формується командами розгалуження у відповідній ділянці програми.

Так, ділянці коду розглянутого прикладу:

```
if (x<2) f=x+3;
else if ((x>=2) && (x<12)) f=2*x-3;
else f=-3*x+7;
```

відповідають три дуги, які виходять з вершини X. Результати обчислень I_1, I_2, I_3, які залежать від змінної X, перевіряються у вузлах R_1, R_2, R_3 відповідно. У випадку виконання операцій I_1 реалізується наступне розгалуження, що подано рядками тексту програми:

```
if (x<2./3.*Pi) w=sin(x+Pi/3);
```

```
else w=sin(Pi*x)+2;
```

Далі незалежно від того, оператори якої з гілок виконувалися, виробляється загальна для всіх транзакцій операція підсумовування:

```
F=f+w;
```

Її виконання відбувається на різних дугах (об'єкти $I_{6A}, I_{6B}, I_{6C}, I_{6D}$), проте всі вони відповідають однієї і тієї же ділянці програмного коду, отже, коректне виконання операцій на одній з них виключає можливість помилки на трьох інших. У наступних стадіях діагностування програмного засобу, ці об'єкти об'єднані і зображені у вигляді одного - I_6. Отриманий результат перевіряється в кінцевій вершині Y.

Спосіб подання алгоритму програми за допомогою графової структури дозволяє наочно продемонструвати всі можливі варіанти виконання програми, а також полегшує проведення наступного етапу пошуку дефекту в програмному коді і створенні мінімального тесту.

6.5.2. Синтез та аналіз тестів.

Множина дуг записується у формі ДНФ вершин, де кожен терм являє собою одновимірний шлях від вхідного порту до виходу, який також покриває підмножина внутрішніх ліній: $P = X14Y \vee X15Y \vee X2Y \vee X3Y$. У сукупності одновимірні шляхи, подані у вигляді ДНФ, покривають всі можливі транзакції - вершини і дуги в графі. Кожній дузі ставиться у відповідність сукупність фрагментів коду або операторів - інструкцій, що активізують, записаних за допомогою диз'юнкції. Так, шлях X14Y активізує виконання операцій на дугах I_1, I_4, I_{6A}. При цьому дуги I_1 і I_{6A} мають тільки по одному оператору, а ідентифікатору I_4 відповідають послідовне виконання трьох операцій. Тест $P_1 = [(1)(1 \vee 4 \vee 5)(1)]$, що активізує шлях X14Y, забезпечує перевірку коректності виконуваних операцій. Таким чином, можна записати тест мінімального покриття всіх вершин і дуг графа за допомогою команд, які активізують дуги графа, а значить просування даних до точок спостереження:

$$P = [(1)(1 \vee 4 \vee 5)(1)] \vee [(1)(1 \vee 2 \vee 5)(1)] \vee [(2 \vee 3)(1)] \vee [(1 \vee 2)(1)].$$

Подальше перетворення ДНФ пов'язано з розкриттям дужок з метою отримання повного тесту, що дозволяє перевіряти транзакції в графі, які покривають всі вершини і дуги в різних поєднаннях:

$$P = (111 \vee 141 \vee 151) \vee (111 \vee 121 \vee 151) \vee (21 \vee 31) \vee (11 \vee 12).$$

Отриманий тест є надлишковим, що для програмного продукту великої розмірності не завжди прийнятно через значну кількість тестових наборів. Тому дуже важливо вміти створювати тест мінімальної довжини, але із заданою глибиною діагностування, що задовольняє користувача. Такий тест формується шляхом вирішення задачі покриття всіх вершин і дуг графа, і ак-

тивізації сукупностей фрагментів кодів. При проведенні тестування вважається, що апаратні компоненти, задіяні у програмному засобі, є працездатними.

6.5.3. Побудова таблиці несправностей.

ТН орієнтована на перевірку сукупностей фрагментів кодів на дугах, що створюють шляхи активізації даних до точок спостереження - вершин графа. За результатами порівняння експериментальних даних тестованого програмного засобу та очікуваних реакцій формується вектор експериментальної перевірки V. В разі неспівпадання результатів на спостережуваної лінії відповідна координата ВЕП приймає одиничне значення 1 для розглянутого тестового набору. Нижче зображена ТН сукупностей фрагментів кодів на повному тесті

$$P = X14Y \vee X15Y \vee X2Y \vee X3Y ,$$

де тестові набори записані в загальному вигляді (сукупність одновимірних шляхів активізації):

T_i / I_j	I_{11}	I_{22}	I_{23}	I_{31}	I_{32}	I_{41}	I_{44}	I_{45}	I_{51}	I_{52}	I_{55}	I_{61}	V
X14Y	1					1	1	1				1	0
X15Y	1								1	1	1	1	1
X2Y		1	1									1	0
X3Y				1	1							1	0
Faults									1	1	1		

Запис I_{jk} означає виконання оператора, що знаходиться на дузі I_j та має індекс k. Наприклад, I_{22} означає виконання последовності операторів дуги I_{22} при активізації шляху X2Y та операції множення, що відповідає фрагменту вихідного коду програми:

```
else if ((x>=2) && (x<12)) f=2*x-3;
```

Глибина діагностування для такого тесту при значенні вектора експериментальної перевірки V = (0100) визначається трьома можливими дефектми: $F = I_{51}, I_{52}, I_{55}$. Одиничне значення ВЕП для розглянутого тест-вектора означає, що при подачі другого набору відбувається активізація виконання відповідних операцій. Найменша множина термів ДНФ, яка розрізняє всі одиночні дефекти програмних фрагментів графа регістрових передач, становить мінімальний тест діагностування. Наступна сукупність термів - тут збігається з повним тестом - розрізняє дефекти всіх інструкцій, заданих в ДНФ:

$$P = (111 \vee 141 \vee 151) \vee (111 \vee 121 \vee 151) \vee (21 \vee 31) \vee (11 \vee 12).$$

Неможливість спрощення обумовлена тим, що виключення будь-якого з термів не забезпечує активізацію одного або декількох фрагментів. Далі будується повна і розширена таблиця несправностей, що формується за

допомогою записаної вище сукупності термів. Кожен з отриманих тестових наборів розбивається на складові частини - терми. Перший тестовий набір $(111 \vee 141 \vee 151)$ складається з трьох термів: (111), (141) і (151). Кожен з них посідає свою позицію в стовпці. За горизонталлю відкладають всі можливі операції, що виконуються в програмі, які умовно позначені I_{ik}, де j - ідентифікатор дуги в графі, k - оператор, виконануючий перетворення над даними на j-й дузі. Розглядається шлях графа, до якого належить досліджуваний терм. Наприклад, терм (141) відноситься до першого тестового набору, активізує шлях X14Y.

Розширена таблиця несправностей має вигляд:

$T_i \backslash I_i$	I_{11}	I_{22}	I_{23}	I_{31}	I_{32}	I_{41}	I_{44}	I_{45}	I_{51}	I_{52}	I_{55}	I_{61}	V
111_1	1					1						1	0
141	1						1					1	0
151_1	1							1				1	0
111_2	1								1			1	1
121	1									1		1	1
151_2	1										1	1	1
21_1		1										1	0
31			1									1	0
11				1								1	0
21_2					1							1	0

Кожна з цифр терма позначає виконання однієї з операцій на відповідній дузі графа. Перша цифра 1 забезпечує активацію оператора $\{1\}$ I_1, тому навпроти відповідного стовпця ставиться одиниця. Значення стовпців розширеної ТН переносяться з таблиці несправностей фрагментів кодів, визначеної на повному узагальненому тесті. Тільки тепер значення координати записується для кожного з термів тесту. Отримання розширеної таблиці несправностей дозволяє більш наочно продемонструвати результати впливу кожного з тестових наборів, а також спрощує подальше проведення процедури пошуку дефекту для визначення місцезнаходження несправності з наперед заданою глибиною діагностування.

6.5.4. Визначення діагнозу.

Відповідно до числа одиниць у векторі експериментальної перевірки V формується кількість діз'юнктівних термів КНФ. Кожен терм є порядковий запис дефектів через логічну операцію АБО, що впливають на спотворення вихідних сигналів функціональності. Далі здійснюється перетворення КНФ до ДНФ за правилами алгебри логіки, що дає наступний результат:

$$F = (I_{11} \vee I_{51} \vee I_{61})(I_{11} \vee I_{52} \vee I_{61})(I_{11} \vee I_{55} \vee I_{61}) =$$
$$= I_{11} \vee I_{11}I_{55} \vee I_{11}I_{61} \vee I_{11}I_{52} \vee I_{11}I_{52}I_{55} \vee I_{11}I_{52}I_{61} \vee I_{61}I_{11} \vee$$
$$\vee I_{11}I_{61}I_{51} \vee I_{11}I_{61} \vee I_{51}I_{11} \vee I_{11}I_{51}I_{55} \vee I_{11}I_{51}I_{61} \vee I_{11}I_{51}I_{52} \vee$$
$$\vee I_{51}I_{52}I_{55} \vee I_{51}I_{52}I_{61} \vee I_{51}I_{61}I_{11} \vee I_{55}I_{61} \vee I_{51}I_{61} \vee I_{11}I_{61} \vee I_{11}I_{55}I_{61} \vee$$
$$\vee I_{11}I_{61} \vee I_{11}I_{52}I_{61} \vee I_{52}I_{55}I_{61} \vee I_{52}I_{61} \vee I_{61}I_{11} \vee I_{61}I_{51} \vee I_{61}.$$

Для спрощення отриманого сполучення можливих дефектів використовуються закони булевої алгебри:

$$A \wedge A = A;\ A \vee B = B \vee A;\ (A \vee B)C = AC \vee BC;$$
$$(A \vee B) \vee C = A \vee (B \vee C);\ A \vee A = A;\ (A \wedge B) \vee A = A;\ (A \vee B) \wedge A = A,$$

щчто дпозволяєет отримати вираз:

$$F = I_{11} \vee I_{11}I_{55} \vee I_{11}I_{61} \vee I_{11}I_{52} \vee I_{11}I_{52}I_{55} \vee I_{11}I_{52}I_{61} \vee I_{61}I_{11} \vee I_{11}I_{61}I_{51} \vee$$
$$\vee I_{11}I_{61} \vee I_{51}I_{11} \vee I_{11}I_{51}I_{55} \vee I_{11}I_{51}I_{61} \vee I_{11}I_{51}I_{52} \vee I_{51}I_{52}I_{55} \vee I_{51}I_{52}I_{61} \vee$$
$$\vee I_{51}I_{61}I_{11} \vee I_{55}I_{61} \vee I_{51}I_{61} \vee I_{11}I_{61} \vee I_{11}I_{55}I_{61} \vee I_{11}I_{61} \vee I_{11}I_{52}I_{61} \vee$$
$$\vee I_{52}I_{55}I_{61} \vee I_{52}I_{61} \vee I_{61}I_{11} \vee I_{61}I_{51} \vee I_{61} = I_{11} \vee I_{51}I_{52}I_{55} \vee I_{61}.$$

Далі вилучаються такі елементи I_{jk} з F, які виконуються при використанні інших тестових наборів із значенням $V_i = 1$. Формується множина об'єктів, що містять ті операції, які однозначно і коректно виробляють перетворення даних при виконанні програми:

$$H = \{X14Y, X2Y, X3Y\} = \{(141) \vee (151) \vee (21_1) \vee (31) \vee (11) \vee (21_2)\} =$$
$$= I_{11} \vee I_{22} \vee I_{23} \vee I_{31} \vee I_{32} \vee I_{44} \vee I_{45} \vee I_{61}.$$

В результаті спрощення залишається єдиний терм ДНФ:

$$F' = F \setminus H = (I_{11} \vee I_{51}I_{52}I_{55} \vee I_{61}) \setminus (I_{11} \vee I_{22} \vee I_{23} \vee I_{31} \vee I_{32} \vee I_{44} \vee I_{45} \vee I_{61}) =$$
$$= I_{51}I_{52}I_{55},$$

Це означає, що програмний засіб помилково функціонує при виконанні одного з операторів {1,2,5} на дузі I_5.

Помилка дійсно була здійснена в лінійній ділянці програми, що належить дузі з послідовністю операторів I_5, а саме I_{51} – виконання операції замість підсумовування.

Отримання більш точного діагнозу (аж до оператора) можливе при використанні більшої кількості контрольних точок, що ускладнює пошук дефектів через необхідність складання довших тестів. Запропонований метод дозволяє виробляти аналіз програмного засобу на відмінність помилок в коді і допомагає визначити її місце розташування. Тестування і верифікація програмного продукту є однією з головних проблем при програмуванні, вирішення якої підвищує якість функціонування програмного засобу та доз-

воляє уникнути непередбачених результатів роботи. Запропонований метод грунтується на зображенні алгоритму програми за допомогою графової структури, де дуги є послідовності операцій або фрагменти кодів, а вершини - точки моніторингу інформації для складання асерцій. Створення мінімальної кількості тестових наборів дозволяє значно зменшити часові витрати на пошук помилки. При цьому тести повинні покривати всі можливі транзакції. Число точок контролю також має бути мінімальним, але достатнім для постановки діагнозу із заданою глибиною діагностування.

6.6. Висновки

Розглянуто інноваційні технології тестопридатного проектування програмних і апаратних продуктів [27-32], орієнтовані на ефективну розробку тестів і верифікацію компонентів цифрових систем на кристалах.

1. Показані основні напрями використання технологій тестопридатного проектування цифрових систем на кристалах в задачах тестування та верифікації програмних продуктів.

2. Подана універсальна модель програмного і апаратного компонента у вигляді орієнтованого графа регістрових передач та управління, на якому можна вирішувати завдання тестопридатного проектування, синтезу та аналізу тестів.

3. Розроблено технологічну модель асерцій для програмного продукту, що широко використовується для тестування і верифікації цифрових систем на кристалах.

4. Запропоновано метрику оцінювання тестопридатності (керованості і спостерігаємості) програмних і апаратних продуктів, моделі яких зображені орієнтованими графами регістрових передач та управління.

5. Запропоновано комплекс технологічних заходів і рекомендацій, орієнтованих на тестопридатності аналіз і синтез програмних продуктів.

6. Практична значущість запропонованих методик і моделей полягає у високій зацікавленості софтверних компаній в інноваційних рішеннях проблеми ефективного тестування та верифікації програмних продуктів, запропонованих вище.

6.7. Література

1. Abramovici M., Breuer M.A. and Friedman A.D. Digital System Testing and Testable Design. Computer Science Press.– 1998.– 652 p.

2. Bayraktaroglu Ismet, Orailoglu Alex. The Construction of Optimal Deterministic Partitionings in Scan-Based BIST Fault Diagnosis: Mathematical Foundations and Cost-Effective Implementations // IEEE Transactions on Computers.– 2005.– P.61-75.

3. Douglas Densmore, Roberto Passerone, Alberto Sangiovanni-Vincentelli. A Platform-Based taxonomy for ESL Design // Design&Test of computers.– September-October, 2006.– P. 359-373.

4. Francisco DaSilva, Yervant Zorian, Lee Whetsel, Karim Arabi, Rohit Kapur. Overview of the IEEE P1500 Standard // ITC International Test Conference.– 2003. – P.988–997.
5. Huang Rei-Fu, Chen Chao-Hsun, Wu Cheng-Wen. Economic Aspects of Memory Built-in Self-Repair // IEEE Design and Test of Computers.– 2007.– P. 164-172.
6. Rashinkar P., Paterson P., Singh L. System-on-chip Verification: Methodology and Techniques.–Kluwer Academic Publishers.– 2002.– 324 p.
7. Rossen K. Discrete Mathematics and its Applications.– McGraw Hill.–2003.– 824 p.
8. Zorian Yervant. What is Infrustructure IP? // IEEE Design & Test of Computers.– 2002.– P. 5-7.
9. Zorian Yervant, Gizopoulos Dmytris. Gest editors' introduction: Design for Yield and reliability // IEEE Design & Test of Computers.– 2004.– P. 177-182.
10. Zorian Y., Shoukourian S. Embedded-Memory Test and Repair: Infrastructure IP for SoC Yield // IEEE Design and Test of Computers.– 2003.– P. 58-66.
11. Zorian Yervant, Shoukourian Samvel. Embedded-Memory Test and Repair: Infrastructure IP for SoC Yield // IEEE Design and Test of Computers.– 2003.– P. 58-66.
12. Zorian Yervant. Guest Editor's Introduction: Advances in Infrastructure IP // IEEE Design and Test of Computers.– 2003.– 49 p.
13. Бондаренко М.Ф., Кривуля Г.Ф., Рябцев В.Г., Фрадков С.А., Хаханов В.И. Проектирование и диагностика компьютерных систем и сетей.– К.: НМЦ ВО.– 2000.– 306 с.
14. Пархоменко П.П., Согомонян Е.С. Основы технической диагностики (Оптимизация алгоритмов диагностирования, аппаратурные средства) / Под ред. П.П. Пархоменко.– М.: Энергия.– 1981.– 320 с.
15. Thatte S.M., Abraham J.A. Test generation for microprocessors // IEEE Trans. Comput.– 1980.– C-29. No 6.– P. 429-441.
16. Шаршунов С.Г. Построение тестов микропроцессоров. 1. Общая модель. Проверка обработки данных // Автоматика и телемеханика.– 1985.– №11.– С. 145-155.
17. Чипулис В.П., Шаршунов С.Г. Построение тестов микропроцессоров. 2. Проверка хранения и передачи данных // Автоматика и телемеханика.– 1986.–№1.– С. 139-150.
18. V.I.Hahanov, S.V.Chumachrnko, WGharibi, E.Litvinova. Algebra-logical method for SoC embedded memory repair // Proceedings of the 15 International Conference «Mixed design of integrated circuits and systems», Poland.– 2008.– P. 481- 486.
19. Jerraya A.A. System Level Synthesis SLS. TIMA Laboratory. Annual Report.– 2002.– P. 65-75.

20. Frank Ghenassia. Transaction Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems. Published by Springer.– 2005.– 282 p.
21. Bergeron, Janick. Writing testbenches: functional verification of HDL models.– Boston: Kluwer Academic Publishers.– 2001.– 354 p.
22. Shameem Akhter, Jason Roberts. Multi-Core Programming. Intel Press.– 2006.– 270 p.
23. Richard Gerber, Aart J.C. Bik, Kevin B. Smith, Xinmin Tian. The Software Optimization Cookbook. Intel Press.– 2006.– 430 p.
24. Foster H., Krolnik A., Lacey D., Assertions-based Design. Kluwer Academic Publishers.– 2003.– 392 p.
25. Rashinkar P., Paterson P., Singh L. System-on-chip Verification: Methodology and Techniques.– Kluwer Academic Publishers.–2002.– 393 p.
26. Meyer A.S. Principles of Functional Verification. Elsevier Science.– 2004.– 206 p.
27. Хаханов В.И., Каминская М.А., Егоров А.А., Побеженко И.А. Повышение качества теста на основе технологии Boundary Scan // Радиоэлектроника и информатика.– Харьков.– 2004.– №3.– С. 85-90.
28. Хаханов В.И., Каминская М.А., Зайченко С.А. Верификация цифровых устройств на основе использования анализа тестопригодности и асерционных бибилиотек // АСУ и приборы автоматики.– Харьков.– 2007.– №140.– С.75-83.
29. Kaminska M., Hahanov V., Kulak E., Guz O. Hierarchical Analysis of Testability for SoC // Proceedings of the 4th East-West Design and Test Workshop.– Russia.– 2006 .– P. 226-230.
30. Hahanov V., Kaminska M., Fomina E. Testability Analysis of Digital Design Verification // Proceedings of the 10th Biennial Baltic Electronic Conference BEC2006.– Tallinn, Estonia.– 2006.– P. 171-175.
31. Maryna Kaminska, Vladimir Hahanov, Anna Hahanova, Alexander Parfentiy Fault Coverage Improving based on Testability Analysis of the VHDL Code // Proceedings of the 9th International Conference The Experience of Designing and Application of CAD systems in Microelectronics.– Ukraine.– 2007.– P. 454-456.
32. Maryna Kaminska, Roman Prikhodchenko, Artem Kubirya, Pavel Mocar Testability analysis method for hardware and software based on assertion libraries // Proceedings of the 6th East-West Design and Test Symposium 2008.– Lviv. Ukraine.– 2008.– P.163-167.

7. ИЕРАРХИЧЕСКИЕ МОДЕЛИ ТЕСТИРОВАНИЯ СЛОЖНЫХ ЦИФРОВЫХ СИСТЕМ

7.1. Актуальность проблемы

К критическим цифровым системам на кристаллах предъявляются жесткие требования по их высокой надежности функционирования и управлению критическими объектами в реальном масштабе времени [1]. С другой стороны, такие системы, реализующие функции управления, вычисления, преобразования и доставки информации, являются сложными комплексами, состоящими из большого количества взаимодействующих компонентов [2]. Применение цифровых систем для управления критическими технологическими объектами вызывает ряд проблем, существенных с точки зрения безопасности их эксплуатации: структурная и функциональная сложность систем, повышающая вероятность ошибки при разработке; высокая трудоемкость создания диагностического обеспечения; разработка качественных и полных тестов для обнаружения отказов, дефектов и повреждений; выполнение жестких требований к времени поиска дефекта и идентификации неисправного компонента. Поэтому для обеспечения требований нормативных документов по безопасности эксплуатации [1] необходимы эффективные сервисные средства тестирования, диагностики и восстановления работоспособности всех подсистем, а также обеспечение требуемой глубины диагностирования в реальном масштабе времени за минимальное число тестовых проб. С точки зрения проблемы диагностирования программно-аппаратное изделие можно рассматривать как систему, имеющую несколько уровней иерархии. Самый нижний уровень системы представлен функциональными модулями IP-cores, входящими в состав современных интегральных схем (PLD, ASIC). Второй уровень – компоненты, содержащие процессоры, память. Третий – системы и сети на кристллах – SoC, NoC. Четвертый уровень представлен пакетами SoC и NoC – цифровых систем и сетей на кристаллах, обладающих по сравнению с системами на платах высокой частотой синхронизации и быстродействием, низким энергопотреблением, малыми размерами (1-4 см^2), широкой функциональностью, высокой степенью интеграции [3-6].

В технологиях тестирования и диагностирования сложных цифровых изделий наблюдаются сравнительно новые тенденции, определяемые такими направлениями, как: Design for Manufacturability, Design for Testability, Design for Operation, Design for Diagnosis, Design for Verification [7-50].

Важнейшим этапом разработки компонентов цифровых систем является процесс верификации, связанный с устранением всех ошибок проектирования как можно на более ранней стадии, что приводит к значительной эконо-

мии временных и материальных ресурсов (рис. 7.1). Поэтому внесение допустимой тестопригодной избыточности в состав проекта на ранней (системной) стадии разработки является широко распространенным методом повышения эффективности тестирования и диагностирования [8, 15, 33, 35, 39, 44, 48, 51, 52].

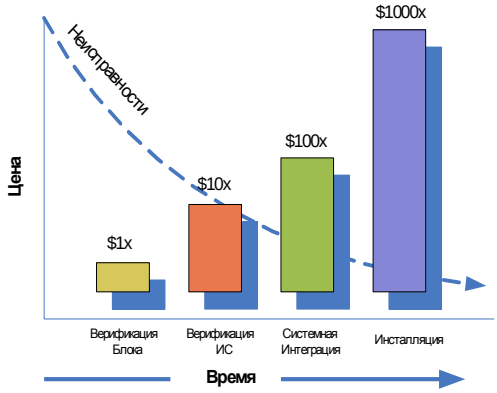


Рис. 7.1. Стоимость ошибок на стадиях проектирования

Другая интересная задача связана с использованием теста верификации, полученного на стадии системного проектирования, для проверки уже готового компонента (изделия) с минимальными дополнительными аппаратурными и программными затратами. Она может быть рассмотрена и решена в контексте использования стандарта IEEE 1500 SECT [7, 9, 33-48]. При этом механизм программно-аппаратной избыточности должен включать дополнительные точки контроля, которые необходимо ввести в проект с помощью Boundary Scan регистра и/или специальных (ad hoc) технологий. Таким образом, создаваемая избыточность проекта может быть многократно использована для проверки компонентов цифровой системы на всех стадиях его жизненного цикла.

Далее рассматривается иерархия системы от первого до четвертого уровней в целях создания моделей и методов их тестирования с заданной глубиной диагностирования.

Цель – существенное уменьшение времени тестирования и диагностирования сложных цифровых систем в процессе производства и эксплуатации на основе создания общей модели организации и проведения диагностического эксперимента, включающего условные и безусловные алгоритмы поиска дефектов с использованием IEEE стандартов тестопригодного проектирования [7-16,31,37,50-60].

Для достижения поставленной цели решаются следующие задачи: 1) выбор приемлемых методов тестирования сложных цифровых систем всех, обозначенных выше, уровней иерархии; 2) разработка иерархической модели орга-

низации и проведения диагностического эксперимента, включающей условные и безусловные алгоритмы поиска дефектов, ориентированные на использование стандартов тестопригодного проектирования; 3) построение ad hoc моделей тестирования, ориентированных на специфику представления цифровых устройств упомянутых уровней иерархии.

7.2. Выбор модели тестирования сложной цифровой системы

Объект исследования – сложная цифровая система, представленная несколькими уровнями иерархии и разрабатываемая на основе IEEE стандартов тестопригодного проектирования. Сложная цифровая система F, как объект тестирования, может быть представлена в виде кортежа: $F =< C, B, P, M >$, где $C = \{c_1, c_2, ..., c_n,\}$ – конечное непустое множество пакетов в системе, $B = \{b_1, b_2, ..., b_m\}$ – множество кристаллов в пакете, $P = \{p_1, p_2, ..., p_k\}$ – совокупность компонентов на кристалле, $M = \{m_1, m_2, ..., m_g\}$ – множество функциональных IP-модулей на чипе (в интегральной схеме). При этом выполняются следующие условия вложения или иерархии:

$$m_{ij}rs \in p_{ij}r \in b_{ij} \in c_i \in F, \ (i = \overline{1, n}; \ j = \overline{1, m}; \ r = \overline{1, k}; \ s = \overline{1, g}). \qquad (7.1)$$

Основное уравнение диагноза для рассматриваемого объекта исследования может быть представлено в следующем виде:

$$D = R^* \wedge L = (R^+ \oplus R^-) \wedge L = [(T \oplus F) \oplus (T \oplus F^*)] \wedge L, \qquad (7.2)$$

где параметры формулы определяются как: D – множество обнаруженных дефектов в процессе тестирования; L(Ti) – список неисправностей, обнаруживаемых тест-вектором $T_i \in T = (T_1, T_2, ..., T_i, ..., T_h)$, представленный в форме таблицы

$$L(t_i) = [L_{ij}], i = \overline{1, p}; \ j = \overline{1, q},$$

где каждому тесту ставится в соответствие подмножество проверяемых дефектов, p – число наблюдаемых выходов, q – количество линий в объекте; $R^-(T_i) = (R_1^-, R_2^-, ..., R_i^-, ..., R_p^-)$ – двоичный вектор элементарной экспериментальной проверки (ВЭП), равный числу наблюдаемых выходов системы; $R^+(T_i) = (R_1^+, R_2^+, ..., R_i^+, ..., R_p^+)$ – эталонный вектор состояния выходов; F – эталонная модель проверяемого объекта; F* – проверяемый объект.

Для организации диагностирования формируются координаты вектора

$$R^*(T_i) = (R_1^*, R_2^*, ..., R_i^*, ..., R_p^*)$$

в соответствии со следующим правилом:

$$R_i^* = R_i^+ \oplus R_i^-. \qquad (7.3)$$

Идеальная модель для диагностирования дефектов цифровой системы представляется в виде следующей структуры: $F = \{T \times L \times R\}$. Однако мощность такой трехмерной таблицы определяется выражением $|F| = h \times 2q \times p$. Если учесть, что размерность теста, в лучшем случае, есть квадрат от числа линий, то данное выражение приводится к следующему виду: $|F| = \{q^2 \times 2q \times p\} = 2q^3 p \approx q^4$.

Если число линий в реальных объектах от 100 000 до 1000 000, то размерность трехмерной таблицы диагностирования будет расположена в интервале значений $10^{20} \geq |F| \geq 10^{24}$. Естественно, что такой объем диагностической информации практически невозможно обработать в целях поиска дефекта в цифровой системе. Возникает проблема сжатия данных до приемлемых размеров. Практически сжимать можно все три компонента $\{T, L, R\}$: 1) тест, 2) дефекты и 3) реакции на входные наборы, но при этом теряется или точность установления диагноза, или время проведения тестового эксперимента.

Далее предлагается метод диагностирования с максимально возможной глубиной диагностирования, основанный на выполнении тестового эксперимента над моделью и объектом в реальном масштабе времени, когда исключается из рассмотрения параметр T, а объем информации переходит в плоскость – теряется одно измерение, связанное с длиной теста. В связи с этим вводится следующая модель диагностирования:

$$
\begin{aligned}
&1)\begin{cases} L(T_i) = F \oplus T_i; \\ R^*(T_i) = f(F, T_i, D^*); \\ D(T_i) = g[L(T_i), R^*(T_i)]. \end{cases} \\
&2)\begin{cases} D^s(T_i) = \bigwedge\limits_{\forall j(R_j^*=1)} L_j \,\&\, \overline{\bigvee\limits_{\forall j(R_j^*=0)} L_j}; \\ D^m(T_i) = \bigvee\limits_{\forall j(R_j^*=1)} L_j \,\&\, \overline{\bigvee\limits_{\forall j(R_j^*=0)} L_j}. \end{cases} \\
&3)\begin{cases} D_0^s(T) = D_0^s(T) \vee D^s(T_i) \,\&\, S^+(T_i); \\ D_1^s(T) = D_1^s(T) \vee D^s(T_i) \,\&\, \bar{S}^+(T_i). \end{cases} \\
&4)\begin{cases} D_0^m(T) = D_0^m(T) \vee D^m(T_i) \,\&\, S^+(T_i); \\ D_1^m(T) = D_1^m(T) \vee D^m(T_i) \,\&\, \bar{S}^+(T_i). \end{cases}
\end{aligned}
\tag{7.4}
$$

Здесь первые три равенства представляют процедуры: формирования списков проверяемых дефектов по наблюдаемым выходам объекта; вычисления

фактической реакции устройства на заданную входную последовательность; определения дефектов, присутствующих в изделии на момент тестирования. Вторая система уравнений позволяет определить наличие одиночных или кратных константных дефектов в устройстве путем выполнения двух процедур над содержимым таблицы L проверяемых на входном наборе T_i дефектов и вектора экспериментальной проверки R*. Третья система уравнений позволяет вычислить вектор одиночных дефектов, присутствующих в цифровой системе на момент тестирования, как функцию от вектора S+(T_i) исправного состояния линий и полного теста T. Четвертая система уравнений позволяет накапливать вектор кратных дефектов, присутствующих в цифровой системе на момент тестирования, также как функцию от вектора S+(T_i) исправного состояния линий. Результатом выполнения диагностического эксперимента, проводимого в реальном масштабе времени, является множество дефектов, одиночных или кратных, которые имеются в цифровой системе. При этом затраты памяти для хранения информации и проведения диагностического эксперимента определяются выражением:

$$Q^M = (p^x \times q^2) + (q \times p) + 7q + 3p. \qquad (7.5)$$

Если использовать встроенный генератор тестов, для которого нет необходимости хранить информацию, то объем памяти будет минимальным:

$$Q_A^M = (q \times p) + 7q + 3p. \qquad (7.6)$$

Естественно, что и вычислительная сложность проведения диагностического эксперимента является функцией от объема обрабатываемой памяти, представленной выражениями (7.5) и (7.6).

На рис. 7.2 представлена структура вычислителя для проведения диагностического эксперимента, которая построена на основе уравнений (7.4). Достоинством устройства является выполнение операций над таблицами и векторами, что легко реализуется как в компьютере, так и в кристалле PLD.

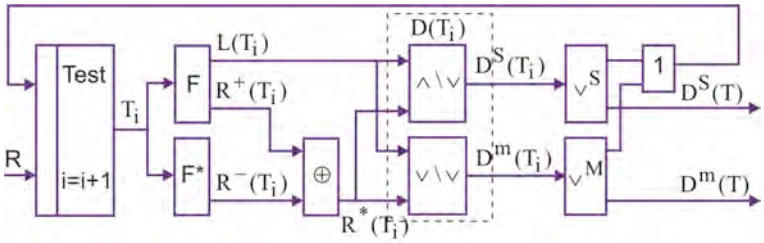


Рис. 7.2. Структурная модель диагностирования цифровой системы

В качестве примера структур данных, необходимых для проведения диагностического эксперимента, ниже предлагаются таблицы и векторы, ориентированные на иллюстрацию выполнения процедур, определенных упомянутым вычислителем (рис. 7.2) и уравнениями (7.4).

$$L(T_i) = [L_{ij}] \quad q = 8 \quad R^*(t) = R^-(t) \oplus R^+(t)$$

$$p = 4 \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ \hline \end{array} \quad \begin{array}{|c|} \hline 1 \\ \hline 0 \\ \hline 1 \\ \hline 0 \\ \hline \end{array} \quad \begin{array}{|c|} \hline 0 \\ \hline 0 \\ \hline 1 \\ \hline 1 \\ \hline \end{array} \quad \begin{array}{|c|} \hline 1 \\ \hline 0 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}$$

$$D^s(T_i) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$D^m(T_i) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

$$S^+(T_i) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array}$$

$$D_0^s(T) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$D_1^s(T) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$D_0^m(T) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$D_1^m(T) = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Здесь рассматривается объект диагностирования, имеющий 4 выхода (p=4), где каждому из них ставится в соответствие вектор проверяемых дефектов, равный общему числу линий в объекте, в данном случае q=8. Проверяемость дефекта в рассматриваемой таблице $L(T_i)$ идентифицируется символом 1, непроверяемость – нулем. Тип константного дефекта определяется инверсией от исправного состояния соответствующей линии вектора $S+(T_i)$. Состояния координат вектора экспериментальной проверки $R^*(T_i)$ вычисляются как сумма по модулю 2 векторов: исправного состояния модели $R+(T_i)$ и фактических реакций $R-(T_i)$ выходов реального объекта. Таким образом, единичное (нулевое) значение координаты вектора $R^*(T_i)$ идентифицирует несовпадение (совпадение) по выходам эталонной и экспериментальной реакций и позволяет точно определять дефекты в цифровой системе. При этом используются две стратегии: поиск одиночных и кратных дефектов на каждом тест-векторе, отображаемых векторами $D^s(T_i), D^m(T_i)$ соответственно. Накопление информации, по мере подачи входных последовательностей, происходит уже раздельно в векторах

$$D_0^s(T), D_1^s(T), D_0^m(T), D_1^m(T),$$

которые формируют списки одиночных и кратных, нулевых и единичных константных дефектов. При этом используются системы уравнений 3 и 4 из выражения (7.4).

Система уравнений (7.4) становится существенно более простой, если вместо векторов и таблиц будут фигурировать списки неисправностей:

$$1)\begin{cases} L(T_i) = F \oplus T_i; \\ R^*(T_i) = f(F, T_i, D^*); \\ D(T_i) = g[L(T_i), R^*(T_i)]. \end{cases}$$

$$2)\begin{cases} D^s(T_i) = \bigcap\limits_{\forall j(R_j^*=1)} L_j \setminus \bigcup\limits_{\forall j(R_j^*=0)} L_j; \\ D^m(T_i) = \bigcup\limits_{\forall j(R_j^*=1)} L_j \setminus \bigcup\limits_{\forall j(R_j^*=0)} L_j. \end{cases}$$

$$3)\begin{cases} D^s(T) = D^s(T) \cup D^s(T_i); \\ D^m(T) = D^m(T) \vee D^m(T_i). \end{cases}$$

Однако такое упрощение модели не приводит к меньшей вычислительной сложности решения задачи, наоборот, существенно усложняется алгоритм обработки списков по сравнению с их векторным представлением.

Таким образом, для иерархической модели сложной цифровой системы экономичнее проводить диагностический эксперимент в реальном масштабе времени путем выполнения моделирования модели и реального объекта на каждом входном наборе в целях определения и сравнения их реакций на каждую входную последовательность. В этом случае необходимо хранить информацию только о состоянии выходов и результаты моделирования неисправностей на одном тест-векторе.

Формат данных может быть выбран конкретным разработчиком. Что целесообразнее: уменьшение объема информации при использовании списков неисправностей, но с одновременным усложнением процедур диагностирования и операций над списками, или технологичные алгоритмы на основе регистровых операций при существенном увеличении затрат памяти. Поскольку две упомянутые реализации имеют только формальные отличия, то можно предоставить квалифицированному разработчику два режима, которые он может применять в каждом конкретном случае разработки тестового обеспечения.

Предложенная модель тестирования может быть также использована при ее реализации на дереве контроллеров, управляющих BS-регистрами, которые позволяют сканировать только очередную точку контроля, необходимую для установления точного диагноза.

7.3. Анализ методов тестирования на основе IEEE стандартов

Недостаток существующих распространенных методов проектирования заключается в том, что не всегда поддерживается реализация нефункциональных свойств изделия, каковыми являются избыточные средства тестирования и диагностирования. К тому же избыточность требует определенных временных и материальных затрат, которые окупаются лишь в процессе экс-

плуатации, когда возникают нештатные ситуации, связанные с повреждениями или отказами. Тем не менее основной тенденцией при проектировании сложных цифровых систем является учет требований IEEE стандартов по тестопригодности и надежности. Особенно это касается применений в критических отраслях. Рассмотрим упомянутые стандарты в плане улучшения процесса и результата диагностирования сложных иерархических систем. Критерий эффективности методов тестирования - минимум времени диагностирования с наперед заданным качеством, определяемым глубиной поиска дефекта.

В существующих методах тестирования применяются элементарные проверки, позволяющие уточнить место и вид дефекта в системе. Временные затраты элементарной проверки при выполнении условного диагностического эксперимента определяются выражением

$$T_d = T_s + T_k + T_t + T_r, \tag{7.7}$$

где T_s – время поиска координаты микросхемы, платы, крейта, шкафа для установки (сигнатурного) зонда; T_k – время проверки наличия связи между зондом и выводом для наблюдения рекции; T_t – время подачи теста на объект диагностирования, снятия экспериментальной реакции в точке контроля и сравнения ее с эталонной; T_r – время восстановления работоспособности. Материальные затраты на реализацию элементарной зондовой проверки пропорциональны времени T_d. Естественно, что наиболее весомозначными компонентами являются T_s и T_r. Второй компонент трудно исключить из процедуры диагностирования, поскольку он связан с человеческим фактором. Первый же можно устранить из диагностического эксперимента путем исключения зондирований как элементарных проверок. Для этого следует перейти к безусловному диагностическому эксперименту, исключающему неоднозначность тестирования и установления диагноза путем введения дополнительных линий наблюдения на основе использования существующих IEEE-стандартов.

Какие же линии нуждаются в дополнительном наблюдении? Если коротко, то те, которые позволяют разделить эквивалентные неисправности [61] и привести их к собственным подмножествам на существующем тесте диагностирования. Например, если число наблюдаемых выходов – p, а длина теста – h, то q, как число неисправностей, которое можно обнаружить и различить, определяется формулой

$$q = 2^{p \times h} \ (p \neq 0, h \geq 2). \tag{7.8}$$

Однако эквивалентные неисправности – это хорошо для проектов, насчитывающих миллионы вентилей. Они дают возможность существенно уменьшить длину теста (параметр h), но при этом ухудшить его диагностические свойства. Для компенсации или восстановления упомянутых свойств необ-

ходимо увеличивать параметр p в формуле (7.8). Таким образом, дополнительная наблюдаемость внутренних линий дает возможность существенно повысить глубину диагностирования цифрового изделия и в пределе получить возможность различать каждый дефект в объекте. С другой стороны, уменьшение (увеличение) одного из параметров в (7.8) должно быть скомпенсировано увеличением (уменьшением) второго. В пределе на паре входных наборов можно различить все неисправности, если этому способствует логика цифрового устройства (например, сумматор, имеющий 3 наблюдаемых входа и два выхода). Но при этом в пределе все линии должны быть наблюдаемы.

Для формализации процесса поиска дополнительных линий наблюдения вводится модель, зависящая от следующих параметров:

$$Z = f\,(S, T, C, O, T^*), \tag{7.9}$$

где S – структура объекта; T – тест; C и O – управляемость и наблюдаемость линий; T* – тестируемость изделия.

Общее количество наблюдаемых выходов (априорных и дополнительных) определяется следующим выражением:

$$Y = \{Z, R, y\}, \tag{7.10}$$

где Z – количество дополнительных линий в объекте, подлежащих определению по методу CAMELOT [62]; R – множество сходящихся разветвлений, которые не проверяются базовым тестом; y – априорные выходы схемы.

Дополнительные выходные линии определяются путем вычисления наблюдаемости $O(X_i)$ всех переменных схемы. После этого необходимо выбрать совокупность таких линий X_i, которые удовлетворяют наперед заданному пороговому значению наблюдаемести $O(X_i)$ или тестируемости $T^*(X_i)$ цифровой схемы:

$$X_i \in Y \leftarrow [O(X_i) \le \{O_{max}, T^*_{max}\} \vee X_i \in R]. \tag{7.11}$$

Из множества Y следует исключать все входные линии схемы. Кроме того, при прочих равных характеристиках тестопригодности предпочтение следует отдавать сходящимся разветвлениям R. В целях применения данного выражения необходимо вычислять значения управляемости, наблюдаемости и тестопригодности для каждой линии схемы. Для этого необходимо использовать следующие определения.

1. Управляемость – количественная мера способности устройства генерировать на рассматриваемой линии значение 0 или 1, которая зависит от логической функции устройства и уменьшается по мере отдаления линии от внешних входов схемы:

$$C(Y) = C_i \times C^*(X_i), \tag{7.12}$$

где $C_i = \left|N_i^0 - N_i^1\right| / (N_i^0 + N_i^1)$, – коэффициент передачи управляемости; $N_i^0, (N_i^1)$ – число всех способов установки логического нуля (единицы) на линии; $C^*(X_i) = \sum_{i=1}^{n} C(X_i) / \sum_{i=1}^{n} X_i$ – среднее арифметическое управляемостей входов примитива.

2. Наблюдаемость – количественная мера способности устройства транспортировать состояние рассматриваемой линии на внешние выходы схемы, которая зависит от логической функции устройства и уменьшается по мере отдаления линии от внешних выходов схемы:

$$O(Y) = O_i \times O(X_i) \times C^*(X_i), \tag{7.13}$$

где $O_i = \left|T^a - T^b\right| / (T^a + T^b)$ – коэффициент передачи наблюдаемости; T^a – число наборов, активизирующих X_i на выход примитива, T^b – число наборов, блокирующих активизацию X_i на выход примитива; наблюдаемость внешних выходов схемы $O(X_i) = 1$.

3. Тестируемость – мультипликативная количественная оценка способности устройства транспортировать управляемость (наблюдаемость) рассматриваемой линии на внешние входы (выходы) схемы, которая зависит от логической функции устройства, его структурной сложности и уменьшается по мере отдаления линии от внешних входов (выходов) схемы:

$$T^* = C(Y) \times O(Y). \tag{7.14}$$

Таким образом, представленные выше критерии тестопригодности дают возможность не только оценить проектируемое изделие, но и определить узкие места в проекте с точки зрения управляемости, наблюдаемости и тестируемости. Применение стандартов Boundary Scan IEEE 1149.1 и IEEE P1500 [11-13,47] обеспечивает наблюдаемость (управляемость) линий объекта, благодаря технологии их сканирования, незначительно увеличивающей время проведения диагностического эксперимента.

Что касается иерархического проекта, то его структура может быть представлена в виде мультипликативного дерева, где каждая вершина – компонент по отношению к верхнему уровню иерархии – имеет одну входящую дугу и несколько исходящих (рис. 7.3).

Полный идентификатор компонента системы $F = \left[F_{ij}^{rs}\right]$ имеет четыре индекса. Нижние индексы (ij) служат для обозначения уровня иерархии и порядкового номера компонента в нем. В целях установления связи с его родительской вершиной имеются два верхних индекса – rs, которые задают

предшествующий уровень иерархии и номер элемента в нем. Каждому компоненту сложной цифровой системы ставится в соответствие тест, дефекты и правила установления диагноза:

$$\begin{cases} F = [F_{ij}^{rs}],\ (i = \overline{0,5};\ j = \overline{1, k_j};\ r = \overline{0,4};\ s = \overline{1, k_{j-1}}); \\ T = [T_{ij}^{rs}];\ L = [L_{ij}^{rs}]; \\ F_{ij}^{rs} \oplus T_{ij}^{rs} = L_{ij}^{rs}. \end{cases} \tag{7.15}$$

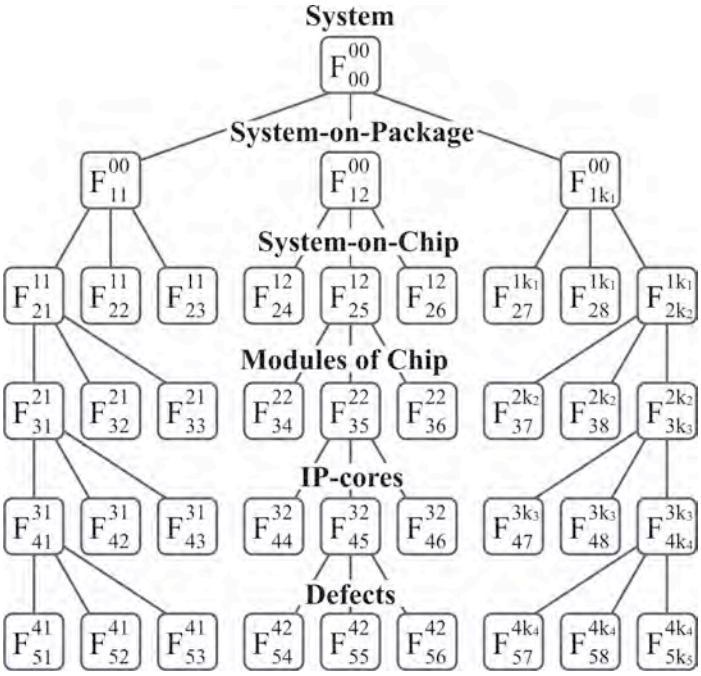


Рис. 7.3. Иерархическая модель сложной цифровой системы

Применяются две стратегии тестирования. Первая использует тест проверки функциональности, который позволяет определить исправность блока (системы) или зафиксировать непрохождение теста. Лишь на последнем уровне иерархии тест дает список неисправностей, присутствующих в системе. Процедура диагностирования в худшем случае предполагает тестирование всех компонентов объекта. Вторая стратегия – более избирательная благодаря использованию сложных и объемных тестов диагностирования. Тестирование текущего блока дает точный адрес компонента, в котором далее следует искать неисправность.

Тест более высокого уровня должен указывать на неисправные компоненты на один уровень ниже:

$$
\begin{aligned}
&F_{1j}^{rs} \oplus T_{1j}^{rs} = L_{1j}^{rs} = \{L_{21}^{1j}, L_{22}^{1j}, \ldots, L_{2k_j}^{1j}\};\\
&F_{2j}^{rs} \oplus T_{2j}^{rs} = L_{2j}^{rs} = \{L_{31}^{2j}, L_{32}^{2j}, \ldots, L_{3k_j}^{2j}\};\\
&F_{3j}^{rs} \oplus T_{3j}^{rs} = L_{3j}^{rs} = \{L_{41}^{3j}, L_{42}^{3j}, \ldots, L_{4k_j}^{3j}\};\\
&F_{4j}^{rs} \oplus T_{4j}^{rs} = L_{4j}^{rs} = \{L_{51}^{4j}, L_{52}^{4j}, \ldots, L_{5k_j}^{4j}\}.
\end{aligned}
\tag{7.16}
$$

Это дает возможность существенно уменьшить длину теста для верхнего компонента дерева, а также определить локальные тесты для вершин более низкого уровня. Плата за такие удобства – доступность каждого модуля иерархической системы по входам и выходам, благодаря технологии граничного сканирования. Кроме того, иерархия позволяет значительно уменьшить количество рассматриваемых (моделируемых и диагностируемых) дефектов за счет разделения целостного объекта на части. Уменьшается также общая длина теста и время его построения. Если учесть, что вычислительная сложность решения задачи синтеза теста есть функция от количества эквивалентных вентилей ($Q = n^2$), то для устройства, содержащего миллион линий, данная оценка будет иметь значение Q = 1012. Если же разбить устройство на 100 компонентов, то вычислительная сложность синтеза тестов будет определяться оценкой $Q = 100 \times (10^4)^2 = 10^{10}$. Аналогично может быть существенно уменьшен объем таблиц неисправностей при иерархическом описании модели объекта. Суммарный объем всех таблиц дефектов компонентов всегда будет значительно меньше, чем наличие одной для всей цифровой системы. Еще одна стратегия подготовки теста для системы связана с рассмотрением моделей нижнего уровня иерархии. Для каждого IP-cores строится тест проверки неисправностей константного типа. Затем, поднимаясь по иерархии вверх, уже построенные тесты объединяются в соответствии с их принадлежностью к чипу, плате, крейту. Следующая структура взаимодействия технологии граничного сканирования и иерархической модели объекта иллюстрирует возможность доступа ко входам и выходам каждого компонента цифровой системы в процессе выполнения процедуры диагностирования (рис. 7.4).

Модель B/S-контроллера является универсальной по отношению к одной родительской вершине и ближайшим ее компонентам-преемникам:

$$F_{ij}^{rs} = \{F_{i+1,1}^{ij}, F_{i+1,2}^{ij}, \ldots, F_{i+1,k_j}^{ij}\}. \tag{7.17}$$

Например, B/S-контроллер $C_{31}^{21} \approx F_{31}^{21} = \{F_{41}^{31}, F_{42}^{31}, F_{43}^{31}\}$ (см. рис. 7.4) осуществляет сканирование данных в пределах одного чипа, содержащего три

функциональных блока – IP-cores: $\{F_{41}^{31}, F_{42}^{31}, F_{43}^{31}\}$. Применив формулу (7.17) к модели системы (см. рис. 7.4), можно получить иерархическую структуру B/S- контроллеров, представленную на рис. 7.5.

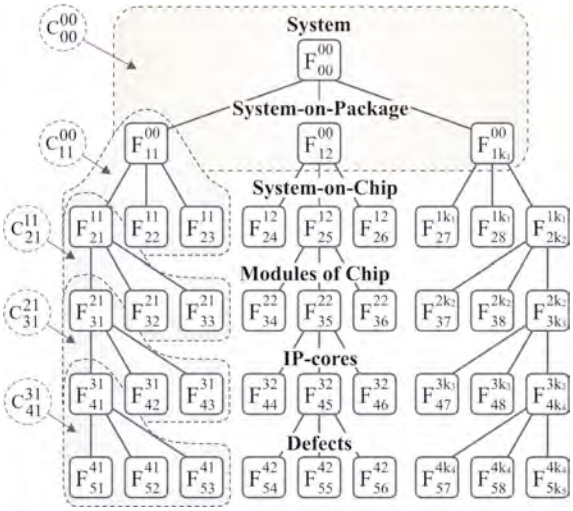


Рис. 7.4. BS-wrapper иерархической модели цифровой системы

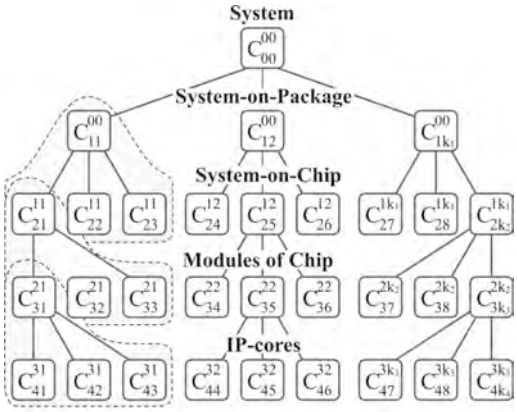


Рис. 7.5. Иерархическая структура B/S-контроллеров сложной системы

Несмотря на кажущуюся сложность такой архитектуры, аппарутурная избыточность B/S-системы тестирования и диагностирования составляет порядка 2-5 % от общей функциональности, что является допустимым для современных chip set таких фирм как Xinix, Altera и привлекательным для заказчиков сложных систем управления критическими технологиями. Что касается ухудшения характеристик быстродействия функциональных модулей, то она составляет не более 3% от расчетных характеристик, без использования тех-

нологий граничного сканирования. Однако существуют определенные временные затраты при выполнении диагностического эксперимента, связанные со сканированием входных и выходных данных конкретного блока. Процесс замедления тестирования зависит от количества наблюдаемых точек (разрядов регистров граничного сканирования) и связан с ним следующим выражением:

$$Q(F) = \sum_{j=1,5} k_j \times p_j \times t^*. \qquad (7.18)$$

Здесь k_j – число регистров граничного сканирования на j-м уровне иерархии; p_j – количество разрядов в регистрах j-го уровня иерархии; t* – время сдвига информации в регистре. При наличии в проекте 1 000 000 эквивалентных вентилей число наблюдаемых линий может достигать 1 000. В данном случае аппаратурная избыточность будет равна 2,1%, скорость тестирования уменьшится в 1 000 раз и составит 1 мс/тест-вектор.

Интерес представляет также аппаратурная избыточность, необходимая для включения B/S регистров на все уровни описания проекта. Для тестирования с использованием технологии граничного сканирования необходимы следующие аппаратурные затраты:

$$\begin{aligned}
&Q^h = Q^{bsr} + Q^{tap} + Q^{in\text{-}out};\\
&Q^{bsr} = q^{bsr} \times (\sum_s \sum_r \sum_j \sum_i q^{core}_{ijrs} + \sum_s \sum_r \sum_j q^{comp}_{jrs} + \sum_s \sum_r q^{chip}_{rs} + \sum_s q^{pack}_s + q^{syst});\\
&Q^{tap} = q^{tap} \times (1 + n^{core} + n^{comp} + n^{chip} + n^{pack} + n^{syst});\\
&Q^{in\text{-}out} = q^{in\text{-}out} \times (1 + n^{core} + n^{comp} + n^{chip} + n^{pack} + n^{syst}).
\end{aligned} \qquad (7.19)$$

Здесь $Q^{bsr}, Q^{tap}, Q^{in\text{-}out}$ – суммарные аппаратные затраты на реализацию B/S регистров для всех уровней иерархии (IP-core, chip, board, crate); затраты на реализацию TAP-контроллеров, выполняющих управление тестированием компонентов на каждом уровне иерархии; дополнительные затраты на входы и выходы TAP-контроллеров, а также на межсоединения между компонентами; параметры: $q^{core}_{ijrs}, q^{comp}_{jrs}, q^{chip}_{rs}, q^{pack}_s, q^{syst}$ обозначают длину B/S регистра в каждом core, comp, chip, pack, syst, параметры: $1 + n^{core} + n^{comp} + n^{chip} + n^{pack} + n^{syst}$ обозначают необходимое количество TAP-контроллеров в каждом компоненте (IP-core, chip, board, crate, system) иерархии; $q^{bsr}, q^{tap}, q^{in\text{-}out}$ – аппаратурные затраты на реализацию одного разряда B/S регистра, TAP-контроллера, входов-выходов блока и межсоединений для одного иерархического компонента системы. Учитывая,

что $Q^{in\text{-}out}$ не имеет вентильного эквивалента, суммарные аппаратурные затраты можно выразить в виде следующего выражения:

$$Q^{h} = Q^{bsr} + Q^{tap} = q^{bsr} \times 20 + q^{tap} \times 500. \qquad (7.20)$$

Здесь коэффициенты 20 и 500 определяют средние затраты по числу вентилей для аппаратурной реализации одного разряда B/S-регистра и одного TAP-контроллера. Для реальных объектов, построенных по иерархическому принципу, аппаратурная избыточность не будет превышать 3 %, что вполне допустимо при создании сложных систем. Разбиение проекта, содержащего 1 000 000 вентилей на 20 IP-cores, дает избыточность в 20 000 вентилей, что составляет 2% экстрааппаратуры.

7.4. Анализ архитектур IEEE BS стандартов

Три параметра: выход годных изделий (yield ratio) при производстве, time-to-market при проектировании, качество и надежность при эксплуатации являются определяющими при оценке эффективности средств тестирования цифровой системы. Естественно, что революционные преобразования в нано- и субмикронных технологиях изготовления кристалла и их использование для создания SoC и NoC создают новые проблемы тестирования и диагностирования сложных цифровых систем и сетей [8,9]. Решение одной из них связано с уменьшением количества дефектных изделий DL в зависимости от качества теста T [10]: $DL = 1 - Y^{(1\text{-}T)}$. При тесте, покрывающем 100% всех неисправностей, выход годных изделий будет максимальным, что означает – на рынке не появятся дефектные изделия. Что касается time-to-market, то и здесь одним из существенных критериев является качество теста, влияющее на время верификации цифрового проекта или тестирования готового изделия. В то же время качество покрытия неисправностей тестом является определяющим для обеспечения надежности работающего изделия, поскольку несвоевременное диагностирование возникшего дефекта может привести к катастрофическим последствиям. Для решения упомянутых проблем ведущие фирмы все более используют мировой опыт тестопригодного проектирования, обобщенный в стандарты общества IEEE [11-14,47]. Основная идея стандартов граничного сканирования – упрощение наблюдения и управления внутренними линиями изделия за счет введения избыточности в виде Boundary Scan регистра и Test Access Port контроллера. Естественно, что первоначально ориентированная на диагностирование межсоединений на платах стратегия граничного сканирования далее получила распространение как вверх по иерархии, до сложных цифровых систем на крейтах, так и вниз – до функциональных модулей микросхемы, обозначенных IP-cores. Архитектуры сканирования информации (входные тесты и реакции линий на них) в процессе тестирования и диагностирования близки по идеологии, но отличаются лишь точкой их приложения – различным объектом тестирования. Все четыре типовых B/S- структуры представлены на рис. 7.6.

Первая из них (см.рис. 7.6, а) ориентирована на решение задач тестирования и диагностики внутри одного чипа, размерностью более 1 000 000 вентилей. Функциональность микросхемы разбивается на отдельные компоненты, для которых строятся тесты и алгоритмы диагностирования. Другое возможное решение не связано с разбиением – наблюдаемыми (управляемыми) становятся критические внутренние линии, которые находятся с помощью известных методов оценки тестопригодности (управляемость, наблюдаемость) [10,62].

Вторая структура (см. рис. 7.6, б) отслеживает стратегию проверки межсоединений на кристалле, а также тестирование и поиск дефектов внутри каждой микросхемы.

Третья схема (см. рис. 7.6, в) ориентирована на своевременное и быстрое тестирование и диагностику кристаллов, объединенных понятием пакета, что очень существенно при управлении критическими технологиями.

Четвертая структура (см. рис. 7.6, г) предназанчена для тестирования системы в целом, объединенной понятием Package on Package.

IEEE 1149.1-2001 Standard Test Access Port and Boundary Scan Architecture [15] определяет архитектуру и функции тестовой логики, которая встраивается в кристалл и предоставляет стандартные средства для: тестирования межсоединений в пакете между кристаллами; проверки самого чипа; наблюдения и управления значениями входов и выходов логики в режиме нормальной работы.

Тестовая логика ориентирована на решение задач диагностирования, содержит регистр граничного сканирования (BSR, Boundary Scan Register) и тестовый порт (TAP, Test Access Port). Схемотехнические решения, определенные в стандарте, позволяют загружать в микросхему инструкции и входные проверяющие воздействия, а также наблюдать результаты выполненных тестовых операций в виде реакций линий, присоединенных к разрядам B/S-регистра. Вся информация (инструкции, тестовые воздействия и реакции) передается в режиме последовательной передачи данных.

Проблема тестирования любой цифровой системы, состоящей из нескольких взаимосвязанных компонентоы, заключается в решении трёх задач: проверка функциональности каждого компонента; все модули имеют корректное соединение; вся система выполняет требуемую функцию. Такой подход можно применить не только для пакетов, составленных из чипов, но и к любым сложным системам, состоящим из множества более простых функциональных модулей. IEEE 1149.4-1999 Standard for a Mixed-Signal Test Bus [12] определяет средства тестирования и протоколы проверки аналогово-цифровых компонентов для решения следующих задач: тестирование межсоединений, обрывов, замыканий, временных параметров, цифровых и аналоговых характеристик элементов на печатной плате.

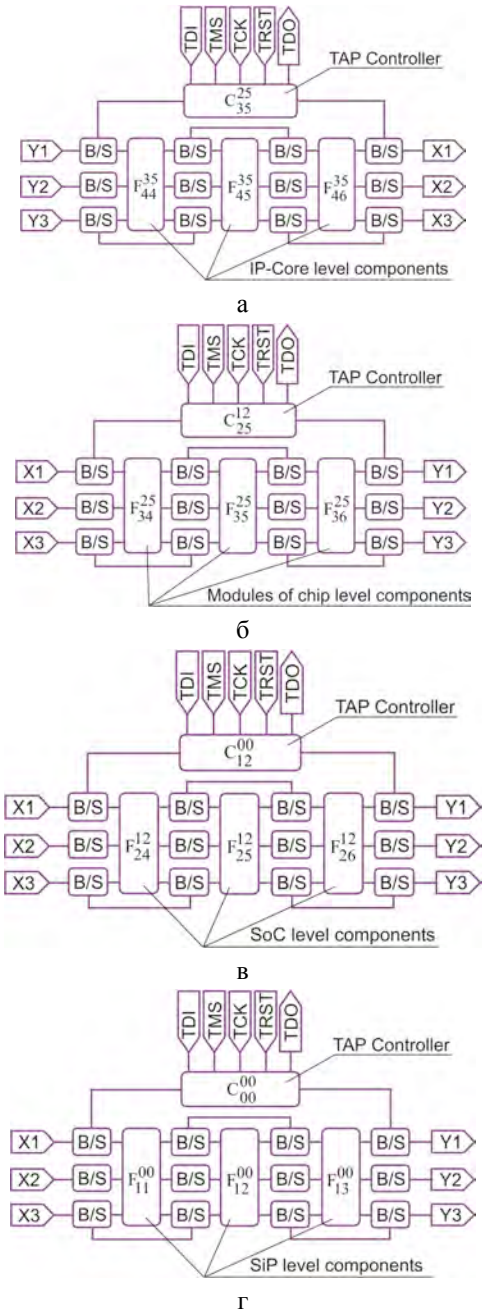


Рис. 7.6. Структурные схемы IEEE стандартов сканирования

IEEE 1149.6-2003 Standard for Boundary-Scan Testing of Advanced Digital Networks [13] расширяет стандарт IEEE Std 1149.1 в целях улучшения возможностей тестирования дифференциальных межсоединений и/или цепей питания переменного тока в интегральных схемах на печатных платах и определяет дополнительные элементы, позволяющие проводить диагностирование с высокой разрешающей способностью.

Основной функцией IEEE 1500 Standard for Embedded Core Test (SECT) [7,47] является стандартизация интерфейса доступа для тестирования внутреннего модуля, входящего в состав цифровой системы на кристалле, и изоляции его от окружающей среды. Он позволяет также тестировать среду, внешнюю по отношению к модулю, путем изоляции последнего и обеспечения его прозрачности.

Основное достоинство тестопригодных стандартов заключается в обеспечении сканирования, управления и наблюдения внутренних переменных цифровой системы на плате и в кристалле, включая переменные функциональных блоков, размещенных на кристалле. Интерфейс стандарта IEEE 1500 SECT представлен на рис. 7.7.

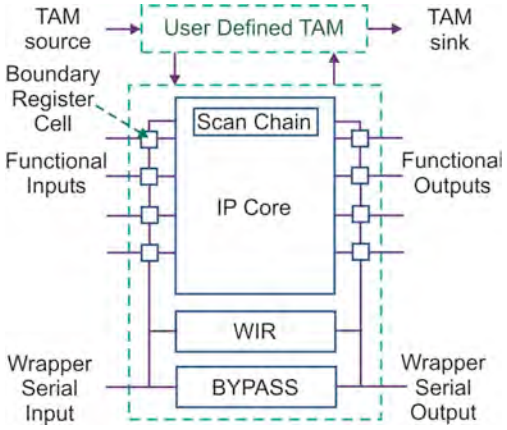


Рис. 7.7. Standard IEEE P1500 Wrapper Components

Каждый из компонентов системы подключается к TAP-контроллеру и тестируется как независимая схема, которая должна иметь отдельный тест, TAP-контроллер, регистр команд, boundary scan регистр и дополнительную логику для декодирования команд. Использование стандарта позволяет проверять функциональность модуля, ошибки межсоединений, определять место, причину и вид дефекта. Основным компонентом Boundary Scan регистра [10-14] является ячейка сканирования, представленная на рис. 7.8. Ее следует рассматривать как аппаратурную избыточность для каждой линии, которая подлежит наблюдению и управлению. Поэтому число таких линий в проекте должно быть ограниченным.

Кроме того, следует учитывать временной фактор тестирования, связанный с использованием Boundary Scan регистра. Чем больше разрядов L(R) он имеет, тем существеннее время, затрачиваемое на тестирование, зависящее от числа входов модуля N(X) и длины теста L(T), определяемой следующей формулой:

$$Q = N(X) \times L(T) + L(R) \times L(T).$$

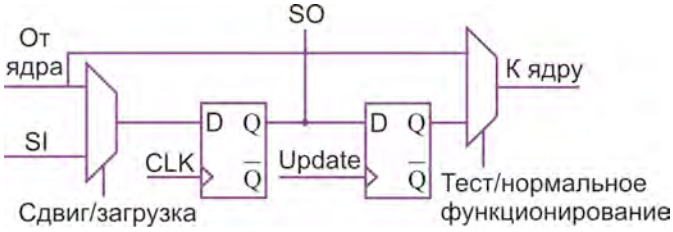


Рис. 7.8. Ячейка регистра граничного сканирования

7.5. Усовершенствование технологий, моделей и методов тестирования сложных цифровых систем

В целях минимизации времени тестирования модулей (IP-cores) в [14] предложено существенное дополнение к IEEE Boundary Scan стандарту. Его суть заключается в следующем. Каждый IP-core имеет собственный встроенный тест самопроверки – Built-in Self Test (BIST), который генерирует псевдослучайные входные последовательности. Это дает возможность тестировать IP-cores паралельно. При использовании детерминированного теста, хранящегося в памяти, вводятся периоды простоя в целях избежания конфликтных ситуаций. Для подачи тестов к модулям используется шина AMBA (Advanced Microcontroller Bus Architecture). Автором предложена методология оптимальной комбинации псевдослучайного и детерминированного тестов для всех IP-cores системы. Стандарты следует рассматривать как базовые средства, которые могут быть успешно дополнены ad hoc технологиями для решения сложных задач синтеза тестов, моделирования неисправностей, верификации и диагностирования некорректностей и дефектов.

Далее предлагаются специализированные технологии тестирования SiP, как сложной цифровой системы, развивающие гибридную методологию [15,49], сочетающую IEEE стандарты и эвристики.

Система тестирования, представленная на рис. 7.9, ориентирована на использование шины в качестве средства доставки проверяющих последовательностей от test-процессора и реакции проверяемого компонента обратно. При такой архитектуре, используя Decoder, можно выполнять тестирование N IP-cores или интегральных схем, а также самого ARM-процессора. Подготовка к диагностическому эксперименту заключается в предварительном создании тестов и получении эталонных реакций для всех модулей цифровой системы. Диагностическое обеспечение (тест, реакции и алгоритмы поиска дефектов) сохраняется в одноименном процессоре. При выполнении проце-

дуры тестирования управление передается от ARM- к test-процессору с прерыванием выполнения основных функций цифровой системы.

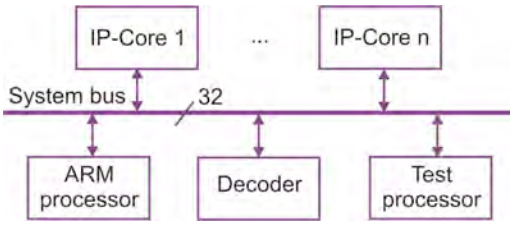


Рис. 7.9. Шинная архитектура тестирования цифровой системы

Преимущества такой системы заключаются в простоте конфигурации, возможности диагностирования с точностью до микросхемы (IP-core) и высоком быстродействии, определяемом тактовой частотой и разрядностью шины передачи данных:

$$\varphi = \frac{k_t \times n_t}{f \times r}, \tag{7.21}$$

где k_t, n_t, f, r – число строк и столбцов (размерность) теста, тактовая частота и разрядность шины передачи данных соответственно.

Недостаток указанной архитектуры заключается в невозможности определить место и вид дефекта внутри микросхемы.

В целях устранения этого недостатка на рис. 7.10 предложена структура, сочетающая преимущества шинной организации обмена данными между IP-cores с высокой глубиной диагностирования, имеющей место в стандарте IEEE 11.49. Платой за качество диагноза является существенное замедление тестового эксперимента, длительность которого определяется формулой:

$$\varphi = \frac{k_t \times n_t}{f \times r \times L_{bsr}}, \tag{7.22}$$

где L_{bsr} – длина boundary scan регистра тестируемого IP-core.

Следует заметить, что такой подход ориентирован на проверку функциональностей компонентов системы и не рассматривает вопросы тестирования соединений между IP-cores, которые здесь отсутствуют.

Следующая архитектура, представленная на рис. 7.11, объединяет преимущества шинной организации, Boundary Scan стандарта и встроенных средств тестирования BIST.

Это дает возможность значительно уменьшить время выполнения тестового эксперимента при высокой глубине диагностирования дефекта в пределах одного компонента системы:

$$\varphi = \frac{\eta(k_t \times n_{t)}}{f \times r \times L_{bsr}}, \tag{7.23}$$

где η – коэффициент, учитывающий часть детерминированного теста в общей длине входных наборов (плюс псевдослучайные векторы) для проверки рассматриваемого IP-core.

Естественно, что для верификации всех компонентов цифровой системы в целях диагностирования необходимо выполнить N таких экспериментов.

Общая структура процесса тестирования сложной иерархической цифровой системы в целях проверки работоспособности и установления места и вида дефекта изображена на рис. 7.12.

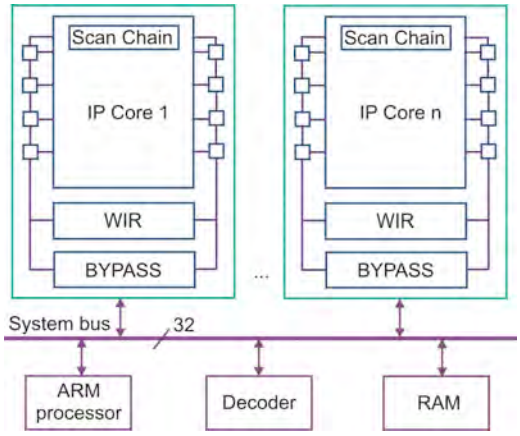


Рис. 7.10. Шинная архитектура и Boundary Scan стандарт

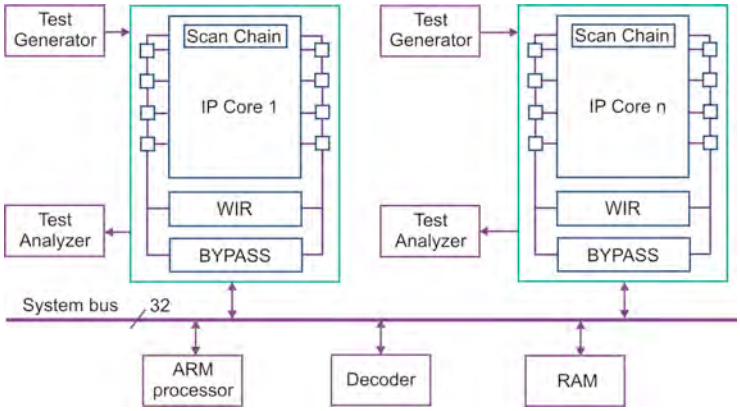


Рис. 7.11. Шинная архитектура, BIST и Boundary Scan стандарт

Принципы организации системы тестирования:

1) Использование наиболее подходящих IEEE стандартов [2, 7-9] для верификации компонентов рассматриваемого уровня.

2) Установление диагноза заданной глубины в автоматическом режиме, без использования условных алгоритмов поиска дефектов.

3) Диагноз выполняется при условии, что только один компонент может быть неисправным.

4) После восстановления работоспособности процедура тестирования начинается всегда с верхнего уровня иерархии к нижнему – нисходящее диагностирование.

5) Процедура тестирования может начинаться с любого уровня иерархии и заканчиваться на уровне, желаемом разработчиком.

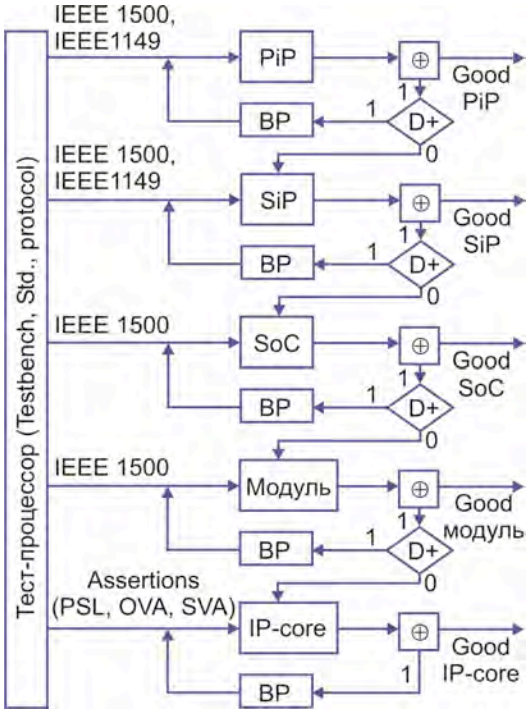


Рис. 7.12. Структура тестирования цифровой системы (BP – восстановление работоспособности; D+ – достаточная глубина диагностирования)

Тест-процессор регламентирует подачу входных последовательностей необходимого формата с заданными свойствами для конкретных компонентов рассматриваемого уровня иерархии $\{C, B, P, M\} \in F$. Каждый компонент имеет собственный стандарт представления теста для проверки и диагностирования дефектов заданного класса. Тестирование поддерживает автоматический режим поиска дефекта с большой глубиной диагностирования, до уровня группы вентилей или фрагмента программного кода. По желанию пользователя процесс может быть остановлен при достижении заданной глубины диагностирования. После восстановления работоспособности процедура диагностирования выполняется с верхнего уровня организации диагностического эксперимента.

7.6. Повышение качества теста на основе технологии IEEE BS

Предлагается технология повышения качества тестовых последовательностей в процессе моделирования неисправностей цифровых систем, основанная на IEEE стандартах тестопригодного проектирования, использующих Boundary Scan регистр, и тестопригодном анализе структуры схемы в целях введения дополнительных точек контроля, вычисленных по технологии CAMELOT.

Актуальность исследования определяется необходимостью повышения быстродействия средств моделирования, улучшения качества теста и уменьшения его размерности для цифровых систем на кристаллах, имеющих миллионы вентилей. Высокие затраты, обусловленные трудоемкостью верификации функционально- и структурно-сложных схем, могут достигать 70% от общего времени разработки проекта. Для снижения таких затрат был разработан стандарт IEEE 1149.1 Boundary Scan (IEEE 1500 SECT), предназначенный для уменьшения времени синтеза тестов, моделирования неисправностей и диагностирования физических дефектов на стадиях производства и эксплуатации цифровых изделий. Однако соединить преимущества упомянутого стандарта с известными методами генерации тестов и анализа их качества – довольно непростая проблема, связанная с оптимизацией функционала f = <T, F, Y>, заданного параметрами: <тест, проверяемые дефекты, выходы схемы>. Частичное решение этой проблемы рассматривается ниже.

Объект тестирования – цифровая система, представленная структурно-функциональной моделью. Цель – получение минимального теста путем введения дополнительных наблюдаемых линий, используемых в технологии Boundary Scan. Задачи: 1) минимальное увеличение количества наблюдаемых линий, обеспечивающего повышение качества фиксированного теста до 100%; 2) минимизация фиксированного теста, имеющего 100% качество, путем увеличения количества наблюдаемых линий и решения задачи покрытия; 3) определение минимального числа дополнительных наблюдаемых линий цифрового устройства на основе метода CAMELOT.

7.6.1. Boundary scan архитектура.

С начала 80-х годов активно внедряются технологии тестопригодного, а в последнее время и производственно-пригодного проектирования цифровых изделий, ориентированного на непрерывность процесса превращения проекта в производимое электронное изделие. Такая тенденция, впервые отмеченная в 2001 году, получила название «Design for Manufacturability – проектирование для производства» [63]. Это означает не только встроенное самотестирование (BIST), но и встроенное самовосстановление (BISR), а также отказоустойчивость и надежность. Концепция WYSIWYG – what you see is what you get – осталась в прошлом. При этом проблема логического проектирования, связанная с отказоустойчивостью и ремонтопригодностью,

переносится в плоскость введения избыточности на благо производства – Manufacturability.

Для решения упомянутой проблемы широко используются стандарты Boundary Scan IEEE 1149.1 [10] и IEEE 1500 SECT в целях упрощения тестирования печатных плат, кристаллов и пакетов, которые имеют также и общее наименование – JTAG, по названию группы его создателей – Joint Test Action Group. В 1994 году в него включено описание языка Boundary-Scan Description Language (BSDL). В настоящее время большинство выпускаемых электронных устройств поддерживает boundary scan технологию тестирования (Xilinx, Atmel, Altera). Методика тестирования с использованием boundary scan архитектуры реализована практически во всех типах электронных чипов: CPU, ASIC, PLD. Плата устройства представлена конструктивными примитивами, каждый из которых окружается boundary scan регистром и имеет TAP-контроллер. Boundary scan технология дает возможность тестировать как межсоединения, так и конструктивные элементы, находящиеся на плате цифрового устройства.

Для расширения концепции Boundary Scan на функциональные примитивы в масштабе одного чипа был разработан стандарт IEEE P1500 [64]. Он имеет две составляющие: техническое обеспечение и язык тестирования ядра (core test language – CTL). Основное достоинство стандарта заключается в том, что он может обеспечить сканирование (наблюдение) не только выходных, но и внутренних переменных цифровой системы как на плате, так и на кристалле (рис. 7.13).

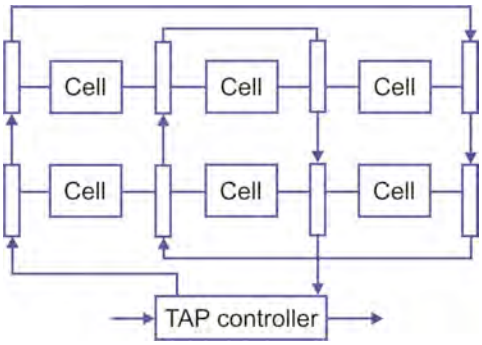


Рис. 7.13. Сканирование компонентов кристалла

Инструкции, поддерживаемые стандартом, обеспечивают режимы сканирования и тестирования компонентов System on Chip (SoC). Каждый из них подключается к TAP-контроллеру и тестируется как независимая схема, которая должна иметь отдельный тест, TAP-контроллер, регистр команд, boundary scan регистр и дополнительную логику для декодирования команд. Использование стандарта позволяет проверять как дефекты, так и функционирование устройства. Недостаток: существенное увеличение времени тестирования SoC.

Для достижения поставленной цели необходимо обеспечить сканирование внутренних линий схемы, которые являются слабоуправляемыми и наблюдаемыми (см. рис. 7.1) с позиции тестируемости [62].

Какие возможности дает технология boundary scan для повышения быстродействия средств синтеза тестов и их анализа в целях оценки их качества относительно одиночных константных неисправностей? Для ответа на данный вопрос необходимо, прежде всего, формализовать процесс тестирования цифрового устройства.

7.6.2. Математическая модель тестирования дефектов.

Проблема получения теста для проверки заданных неисправностей связана, во-первых, с разработкой эффективных генераторов входных последовательностей, во-вторых, с созданием достаточно быстродействующих симуляторов дефектов, в-третьих, с ограничениями на число наблюдаемых выходов цифрового устройства. В общем случае формализация проблемы может быть представлена в виде следующего функционала [67,68]:

$$f = \langle T, F, Y \rangle, \tag{7.24}$$

где $T = (T_1, T_2, \ldots, T_n)$ – тест (его мощность равна n); F – проверяемые дефекты с предельной мощностью 2k (их количество или процентное соотношение проверенных к общему числу линий в схеме k); Y = {y, Z} – выходы схемы y, дополненные наблюдаемыми (сканируемыми) внутренними линиями Z.

На основе введенного функционала можно формализовать решение шести практически значимых задач, представленных в табл. 7.1.

Таблица 7.1

No	T	F	Y	Functions
1	↑	↑	–	$F = f(T, Y = const)$
2	–	↑	↑	$F = f(T = const, Y)$
3	↓	–	↑	$T = f(F = const, Y)$
4	↑	–	↓	$Y = f(T, F = const)$
5	↓	↓	–	$T = f(F, Y = const)$
6	–	↓	↓	$Y = f(T = const, F)$

Краткое описание задач, являющихся производными трехмерного функционала (7.24), может быть представлено в следующем виде:

1. $F = f(T, Y = const)$ – классическая задача синтеза теста, когда повышение его качества достигается путем увеличения его размерности. На рис. 7.14 изображен график функциональной зависимости 1 (в соответствии с нумерацией, приведенной в табл. 7.1) качества теста от его размерности, где P(T) – мощность теста, FC(%) – полнота покрытия (Fault Coverage) одиночных константных неисправностей заданным тестом. Здесь и далее на других гра-

фиках рассматривается пример цифровой схемы, содержащей 432 вентиля и 7 выходов, для которой построен псевдослучайный тест мощностью 2000 векторов с качеством 65%.

-

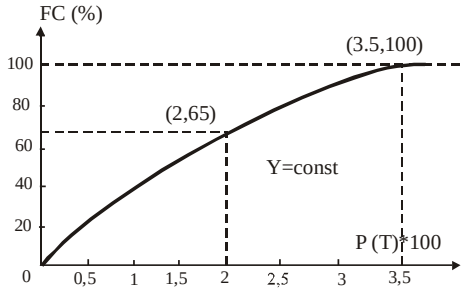


Рис. 7.14. График зависимости 1

2. $F = f(T = const, Y)$ – задача, связанная с тестопригодным проектированием, когда практически невозможно достигнуть требуемого качества теста путем повышения его мощности: в этом случае необходимо вводить дополнительные линии наблюдения (контроля). Их наличие позволяет повысить качество теста до требуемой величины, о чем свидетельствует функциональная зависимость, представленная на рис. 7.15, где дополнительные линии наблюдения генерировались случайным образом. Здесь введение 36 дополнительных выходов позволило повысить качество сгенерированного теста от 65 до 100 %.

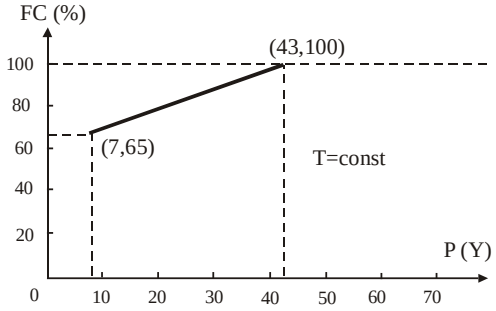


Рис. 7.15. График зависимости 2

3. $T = f(F = const, Y)$ – уменьшение длины теста путем увеличения числа наблюдаемых линий при сохранении достигнутого качества проверяющих последовательностей – используется, когда для проекта более критичной является размерность теста. График зависимости, представленный на рис. 7.16, иллюстрирует функцию предпочтения при фиксированной полноте теста: следует ли уменьшать длину теста за счет увеличения числа наблюдаемых выходов или наоборот. Любая точка, принадлежащая данной кривой, имеет

две координаты (длину теста, число наблюдаемых выходов), которые формируют одинаковое качество теста, в данном случае, равное 100%.

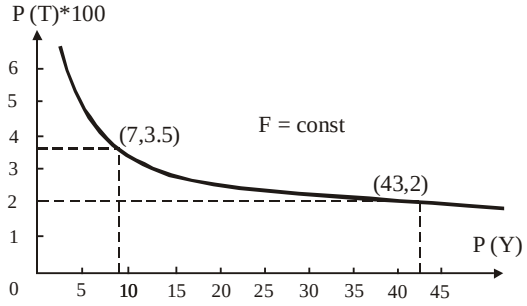


Рис. 7.16. График зависимости 3

4. $Y = f(T, F = const)$ – уменьшение числа наблюдаемых линий (выходов) путем увеличения длины теста при сохранении достигнутого качества проверяющих последовательностей – используется, когда для проекта более критичным является количество выходов или сканируемых линий. График зависимости, представленный на рис. 7.17, иллюстрирует инверсную по отношению к предыдущей функцию предпочтения при фиксированной полноте теста: следует ли уменьшать число наблюдаемых выходов за счет увеличения длины теста или наоборот. Здесь также любая точка, принадлежащая кривой, задает координаты (длину теста и число наблюдаемых линий), соответствующие одинаковому качеству теста.

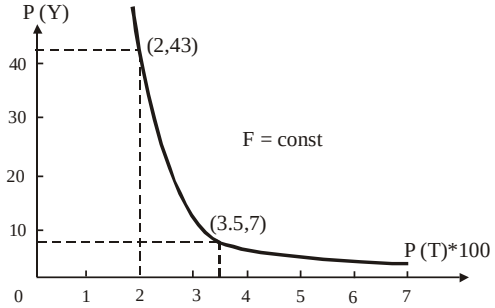


Рис. 7.17. График зависимости 4

5. $T = f(F, Y = const)$ – уменьшение длины теста благодаря более оптимальному решению задачи покрытия всех неисправностей найденным подмножеством тестовых последовательностей, обладающих тем же качеством, что и исходный тест. В соответствии с представленной функциональной зависимостью (рис. 7.18) существует некоторая избыточность теста (люфт) для покрытия заданного множества неисправностей. Иными словами, одна и та же неисправность может быть проверена несколькими тестовыми векторами, что является позитивным фактором для диагностирования дефектов, но из-

быточно для проверки неисправностей. Устранить существующую избыточность можно путем решения задачи покрытия (методами Квайна, Петрика), которая позволяет получить квазиоптимальное решение, не прибегая к полному перебору.

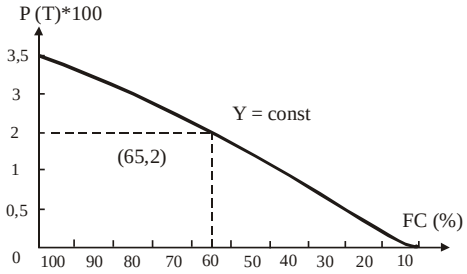


Рис. 7.18. График зависимости 5

6. $Y = f(T = const, F)$ – уменьшение числа наблюдаемых линий (выходов), благодаря более оптимальному решению задачи определения дополнительных выходов или исключению несущественных линий из списка наблюдаемых (сканируемых) при сохранении качества теста. В соответствии с представленной функциональной зависимостью (рис. 7.19) может существовать некоторая избыточность наблюдаемых выходов для проверки заданного множества неисправностей. Иными словами, одна и та же неисправность может быть проверена по нескольким выходам, что является позитивным фактором для диагностирования дефектов, но избыточно для их проверки. Устранить существующую избыточность выходных линий можно также путем решения задачи покрытия, когда каждой неисправности, проверяемой на тест-векторе, ставится в соответствие множество выходов, на которых она проверяется. Затем находится минимальное количество выходов, необходимых для проверки заданных дефектов, что и будет решением данной задачи.

Первые две задачи ориентированы на достижение требуемого качества теста (например, 100%) в режиме дополнения векторов или линий наблюдения; остальные четыре – на квази оптимизацию функционала (7.24) или минимизацию параметров длины теста и числа наблюдаемых линий при сохранении достигнутого качества. В этом случае уместно говорить о двух фазах стратегии синтеза теста: 1) режим дополнения (расширения) теста и линий наблюдения; 2) режим минимизации теста и линий наблюдения. Основные шаги стратегии и выбора режимов представлены на рис. 7.20.

После моделирования неисправностей, проверяемых сгенерированным тестом, оценивается его качество покрытия дефектов. Если FC не равно 100%, то работает фаза 1, содержащая 2 способа повышения FC: 1) псевдослучайно или по методу CAMELOT [62] определяются дополнительные линии наблюдения или выходы; 2) выполняется генерирование дополнительных тестовых последовательностей. Если FC равно 100%, включается фаза 2, содержащая следующие альтернативы: 1) минимизация теста путем решения задачи по-

крытия; 2) минимизация количества наблюдаемых выходов, достаточного для проверки всех неисправностей, формирующих качество теста.

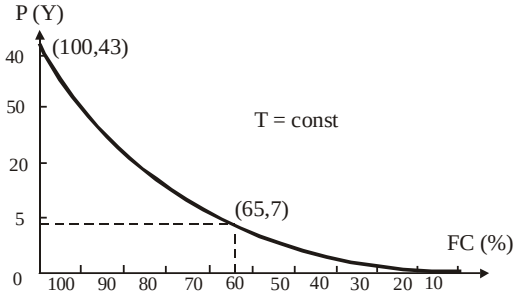


Рис. 7.19. График зависимости 6

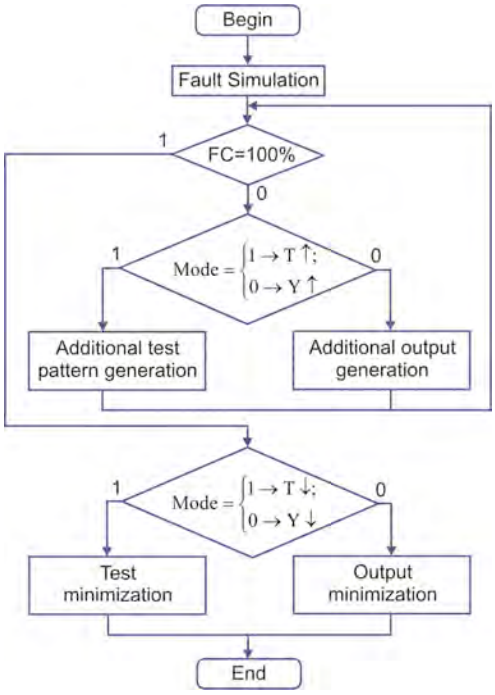


Рис.7.20. Две фазы стратегии синтеза теста

7.6.3. Структурный анализ схемы.

В соответствии с ранее сформулированной задачей необходимо определить достаточное и по возможности минимальное количество дополнительных линий наблюдения в целях повышения качества уже сгенерированного теста до наперед заданной величины. Определять их случайным образом не есть эффективное решение задачи. Существуют уже разработанные правила и методики определения линий, таких как сходящиеся разветвления, имеющие

низкую оценку тестируемости (управляемости и наблюдаемости). Поэтому необходимо воспользоваться уже накопленным исследователями опытом.

Для формализации процесса поиска дополнительных линий наблюдения используется модель тестопригодности (управляемость, наблюдаемость, тестируемость), описанная формулами (7.9) – (7.14).

Пример 7.1. Для схемы, представленной на рис. 7.21, построить тест проверки одиночных неисправностей с использованием структурного анализа тестопригодности для определения дополнительных выходов, преобразующий базовый неполный тест в полный и имеющий 100% качество.

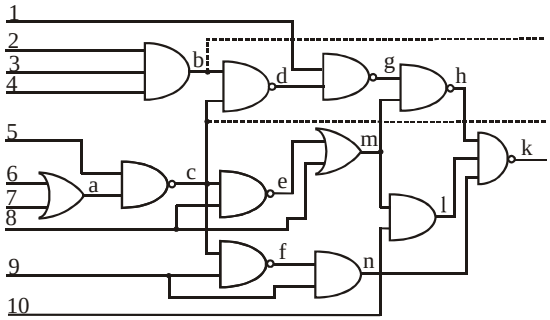


Рис. 7.21. Схема для структурного анализа

В соответствии с правилами (7.12) – (7.14) можно вычислить контролируемость, наблюдаемость и тестируемость для а-линии:

$$C(a) = C_i \cdot ((C(6) + C(7)/2) = 0{,}5 \cdot 1 = 0{,}5;$$

$$O(a) = O_i \cdot O(c) \cdot C(5) = 0{,}5 \cdot 0{,}0033 = 0{,}0017;$$

$$T(a) = O(a) \cdot C(a) = 0{,}5 \cdot 0{,}0017 = 0{,}00085.$$

Аналогично вычисляются показатели тестопригодности для всех остальных линий схемы (табл. 7.2).

Для наглядности изменения значений тестопригодности по мере удаления линии от внешних входов и выходов ниже представлены диаграммы (рис. 7.22), по которым можно визуально определить дополнительные выходы схемы в целях повышения качества базового теста.

Из диаграммы видно, что при T_{max} = 0,0033, только три внутренние линии (a, b, c) по величине наблюдаемости (0,0017; 0,0013; 0,0033) удовлетворяют выражению (7.14). Среди них линия “c” является к тому же сходящимся разветвлением. Их дополнение к множеству выходных линий позволяет сделать базовый неполный псевдослучайный тест, содержащий 56 векторов (85%), полным и имеющим 100% качество. Однако после применения фазы минимизации количества наблюдаемых линий (см. рис. 7.20) из множества Y исключается линия “a”. Таким образом, окончательное решение – множество дополнительных наблюдаемых линий, превращающих тест в полный, есть Y= {b,c}.

Таблица 7.2

№	$C(X_i)$	$O(X_i)$	$T^*(X_i)$
1	1	0,001	0,001
2	1	0,0005	0,0005
3	1	0,0005	0,0005
4	1	0,0005	0,0005
5	1	0,0012	0,0012
6	1	0,0008	0,0008
7	1	0,0008	0,0008
8	1	0,0023	0,0023
9	1	0,107	0,107
10	1	0,105	0,105
a	0,5	0,0017	0,00085
b	0,75	0,0013	0,0009
c	0,375	0,0033	0,0012
d	0,281	0,007	0,0019
e	0,344	0,006	0,0021
f	0,344	0,0134	0,0046
g	0,320	0,014	0,0045
h	0,164	0,084	0,0137
l	0,334	0,0625	0,0209
m	0,336	0,0134	0,0045
n	0,336	0,0622	0,139
k	0,139	1	1

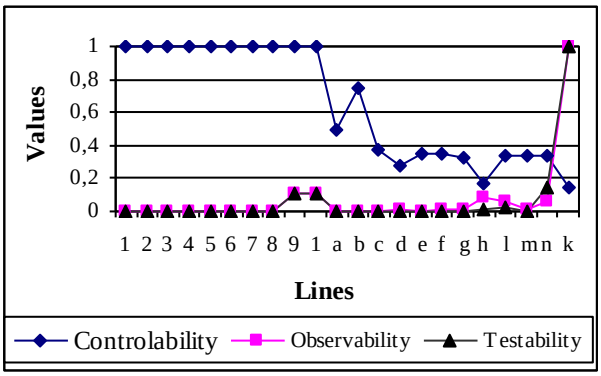


Рис. 7.22. Диаграмма тестопригодности схемы

7.7. Выводы

Проанализированы международные стандарты общества IEEE, пригодные для тестирования и диагностирования программно-аппаратных компонентов электронных цифровых систем широкого профиля: начиная от функциональных модулей внутри кристалла и заканчивая сложными системами, состоящими из совокупности систем в пакетах.

Рассмотренные стандарты IEEE Std 1149.1, 1149.4, 1149.6 и 1500 и технические предложения SJTAG предоставляют широкие возможности для решения задач: тестирование функциональности компонента; тестирование соединений между кристаллами и внутри него; тестирование системы, состоящей из совокупности кристаллов и/или пакетов. Объектами тестирования являются: аналоговые и цифровые модули; узлы смешанного типа; дискретные элементы; память, состоящие из перечисленных выше компонентов.

Наиболее существенные результаты, описанные в разделе:

1) Аналитическая и структурная модели диагностирования SiP как сложной цифровой системы, которые ориентированы на выполнение тестового эксперимента в реальном масштабе времени с использованием средств моделирования неисправностей, позволяющих установить точный диагноз за несколько секунд.

2) Блочно-иерархический метод тестирования и диагностирования программно-аппаратных комплексов на основе их представления в виде иерархии относительно простых блоков, для которой разработана сеть контроллеров граничного сканирования, дающая возможность управлять процессом тестирования сложных объектов в реальном масштабе времени и существенно (на 50-70%) уменьшать время проектирования и верификации цифровых изделий.

3) Сформулирована математическая модель тестирования дефектов, позволяющая соединить решение шести наиболее важных задач технической диагностики. На основе упомянутой модели разработана стратегия синтеза тестов и моделирования неисправностей при использовании дополнительных линий наблюдения, включающая две фазы: расширения и уменьшения мощностей тестов и наблюдаемых выходов.

4) Предложена процедура поиска и выбора дополнительных линий наблюдения на основе методологии CAMELOT, позволяющей априори получать информацию о тестируемости, управляемости и наблюдаемости линий цифровой схемы.

5) Стандарты Boundary Scan IEEE 1149.1 и IEEE P1500 дают возможность увеличивать количество наблюдаемых выходов схемы, благодаря технологии сканирования состояний внутренних линий, незначительно повышающей время проведения тестирования. Упомянутая технология предоставляет

возможность использовать внутренние линии как внешние выходы устройства для существенного (в десятки раз) повышения быстродействия моделирования неисправностей.

6) Сложность получения полного проверяющего теста для цифровых систем на кристаллах может не удовлетворять пользователя по параметру времени генерации тестов. Введение дополнительных линий наблюдения может обеспечить 100% полноту базового теста в рамках отведенного для генерации времени. Значительная мощность теста для сложных цифровых изделий может быть неприемлема для пользователя. Ее можно существенно уменьшить путем введения дополнительных наблюдаемых линий с последующим решением задачи покрытия, более минимальным, тестом всех ранее проверенных дефектов.

7) Проверка предложенных стратегий синтеза тестов и моделирования неисправностей на основе введения дополнительных наблюдаемых линий осуществлена на 10 схемах из библиотеки ISCAS'2000 с помощью средств синтеза тестов и моделирования неисправностей SIGETEST [65]. Результаты: 1) При средней полноте теста 80% необходимо найти и ввести 2-5% дополнительных линий наблюдения, чтобы тест трансформировался в полный. 2) В процессе применения фазы минимизации количества (тест-векторов) наблюдаемых линий мощность множества (T) Y может быть уменьшена на (20-30%) 10-20% соответственно.

8) Фирма Xilinx предложила разработчикам программное средство ChipScope Pro [66], которое позволяет встраивать ядро логического анализатора в проект цифровой системы. Оно дает возможность пользователю наблюдать сигналы на всех внутренних линиях кристалла FPGA. Учитывая такое новшество фирмы Xilinx, разработки, представленные выше, имеют большую практическую значимость, ориентированную на рынок электронных технологий. В самом деле, ядро логического анализатора дает возможность наблюдать все линии. Но такой Core будет иметь соизмеримые с проектом размеры. Следовательно, для их уменьшения будут весьма полезными методы поиска необходимого и достаточного числа наблюдаемых внутренних линий.

Таким образом, технологии граничного сканирования позволяют разработчикам быть на уровне современных требований и тенденций при проектировании, верификации, отладке, изготовлении, тестировании, внутрисхемном программировании и эксплуатации различных систем. Соответствие конечного программного и/или аппаратного продукта международным стандартам IEEE способствует его успешному продвижению на мировом рынке.

Научная новизна полученных результатов заключается в разработке специализированных ad-hoc технологий, моделей и методов тестирования, объединяющих существующие IEEE стандарты со специальными схемотехническими решениями, позволяющими в несколько раз уменьшить время

диагностирования сложных SiP систем управления критическими технологиями [51-60].

Практическая значимость раздела заключается в существенном (на 20-50%) уменьшении времени поиска дефектов в сложной цифровой системе на основе использования предложенной иерархической модели процесса диагностирования и специальных технологий граничного сканирования IEEE стандартов проектирования [56,58-60]. При этом глубина диагностирования цифрового изделия повышается до 30% при увеличении числа наблюдаемых линий на 3%. Улучшаются также и другие характеристики, такие как: качество теста, тестопригодность, время моделирования и генерации теста.

7.8. Литература

1. Ваде Гриби, Елисеев В.В., Хассан Ктейман, Хаханов В.И., Хаханова И.В. Анализ неисправностей при проектировании дискретных логических сетей // Бионика интеллекта. – 2004. – № 64/1.– С. 61- 67.

2. Хаханов В.И., Елисеев В.В., Обризан В.И., Ваде Гриби, Хассан Ктиаман. Ассерт-метод верификации цифровых систем на основе стандарта IEEE 1500 SECT // АСУ и приборы автоматики. – 2005. – Вып. 132. – С. 93-105.

3. Grant Martin. The Reuse of Complex Architectures. IEEE Design and Test of Computers. November-December 2002. – P. 4-6.

4. Rajesh K. Gupta and Yervant Zorian. Introducing Core-Based System Design. IEEE Design & Test of Computers. November-December 1997. – P. 15-25.

5. Yervant Zorian. What is Infrustructure IP IEEE Design & Test of Computers. May-June 2002. – P. 5-7.

6. Benini L. and Micheli G.D. Networks on chips: A new soc paradigm. IEEE Computer. – 2002. – Vol. 35, No. 1. – P. 70–78.

7. IEEE P1500/D11. January 2005. Draft Standard Testability Method for Embedded Core-based Integrated Circuits. New York. – 2005. – 138 p.

8. Yervant Zorian. Advances in Infrustructure IP. IEEE Design & Test of Computers. – May-June 2003. – P. 49-56.

9. Yervant Zorian. Test Requirements for Embedded Core-Based Systems and IEEE P1500. In Proceedings IEEE International Test Conference (ITC). Washington, DC. November 1997. IEEE Computer Society Press. – P. 191-199.

10. Abramovici M., Breuer M.A. and Friedman A.D. Digital systems testing and testable design. Computer Science Press. – 1998. – 652 p.

11. IEEE Std 1149.1-2001. Standard Test Access Port and Boundary-Scan Architecture. New York. – 2001. – 208 p.

12. IEEE Std 1149.4-1999. IEEE Standard for a Mixed-Signal Test Bus. New York., 2000. – 84 p.

13. IEEE Std 1149.6-2003. Standard for Boundary-Scan Testing of Advanced Digital Networks. New York. – 2003. – 139 p.

14. Gert Jervan, Petru Eles, Zebo Peng, Raimund Ubar, Maksim Jenihhin. Test Time Minimization for Hybrid BIST of Core-Based Systems. Proceedings of the 12th Asian Test Symposium (ATS'03). – P.318-323.

15. Papachristou C.A., Martin F., and Nourani M. Microprocessor based testing for core-based system on chip. In Proceedings of the 36th ACM/IEEE conference on Design automation conference // ACM Press. – 1999. – 586-591 p.

16. Park S.,T.Kim. A new IEEE 1149.1 boundary scan design for the detection of delay defects // Proc. DATE'2000. – P. 458-462.

17. Nadeau-Dostie B., Cote J., Hulvershorn H., Pateras S. An Embedded Technique for At-Speed Interconnect Testing // Proc. ITC'1999. – P.431-438.

18. Shin J., Kim H., Kang S. At-Speed Boundary-Scan Interconnect Testing in a Board with Multiple System Clocks // Proc. of Design, Automation and Test in Europe (DATE '99) Conf. March 09-12. – 1999. – Munich. Germany. – P. 473-477.

19. Pendurkar R., Chatterjee A., Zorian Y. Switching Activity Generation with Automated BIST Synthesis for Performance Testing of Interconnects. IEEE TransCAD / ICS. – Vol. 20, No.9. – 2001. – 6p.

20. Attarha A., Nourani M. Testing Interconnects for Noise and Skew in Gigahertz SoC // Proc. of ITC'2001. – P. 305-314.

21. Jutman A. Shift Register Based TPG for At-Speed Interconnect BIST. Proc. of 24th International Conference on Microelectronics (MIEL'04) // Serbia and Montenegro. – 2004. – P. 751-754.

22. Jutman A. At-Speed On-Chip Diagnosis of Board-Level Interconnect Faults // Proc. of 9th European Test Symposium (ETS'04). – France. – 2004. – P. 2-7.

23. The International Technology Roadmap for Semiconductors. 2003 Update: Test and Test Equipment. URL: http: //public.itrs.net/

24. Hassan A.J., Rajski J., Agrawal V.K. Testing and Diagnosis of Interconnects. Proc. Int'l Test Conf. – 1988. – P. 126-137.

25. C. Su, S.W.Jeng, Y.T. Chen. Boundary scan BIST methodology for reconfigurable systems. Proc. ITC'98. – P. 774-783.

26. W. Feng, F.J. Meyer, F. Lombardi. Novel control pattern generators for interconnect testing. Proc. Int'l Symp. Defect and Fault Tolerance in VLSI Systems. – 1999. – P. 112-120.

27. L. Jin. The driver/receiver conflict problem in interconnect testing with boundary-scan. Proc of Asian Test Symposium (ATS'93). Beijing. – 1993. – P.210-214.

28. A. Hassan, V. Agarwal, B. Nadeau-Dostie, and J. Rajski. BIST of PCB interconnects using boundaryscan architecture. IEEE Trans.Computer-Aided Design. – Vol. 11. – P. 1278–1287.

29. Ulf Pillkahn. Structural test in a board self test environment. Proc. IEEE Int. Test Conf. (ITC'2000). Atlantic City. NJ. USA. – 2000. – P. 1005-1012.

30. W.H. Kautz. Testing of Faults in Wiring Interconnects. IEEE Trans. Computers. – 1974. – Vol. 23, No. 4. – P. 358-363.

31. P.T. Wagner. Interconnect Testing with Boundary Scan. Proc. Int'l Test Conf. (ITC 87). IEEE Press. – 1987. – P. 52-57.

32. E.J. Marinissen et al. Towards a Standard for Embedded Core Test: An Example. Proc. IEEE Int'l Test Conf. (ITC 99). IEEE Press. Piscataway. NJ. – 1999. – P. 616-627.

33. Stephen Pateras. IP for Embedded Diagnosis. IEEE Design and Test of Computers. – 2002. – P. 46-56.

34. Ing-Jer Huamg, Chung-Fu Kao, Hsin-Ming Chen, Ching-Nan Juan, and Tai-An Lu. A Retargetable Embedded In-Circuit Emulation Module for Microprocessors. IEEE Design and Test of Computers. – 2002. – P. 28-49.

35. Grant Martin. The Reuse of Complex Architectures. IEEE Design and Test of Computers. – 2002. – P. 4-6.

36. Wolfgang Raab, Nico Bruels, Ulrich Hachmann, Jens Harnisch, Ulrich Ramacher, Christian Sauer and Axel Techmer. A 100-GOPS Programmable Processor for Vehicle Vision Systems. IEEE Design and Test of Computers. – 2003. – P. 8-18.

37. Peter Petrov and Alex Orailoglu. Application-Specific Instruction Memory Customizations for Power-Efficient Embedded Processors. IEEE Design and Test of Computers. – 2003. – P. 18-25.

38. Tiehan Lv, I. Burak Ozer, Srimat T. Chakradhar, Jiang Xu and Wayne Wolf, Jorg Henkel. S Methodology for Architectural Design of Multimedia Multiprocessor SoCs. IEEE Design and Test of Computers. – 2005. – P. 18-28.

39. Bergeron Janick. Writing testbenches: functional verification of HDL models. Boston: Kluwer Academic Publishers. – 2001. – 354 p.

40. R. Chandramouli and Stephen Pateras. Testing Systems on a Chip. IEEE Spectrum. – 1996. – P. 42-47.

41. S.G. Hemmady, T.L. Anderson, Y. Zorian. Verification and Testing of Embedded Cores. Proc. Design SuperCon. On-Chip Design. – 1997. – P 119-122.

42. Samvel Shoukourian. A Unified methodology for Offline and Online Testing. IEEE Design & Test of Computers. – 1998. – P. 73-79.

43. Yervant Zorian, Erik Jan Marinissen, and Sujit Dey. Testing Embedded Core-Based System Chips. In Proceedings IEEE International Test Conference (ITC). Washington. 1998. IEEE Computer Society Press. – P. 130-143.

44. Eric Jan Marinissen and Yervant Zorian. Challenges in Testing Core-Based System ICs. IEEE Communications Magazine. – 1999. – P. 37(6): 104-109.

45. S.K. Goel and E.J. Marinissen. A Novel Test Time Reduction Algorithm for Test Architecture Design for Core-Based System Chips. In Proceedings of IEEE European Test Workshop ETW'02. – 2002. – P. 7-15.

46. Yervant Zorian. Advances in Infrustructure IP. IEEE Design & Test of Computers. – 2003. – P. 49.

47. Erik Jan Marinissen, Rohit Kapur, Maurice Lousberg, Teresa McLaurin, Mike Ricchetti, Yervant Zorian. On IEEE P1500's Standard for Embedded Core Test. Journal of Electronic Testing: Theory and Applications (JETTA). Kluwer Academic Publishers. – Volume 18. – 2002. – 12p.

48. Erik Jan Marinissen, Sandeep Kumar Goel, and Maurice Lousberg. Wrapper Design for Embedded Core Test // Proc. IEEE International Test Conference (ITC). – Atlantic City, NJ. – 2000. IEEE Computer Society Press. – P. 911-920.

49. Guru Rao C.V., D. Roy Chowdhury. Testing of Embedded Core Based Systems Using Linear Cellular Automata // Proc. of Cellular Automata Symposium 2001. – No 01-63. – Yokagama National University. Japan. – 2001. – P. 108-113.

50. Harry Bleeker, Peter van den Eijnden, and Frans de Jong. Boundary-Scan Test – A Practical Approach. – Kluwer Academic Publishers. – Dordrecht. The Netherlands. – 1993. – 264 p.

51. Хаханов В.И., Елисеев В.В., Парфентий А.Н. EDA-market и технологии проектирования SOC. Материалы международной конференции "Единое информационное пространство. Днепропетровск. – 2004. – С. 201-204.

52. Alexandr Yegorov, Irina Hahanova, Olga Melnikova, Vladimir Yeliseyev. Eda-Market and Soc Design Verification Technologies. Proceedings of the 8th International Conference CADSM 2005. Lviv, Ukraine. – 2005. – P. 352-355.

53. Hahanova I., Obrizan V., Ghribi W., Yeliseev V., Ktiaman H., Guz O. Hierarchical Hybrid Approach to Complex Digital Systems Testing // Proc. of the 3rd East-West Design and Test Workshop. – Odessa. Ukraine. – September 15-19, 2005. – P. 254-256.

54. Kaminska M.O., Kulak E.N., Guz O.A., Yeliseev V.V. Probabilistic testability measure before pseudorandom test generation. Proceedings of the 12th International Conference "Mixed Design of Integrated Circuits and Systems". Gdynia. – P. 591-594.

55. Hahanov V., Obrizan V., Yeliseev V., Ghribi V. Hierachical testing of complex digital systems. Proceedings of the International Conference TCSET 2006.- Lviv, Ukraine. – 2006. – P. 426-429.

56. Хаханов В.И., Елисеев В.В. Применение IEEE стандартов для тестирования программно-технических комплексов // Радиоэлектронные и компъютерные системы. – 2006. – № 6(18). – С. 163-171.

57. Vladimir Hahanov, Vladimir Yeliseev, Anna Hahanova, Dmytro Melnik. Hierarchical Systems Testing based on Boundary Scan Technologies // Proc. of the 4th East-West Design and Test Workshop. – Sochi. Russia. – September 15-19, 2006. – P. 53-58.

58. Хаханов В.И., Елисеев В.В., Обризан В.И., Ваде Гриби, Хасан Ктиаман. Иерархическое тестирование программно-технических комплексов // АСУ и приборы автоматики. – 2006. – Вып. 134. – С. 93-102.

59. Елисеев В.В., Хаханов В.И., Парфентий А.Н., Хаханова А.В. Модели диагностирования иерархических систем на основе технологий граничного сканирования // АСУ и приборы автоматики. – 2007. – Вып. 138. – С. 86-97.

60. Hahanov V., Yeliseev V., Krasovskaya A., Melnik D. Testing model of complex digital systems based on boundary scan technology // Proc. of the 4th East-West Design and Test Workshop. – Sochi. Russia. – September 15-19, 2006. – P. 254-256.

61. Автоматизация диагностирования электронных устройств / Ю.В. Малышенко и др./ Под ред.В.П.Чипулиса. – М.: Энергоатомиздат, 1986. – 216с.

62. Беннеттс Р. Дж. Проектирование тестопригодных логических схем. – М.: Радио и связь, 1990. – 175 с.

63. Yervant Zorian. Dmytris Gizopoulos. Gest editors' introduction: Design for Yield and reliability// IEEE Design & Test of Computers. – May-June, 2004. – P. 177-182.

64. Teresa McLaurin, Souvik Ghosh. ETM10 Incorporates Hardware Segment of IEEE P1500 // IEEE Design & Test of Computers. – 2002. – P. 8-13.

65. Hahanov V., Hyduke S. Topological BDP faultsimulation method. Proceeding of EuroMicro Symposium. France, Rennes: INRIA. – 2004. – P. 356-359.

66. Horgan Jack. FPGA Direction. – EDA Cafee. – August 2 - 6, 2004.

67. Хаханов В.И., Каминская М.А., Егоров А.А., Побеженко И.А. Повышение качества теста на основе технологии Boundary Scan // Радиоэлектроника и информатика.– Харьков.– 2004.– №3.– С. 85-90.

68. Kaminska M.A., Melnikova O.V., Sami Ulan Khan, W. Ghribi Improving test quality by applying Boundary Scan technology // Proceedings of the 2nd East-West Design and Test Workshop 2004.– Ukraine.– 2004.– P.263-269.

8. ПОИСК ДЕФЕКТОВ НА ОСНОВЕ АНАЛИЗА ТАБЛИЦ НЕИСПРАВНОСТЕЙ

Таблицы неисправностей являются универсальным инструментом для диагностирования дефектов любой природы. В контексте рассмотрения цифровых изделий важно иерархически дифференцировать типы дефектов: одиночные или кратные неисправности – для логических модулей и памяти, макродефекты – для компонентов, входящих в состав SoC или SiP. Здесь критерием адекватности модели дефекта может служить факт его фиксации, который конструктивно влияет на последующее восстановление работоспособности компонента или системы в целом. Что касается систем на кристаллах, то реализация встроенных тестовых методов диагностирования предполагает наличие инфраструктуры сервисного обслуживания, минимально включающего: интерфейс связи с объектом диагностирования, средства подачи тестов и снятия реакции изделия, логические стимуляторы и анализаторы для поиска дефектов внутри цифрового изделия. В условиях высокой стоимости отбраковки компонентов вычислительной техники и выполнения процедур диагностирования далее предлагаются методы структурного условного поиска дефектов, которые достаточно просто реализовать для решения практических задач в целях минимизации временных и материальных затрат восстановления работоспособности компонентов цифровых изделий на кристаллах.

8.1. Двузначные таблицы неисправностей

Классические таблицы неисправностей (ТН) [1] заданы на множестве входных последовательностей, для каждой из которых определен вектор проверяемых дефектов на совокупности нулевых и единичных ОКН, координата которого отмечена единицей, если соответствующий данному столбцу дефект проверяется. В этом случае объем ТН определяется выражением: $W = 2 \times n \times k$, где n – количество эквипотенциальных линий; k – число тестовых наборов. Практическое использование таких ТН для поиска дефектов ограничивается размерностью таблиц и неточностью определения кратных дефектов. Для их диагностирования существуют достаточно простые для понимания методы исключения и приближения.

Метод исключения использует идею устранения из списка подозреваемых такую ОКН, которая не изменяет состояния наблюдаемых выходов на хотя бы одном тестовом наборе, на котором она проверяется. Иллюстрацией сказанному может служить ТН – табл. 8.1, которая соответствует схеме, изображенной на рис.8.1.

Чтобы использовать ТН для определения технического состояния объекта, необходимо получить вектор экспериментальной проверки (ВЭП) V, длина которого равна числу k входных наборов, где $V_i = 1$, если реакция хотя бы одного наблюдаемого выхода ОД на i-м тестовом векторе отлична от эта-

лонной, и $V_i = 0$ – в противном случае. (Нулевые координаты таблицы представлены точками).

Таблица 8.1

	Stack-at-Faults – Одиночные неисправности линий																				
12345	0										1										V
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	
00100	.	.	.	.	.	.	.	.	.	.	1	.	.	.	1	.	1	.	1	1	1
10110	1	.	1	.	.	1	1	.	.	1	.	.	.	.	.	.	.	.	.	.	0
11010	1	1	.	.	.	1	1	.	.	1	.	.	.	.	.	.	.	.	.	.	0
10001	.	.	.	.	.	.	.	.	.	.	.	1	1	1	.	1	1	1	1	1	0
00011	.	.	.	1	1	.	.	1	1	1	.	.	.	.	.	.	.	.	.	.	1
01101	.	.	.	.	1	1	.	1	1	1	.	.	.	.	.	.	.	.	.	.	0
10000	.	.	.	.	.	.	.	.	.	.	.	1	1	.	.	1	1	.	1	1	0
00001	.	.	.	.	.	.	.	.	.	.	.	1	1	1	.	1	1	1	1	1	0

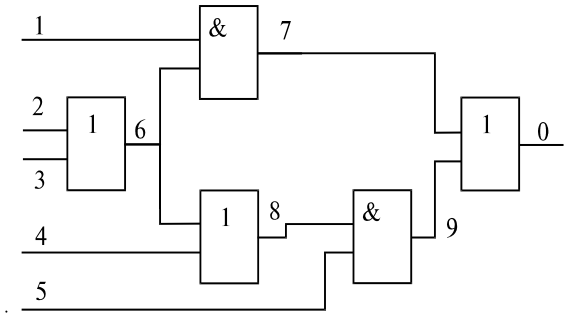


Рис.8.1. Пример цифровой схемы

Определение ОКН в схеме связано со сравнением ВЭП с каждым столбцом ТН, когда при их совпадении делается вывод о наличии в схеме неисправности, соответствующей данному столбцу.

Если же вектор V не имеет аналогов в столбцах ТН, можно предположить наличие в схеме кратных дефектов. В этом случае упомянутый выше метод исключения должен устранить из полного списка подозреваемых дефектов (СПД) S_d такие неисправности, которые проверяются на наборе i, но при этом имеют $V_i = 0$:

$S=S_d\backslash\{\{10, 30, 60, 70, 00\}; \{10, 20, 60, 70, 00\}; \{21, 31, 41, 61, 71, 81, 91, 01\}; \{50, 60, 80, 90, 00\}; \{21, 31, 61, 71, 91, 01\}; \{21, 31, 41, 61, 71, 81, 91, 01\}\}$.

Учитывая, что S_d есть все двадцать ОКН, после теоретико-множественного вычитания получаем список: S={40, 11, 51}. Данный результат следует рассматривать как наличие в схеме хотя бы одной фактически присутствующей неисправности из S.

Метод приближения использует идею максимального сходства вектора V с одним или несколькими столбцами ТН по минимальному кодовому расстоянию в понимании Хэмминга. Для упомянутого примера при заданном кодовом расстоянии, равном 1, между V и столбцами ТН получаем список подозреваемых неисправностей S={11, 40, 51}, каждая из которых отличается от вектора V только в одном разряде.

Таким образом, для данной схемы оба метода дали одинаковые результаты при фактическом кратном дефекте {30, 51}. Однако для схем с произвольной структурой рассмотренные методы могут не выявить даже одного представителя кратной неисправности. Для двухуровневой схемы в список S войдет хотя бы одна фактическая неисправность. Для цифровой схемы без разветвлений при кратности дефектов не более трех в список S войдет хотя бы одна неисправность.

8.2. Анализ многозначных таблиц неисправностей

Для уменьшения размерности ТН вводится многозначный алфавит описания ее координат: A={0,1,X,U}. В результате этого определяется число столбцов таблицы неисправностей, равное количеству существенных переменных. Тогда объем ТН будет $W = n \times k$, а ее координата $T_{ij} = a$, если тест-вектор i проверяет константу L_a на линии L, где a={0,1}. Пример многозначной таблицы неисправностей (МТН) для схемы (см. рис. 8.1) изображен на рис. 8.2.

12345	1	2	3	4	5	6	7	8	9	0	12
00100	1	.	.	.	1	.	1	.	1	1	10
10110	0	.	0	.	.	0	0	.	.	0	00
11010	0	0	.	.	.	0	0	.	.	0	00
10001	.	1	1	1	.	1	1	1	1	1	01
00011	.	.	.	0	0	.	.	0	0	0	10
01101	.	.	.	.	0	0	.	0	0	0	00
10000	.	1	1	.	.	1	1	.	1	1	00
00001	.	1	1	1	.	1	1	1	1	1	01

Рис.8.2. Многозначная таблица неисправностей

Клеточки МТН, отмеченные точкой, соответствуют пустому множеству (U) проверяемых на линиях дефектов. Очевидно, в построенной таблице неисправностей нет места символам X, поскольку в этом случае тест-вектор должен проверять одновременно константу нуля и единицы, что невозможно для комбинационной схемы.

Диагностический эксперимент на основе использования МТН дифференцируется в два независимых подхода, определяемых наличием в объекте одиночного или кратного константного дефекта.

В первом случае вектор подозреваемых неисправностей (ВПН) одиночного константного типа вычисляется по формуле

$$D^s = \bigcap_{V_1} T_i \setminus \bigcup_{V_0} T_i, \qquad (8.1)$$

где $D^s = (D_1, D_2, ..., D_j, ..., D_n)$ – дефекты одиночного константного типа при $D_j = \{0,1\}$; $\{V_1, V_0\} = V$ – подмножества нулевых и единичных координат вектора экспериментальной проверки; $\bigcap T_i = D^1$ – обозначение пересечения между собой всех строк МТН, которые отмечены единичным значением ВЭП; $\bigcup T_i = D^0$ – объединение строк МТН, имеющих нулевое значение V.

Список подозреваемых дефектов формируется теоретико-множественным вычитанием из пересечения всех неисправностей, которые могут дать отрицательный результат при подаче тестовых наборов, объединения таких ОКН, которые теоретически по результатам проверки не искажают состояния выходов схемы, определяемые как вектор отсутствующих неисправностей (ВОН). Для ВЭП V_1 выполнение процедур пересечения и объединения [2,12] дает следующий результат:

$D^1 = \varnothing$; $D^0 = (0XX10XXXXX)$, а разность $D^s = D^1 \setminus D^0 = \varnothing$, что свидетельствует о невозможности получения реакции, обозначенной вектором V_1, в предположении наличия в схеме ОКН.

Для вектора V_2 выполнение процедуры (8.1) дает результат

$$D^1 = (.111/11111) \setminus D^0(0XX00XX0XX) = D^s(...1...1..).$$

Вывод: в схеме присутствует одна из неисправностей {41, 81}.

При подходе, предполагающем наличие кратного константного дефекта, вычислительная формула имеет вид

$$D^M = \bigcup_{V_1} T_i \setminus \bigcup_{V_0} T_i, \text{ где } \bigcup_{V_0} T_i = D^2. \qquad (8.2)$$

Иначе необходимо определить объединения векторов дефектов, соответствующих единичным и нулевым координатам ВЭП, и выполнить между ними операцию теоретико-множественного вычитания. Для вектора V1 по (8.2) вычислим кратные константные дефекты эквипотенциальных линий:

$$D^2(1..0X.1.XX) \setminus D^0(0XX10XXXXX) = D^M(1..1.....).$$

Вектор V_1 может быть получен при наличии в схеме подмножества дефектов {11, 40, 51}. Какие неисправности и в каком сочетании фактически присутствуют в схеме? Это может быть хотя бы один дефект из списка для

двухуровневой схемы; для цифрового устройства без разветвлений при кратности не более трех ОКН также присутствует одна неисправность из полученного множества [3].

Для вектора V_2 вычисление по (8.2) дает результат:

$$D^2(.111.11111)/D^0(0XX10XXXXX) = D^M(1..1.....).$$

Присутствие одновременно дефектов {41, 81} может дать результат проверки, соответствующий вектору V_2.

Объединение обеих процедур диагностирования одиночных и кратных константных дефектов определяет следующий, один из возможных вариантов алгоритма тестирования объекта на основе использования МТН.

1. Выполнение элементарной проверки и формирование вектора V по всем координатам: $V_j = 1$, если на наблюдаемых выходах имеет место несовпадение эталонной и экспериментальной реакций схемы на тестовом наборе j, $V_j = 0$ – в противном случае. Если все $V_j = 0$ – делается вывод об исправности изделия, после чего выполняется п. 9.

2. Вычисление результатов пересечения D^1 и объединений: D^0, D^2.

3. Получение вектора подозреваемых дефектов D^S по (8.1).

4. Если $D^S = \varnothing$, перейти к вычислению списка D^M, в противном случае следует выполнить п. 9.

5. Определение списка D^M на основе анализа формулы (8.2).

6. Анализ вектора D^M в целях формирования взаимно противоречивых групп дефектов.

7. Использование зондовых методов диагностирования в целях уточнения фактических неисправностей.

8. Устранение дефектов и переход к выполнению п. 1.

9. Окончание процедуры диагностирования.

При наличии в векторе подозреваемых дефектов символов X выполняется п. 6. Это связано с невозможностью одновременного существования в реальной схеме констант 0 и 1 на одной линии. Поэтому выполняется разбиение D^M на подмножества, число которых равно 2n (n – число символов X). Например, вектор 01XX10 будет представлен наборами: 010010, 010110, 011010, 011110.

8.3. Структурный анализ многозначных таблиц неисправностей

Проблема безусловного или незондового диагностирования технического состояния цифрового изделия заключается в минимизации области суще-

ствования неисправностей наперед заданного класса, которая называется множеством эквивалентных дефектов |D|, при допустимых параметрах длины теста |T| и числа наблюдаемых выходов |Z|. Функционально зависимость упомянутых компонентов определяется выражением $|D|=1/(|T|*|Z|)$. Естественно, чем больше длина теста и количество наблюдаемых выходов, тем выше разрешающая способность алгоритма диагностирования, которая в пределе должна обеспечивать определение неисправного компонента только по фактическим реакциям наблюдаемых выходов без зондирования линий.

В зависимости от стадии жизненного цикла цифрового изделия существуют две стратегии проектирования алгоритма поиска дефектов. Для создаваемого объекта затраты на создание диагностического обеспечения (ДО), включающего тесты, алгоритмы поиска дефектов могут быть уменьшены благодаря упрощению решения задачи генерации полного теста за счет применения дополнительных линий наблюдения, выводимых на внешний разъем. Если процедура создания ДО ориентирована на уже спроектированное изделие, то следует максимально использовать только имеющиеся внешние наблюдаемые линии и увеличивать длину теста для получения требуемой глубины диагностирования дефектов. При фиксации параметров |T| и |Z|, что имеет место в алгоритмах поиска дефектов, основанных на анализе уже заданных таблиц неисправностей, можно увеличить глубину диагностирования благодаря использованию структуры функционально-гальванических связей вход-выходных переменных элементов (модулей, кристаллов) цифрового изделия. Решение задачи в упомянутой постановке и предлагаемой ниже, является развитием метода, описанного в [3].

Максимально адекватной реакцией цифрового объекта на тест-вектор T_i (i=1,k) при наличии дефекта является вектор-строка значений выходных наблюдаемых линий изделия Z_i. Учитывая, что длина упомянутого вектора, равная n_y, может быть значительной (сотни линий), а количество даже одиночных константных неисправностей (ОКН) (число дефектов линий произвольной кратности задается выражением $\sum_{i=1}^{n}(C_n^i \times 2^i)$, C_n^i – число сочетаний) определяется как удвоенное число эквипотенциальных линий (2n), то объем таблицы функций неисправностей (ТФН) [4], представляющей исходную информацию для безусловного поиска дефектов, вычисляется по формуле $Q_{ТФН} = k \times 2n \times n_y$. Одним из вариантов уменьшения размерности ТФН является построение таблицы неисправностей (ТН), которая имеет меньший объем диагностической информации $Q_{ТН} = k \times 2n$, благодаря снижению адекватности результатов моделирования ОКН. Учитывая, что реакция объекта на тест-вектор при наличии неисправности сводится здесь уже к одному биту вместо n_y в ТФН, то относительно каждого дефекта имеется инфор-

мация типа «обнаружен (1) – не обнаружен (0)». Естественно, что глубина безусловного диагностирования неисправностей при использовании ТН значительно уменьшается, что отрицательно сказывается на стоимости процедуры восстановления работоспособности цифрового изделия, поскольку для уточнения места неисправности необходимо использовать дорогостоящие процедуры зондового диагностирования вычислительных устройств.

В качестве компромиссного варианта предлагается метод тестового диагностирования одиночных и кратных константных дефектов в цифровых устройствах на основе использования многозначных таблиц неисправностей (МТН) [2,12], отличающихся от ТН вдвое меньшим числом столбцов без потери свойств диагностирования. Для повышения глубины поиска неисправностей используется анализ структуры объекта, представленной в виде множества вектор-списков (XY) матрицы смежностей графа функциональных гальванических связей, который в совокупности с МТН меньше по объему ($Q = k * n + |XY|$) классической таблицы функций неисправностей [3], а его (графа) построение не составляет труда.

Исходная информация представлена многозначной таблицей неисправностей (МТН) $D = \| D_{ij} \|$, определенной на множестве входных тестовых последовательностей $T = (T_1, T_2, ..., T_i, ..., T_k)$, для каждой из которых задана вектор-строка проверяемых одиночных константных неисправностей (ОКН) $D_i = (D_{i1}, D_{i2}, ..., D_{ij}, ..., D_{in})$. Координата D_{ij} определена на алфавите A={0,1,X,$\varnothing$}. $D_{ij} = 0(1)$, если тест-вектор T_i проверяет константу 0(1) на линии j. Символ X означает проверку константной неисправности нуля и единицы, что присуще только схемам с памятью, когда проверяющие способности текущего вектора объединяются с неисправностями, проверяемыми на предыдущих входных наборах. Значение $D_{ij} = \varnothing$ определяет отсутствие проверяемых на линии j неисправностей тестом T_i.

Матрица экспериментальной проверки (МЭП) $V = \| V_{ir} \|$ принимает значения из множества {0,1}; $r = n'+1, n$; n' – число невыходных линий; $n_y = n - n' + 1$ – длина вектора упомянутых линий; $i = \overline{1, k}$; k – длина теста. Координата $V_{ir} = 1$, если $T_{ir} \neq Z_{ir} \in Z$, $Z = \| Z_{ir} \|$ – матрица ожидаемых двоичных состояний наблюдаемых линий; $V_{ir} = 0$, если $T_{ir} = Z_{ir}$. Учитывая специфику анализа структуры объекта, связанную с последовательным моделированием подграфов примитивов, в качестве формы описания структуры цифрового устройства используются списки векторов вход-выходных линий примитивов (X_t, Y_t), при условии, что $\forall j (Y_{tj} = f(X_t))$, где $t = \overline{1, p}$; p – число функциональных ПЭ в цифровом устройстве.

Матрица действительных дефектов (МДД) $M = \| M_{ij} \|, M_{ij} = \{0,1\}$, где $M_{ij} = 1$, если дефект D_{ij} не противоречит строке V_i матрицы экспериментальной проверки, по формату соответствует структуре $\| D_{ij} \|$ и является смоделированной (вычисляемой) формой задания графа эквипотенциальных линий схемы.

Иначе, при $M_{ij} = 1$ состояние линии j на тесте T_i подозревается в наличии одиночной константной неисправности, которая транспортируется до наблюдаемых выходов и формирует реакцию, не противоречивую по отношению к строке V_i. Если $M_{ij} = 0$, то дефект (если он имеется в МТН), определяемый координатой D_{ij}, не способен создать значения компонентов вектора V_i, а поэтому он должен быть исключен из списка подозреваемых неисправностей.

В результате диагностического эксперимента по наблюдаемым выходам формируется МЭП $V = \| V_{ij} \|$. Для каждой строки $V_i \in V$ вычисляется подграф (строка M_i) эквипотенциальных линий цифровой схемы, неисправности которых могут сформировать двоичный вектор технического состояния наблюдаемых выходов V_i. Построение таблицы M для заданной матрицы экспериментальной проверки V с использованием графа объекта основано на следующем алгоритме структурного моделирования:

1. Состояния наблюдаемых линий определяются в соответствии с выражением $\forall i, r(M_{ir} = V_{ir})$ и не могут быть модифицированы с помощью двух следующих ниже правил, определяемых пунктами 2 и 3.

2. Входные линии примитива подозреваются в существовании на них неисправностей, если имеется выходная дефектная линия: $" \forall j(M_{ij}(X_t) = 1)$ при $\exists j(M_{ij}(Y_t) = 1)$.

3. При различных технических состояниях (исправное и неисправное) входов одного потенциала разных примитивов состояние линии подозревается в наличии дефекта $M_{ij} = M_{ij}(X_t) \vee M_{ij}(X_f)$.

С учетом приведенных определений процедура вычисления состояний линий схемы – координат матрицы M – включает два пункта:

1) Определение значений внешних наблюдаемых выходов как результатов сравнения экспериментальной и ожидаемой реакций объекта на тест

$$M_{ir} = Z_{ir} \oplus T_{ir} = V_{ir}.$$

2) Последовательная обработка примитивов схемы методом обратного прослеживания от внешних выходов схемы к входам в целях определения состояний всех координат M_{ij} на основе трех упомянутых выше правил.

После вычисления матрицы действительных дефектов M для полученной V выполняется процедура поиска одиночных D' или кратных D" дефектов в соответствии с выражениями:

$$D'_{j|j=1,k} = \Big(\bigcap_{\forall i(Mij=1)} Dij\Big) \setminus \Big(\bigcup_{\forall i(Mij=0)} Dij\Big); \qquad (8.3)$$

$$D''_{j|j=1,k} = \Big(\bigcup_{\forall i(Mij=1)} Dij\Big) \setminus \Big(\bigcup_{\forall i(Mij=0)} Dij\Big). \qquad (8.4)$$

Вектор одиночных константных дефектов D' определяется векторной разностью между пересечением координат столбцов матрицы D_j, отмеченных единичными значениями соответствующих M_{ij}, и объединением всех остальных координат по столбцам матрицы D. Для выполнения разности между двумя координатами $D_{rj} \setminus D$ существуют правила:

$$0 \setminus 0 = \varnothing,\ 0 \setminus 1 = 0,\ 0 \setminus X = \varnothing,\ 1 \setminus 0 = 1,\ 1 \setminus 1 = \varnothing,\ X \setminus 0 = 1,\ X \setminus 1 = 0,\ X \setminus X = \varnothing\ .$$

Получение вектора кратных неисправностей D" задается разностью объединения координат столбцов матрицы D_j, отмеченных единичными значениями соответствующих M_{ij}, и объединением всех остальных координат по столбцам матрицы D.

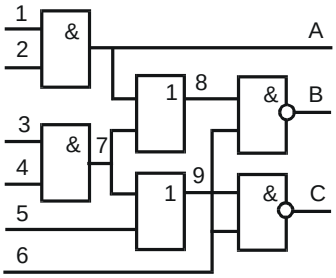
Пример практического применения указанных формул для схемы представлен на рис. 8.3, где темным фоном в таблице показаны координаты, определяющие $M_{ij} = 1$, светлым – $M_{ij} = 0$.

Среди пяти подозреваемых неисправностей, заданных в результирующей последней строке, имеются фактические дефекты $\{1^0, 3^1, 6^1\}$, которые присутствуют в схеме. В общем случае в векторе кратных дефектов содержится хотя бы одна одиночная составляющая из кратного дефекта, фактически присутствующего в схеме [3]. Безусловные методы [4] анализа дефектов в лучшем случае для данной схемы определяют множество, содержащее 9 неисправностей.

При условии наличия в схеме одиночного константного дефекта обработка табличных данных по формуле (8.1) дает неисправности: $\{1^0, 2^0, A^0\}$. Для уточнения полученного результата необходимо выполнить моделирование каждого из дефектов полученного списка в целях определения фактических искажений выходов.

Если при реализации упомянутой процедуры для анализируемого дефекта получается совпадение матрицы экспериментальной проверки $\| V_{ij} \|$ с матрицей моделируемой проверки $\| V^{D_i} \|$ при внесении каждой неисправности

$\| V_{ij} \| = \| V^{D_i}{}_{ij} \|$, то делается вывод о фактическом существовании в объекте дефекта D_i. Для рассматриваемого примера каждый из дефектов списка $\{1^0, 2^0, A^0\}$ не выдерживает проверки при моделировании, поскольку ни один из них не способен исказить выходы на тест-векторах (1, 2, 5), поэтому окончательный список одиночных дефектов есть пустое множество.



Тест	Неисправности линий												МП		
	1	2	3	4	5	6	7	8	9	A	B	C	A	B	C
T1=111101	0	0	0	0	.	0	0	0	0	0	1	1	1	0	0
T2=010101	1	.	1	.	1	.	1	1	1	1	0	0	0	1	1
T3=101001	.	1	.	1	1	.	1	1	1	1	0	0	0	0	0
T4=000011	.	.	.	.	0	0	1	1	0	1	0	1	0	0	0
T5=111110	0	0	.	.	.	1	.	.	.	0	0	0	1	1	1
D"(1)=	0	0	1	.	1	1	1	1	1	0	0	0			
D"(0)=	1	1	0	X	X	0	X	X	X	1	X	X			
D"=	0	0	1	.	.	1	.	.	.	0	.	.			

Рис. 8.3. Трехвходовая схема и таблица неисправностей

На основе использования системы векторов M, матрицы экспериментальных проверок V и в соответствии формулами (8.3), (8.4) вычисления D' и D" можно оформить в структурный алгоритм анализа МТН для поиска дефектов.

1. Подача теста на входы устройства и снятие реакций с выходных наблюдаемых линий – определение матрицы Z.

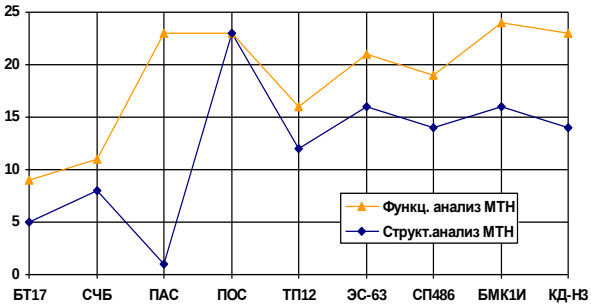
2. Вычисление матрицы экспериментальной проверки $V_{ir} = T_{ir} \oplus Z_{ir}$.

3. Формирование матрицы M путем анализа графовой структуры с применением упомянутых выше трех правил моделирования.

4. Вычисление списка D' или D" соответственно подозреваемых одиночных или кратных константных неисправностей по (8.3), (8.4).

5. Уточнение списка полученных одиночных неисправностей путем их анализа в целях получения матрицы моделируемой проверки для каждого дефекта с последующим сравнением ее с МЭП.

Алгоритм анализа МТН с применением моделируемой структуры объекта в среднем на 30% уменьшает количество подозреваемых дефектов (рис. 8.4) при диагностировании одиночных и кратных константных неисправностей, по сравнению с диагностированием по формулам (8.1), (8.2). При этом используются значительно меньшие объемы информации, простые и технологичные машинно-ориентированные процедуры обработки таблиц, что позволяет включить его в ряд встроенных технологий тестирования и поиска дефектов на этапах эксплуатации и производства цифровых объектов. Однако для работы алгоритма необходима довольно трудоемкая процедура получения МТН с помощью программных или аппаратных средств моделирования дефектов введенного класса.



Ì î äóëü	ÀÓ17	Ñ×Á	Ì ÀÑ	Ì Ì Ñ	ÓÌ 12	Ý Ñ-63	ÑÌ 486	ÀÌ Ê 1Ê
Ôóí êö. àí àëèç Ì ÒÍ	9	11	23	23	16	21	19	24
Ñòðóêò.àí àëèç Ì ÒÍ	5	8	1	23	12	16	14	16

Рис. 8.4. Анализ функционального и структурного диагностирования

Таким образом, выше были предложены универсальные алгоритмы безусловного диагностирования одиночных и кратных константных дефектов по фактическим реакциям выходов цифрового модуля в процессе тестового диагностирования на основе выполнения процедуры объединения пересечений. Для повышения глубины безусловного диагностирования цифровых модулей предложена модель анализа многозначных таблиц неисправностей на основе использования графа структурно-функциональных связей, который позволяет уменьшить область подозреваемых неисправных линий и элементов до 30 %, благодаря дифференцированному подходу к анализу проявления дефектов на наблюдаемых выходах цифровой структуры.

8.4. Проектирование условных алгоритмов поиска дефектов

8.4.1. Сигнатурный алгоритм обратного прослеживания.

Алгоритм диагностирования не дифференцируется строго на процедуры контроля и поиска дефектов, но описывается уравнением диагноза:

$$F*(F,T,D)|_{F,T,g*} = \cup_i (g(T,F) \cap g*(T,F,D_i)) = \varnothing,$$

где g, g*– эталонная и экспериментальная реакции наблюдаемых линий ЦУ; F, T, D – функционал, проверяющий тест, наперед заданные дефекты.

Использование сигнатурного анализа [5,6] определяет элементарную проверку $P_i = g \times (T,F,D_i)$ как получение сжатой реакции при подаче проверяющего теста T на выбранном наблюдаемом контакте i схемы. Проверка P_i положительна, если в точке контроля i сигнатура эталонная, вычисленная при отсутствии неисправностей, равна экспериментальной: $S_{эт} = S_{экс}$. Иначе $S_{эт} \neq S_{экс}$ проверка P_i будет отрицательной. В общем случае идентификация проверки определяется пересечениями:

$$Pi = \begin{cases} P_i^+ \leftarrow [g*(T,F,Di) \cap g(T,F) = g(T,F)]; \\ P_i^- \leftarrow [g*(T,F,Di) \cap g(T,F) = \varnothing]. \end{cases}$$

Точкой контроля i является внешний контакт примитива, чипа, цифрового изделия. Два логических исхода проверки задают альтернативные пути поиска дефекта или определения технического состояния объекта:

$$S = \begin{cases} D_0, D^+\} \rightarrow P_i = P_i^+; \\ D^- \rightarrow P_i = P_i^-, \end{cases}$$

где $S = D_0 \cup D$; $D^+ \cup D^- = D$; $D^+ \cap D^- = \varnothing$, D_0– исправное техническое состояние; D – предполагаемая область существования дефектов; $D^+ (D^-)$– область неисправностей, определяемая проверкой $P^+ (P^-)$.

Поэтому алгоритм диагностирования методом обратного прослеживания является условным и зондовым, который использует сигнатурный логический анализатор. Длительность и стоимость выполнения условного диагностического эксперимента на порядок выше, чем безусловного. В отличие от метода половинного деления [4,5] он ориентирован на поиск не только логических дефектов, но и на обнаружение разрывов линий. Поэтому минимизация числа контрольных точек в предлагаемом далее алгоритме поиска дефектов не используется. А стратегия обратного прослеживания неисправности по структуре объекта, связана с введенным классом макро-дефектов, которые распознаются при последовательном выполнении проверок линий функционально-логического пути, что ориентирует предлагаемый алгоритм для восстановления работоспособности цифровых модулей на стадии их производства при использовании в качестве средства диагностирования «ложе из гвоздей» или автоматический зонд [7].

Диагностическая модель устройства представлена графом функционально-гальванических связей выходных контактов кристалла и цифрового устройства в целом:

$$G = \{L_{ij} \in L, F_{tr}^{k} \in F\},$$

где L_{ij} – (дуга) гальваническое отношение одного потенциала между контактами различных конструктивов или примитивов; F_{tr} – (дуга или дуги F_{tr}^{k} k-го ПЭ) функциональное отношение между входной t и выходной r переменной. Для построения алгоритма диагностирования необходимы: полный проверяющий тест относительно одиночных константных неисправностей T, эталонные сигнатуры $S_{эт}$ всех контактов кристалла, граф функционально-гальванических связей F^{G} цифрового модуля:

$$W = \{G, T, S_{эт}\}.$$

В целях приближения моделей неисправностей к реальным изделиям вводятся следующие типы состояний объекта:

$$D = \{D_0, D_1, D_2, D_3, D_4, D_5\},$$

где D_0 – исправное состояние объекта определяется совпадением $S_{эт} = S_{экс}$ по всем выходным контактам устройства:

$$D_0 \leftarrow (g(T,F) \cap g^*(T,F,D_i) = g(T,F));$$

D_1 – отсутствие гальванической связи между контактами различных компонентов цифровой системы:

$$D_1 \leftarrow [S_{эт}(L_i) = S_{экс}(L_i) \,\&\, S_{эт}(L_j) \neq S_{экс}(L_j)];$$

D_2 – неисправность на выходе k-компонента или внутри него:

$$D_2 \leftarrow \{\forall t[S_{эт}(F_t^k) = S_{экс}(F_t^k)] \,\&\, \exists r[S_{эт}(F_r^k) \neq S_{экс}(F_r^k)]\};$$

D_3 – отсутствие связи между входом и шиной константы $\equiv 0, \equiv 1$:

$$D_3 \leftarrow [S_{эт}(\{\equiv 0, \equiv 1\}) = S_{экс}(\{\equiv 0, \equiv 1\}) \,\&\, S_{эт}(L_i = \{\equiv 0, \equiv 1\}) \neq S_{экс}(L_j)];$$

D_4 – неисправность внешнего входного контакта цифровой системы:

$$D_4 \leftarrow [S_{эт}(L_i) \neq S_{экс}(L_i) \,\&\, S_{эт}(L_j) \neq S_{экс}(L_j)];$$

D_5 – неисправность выходов компонентов цифровой системы, объединенных в монтажную логику:

$$D_5 \leftarrow \{\forall i[S_{эт}(L_i) \neq S_{экс}(L_i)] \,\&\, [S_{эт}(L_j) \neq S_{экс}(L_j)]\}.$$

Далее выполняется построение алгоритма поиска дефектов методом обратного прослеживания для схемы, изображенной на рис. 8.5.

Априорно все доступные контакты идентифицируются порядковыми номерами или топологическими идентификаторами <координата компонента, номер контакта>. Последовательность диагностирования контактов на топологии схемы определяется двумя направлениями: «справа – налево» – при отрицательной проверке; «снизу – вверх» – при положительной. Направления могут быть и другими. Такая стратегия задает в качестве очередной точки контроля при отрицательной проверке P_j^- на выходе элемента его первый нижний входной контакт. При фиксации положительного результата P_j^+ на входном контакте примитива очередная точка контроля задается расположенной выше входной ножкой. Если в последнем случае проверка отрицательная, очередная точка контроля определяется выходным контактом элемента, связанного с рассматриваемым входом. Если в диагностическом эксперименте рассматривается очередная точка контроля P_i, не являющаяся выходным контактом схемы, то все предыдущие проверки линий-преемников имели отрицательный результат.

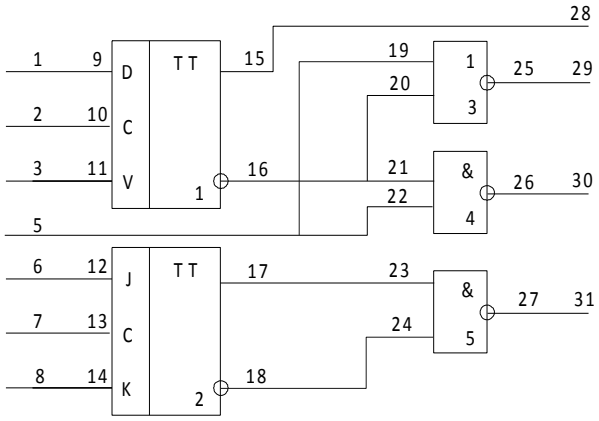


Рис. 8.5. Схема для поиска дефектов методом обратного прослеживания

Для предложенного примера (см. рис. 8.5) построенный алгоритм диагностирования представлен в табл. 8.2.

Таблица 8.2

Pi	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
Pi+	30	29	28	D 0	D 1	D 1	D 1	23	D 2	21	D 2	19	D 2	D 1	D 1
Pi-	27	26	25	15	24	22	20	18	17	5	16	16	5	14	14
Sign	21F E	31F E	41F E	51F E	21F E	31F E	41F E	11F 4	1133	11F 1	1110	1110	11F 1	11F 4	1133
Pi	16	15	14	13	12	11	10	9	8	7	6	5	3	2	1
Pi+	D 1	D 1	13	12	D 2	10	9	D 2	D 1	D 1	D 1	D 1	D 1	D 1	D 1
Pi-	11	11	8	7	6	3	2	1	D 4	D 4	D 4	D 4	D 4	D 4	D 4
Sign	1110	51F E	115E	110E	133E	11H E	11P E	13F E	115E	110E	133E	11F 1	11H E	11P E	13F E

Первой точкой контроля является контакт 31. Он имеет эталонную сигнатуру $S_{эт} = 21FE$. Все другие контакты также имеют собственные сигнатуры.

Результат сравнения эталонной и экспериментальной сигнатур формирует точки контроля на следующем шаге диагностирования при двух альтернативах: P^{+} – проверка положительная; P^{-} – проверка отрицательная.

Состояние D_0 фиксируется, если все выходы дают положительную проверку; D_1 – на гальванической связи зафиксирована положительная проверка при отрицательной предыдущей; D_2 – при отрицательной проверке на выходе все входы имеют положительный результат сравнения сигнатур; D_3 – отрицательная проверка на входе, связанном с шинами $\equiv 0, \equiv 1$; D_4 – отрицательная проверка на внешнем входе цифрового устройства.

Таблица графа поиска дефектов имеет число проверок, равное количеству наблюдаемых контактов и позволяет осуществлять диагностический эксперимент в диалоговом режиме без наличия принципиальной электрической схемы объекта. Функции контроллера управления при диагностировании заключаются в инициализации команд для подачи теста, подключении зонда сигнатурного анализатора в заданную координату точки контроля, принятии окончательного решения о состоянии объекта, ремонте дефектного блока.

Диагностирование дефектов путем анализа табл. 8.2 демонстрируется на примере разрыва связи между контактами 8 и 14. Проверка начальной координаты 31 дает отрицательный результат, что определяет очередной точкой контроля контакт 27. Его проверка имеет отрицательное значение, следовательно, выполняется переход на координату 24. Далее фиксируются отрицательные проверки на линиях 18, 14, после чего на контакте 8 фиксируется положительный результат. Вывод: техническое состояние D_1 – нет связи между контактами 8 и 14.

Применение алгоритма обратного прослеживания для поиска дефектов на стадии эксплуатации цифровых модулей связано с уменьшением количества точек наблюдения для установления диагноза. В этом случае необходимо выполнять структурный анализ схемы относительно элементарных проверок внешних выходов. В предположении наличия в схеме одиночного макродефекта, что характерно для объекта в период эксплуатации, наличие неисправности ограничивается областью $G^{\cap}$, получаемой пересечением подсхем G_j, относящихся к тем наблюдаемым выходам L_j^Y, на которых зафиксированы отрицательные проверки:

$$G^{\cap} = \forall j\{\cap G_j \leftarrow [S_{эт}(L_j^Y) \neq S_{экс}(L_j^Y)]\}, (j = \overline{1,m}).$$

Здесь G_j – подграф функционально-гальванических связей – предшественник для линии L_j^Y; m – число внешних выходов схемы.

Из подозреваемой области $G^{\cap}$ следует исключать линии (подграфы), которые имеют логические пути до наблюдаемых выходов схемы с положительным результатом испытания:

$$G = G^{\cap} \setminus \forall j\{\cup G_j \leftarrow [S_{эт}(L_j^Y) = S_{экс}(L_j^Y)]\},\ (j = \overline{1,m}).$$

При допущении наличия кратных дефектов в схеме область существования неисправностей равна объединению подсхем, относящихся к неисправным выходам, за исключением объединения подсхем, логически связанных с исправными выходами:

$$G = \forall j\{\cup G_i \leftarrow [S_{эт}(L_i^Y) \neq S_{экс}(L_i^Y)]\} \setminus \forall j\{\cup G_i \leftarrow [S_{эт}(L_i^Y) = S_{экс}(L_j^Y)]\}.$$

Для использования упомянутых правил необходимо иметь тест, который активизирует проверку ОКН относительно всех возможных выходов схемы.

Стоимость процесса поиска дефектов оценивается по числу зондирований, где каждое из них имеет собственную оценку по времени, определяемую коэффициентом k:

$$Q = \sum_{i=1}^{n} k \times P_i,\ \ (k = 1)$$

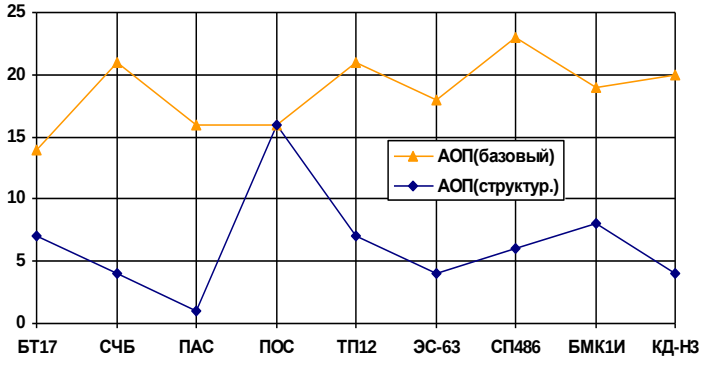
есть частный случай для одинаковой стоимости всех зондирований.

При разрыве связи между линиями 2 и 10 (см. рис. 8.5) последовательность выполнения проверок без структурного анализа имеет цену, равную 8:

$$(P_{30}^-, P_{26}^-, P_{22}^+, P_{21}^-, P_{16}^-, P_{11}^+, P_{10}^-, P_2^+) \rightarrow D_1(10,2).$$

С использованием структурного анализа стоимость диагностирования равна 3: $(P_{11}^-, P_{10}^-, P_2^+) \rightarrow D_1(10,2)$, что в 2,6 раза меньше базового алгоритма.

Выборочная статистика (на девяти схемах) числа зондирований для установления диагноза дефекта D1 на входе ЦУ представлена на рис. 8.6.



Ì î äóëü	ÁÒ17	N× Á	Ì ÁÑ	Ì Î Ñ	ÒÌ 12	ÝÑ-63	ÑÌ 486	ÁÌ Ê 1Ê	Ê Á-Ì 3
ÀÎ Ï (áàçî âûé)	14	21	16	16	21	18	23	19	20
ÀÎ Ï (ñòðóêòóð.)	7	4	1	16	7	4	6	8	4

Рис. 8.6. Стоимости диагностирования дефекта по алгоритму обратного прослеживания без- и со структурным анализом

Средняя оценка относительного уменьшения числа зондирований для установления диагноза по данным таблицы, представленной на рис. 8.69, составляет $q = \mathrm{card}(\text{АОП}_б) / \mathrm{card}(\text{АОП}_с) = 2{,}75$.

8.4.2. Алгоритм половинного деления неисправной области.

Реализация тестовых методов диагностирования цифровых систем на кристаллах [4,8] предполагает наличие инфраструктуры сервисного обслуживания, включающего интерфейс связи с объектом диагностирования, средства подачи тестов и снятия реакции изделия, логические стимуляторы и анализаторы для поиска дефектов внутри кристалла или его компонента.

Функциональное диагностирование не требует дорогостоящего сервисного диагностического оборудования, но его разрешающая способность – глубина поиска дефектов – значительно ниже, чем у тестовых методов. В условиях высокой стоимости отбраковки компонентов SoC и выполнения процедуры диагностирования далее предлагается оптимизированный алгоритм условного поиска дефектов с процедурами безусловного структурного моделирования результатов элементарных проверок для цифровых структур, который достаточно просто реализовать для решения практических задач в целях минимизации временных и материальных затрат восстановления работоспособности вычислительного устройства.

Для изложения стратегии половинного деления связных графов цифровых структур вводится вектор моделирования подозреваемой неисправной области $V = (V_1, V_2, ..., V_i, ..., V_n)$, где n – общее число точек контроля цифрового модуля, а подмножество наблюдаемых выходных контактов устрой-

ства обозначим в виде $V^Y \subset V$, $V^Y \neq \varnothing$. Значение координаты вектора $V_i \in V$ формируется в соответствии с выражением:

$$(V_i = 1) \leftarrow [S_{эт}(L_i) \neq S_{эк}(L_i) \,\&\, S_{эт}(L_j^Y) \neq S_{эк}(L_j^Y) \,\&\, L_j^Y = f(L_i)];$$

в противном случае – $V_i = 0$.

В результате диагностического эксперимента по наблюдаемым линиям формируется подмножество $V^Y \subset V$ вектора моделирования:

$$V_i^Y = \begin{cases} 1 \leftarrow S_{эт}(L_i^Y) \neq S_{экс}(L_i^Y); \\ 0 \leftarrow S_{эт}(L_i^Y) = S_{экс}(L_i^Y). \end{cases}$$

Для $V^Y \subset V$ вычисляется подграф точек контроля цифровой схемы $V' \subseteq V$, неисправные сигнатуры которых могут сформировать подмножество $V^Y \subset V$. Построение вектора моделирования неисправной области V для вычисленного подмножества $V^Y \subset V$ с использованием графа цифровой схемы основано на правилах процедуры структурного моделирования.

1. Состояния наблюдаемых линий определяются в соответствии с выражением для вычисления V_i^Y и не могут быть модифицированы.

2. Входные контакты примитива t подозреваются неисправными, если существует выходной дефектный контакт: $\forall i(V_i(X_t) = 1) \leftarrow (\exists i(V_i(Y_t) = 1)$.

3. При различных технических состояниях (исправное и неисправное) входов одного потенциала разных примитивов t и f значение линии разветвления подозревается в наличии дефекта: $V_i = V_i(X_t) \vee V_i(X_f)$.

С учетом приведенных правил процедуры структурного моделирования и вычисленного подмножества $V^Y \subset V$ алгоритм определения состояний невыходных координат вектора моделирования неисправной области V заключается в последовательной обработке примитивов ацикличной (без обратных связей) схемы методом прослеживания от внешних выходных контактов схемы к входным.

Для примера, представленного на рис. 8.5, далее демонстрируется выполнение процедуры структурного моделирования, исходя из условия, что

$$V^Y = \{V_{28} = 1, V_{29} = 0, V_{30} = 1, V_{31} = 0\}.$$

Результат моделирования на основе выполнения правил 2 и 3 дает вектор:

V = (1 2 3 5 9 10 11 15 16 21 22 26 28 30).

На подграфе $G'(V)$, который представлен матрицей $M = \| M_{ij} \|$, организуется условный диагностический эксперимент с выбором точки контроля, делящей подозреваемую область пополам.

Процедура выбора контрольной точки для связной графовой структуры.

При изложении стратегии выбора очередного контакта для зондирования схемы вводится обозначение количества единичных координат в вектор-строке предшественников для точки контроля i матрицы $M = \| M_{ij} \|$ контрдостижимостей через $A_i = \mathrm{card}(M_{ij})|_{j=\overline{1,n}}$; в вектор-столбце преемников – через $B_i = \mathrm{card}(M_{ji})|_{j=\overline{1,n}}$. Тогда при стратегии выбора контрольной точки, делящей подозреваемую область пополам, необходимо максимизировать по параметру i выражение $(A_i + B_i)/n$ (n – мерность матрицы M, соответствующей подграфу $G'(V)$), которое показывает мощность делимого подграфа $(A_i + B_i)$ по отношению к исходному $G'(V)$. Кроме того, минимизация разности предшественников и преемников точки контроля i $(A_i - B_i)/n$ показывает точность половинного деления структуры $G'(V)$. В результате получается функционал

$$f = \mathrm{MAX}|_i \, [(A_i + B_i)/n - (A_i - B_i)/n] = \mathrm{MAX}|_i \, [(2/n)(B_i)].$$

Учитывая, что коэффициент ($2/n$) есть константа, ее можно исключить из выражения для определения f, считая при этом, что вычисление мощности B_i связано с выбором минимального значения из компонентов A_i, B_i:

$$f = \mathrm{MAX}|_i \, [\mathrm{MIN}(A_i, B_i)].$$

Далее используется вычисление функционала f после каждой элементарной проверки для последующего половинного деления подозреваемой области на примере схемы (см. рис. 8.5) и матрицы контрдостижимостей:

$$Q = \left\| \begin{array}{c|ccccccccccccccc}
 & 1 & 2 & 3 & 5 & 9 & 10 & 11 & 15 & 16 & 21 & 22 & 26 & 28 & 30 \\
\hline
1 & 1 & & & & & & & & & & & & & \\
2 & & 1 & & & & & & & & & & & & \\
3 & & & 1 & & & & & & & & & & & \\
5 & & & & 1 & & & & & & & & & & \\
9 & 1 & & & & 1 & & & & & & & & & \\
10 & & 1 & & & & 1 & & & & & & & & \\
11 & & & 1 & & & & 1 & & & & & & & \\
15 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 & & & & & & \\
16 & 1 & 1 & 1 & & 1 & 1 & 1 & & 1 & & & & & \\
21 & 1 & 1 & 1 & & 1 & 1 & 1 & & 1 & 1 & & & & \\
22 & & & & 1 & & & & & & & 1 & & & \\
26 & & & & 1 & & & & & & & 1 & 1 & & \\
28 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 & & & & & 1 & \\
30 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 & & 1
\end{array} \right\|$$

В результате получается последовательность точек контроля для определения дефекта $D_1(10,2)$:

$$(P_{16}^{-}, P_9^{+}, P_{10}^{-}, P_2^{+}) \to D_1(10,2).$$

Количество проверок равно четырем, что больше на одну, чем при структурном анализе, что естественно, поскольку процедура структурного моделирования ориентирована не только на поиск одиночных дефектов.

Таким образом, предложенный алгоритм реализуют условное диагностирование макродефектов на основе использования процедуры структурного моделирования, метода половинного деления подозреваемой области, когда очередная точка контроля при анализе матрицы контрдостижимостей вычисляется по критерию maxmin.

Для функциональных дефектов внутри конструктивных или примитивных элементов на основе двух предлагаемых далее стратегий можно определить рациональную методику построения допустимого решения при условном поиске дефектов по взвешенному дереву диагностирования.

Выбор очередной точки контроля для цифровых схем, структура которых содержит в том числе и несвязные подграфы, при стратегии половинного деления определяется минимумом следующего функционала (функции предпочтения):

$$f_1 = \mathrm{MIN}_j \,|\, (D_{i-1} \wedge M_j) - (D_{i-1}/2), \qquad (8.5)$$

где D_{i-1} – вектор подозреваемых дефектов, полученный после зондирования предыдущей координаты i-1 при выполнении проверки H_{i-1}; M_j – строка матрицы достижимостей $M = \| M_{ij} \|$, которая определяет векторы единичных координат, идентифицирующих связь (функционально-гальваническую) точки контроля, соответствующей номеру столбца, с вершиной подграфа, определяемой диагональной координатой M_{ii}.

При наличии в схеме одиночного дефекта по результатам проверки состояний выходных контактов схемы формируется вектор $V = (V_1, V_2, ..., V_i, ..., V_k)$, когда $V_i = 0$, если на i-м наблюдаемом выходе зафиксировано равенство эталонной и экспериментальной сигнатур $S_{эт} = S_{экс}$ и значение переменной $V_i = 1$ – в противном случае.

Пересечение строк матрицы M, отмеченных единичными значениями, задает область существования дефекта примитивного элемента

$$D = \cap |_{V_i = 1} M_i,$$

внутри которого далее организуется система проверок с учетом веса кандидата в точки контроля, определяемого числом нагруженных на нее контактов-предшественников. Функция предпочтения определяет такую линию,

которая разбивает пространство подозреваемых дефектных координат на две более равные группы. В предположении существования в схеме кратного дефекта подозреваемая область на первом шаге определяется объединением строк, соответствующих единичным значениям координат выходных линий вектора экспериментальной проверки: $D = \cup|_{V_i=1} M_i$.

Суммарные затраты на создание бинарного дерева поиска дефектов в зависимости от числа линий объекта n задаются выражением $C = (\sum n - i)$ при $(i = \overline{0, n-1})$. Стоимость реализации стратегии есть самая длинная ветвь дерева, которая определяется конъюнкцией вершин:

$$R[D(H_i)] = \wedge_i H_i |_{\max\{D_i^+, D_i^-\}},$$

где $D(H_i) = \{D_i^+, D_i^-\}$ – разбиение линий для точки H_i, D_i^+ – область исправных линий; D_i^- – область существования дефекта. При последовательной структуре соединения элементов (цепь) максимальная стоимость реализации стратегии ограничена целой частью логарифма, округленной в большую сторону: $R[D(H_i)] =]\log_2 n[$, где n – число точек контроля в цепочке элементов. В случае параллельной структуры независимых элементов (схема, содержащая n несвязанных инверторов) максимальное число элементарных проверок равно n-1. Для остальных структур дерево поиска дефектов будет иметь максимальную ветвь проверок, заключенную в интервале

$$]\log_2 n[\le R[D(H_i)] \le n - 1.$$

Для стратегии обратного прослеживания выбор очередной точки контроля осуществляется на основе вычисления функции

$$f_2 = \max_j |D_{i-1} \wedge M_j|, \ (j = \overline{1, n};\ j \ne i - 1),$$

которая находит в качестве очередной линии наблюдения одного из непосредственных предшественников предыдущей проверки при ее отрицательном результате. Если H_{i-1} была положительной, то до вычисления f_2 выполняется коррекция подозреваемой области $D_{i-1} = D_{i-2}^- \setminus D_{i-1}^+$.

Стратегия обратного прослеживания для структуры типа «цепь» имеет n-1 элементарную проверку, для множества независимых друг от друга элементов – одну точку контроля. В общем случае стоимость реализации стратегии обратного прослеживания находится в интервале

$$1 \le R[D(H_i)] \le n - 1.$$

Учитывая различные нижние оценки стоимости реализации стратегий f_1 и f_2 для произвольных структур, необходимо воспользоваться критерием минимакса в целях выбора такого дерева поиска дефектов, которое минимизи-

рует максимальную последовательность элементарных проверок на двух предложенных стратегиях в предположении реализации самой неблагоприятной ситуации:

$$R[f(D(H_i)] = \min\{\max R[f_1(D(H_i))],\ \max R[f_2(D(H_i))]\}.$$

Использование минимакса позволяет строить взвешенное дерево поиска дефектов, которое предпочтительнее для оператора, поскольку при возникновении любого одиночного дефекта на его поиск потребуется примерно одинаковое число элементарных проверок. В свою очередь, наличие длинных ветвей проверок при выполнении диагностического эксперимента приводит к дорогому по стоимости диагнозу. На практике в 80% случаев более предпочтительной является стратегия половинного деления, дающая дерево поиска дефектов, которое хуже оптимального не более чем на 20%.

На рис. 8.7 представлена цифровая структура из пятнадцати функциональных элементов, где в качестве точек контроля выступают семнадцать занумерованных невходных эквипотенциальных линий, принадлежащих контактам микросхем.

По упомянутой структуре алгоритмически строится граф функционально-гальванических связей ЦУ, изображенный на рис. 8.9, который формально задается матрицей достижимостей (рис. 8.8).

Стратегия условного диагностирования с помощью ведомого логического зонда определяется правилом: если элементарная проверка на выходе микросхемы или примитива отрицательная, а проверки всех его входов дали положительный результат, то делается вывод о наличии дефекта в рассматриваемом компоненте. Процедура построения взвешенного дерева поиска дефектов заключается в вычислении такой очередной точки контроля, которая разделит в идеальном случае пополам исходную область существования дефекта. Дихотомическое дерево диагностирования дефектов функциональных элементов, спроектированное по стратегии f1, представлено на рис. 8.10.

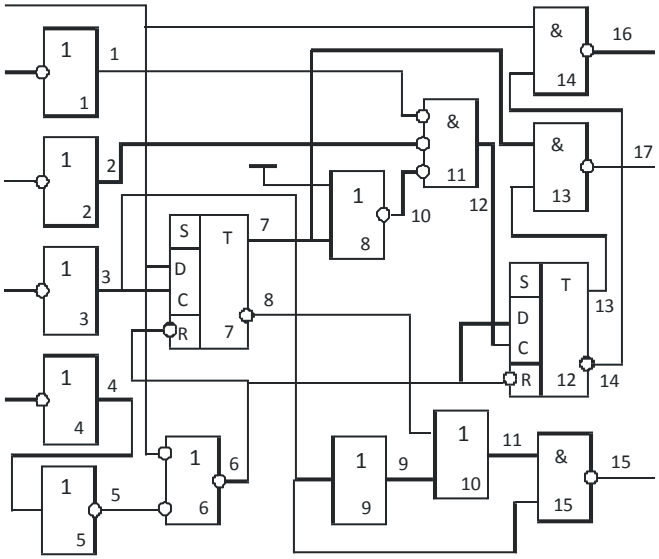


Рис. 8.7. Схема для диагностирования

M =

Mij	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
5				1	1												
6				1	1	1											
7			1	1	1	1	1										
8			1	1	1	1		1									
9			1						1								
10			1	1	1	1	1		1	1							
11			1	1	1	1		1	1		1						
12	1	1	1	1	1	1	1		1	1		1					
13	1	1	1	1	1	1	1		1	1		1	1				
14	1	1	1	1	1	1	1		1	1		1		1			
15			1	1	1	1		1	1		1				1		
16	1	1	1	1	1	1	1		1	1		1		1		1	
17	1	1	1	1	1	1	1		1	1		1	1				1

Рис. 8.8. Матрица достижимостей

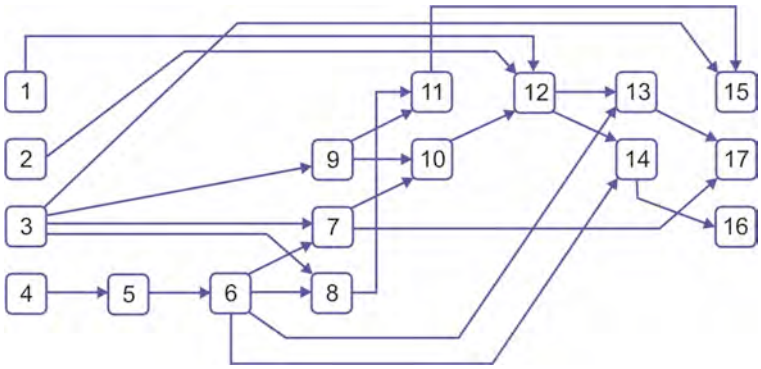


Рис. 8.9. Граф функционально-гальванических связей

Априорно в качестве подозреваемой области выступает вся схема, которая имеет 17 точек контроля. По матрице контрдостижимостей на основе применения (8.5) вычисляется точка 15, делящая на две равные области исходное множество линий, поскольку

$$f_1(H_j)|_{j=15} = (8) - (8,5) = 0,5.$$

Для других точек контроля оценка $f_1(H_j)$ будет большей, чем 0,5. Полученные подмножества

$$D(H_{15}) = \{D_{15}^- = \{3,4,5,6,8,9,11,15\},\ D_{15}^+ = \{1,2,7,10,12,13,14,16,17\}\}$$

определяют области поиска дефектов при фиксации в точке 15 соответственно отрицательной или положительной проверки. Далее для подмножества D_{15}^- в качестве оптимальной, доставляющей минимум функционалу f_1, выбирается проверка $H_8 : D(H_8) = \{D_8^- = \{3,4,5,6,8\},\ \ D_8^+ = \{9,11,15\}\}$, а для D_{15}^+ такой линией будет

$$H_{12} : D(H_{12}) = \{D_{12}^- = \{1,2,7,10,12\},\ D_{12}^+ = \{13,14,16,17\}\}.$$

Признаком окончания ветви является наличие в результирующем подмножестве единственной точки контроля, которая позволяет установить неисправный компонент (см. рис. 8.10), идентифицируемый номером выходной линии дефектного примитива. Построенное дерево поиска дефектов имеет максимальную ветвь, содержащую пять элементарных проверок, что является характеристикой алгоритма диагностирования данного ЦУ.

Стоимость условного диагностирования дефекта ПЭ 3 при структурном моделировании результатов диагноза по выходам

$$\{V_{15} = 0, V_{16} = 1, V_{17} = 1\}$$

дает вектор V=(11111110010111011) по всем линиям, что означает отсутствие неисправных реакций на линиях 8, 9, 11, 15. Это дает основание осуществлять поиск дефектов по начальной матрице

	1	2	3	4	5	6	7	10	12	13	14
1	1										
2		1									
3			1								
4				1							
5				1	1						
6				1	1	1					
7			1	1	1	1	1				
10			1	1	1	1	1	1			
12	1	1	1	1	1	1	1	1	1		
13	1	1	1	1	1	1	1	1	1	1	
14	1	1	1	1	1	1	1	1	1		1

При этом вектор элементарных проверок для установления диагноза – неисправен ПЭ 3 – имеет стоимость $(P_7^-, P_6^+, P_7^+) \to D_2(3)$, что на одно зондирование меньше, чем во взвешенном дереве поиска дефектов.

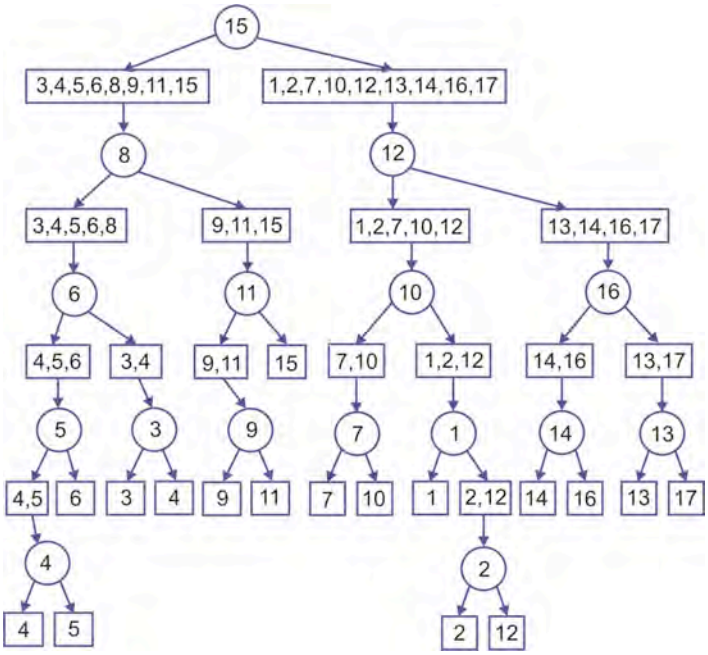
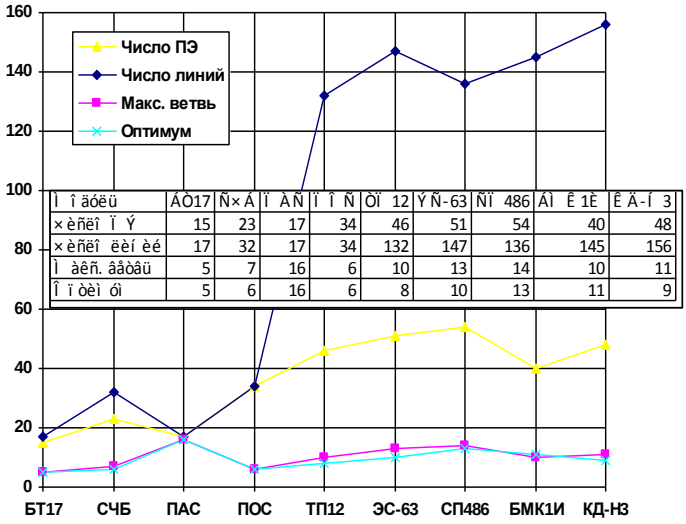


Рис. 8.10. Взвешенное дихотомическое дерево

Таким образом, предложенные алгоритмы реализуют стратегии обратного прослеживания и половинного деления на основе анализа матрицы достижимостей эквипотенциальных линий, что позволяет проектировать условные

процедуры поиска дефектов в неисправных примитивах или конструктивах, с использованием структурного моделирования результатов диагноза по выходам, уменьшающего стоимость поиска дефектов на 15-30%, что иллюстрирует рис. 8.11.



Ì î äóëü	ÁÒ17	Ñ× Á	Ï ÀÑ	Ï Î Ñ	ÒÏ 12	Ý Ñ-63	ÑÏ 486	ÁÌ Ê 1È	Ê Á-Í 3
× èñêî Ï Ý	15	23	17	34	46	51	54	40	48
× èñêî ëèí èé	17	32	17	34	132	147	136	145	156
Ì àêñ. âåòâü	5	7	16	6	10	13	14	10	11
Î ï òèì ôì	5	6	16	6	8	10	13	11	9

Рис. 8.11. Анализ оптимальности деревьев

8.4.3. Стратегия зондирования дефектов на основе анализа МТН.

Временные затраты элементарной проверки при выполнении условного диагностического эксперимента определяются выражением

$$T = T_M + T_B + T_K + T_T,$$

где T_M – время поиска координат необходимого компонента и требуемого контакта (вывода) T_B в целях установки ведомого сигнатурного зонда; T_K – время проверки наличия связи между зондом и выводом; T_T – время подачи теста на объект диагностирования, снятия эталонной реакции в точке контроля, получения экспериментальной сигнатуры и сравнения ее с эталонной. Учитывая, что время выполнения каждой операции имеет определенную стоимость, материальные затраты на реализацию элементарной зондовой проверки составляют

$$T = C_M T_M + C_B T_B + C_K T_K + C_T T_T.$$

В связи с этим целесообразно уменьшать количество условных точек контроля, которое формирует основную стоимость восстановления работоспособности изделия, для чего необходимо использовать диагностическую информацию в виде многозначных таблиц неисправностей.

Анализ МТН применяется для безусловного определения дефектной области в цифровом модуле. Естественно, что после нахождения такой области возникает необходимость в более точной локализации неисправности. В этом случает на помощь приходят условные алгоритмы поиска дефектов на основе анализа структуры функционально-гальванических связей. Предлагается использовать информацию в виде МТН для вычисления очередной точки контроля, которая разделит подозреваемую область на примерно одинаковые подмножества неисправностей.

В результате выполнения процедуры моделирования в целях определения качества теста строится многозначная таблица неисправностей $D = \| D_{ij} \|$, заданная на множестве входных тестовых последовательностей $T = (T_1, T_2, ..., T_i, ..., T_k)$, для каждой из которых задана вектор-строка проверяемых одиночных константных неисправностей (ОКН) $D_i = (D_{i1}, D_{i2}, ..., D_{ij}, ..., D_{in})$. Координата D_{ij} определена на алфавите $A = \{0,1, X, \varnothing\}$. $D_{ij} = 0(1)$, если тест-вектор T_i проверяет константу 0(1) на линии j. Символ X означает проверку константной неисправности нуля и единицы, что присуще только схемам с памятью, когда проверяющие способности текущего вектора объединяются с неисправностями, проверяемыми на предыдущих входных наборах. Значение $D_{ij} = \varnothing$ определяет отсутствие проверяемых на линии j неисправностей тестом T_i. Каждой неисправности D_{ij}, проверяемой на тест-векторе T_i, ставится в соответствие отображение $\Gamma(D_{ij})$ во множество наблюдаемых выходов $Z_i \in Z$, $Z = \| Z_{ir} \|$, по которым проверяется дефект D_{ij}: $\Gamma(D_{ij}) = Z_i$. Координата вектор-строки $Z_{ir} \in Z$ принимает значение 1, если неисправность D_{ij} обнаруживается по выходу Z_{ir}. В противном случае $Z_{ir} = 0$. Обратное отображение определяет множество неисправностей D_i, обнаруживаемых тест-вектором T_i по выходу D_{ij}: $\Gamma^{-1}(Z_{ir}) = D_i$. Координата вектор-строки $D_{ij} \in D$ принимает значение {0,1}, если неисправность $D_{ij} = \{0,1\}$ обнаруживается по выходу Z_{ir}. В противном случае – $D_{ij} = \varnothing$. При таком определении многозначной таблицы неисправностей выполняются аксиомы и тождества, приведенные ниже.

Аксиома 8.1. Объединение всех обратных отображений дает вектор-строку МТН: $\cup|_{r=1,v}\, \Gamma^{-1}(Z_{ir}) = D_i$, где v – количество наблюдаемых выходов.

Аксиома 8.2. Дизъюнкция всех прямых отображений дает единичную вектор-строку матрицы экспериментальной проверки: $\vee|_{j=1,n}\, \Gamma(D_{ij}) = Z_i$, где n – количество эквипотенциальных линий в схеме.

Лемма 8.1. Глубина диагностирования дефекта зависит от количества наблюдаемых эквипотенциальных линий в схеме.

Если в схеме один наблюдаемый выход, то все дефекты D_i, обнаруживаемые на тест-векторе T_i, будут эквивалентными, которые составят список подозреваемых неисправностей. Естественно, на полном тесте T количество эквивалентных неисправностей определяется процедурами (8.3), (8.4). Добавление еще одного наблюдаемого выхода не может не улучшить глубину поиска дефектов, если только добавленный выход не является дублирующим ранее заданную выходную линию. Но в этом случае количество наблюдаемых эквипотенциальных линий останется прежним и равным единице.

Необходимо рассмотреть соотношение влияния наблюдаемых линий в схеме на формирование разрешающей способности фиксированного теста. Пусть заданы две линии Z_{ir} и Z_{il}. Очевидны три случая:

1. Расположение на непересекающихся (параллельных) путях, когда у линий нет общих преемников и предшественников. Тогда их значения создают четыре различных условия: $(Z_{ir}, Z_{il}) = (00,01,10,11)$. Первое условие уменьшает на векторе T_i область существования дефекта на $D_i \setminus \Gamma^{-1}(Z_{ir}) \cup \Gamma^{-1}(Z_{il})$.. Для второго уменьшение связано с мощностью обратного отображения $D_i \setminus \Gamma^{-1}(Z_{ir})$. Для третьего – $D_i \setminus \Gamma^{-1}(Z_{il})$. Для четвертого область существования дефектов не уменьшится, что связано с наличием кратной неисправности, поскольку для одиночной необходимо наличие общих предшественников.

2. Расположение линий на пересекающихся путях, когда у них есть общие предшественники. Их значения создают четыре различных условия: $(Z_{ir}, Z_{il}) = (00,01,10,11)$. Для четвертого условия возможно существование кратного или одиночного дефекта, который проявляется по обоим выходам.

3. Расположение линий на одном логическом пути, когда одна из них есть предшественник, другая – преемник. Их значения создают четыре различных условия: $(Z_{ir}, Z_{il}) = (00,01,10,11)$. Для второго и третьего условий уменьшение подозреваемой области определяется в виде

$$D_i \setminus \Gamma^{-1}(Z_{ir});\ D_i \setminus \Gamma^{-1}(Z_{ir}).$$

Для четвертого условия также возможно существование кратного или одиночного дефекта, который проявляется по обоим выходам.

Таким образом, при фиксированной длине теста повышение глубины диагностирования определяется увеличением числа наблюдаемых линий или, при наличии фиксированного числа выходов, следует выполнять условный диагностический эксперимент, используя в качестве диагностической информации многозначную таблицу неисправностей.

Процедура диагностического эксперимента по МТН уже становится комбинированной, которая включает стадии выполнения условного и безусловного алгоритмов. Иначе ее можно определить как безусловный, на первом шаге, и условный – на последующих. Очередная точка контроля является дополнением к уже существующим линиям, после чего все точки контроля обрабатываются по единой методике. Диагностический эксперимент заканчивается, если на некотором шаге достигнута требуемая точность диагностирования, например, найден неисправный элемент или микросхема.

Следующая структура алгоритма, представленная рис. 8.12, поясняет универсальную процедуру диагностирования на основе анализа МТН.

Рис. 8.12. Алгоритм комбинированного диагностирования

1. Подача тест-векторов $T = (T_1, T_2, ..., T_i, ..., T_k)$ на цифровой модуль и снятие экспериментальной реакции с наблюдаемых выходов или с логического анализатора $Z = \| Z_{ir} \|$, где параметр r количества наблюдаемых выходов инкрементируется на каждой итерации.

2. Сравнение значений на всех наблюдаемых и зондируемых выходах схемы цифрового устройства:

$$Z_{ir} \oplus T_{ir} = V_{ir}.$$

3. Анализ МТН по правилам алгоритма структурного моделирования (8.3) – (8.4) в целях вычисления списка подозреваемых одиночных или кратных константных неисправностей D.

4. Если список $D = \varnothing$, делается вывод (п. 8) об исправности цифрового модуля, естественно при этом

$$\forall(i,r):(Z_{ir} \oplus T_{ir} = 0).$$

Конец процедуры диагностирования цифрового модуля.

5. Если же список $D \neq \varnothing$, то выполняется сравнение

$$\mathrm{card}(D) \leq \mathrm{card}(D_i^c)$$

по мощности, которая должна быть меньше одного из наперед заданных подмножеств

$$D^c = (D_1^c, D_2^c, ..., D_i^c, ..., D_p^c),$$

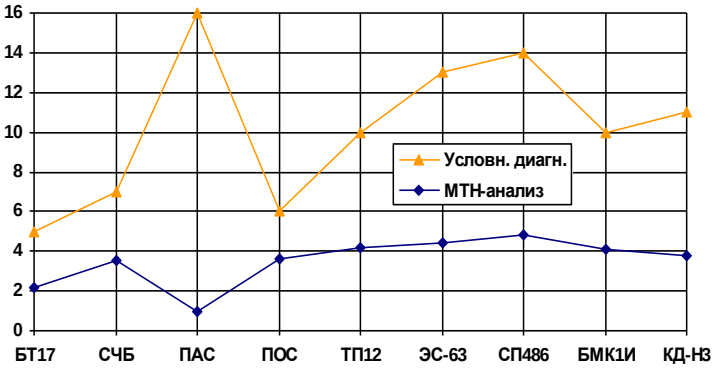
как правило, формирующих список эквивалентных дефектов, заключенных в пределах конструктива D_i^c, число которых равно p. При выполнении данного условия осуществляется процедура восстановления работоспособности (ремонт компонента) в соответствии с п. 7 с последующим переходом к п. 1 алгоритма диагностирования.

6. Если $\mathrm{card}(D) \geq \mathrm{card}(D^c)$, вычисляется очередная точка контроля, а число наблюдаемых линий инкрементируется: r=r+1. Переход к п. 1. Существенным при реализации комбинированного алгоритма диагностирования представляется выполнение п. 6. Какова стратегия определения очередной точки контроля? Учитывая, что в лучшем случае очередная линия проверки должна делить подозреваемую область пополам, стратегия выбора контрольной точки определяется минимумом функции предпочтения:

$$f_1 = \min|(D \wedge M_j) - (D/2)|.$$

Здесь D – вектор подозреваемых дефектов, полученный после зондирования предыдущей координаты; M_j – строка матрицы достижимостей $M = \|M_{ij}\|$, которая определяет векторы единичных координат, идентифицирующих связь (функционально-гальваническую) точки контроля, соответствующей номеру столбца, с вершиной подграфа, определяемой диагональной координатой M_{ii}.

Стратегия выбора точек зондирования обеспечивает нахождение квазиоптимального решения, что есть деление подозреваемой области пополам, если это возможно. Модельно проведенные диагностические эксперименты для 17 внесенных неисправностей на каждом из 9 цифровых модулей показали, что число точек контроля по сравнению с условным диагностированием по стратегии половинного деления при использовании анализа МТН уменьшается в 4 раза (рис. 8.13). Исключив из рассмотрения блок ПАС, реальное число зондирований по остальным модулям уменьшается в 2 раза.



Ì î äóëü	ÀÒ17	Ñ× À	Ì ÀÑ	Ì Ì Ñ	ÒÌ 12	ÝÑ-63	ÑÌ 486	ÀÌ Ë 1Ë	Ë À-Ì 3
Óñëî âí . äèàãí .	5	7	16	6	10	13	14	10	11
Ì ÒÍ -àí àëèç	2,2	3,5	1	3,6	4,2	4,4	4,8	4,1	3,8

Рис. 8.13. Сравнительный анализ числа зондирований

8.4.4. SL-алгоритм диагностирования тестопригодных цифровых структур.

Согласно [9] тестопригодность цифровых систем достигается путем применения методов проектирования цифровых объектов, которые позволяют уменьшить временные и материальные затраты на организацию и проведение диагностического эксперимента. Цифровой объект является тестопригодным, если процедуры генерации тестов, оценки их качества и реализации тестового диагностирования могут быть выполнены при условии соблюдения в установленных пределах финансовых и временных затрат и значений показателей, характеризующих приспособленность схемы к контролю и поиску дефектов, реализации тестового диагностирования.

Методы проектирования тестопригодных схем: 1) анализ структурно-функционального исполнения объекта, численная оценка управляемости и наблюдаемости как меры тестопригодности схемы, которая может использоваться на этапе проектирования; 2) способы структурного проектирования тестопригодных схем, в пределе самотестируемых, основанных на использовании свойств сканируемого пути, обеспечивающего доступ к внутренним точкам схемы, а также эффективные методы генерации тестов; 3) разработка перечня практических руководящих указаний, обеспечивающих сокращение затрат на процедуры генерации тестового диагностирования.

Положительные моменты, связанные с сокращением времени полного цикла проектирования и стоимости тестового диагностирования, достигаются за счет увеличения оборудования (введение контрольных точек, нефункциональных элементов) и дополнительных временных затрат на отдельных этапах проектирования.

Применительно для реализации тестового диагностирования цифровых модулей на основе использования МТН следует заметить, что выполнение эле-

ментарной проверки при зондировании устройства требует значительного времени, наличия высокоточного и дорогостоящего оборудования, а также проведения процедуры высококвалифицированным специалистом. Выходом из данной ситуации может служить устранение физического процесса зондирования, но замена его процедурой сканирования необходимых точек контроля. Для этого в функциональную аппаратуру добавляется избыточность, предназначенная для обеспечения доступа к необходимой точке контроля посредством ее коммутации на наблюдаемый выход сканирования. Применение коммутатора позволит решить проблему вывода на внешний разъем значительного числа контрольных точек, что в условиях высокой стоимости проектирования и изготовления на плате (в микросхеме) каждого вывода можно считать компромиссным и удовлетворительным решением. Структура цифрового модуля с тестопригодной избыточностью для решения задачи поиска дефекта с заданной разрешающей способностью (до компонента или чипа) представлена на рис. 8.14.

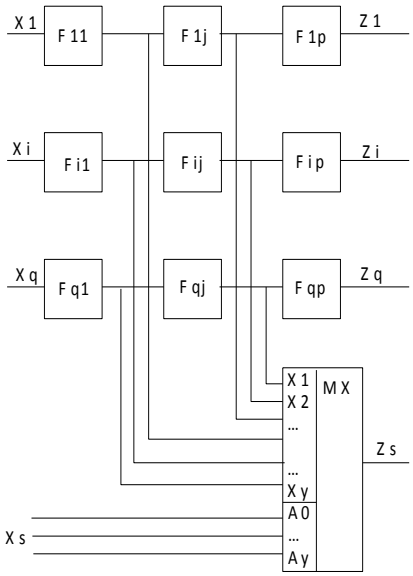


Рис. 8.14. Тестопригодная структура диагностирования

Аппаратурная избыточность определяется наличием мультиплексора, который имеет число информационных входов, определяемое в виде $\mathrm{card}(Xy) = (q-1)p$. Количество адресных входов, которые являются внешними линиями схемы, выходящими на разъем, равно $\mathrm{card}(Xs) = \log_2(q-1)p$. Кроме того, схема имеет выход Z_s для сканирования состояния внутренних линий. Тестопригодность структуры обеспечивает технологичность генерирования тестов, которая сводится к построению проверяющих последовательностей для отдельных функциональных элементов F_{ij} без необходимо-

сти транспортирования проверяющих способностей на внешние функциональные выходы цифрового модуля. При этом проблема доставки тест-сегмента к входам функционального элемента остается актуальной и решается известными способами (например, А-алгоритм Брейера [10,11]). Учитывая, что количество элементов равно pq, длина теста в худшем случае определяется суммой длин тест-сегментов для каждого функционального элемента: $\mathrm{card}(T) = \sum \mathrm{card}(T_{ij}),\ i = \overline{1,q};\ j = \overline{1,p}.$

Упомянутая оценка может быть уменьшена благодаря последовательному соединению элементов, вследствие чего проверка примитива F_{ip} может обеспечивать тестирование всех предшественников:

$$F_{i1}, F_{i2}, \ldots, F_{ip-1}.$$

При этом соотношение тест-сегментов определяется в виде $T_{i1} \subseteq T_{i2} \subseteq \ldots \subseteq T_{ip-1}$. Учитывая произвольность структурной организации цифрового модуля, для возможной минимизации длины теста необходимо выполнять проверку условия $T_{ij} \subseteq T_{il}$, существование которого позволяет устранить избыточность T_{ij}. Кроме того, структура не должна иметь глобальных обратных связей, охватывающих более одного функционального элемента. При наличии таковых следует предусмотреть в проекте технологический или логический их разрыв.

Другое важное преимущество тестирования проекта заключается в необходимости подачи только сегмента T_{ip} для проверки конструктива $F_{ij}, (j = \overline{1,p})$.

К существенному недостатку следует отнести априорное условие исправности мультиплексора и отсутствие неисправностей на его входах и выходах.

SL (Scan Line)-алгоритм диагностирования для тестопригодной структуры с использованием МТН и графа функционально-гальванических связей эквипотенциальных линий представлен следующими пунктами.

1. Задание по управляющим входам X_s режима проверки очередного функционального модуля $F_{ij}, i = \overline{1,q};\ j = \overline{1,p}$.

2. Подача на функциональные входы $(X_1, \ldots, X_q)$ тест-сегмента T_{ip}, $i = \overline{1,q}$.

3. Определение экспериментальной реакции элемента F_{ij} по выходу Z_s и ее сравнение с эталонными значениями с последующим формированием матрицы экспериментальной проверки:

$$Z_r(F_{ij}) \oplus T_r(F_{ij}) = V_r(F_{ij})[r = 1, n(F_{ij})],$$

где r – параметр тест-вектора, $n(F_{ij})$ – длина тест-сегмента для проверки примитива $n(F_{ij})$, $T_r(F_{ij})$ и $Z_r(F_{ij})$ – эталонные и экспериментальные реакции на выходе элемента (F_{ij}), $V_r(F_{ij})$ – вектор столбец матрицы экспериментальной проверки для выхода примитива F_{ij} .

4. Выполнение пунктов 1-3 реализуется для всех примитивов цифрового модуля, после чего осуществляется структурный анализ МТН по полученной матрице экспериментальной проверки V в целях определения множества одиночных или кратных константных неисправностей, локализованных в пределах каждого конструктивного элемента.

Достоинство предложенной процедуры диагностирования тестопригодного проекта заключается в безусловном поиске одиночных и кратных дефектов, искажающих функционирование цифрового модуля, с разрешающей способностью, определяемой конструктивным элементом замены.

Для создания тестопригодного проекта полезными представляются следующие практические рекомендации: 1. Исключать логическую избыточность, которая определяется независимостью выходов схемы от состояния линии на всех входных наборах. 2. Выполнять декомпозицию больших схем с целью уменьшить затраты на генерацию тестов. Функциональные модули желательно делать одновыходовыми в едином конструктивном исполнении. 3. Избегать применения асинхронных схем, в которых используются элементы памяти в форме асинхронных защелок и глобальные обратные связи. 4. Обеспечивать простоту начальной установки элементов памяти, которая должна обязательно предшествовать тестовой программе (при выполнении диагностического эксперимента) и процедуре моделирования. 5. Обеспечивать возможность логического или физического разрыва глобальных ОС, которые усложняют процедуры генерации тестов и моделирования ОКН. 6. Компоновать монтажную схему так, чтобы эквивалентные неисправности размещались в одном корпусе. 7. Избегать использования диагностически неразличимых групп ПЭ (проводное И, ИЛИ, узлы с большим числом разветвлений). 8. Обеспечивать функциональную полноту схемы в одном кристалле (компоненте) цифрового устройства. 9. Выполнять вывод труднонаблюдаемых линий схемы на регистр граничного сканирования данных. 10. Функциональные блоки компоновать таким образом, чтобы логические элементы, расположенные на одном пути, были размещены в одном конструктиве. 11. Число выходов, ответственных за проверку функциональной подсхемы, должно быть минимальным. Каждый такой выход с соответствующей ему подсхемой желательно размещать в отдельном компоненте устройства.

8.5. Выводы

1. Рассмотрен алгоритм обратного прослеживания с использованием сигнатурного анализа и постмоделированием структуры для поиска макродефек-

тов на стадиях производства и эксплуатации цифровых устройств. Процедура диагностирования предполагает наличие: информации о структуре объекта, теста проверки неисправностей.

2. Представлен алгоритм условного диагностирования на основе использования проектируемого взвешенного дерева поиска, которое строится по стратегии половинного деления подозреваемой неисправной области. Глубина диагностирования, до уровня неисправного функционального элемента, достигается на основе проверки наблюдаемых линий конструктивов.

3. Описан алгоритм структурного анализа МТН с использованием дополнительной диагностической информации, получаемой в результате последовательного зондирования точек контроля, определяемых по стратегии половинного деления подозреваемой области. Структурное моделирование многозначной таблицы неисправностей позволяет на 30 % уменьшить список подозреваемых дефектов.

4. Показана структура тестопригодного проекта цифрового модуля, на которой безусловно (незондовыми методами) решается задача диагностирования дефектов с разрешающей способностью до заменяемого конструктивного элемента, благодаря введению избыточности – мультиплексора, коммутирующего линии функциональных элементов на контакт разъема.

8.6. Литература

1. Шейнаускас Р., Таргамадзе А. Мобильное программное обеспечение САПР. Принципы разработки.– Вильнюс: Мокслас, 1989.– 184с.

2. Контроль и диагностика вычислительных устройств и систем / Хаханов В.И., Кривуля Г.Ф., Рысованый А.Н., Монжаренко И.В./ Под ред. В.И. Хаханова/ Харьков: ХВУ, 1997.– 304 с.

3. Автоматизация диагностирования электронных устройств/ Ю.В.Малышенко и др./ Под ред. В.П.Чипулиса.– М.: Энергоатомиздат, 1986.– 216с.

4. Пархоменко П.П., Согомонян Е.С. Основы технической диагностики (Оптимизация алгоритмов диагностирования, аппаратурные средства)/ Под ред. П.П. Пархоменко.– М.: Энергия, 1981.– 320с.

5. Ярмолик В.Н. Контроль и диагностика цифровых узлов ЭВМ.–Минск: Наука и техника, 1989.– 234с.

6. Гордон Д., Надиг Х. Локализация неисправностей в микропроцессорных системах при помощи шестнадцатеричных ключевых кодов// Электроника.– 1977.– №5.– С. 23-33.

7. Байда Н.П., Кузьмин И.В., Шпилевой В.Т. Микропроцессорные системы поэлементного диагностирования.– М.: Радио и связь, 1987.– 256с.

8. Основы технической диагностики / Под. ред. П.П.Пархоменко.– М.: Энергия, 1976.– 460с.

9. Беннеттс Р.Д. Проектирование тестопригодных логических схем/ Пер.с англ. Дербуновича Л.В.– М.: Радио и связь, 1990.– 176с.

10. Breuer M.A., Friedman A.D. Diagnosis and reliable design of digital system.– Woodlound wills, Computerscience press inc., 1976.– 308p.

11. Abramovici, Miron. Digital System Testing & Testable Design.–IEEE.– 1998.– 570p.

12. Хаханов В.И. Техническая диагностика цифровых и микропроцессорных структур: Учебник.– К.: ИСИО, 1995.– 242с.

9. ДВУХТАКТНОЕ КУБИЧЕСКОЕ ИСЧИСЛЕНИЕ

Представлен математический аппарат [1-24] компактного описания функций и структур в форме таблиц, универсальных относительно решения задач анализа, включающих выполнение прямой и обратной импликаций и являющихся базовыми процедурами в методах моделирования, синтеза тестов и диагностирования дефектов цифровых систем. Приведены примеры, иллюстрирующие эффективность алгебр и структур для аналитического (табличного) описания и логического анализа моделей цифровых систем и компонентов.

9.1. Концепция представления моделей цифровых объектов

Математические модели объектов, оперирующие дискретной информацией, представляются триадой компонентов:

$$F = (f, t, h),$$
$$f = (f_1, f_2, \ldots, f_i, \ldots, f_k);$$
$$t = (t_1, t_2, \ldots, t_i, \ldots, t_m);$$
$$h = (h_1, h_2, \ldots, h_i, \ldots, h_n);$$

где f, t, h – дискретные параметры функциональной, временной, структурной адекватности; k, m, n – мощности незамкнутых, конечных множеств известных способов формализации при описании функций, временных особенностей и структур. При этом адекватность (степень соответствия и детализации параметров задания функций, времени и структуры) двух моделей идеальному объекту сравнима, если выполняется условие:

$$F_q = (f_q, t_q, h_q) \geq F_p = (f_p, t_p, h_p) \leftrightarrow (f_q \geq f_p; t_q \geq t_p; h_q \geq h_p).$$

В противном случае –

$$(f_q \geq f_p; t_q \geq t_p; h_q \leq h_p) \vee (f_q \geq f_p; t_q \leq t_p; h_q \geq h_p) \vee$$
$$\vee (f_q \leq f_p; t_q \geq t_p; h_q \geq h_p) \vee (f_q \leq f_p; t_q \leq t_p; h_q \geq h_p) \vee$$
$$\vee (f_q \geq f_p; t_q \leq t_p; h_q \leq h_p) \vee (f_q \leq f_p; t_q \geq t_p; h_q \leq h_p) -$$

модели F_q, F_p несравнимы по метрике использования всех трех компонентов. Для описания функций имеются системный, регистровый и вентильный уровни подробности в описании проекта. Под метрикой временной адекватности подразумевается период синхронизации моделируемых процессов: часы, минуты, секунды, миллисекунды, наносекунды. Под структурной адекватностью понимается уровень иерархии в представлении структуры проекта, иначе – количество элементов, используемое при описании компонента верхнего уровня. Общий относительный критерий адекватности моде-

ли реальному объекту, в соответствии с приведенным описанием сущности, имеет следующий вид:

$$Q = \frac{1}{3}\left(\frac{f^k}{f^{max}} + \frac{t^{min}}{t^k} + \frac{h^k}{h^{max}}\right).$$

Поскольку триада (f, t, h), описывающая модель объекта, подлежит тестированию, что занимает до 70% времени и материальных затрат на создание проекта, то существенным представляется введение в формулу интегральной адекватности параметра полноты или качества q^t / q^{max} предложенного теста, который в пределе должен быть исчерпывающим относительно всех трех компонентов представления модели:

$$Q = \frac{1}{4}\left(\frac{f^k}{f^{max}} + \frac{t^{min}}{t^k} + \frac{h^k}{h^{max}} + \frac{q^t}{q^{max}}\right).$$

Адекватность моделей также может быть сопоставима по одному или двум компонентам обобщенной модели. Определение реального пространства описания модели к идеальному – отношение адекватностей – есть экспериментальная работа, сопряженная со сравнением существующих способов формализации задания параметров (f, t, h). Для анализа и классификации моделей цифровых объектов в целях проектирования диагностического обеспечения интерес представляет множество всех подмножеств универсума F, задающее отображенное в табл. 9.1 многообразие моделей цифровых объектов и компонентов.

Таблица 9.1

Компоненты	Определение модели цифрового устройства
{f}	Функциональная синхронная модель не учитывает временных задержек и структуры объекта
{t}	Временная модель учитывает время прохождения сигналов от входов к выходам без учета структуры и функций
{h}	Структурная модель определяется графом, который не учитывает функций и временных параметров примитивных элементов структуры
{f,t}	Функциональная асинхронная модель есть совокупность взаимосвязанных элементов без учета временных параметров
{f.h}	Структурно-функциональная синхронная модель есть совокупность взаимосвязанных элементов без учета временных параметров
{h.t}	Структурно временная модель имеет структуру в виде графа элементов, для которых определены временные задержки прохождения сигналов
{f,t,h}	Структурно-функциональная асинхронная модель учитывает все параметры и является наиболее адекватной объекту среди всех упомянутых

Компонент f отображает многообразие аналитических, табличных и графических форм представления моделей. Относительно t классификация моделей определяет синхронные (не учитывающие временные параметры объектов и примитивов), асинхронные (использующие реальные или модельные задержки элементов), дельта-троичные (учитывающие модельные задержки примитивных элементов (ПЭ) и разброс времени переключения входных сигналов объекта) и с нарастающей неопределенностью (момент переключения входных сигналов элементов учитывает разброс параметров задержки прохождения сигналов от внешних входов к выходам). Параметр h классифицирует степень подробности задания структуры и идентифицирует: функциональную или автоматную модель для представления поведения комбинационного или последовательностного ПЭ; структурно-функциональную или итеративную, а также чистую структуру, представляющую собой орграф.

Многообразие математических моделей представлено трехмерным пространством, координаты которого задают соотношение степеней адекватности моделей между собой и объектом. Так, из рис. 9.1 следует, что схемное описание, определяемое точкой M2, менее адекватно объекту по сравнению с его отображением координатой M1.

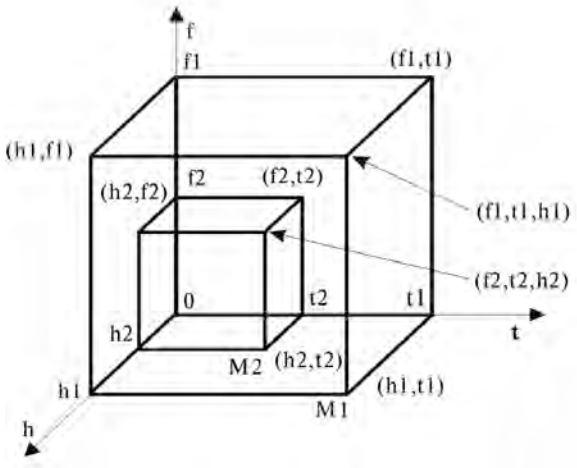


Рис. 9.1. Трехмерное представление моделей

Точность модели увеличивается по мере уменьшения единицы измерения параметров f, t, h. Для функции такая тенденция приводит к использованию дифференциальных уравнений вместо математического аппарата теории переключательных схем в целях описания поведения цифрового объекта на уровне аналоговых сигналов. Трехмерный образ полезен при выборе структур данных с минимальной степенью подробности, отвечающей адекватности поведения объекта для решения конкретной задачи анализа в рамках си-

стемы диагностического обслуживания. Как правило, тема верификации проекта и анализа состязаний рассматривается в рамках самой полной f-t-h-структуры, в то время как генерация проверяющих тестов вполне может обходиться f-h-моделью объекта.

На предложенном множестве структур разработчик операционных средств САПР или систем диагностического обслуживания (СДО) должен выбрать базовую модель, которая отвечает цели исследования по адекватности и обеспечивает приемлемые затраты на ее проектирование и анализ. Исходя из исследований состояния работ в области проектирования различных форм описания поведения цифровых объектов, практического опыта разработки упомянутых систем и ознакомления с техническими характеристиками аналогов можно сделать вывод о целесообразности применения структурно-функциональных (h-f)-моделей вентильного, автоматного, алгоритмического уровней детализации для решения задач: логического синхронного моделирования исправного поведения и неисправностей, генерации статических тестов, проектирования алгоритмов контроля и поиска дефектов. Параметр времени в таких структурах – понятие относительное, определяющее фреймы (такты) автоматных состояний. Длительность каждого модельного такта есть величина, всегда превосходящая максимальную задержку цифровой схемы. При таких условиях адекватность структуры реальному объекту уменьшается по номинальным временным характеристикам, но благодаря этому появляется возможность реального решения задач диагностирования в пространстве <функция, структура, модельное время – последовательность автоматных тактов>. Формализация упомянутых компонентов складывается в систему бинарных отношений примитивных элементов, каждый из которых или все вместе есть автоматная синхронная модель цифровой или микропроцессорной структуры вентильного, функционального и алгоритмического уровней детализации.

Одним из возможных соединений пространства и времени (модельного) может служить формат переменных конечного автомата первого рода в бинарных отношениях двух фреймов:

	Z(t-1)	
X(t)	Z(t)	Y(t)

(9.1)

Это следует из аналитического задания автоматной модели:

$$W=<X, Y, Z, f, g,>,$$

где X, Y, Z – множества входных, выходных и внутренних переменных для кодирования соответствующих состояний автомата; f, g – характеристические функции переходов и выходов, определяющие импликативные отношения на указанных множествах:

$$Z(t)=f[X(t),Z(t-1)]; \; Y(t)=g[X(t),Z(t-1)]. \qquad (9.2)$$

Последним уравнениям соответствует функциональная синхронная модель Хаффмена [1,10], представленная на рис. 9.2.

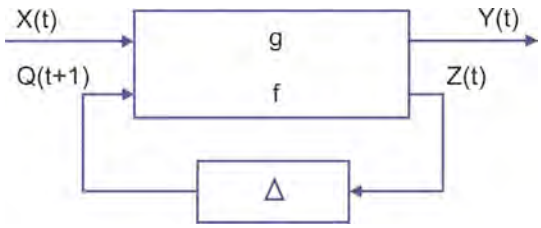


Рис. 9.2. Автоматная модель Хаффмена

Выполнив замену Q(t+1)=Z(t) в выражении (9.2), получим канонические уравнения задания автомата Мили:

$$Q(t+1)=f[X(t),Q(t)];\ Y(t)=g[X(t),Q(t)].$$

Зависимость состояний выходов от значений на входах предопределяет ограниченность использования Мили-структуры для решения задач диагностирования ввиду дополнительного риска возникновения состязаний входных и выходных сигналов. В меньшей степени такой опасности подвержены автоматы Мура, имеющие функции выходов, которые не зависят от состояния входов:

$$Q(t+1)=f[X(t),Q(t)];\ Y(t)=g[Q(t)];$$

благодаря этому автомат будет использоваться в качестве концептуальной или содержательной модели при формировании функционалов – примитивных автоматов. Однако такая структура неудобна для технологии восприятия машиной и человеком из-за различных форматов задания отдельных автоматных переменных: Q задается в двух временных тактах, X и Y – в одном. Следствием этого можно считать неэффективность использования всех переменных X,Q,Y в формировании пространства состояний автомата в двух временных фреймах. Что же можно предложить взамен традиционному формату?

Функции переходов-выходов автомата Мура (Мили, первого рода) будем считать частными случаями вводимого обобщенного модельного (ОМ) автомата [1], задаваемого на формате

X(t-1)	Z(t-1)	Y(t-1)
X(t)	Z(t)	Y(t)

(9.3)

системой отношений

$$Z(t)=f[X(t-1),X(t),Y(t-1),Z(t-1)];$$

$$Y(t)=g[X(t-1),X(t),Z(t-1),Y(t-1)], \qquad (9.4)$$

которой можно поставить в соответствие модифицированную модель Хаффмена, представленную рис. 9.3.

Структура выражений, определяющих поведение ОМ-автомата, позволяет использовать все поля формата (9.3) для расширения пространства состояний и компактной записи таблицы переходов-выходов цифрового объекта. По сравнению с (9.1) для формирования состояний Z(t) и Y(t) дополнительно используются поля X(t-1) и Y(t-1).

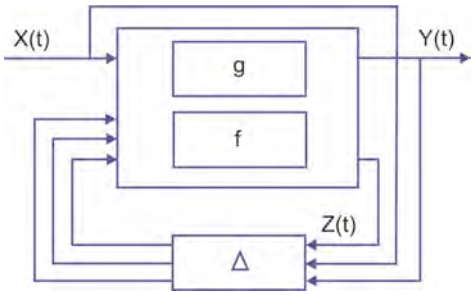


Рис.9.3. Модифицированная модель Хаффмена

Теорема 9.1. *ОМ-автомат не увеличивает количество существенных переменных при записи модели цифрового автомата.*

Во-первых, любое реальное устройство должно соответствовать концепции автомата Мура (Мили, первого или второго рода, или их комбинациям). Однако каждый из упомянутых является подмножеством ОМ-автомата относительно формата входных, внутренних и выходных переменных. Следовательно, не существует устройства, которое имело бы меньшее, чем в соответствующем ему ОМ-автомате, число переменных.

Во-вторых, количество переменных при переходе от произвольной модели к ОМ-структуре может быть уменьшено. Это возможно при использовании дополнительных полей X(t-1) и Y(t-1). При формировании функции выходов Y(t)=g[X(t),Z(t-1)] или системы переходов Z(t)=g[X(t),Z(t-1)] в поле Z может существовать подмножество Z* переменных, непосредственно связанных с полем X, но с задержкой на один такт: Z*(t)=X*(t-1). Поэтому формально и фактически подмножество Z* не является существенным для формирования функций переходов-выходов и его можно исключить из множества Z без ущерба для записи системы переходов. Но в этом случае общее число переменных автомата уменьшится на мощность подмножества Z*. С другой стороны, их добавление в поле X, которое станет двухтактным [X(t-1), X(t)], не увеличит мощности поля X, так как любое подмножество X*(t-1) поглощается полем X(t-1). Поскольку функция выходов в общем случае также зависит от входных линий, то и поле Y может быть уменьшено на мощность подмножества Y*, непосредственно связанного с некоторым подмножеством X*. Аналогично можно использовать поле Y(t-1) для записи переменных Z*, если выполняется Z*(t)=Y*(t-1).

На практике следствия из теоремы 9.1 проявляются в применении дополнительных внутренних переменных Z^* для отображения входного сигнала с задержкой на один автоматный такт $Z^*(t) = X^*(t-1)$, что показано на рис. 9.3. Это означает, что можно выполнить обратный переход, заменив внутреннюю переменную входной, определенной в двух соседних тактах, $X(t-1,t) = Z(t-1,t)$ или выходной $Y(t-1,t) = Z(t-1,t)$.

Например, при описании двухступенчатого DC-триггера с синхронизацией по заднему фронту, задаваемого на векторе существенных переменных (CDYQ) кубами {100S,111S,0XQ0,0XJ1}, перенос переменной первой ступени Y в поле C(t-1) уменьшает формат покрытия (CDQ), что в свою очередь позволяет получить более компактное покрытие {H11, H00, LXS}.

ОМ-автомат есть средство для уменьшения таблиц описания пространства состояний входных, внутренних и выходных линий при максимальном использовании двухфреймового формата автоматных переменных. На практике применение ОМ-автомата дает новые возможности описания устойчивых переходов под управлением фронтальной (передний или задний фронт) синхронизации; записи условий возникновения состязаний по вход-выходным переменным автомата на физически существующих линиях.

9.2. Табличное представление моделей цифровых устройств

Многообразие структур описания поведения конечных автоматов определяется аналитическими, графическими и табличными формами. Первые две – компактны, изобразительны, легко понимаемы человеком, но не технологичны для компьютерного анализа, что требует создания специальных процессоров или мощных программных средств (трансляторов) для машинной интерпретации моделей. Табличная форма является технологичной, универсальной и простой структурой для пользователя и компьютера, не требующей значительных затрат для создания трансляторов, поэтому при проектировании компонентов системы диагностического обслуживания дополнительные ресурсы можно направить на создание новых технологий анализа цифровых объектов.

Естественно, использование других форм описания объектов диагностирования имеет право на существование. Поэтому выбор конкретного математического аппарата для описания функций примитивов в целях последующего решения задач диагностирования есть лишь вопрос вкуса проектировщика и математической моды.

Самой примитивной и универсальной формой является двоичная таблица истинности (переходов), задающая в явном виде импликативные отношения входных, внутренних и выходных булевых переменных на минимально возможном алфавите описания дискретных процессов и явлений, содержащем значения {0,1}:

$$\begin{bmatrix} C_1^{X(t)} C_1^{Z(t-1)} C_1^{Y(t-1)} C_1^{Z(t)} C_1^{Y(t)} \\ C_2^{X(t)} C_2^{Z(t-1)} C_2^{Y(t-1)} C_2^{Z(t)} C_2^{Y(t)} \\ \ldots \\ C_i^{X(t)} C_i^{Z(t-1)} C_i^{Y(t-1)} C_i^{Z(t)} C_i^{Y(t)} \\ \ldots \\ C_q^{X(t)} C_q^{Z(t-1)} C_q^{Y(t-1)} C_q^{Z(t)} C_q^{Y(t)} \end{bmatrix}$$

Существенный недостаток такого описания – большой объем таблиц – может быть сведен к минимуму введением избыточности в алфавит кодирования состояний булевых переменных. Одно из решений предложил Д. Рот [2], создав D-исчисление, в котором неявно присутствовал однотактный четырехзначный алфавит описания автоматных переменных в одном временном фрейме:

$$A^1 = \{0,1, X = \{0,1\}, \varnothing(U)\},$$

где для обозначения высокого импеданса далее вводится символ $Z = \varnothing$.

С позиции теории множеств A^1 есть замкнутый относительно операций пересечения, объединения и дополнения (табл. 9.2) булеан, образованный на универсуме примитивов $\{0,1\}$ взятием множества всех подмножеств. Упомянутые операции можно считать отображением логических функций конъюнкции, дизъюнкции и отрицания в теоретико-множественном алфавите.

Таблица 9.2

Пересечение				
$\cap$	0	1	X	U
0	0	U	0	U
1	U	1	1	U
X	0	1	X	U
U	U	U	U	U

Объединение				
$\cup$	0	1	X	U
0	0	X	X	0
1	X	1	X	1
X	X	X	X	X
U	0	1	X	U

Дополнение				
a	0	1	X	U
$\tilde{a}$	1	0	U	X

Естественно, интерес представляет не сам теоретико-множественный алфавит, известный со времен Г. Кантора и оформленный в основу кубического исчисления (КИ) Р. Миллером [3], но его адаптация к описанию цифровых объектов и вычислительных процессов. Достижения КИ, как математического аппарата геометрического (векторного, табличного) представления булевых функций и методов их анализа в n-мерном векторном пространстве, направлены на обработку моделей комбинационного типа в одном временном такте. Практическое использование однотактного КИ для описания и обработки последовательностных схем является нетехнологичным, по-

скольку применение структуры пространства состояний (9.3) влечет за собой увеличение (почти удвоение) числа модельных переменных, необходимых для кубического задания поведения цифрового устройства в одном временном фрейме, но уже в формате:

X(t-1)	Z(t-1)	Y(t-1)	X(t)	Z(t)	Y(t)

Такая структура приводит к появлению псевдопеременных X(t-1), Y(t-1), Z(t-1), увеличивая объем таблиц и усложняя анализ схемы процедурами двойного преобразования модели цифрового объекта из ее представления в натуральном множестве линий к модельному, где выполняется анализ структуры в целях синтеза диагностической информации, с последующим возвратом к первому для проведения контроля и поиска дефектов.

Отсюда возникает задача преобразования $F^1 \to F^2$ однотактной модели к двухтактной $P(F^1,F^2)=(P^*,P^\wedge,P^\sim)$ на основе использования операторов P* – конкатенации, P^ – минимизации, P~ – поглощения, где система отношений

$$F^1 = \{(t),(X^{t-1}X^tZ^{t-1}Z^tY^{t-1}Y^t),(A^1)\} = \\ = \begin{bmatrix} C^X_{1,t-1}C^Z_{1,t-1}C^Y_{1,t-1}C^X_{1,t}C^Z_{1,t}C^Y_{1,t} \\ C^X_{2,t-1}C^Z_{2,t-1}C^Y_{2,t-1}C^X_{2,t}C^Z_{2,t}C^Y_{2,t} \\ \dots \\ C^X_{i,t-1}C^Z_{i,t-1}C^Y_{i,t-1}C^X_{i,t}C^Z_{i,t}C^Y_{i,t} \\ \dots \\ C^X_{q,t-1}C^Z_{q,t-1}C^Y_{q,t-1}C^X_{q,t}C^Z_{q,t}C^Y_{q,t} \end{bmatrix} \qquad (9.5)$$

представляет общий случай однотактной автоматной модели устройства, заданной кубическим покрытием, которое необходимо привести к виду

$$F^2 = \{(t-1,t),(X,Z,Y),(A^2)\} = \begin{bmatrix} C^X_1C^Z_1C^Y_1 \\ C^X_2C^Z_2C^Y_2 \\ \dots \\ C^X_iC^Z_iC^Y_i \\ \dots \\ C^X_qC^Z_qC^Y_q \end{bmatrix}. \qquad (9.6)$$

Формат модели (9.6) соответствует n-мерному векторному пространству геометрического отображения булевых функций, где каждая точка есть координата n-мерного двоичного вектора, как показано на рис. 9.4.

Введем метрику пространства как $d(C_i, C_j)$, которая определяет расстояние по Хэммингу между двумя заданными векторами (C_i, C_j), точками n-мерного пространства [4, 11], равное числу пустых пересечений между ними $d(C_{it} \underset{t}{\cap} C_{jt}) = \varnothing, t = \overline{1, n}$, с выполнением условий:

1. $d(C_i, C_j) = 0 \leftrightarrow C_i = C_j$.
2. $d(C_i, C_j) = d(Cj, C_i)$.
3. $d(C_i, C_j) + d(C_j, C_r) \geq d(C_i, C_r), \forall i, j, r$.

Следовательно, множество кубов С в соответствии с введенной метрикой формирует метрическое (геометрическое) пространство, которое определяется как булево, где выполняются все основные процедуры анализа КП для проектирования диагностического обеспечения. Куб C_i, имеющий k координат, равных X, называется k-кубом [5]: k-куб представляет собой линию, если k=1; k-куб представляет собой плоскость, если k=2.

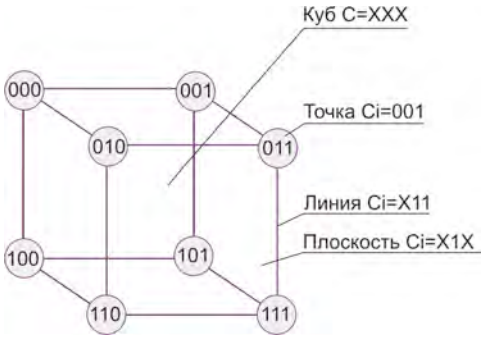


Рис.9.4. Геометрическое представление кубического покрытия

Задача преобразования совокупности однотактных векторов, формирующих кубическое покрытие (КП) F^1-модели, в двухтактное КП F^2-структуры в булевом пространстве рассматривается как трансформация булева пространства, когда дуга, соединяющая две вершины (переход) F1-модели, преобразуется в вершину F^2-модели с помощью оператора конкатенации, как показано на рис. 9.5. После этого возможно свертывание пространства, заключающееся в уменьшении количества простых (двоичных) кубов за счет расширения множества k-векторов при использовании операторов минимизации и поглощения.

Устранение проблемы размерности однотактного КП при описании последовательностных компонентов связано с введением двухтактного кубического исчисления (ДКИ) и соответствующего ему алфавита описания переходов автоматных переменных на естественном формате (9.3) эквипотенциальных

линий, что решает вопросы технологичного и компактного представления и анализа нетривиальных автоматов без введения псевдопеременных. Разработка ДКИ есть результат эволюции трех ветвей проектирования диагностической информации (ДИ): табличных способов представления функций примитивов, методов моделирования неисправностей и исправного поведения синхронных цифровых автоматов, детерминированных алгоритмов генерации тестов на основе построения путей активизации. Каждый из упомянутых компонентов имеет тенденцию к расширению алфавита описания состояний автоматных переменных в двух соседних тактах. Для проектирования моделей это дает возможность формировать компактные КП операционных устройств комбинационного и последовательностного типа, графовых структур управляющих автоматов и микропроцессоров.

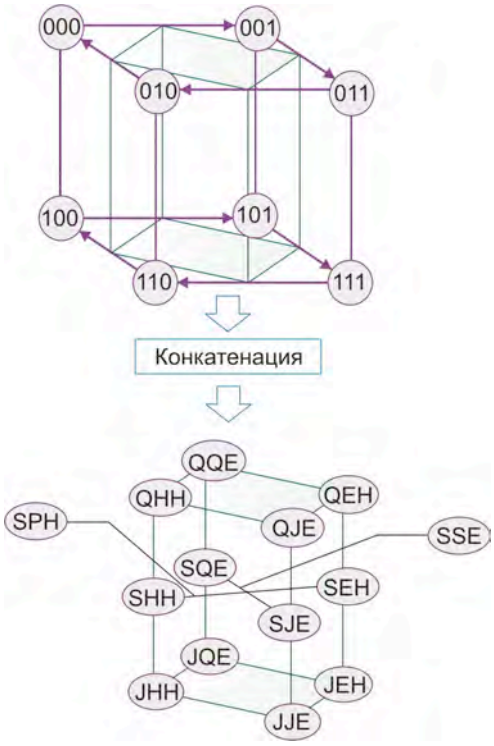


Рис.9.5. Преобразование модели F^1 в F^2

Для алгоритмов моделирования – повышение адекватности анализа логических состояний, технологичность обработки КП на основе процедуры объединения пересечений, использование КП в качестве модели неисправностей для оценки качества тестов. Для тестовой генерации – гарантированная возможность получения проверяющих последовательностей, компактность записи покрытий активизации, значительно уменьшающих время проектиро-

вания теста. Таким образом, ДКИ есть универсальный математический аппарат для описания и анализа операций и структур цифровых и МП-объектов вентильного, функционального и алгоритмического уровней детализации, использующих автоматную концепцию поведения объекта. Его универсальность по отношению к выполнению процедур прямой и обратной импликации не конфликтует с технологичностью восприятия и анализа кубических покрытий компьютером и человеком.

Основным компонентом ДКИ является алфавит A^X [9,3], который задает все возможные переходы двухтактной автоматной переменной в моменты (t-1,t) универсумом примитивов Y={00,01,10,11}. Определение замкнутого относительно теоретико-множественных операций (пересечение, объединение, дополнение) алфавита связано с взятием булеана (множества всех подмножеств) на универсуме Y. Мощность порождаемого таким образом алфавита определяется выражением $Q = 2^n$, где n – количество примитивов в семействе Y. В частности, 16 символов двухтактного алфавита имеют следующую интерпретацию:

A^X = {Q=00,E=01,H=10,J=11,O={Q,H},I={E,J},A={Q,E}, B={H,J}, S={Q,J}, P={E,H},C={E,H,J},F={Q,H,J},L={Q,E,J}, V={Q,E,H},Y={Q,E,H,J},U}.

Символ U обозначает пустое множество $\varnothing$ и служит для замыкания алфавита относительно теоретико-множественных операций для его связи с математическими структурами более высокой иерархии и построения систем доказательств. Для подтверждения замкнутости A^X здесь не приводятся операции пересечения, объединения, дополнения, поскольку данное свойство очевидно по построению.

Имея алфавиты A^1, A^X, определяющие состояния переменных в моменты <t>, <t-1,t>, и в соответствии с принципом симметрии вводится множество A^0={G,T,K={G,T},N}, аналогичное символам A^1, но служащее для описания автоматных переменных в момент <t-1>. Замкнутость алфавита гарантируется выполнением бинарных операций пересечения и объединения, унарной операции дополнения, представленных в табл. 9.3.

Таблица 9.3

Пересечение					Объединение					Дополнение				
∩	G	T	K	U	∪	G	T	K	U	a	G	T	K	U
G	G	U	G	U	G	G	K	K	G	$\tilde{a}$	T	G	U	K
T	U	T	T	U	T	K	T	K	T					
K	G	T	K	U	K	K	K	K	K					
U	U	U	U	U	U	G	T	K	U					

Из трех независимых алфавитов, имеющих различные свойства, создается один универсальный. Объединяющим форматом для них служат два времен-

ных фрейма <t-1,t>. Однотактные символы $a_i^0 \in A^0$, $a_i^1 \in A^1$ к двухтактной форме приводятся с помощью символа Z, который эквивалентен пустому множеству U в одном автоматном такте, но "чуть-чуть” полнее последнего и задает описание переменной в состоянии высокого импеданса или высокого выходного сопротивления. С помощью буквы Z форматы однотактных символов принимают вид: $a_i^0 = (a, Z)$, $a_i^1 = (Z, a)$, где a – однотактные булевы состояния из множества A ={0, 1, X={0,1}, Z}, абстрагированные от времени. При этом символы пустых множеств U,Z,N на двух автоматных тактах определены в следующем виде:

$$Z=UZ;\ N=ZU;\ U=UU;\ UU=ZZ.$$

Состояние ZZ считается неустойчивым пустым множеством, которое на двух автоматных тактах переходит в стабильную «пустоту», определяемую символом U. С учетом приведенных равенств отношения введенных символов пустых множеств регулируются тождествами:

$$Z \cap N = U,\ Z \cup N = ZZ = U,$$
$$Z \cap U = U,\ Z \cup U = Z,$$
$$N \cap U = U,\ N \cup U = N.$$

Взаимодействие других символов на полученном супералфавите $A = \{A^0, A^1, A^X, U\}$ задается теоретико-множественными отношениями пересечения, объединения, дополнения, представленными в табл. 9.4.

Таблица 9.4

Пересечение					Объединение					Дополнение				
$\cap$	A^0	A^1	A^X	U	$\cup$	A^0	A^1	A^X	U	A	A^0	A^1	A^X	U
A^0	A^0	U	A^0	U	A^0	A^0	A^X	A^X	A^0	$\tilde{A}$	A^1	A^0	U	A^X
A^1	U	A^1	A^1	U	A^1	A^X	A^1	A^X	A^1					
A^X	0	A^1	A^X	U	A^X	A^X	A^X	A^X	A^X					
U	U	U	U	U	U	A^0	A^1	A^X	U					

Обобщенные операции полезны для определения аналогичных бинарных и унарных отношений на всех символах алфавита A, представленного табл. 9.5 и рис. 9.6, где показана взаимосвязь одно- и двухтактных символов. Координатные операции пересечения, объединения и дополнения в A-алфавите заданы соответственно в табл. 9.6, 9.7, 9.8.

По определению на каждом множестве A^0, A^1, A^X выполняются все аксиомы и тождества абстрактной математической решетки с дополнениями, та-

кие как ассоциативность, коммутативность, дистрибутивность, идемпотентность, действия с универсумом и пустым множеством, двойного дополнения, правила де-Моргана, поглощения.

Таблица 9.5

Сим-вол	Двухтактная форма	Графовая форма	Комментарии
0	Z0	Z → 0	"Логический 0" в (момент) t
1	Z1	Z → 1	"Логическая 1" в t
X	ZX	0 ← Z → 1	Неопределенность в t
U	UU	U	Неопределяемость
Z(N)	UZ(ZU)	Z N	Полное отключение
G	0Z	0 → Z	"Логический 0" в t-1
T	1Z	1 → Z	"Логическая 1" в t-1
K	XZ	0 → Z ← 1	Неопределенность в t-1
Q	00	↻ 0	Сохранение 0
E	01	0 → 1	Передний фронт
H	10	0 ← 1	Задний фронт
J	11	↻ 1	Сохранение 1
O	X0	↻ 0 ← 1	Установка в 0
I	X1	0 → 1 ↺	Установка в 1
A	0X	↻ 0 → 1	Переход из 0 в X
B	1X	0 ← 1 ↺	Переход из 1 в X
S	{00,11}	↻ 0 1 ↺	Сохранение состояния
P	{01,10}	0 ↔ 1	Изменение состояния
C	{01,10.11}	0 ↔ 1 ↺	Отсутствие сохранения 0
F	{00,10,11}	↻ 0 ← 1 ↺	Отсутствие переднего фронта
L	{00,01,11}	↻ 0 → 1 ↺	Отсутствие заднего фронта
V	{00,01,10}	↻ 0 ↔ 1	Отсутствие сохранения 1
Y	XX	↻ 0 ↔ 1 ↺	Неопределенность в (t-1,t)

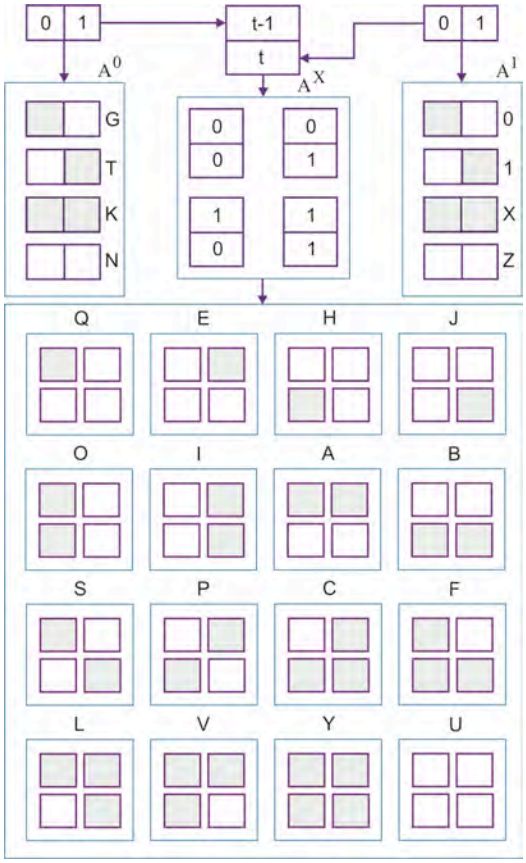


Рис. 9.6. Диаграммы символов А-алфавита

Теорема 9.2. *Супералфавит* A^X, *составленный из замкнутых множеств* A^0, A^1, A^X, *есть замкнутый относительно теоретико-множественных операций алфавит.*

Замкнутость заключается в получении символа $a_i \in \{A^0, A^1, A^X\}$ при выполнении унарной операции дополнения или бинарных: пересечения, объединения. Любой символ $a_i \in A$ в алфавите представлен вектором длиной 2: $a_i = (a^0, a^1)$, где каждая координата определена в алфавите {0,1,X,U,(Z)}. В соответствии с определением операций для A^1-алфавита (см. табл. 9.2) при выполнении координатных отношений пересечения, объединения, дополнения в двухразрядном векторе других символов появиться не может. Тогда для любых подмножеств A^0, A^1, A^X выполнение упомянутых операций в

формате векторов длиной 2 не может дать другого символа, отличного от буквы, принадлежащей к A^0, A^1, A^X, поскольку взаимодействие символов различных подмножеств

$$a_i^0(t-1,t) \in A^0 = (a, Z) = (a * Z);$$

$$a_i^1(t-1,t) \in A^1 = (Z, a) = (Z * a);$$

$$a_i^X(t-1,t) \in A^X = (a, a) = (a * a),$$

где a = {0, 1, X = {0,1}, Z} сводится к координатным операциям над двумерными векторами, которые заданы только символами {0,1,X,U,(Z)}. Следовательно, результат выполнения пересечения, объединения и дополнения не может быть не равен символам из {0,1,X,U,(Z)}. Но все возможные сочетания двух переменных на пяти символах есть операция конкатенации, которая не вносит новых обозначений, что иллюстрируется табл. 9.6. Значит, А-алфавит замкнут относительно теоретико-множественных отношений, что и было показано в табл. 9.4.

Таблица 9.6

∩	0	1	X	U	Z	G	T	K	Q	E	H	J	O	I	A	B	S	P	C	F	L	V	Y
0	0	U	0	U	Z	Z	Z	Z	0	U	0	U	0	U	0	0	0	0	0	0	0	0	0
1	U	1	1	U	Z	Z	Z	Z	U	1	U	1	U	1	1	1	1	1	1	1	1	1	1
X	0	1	X	U	Z	Z	Z	Z	U	1	0	1	0	1	X	X	X	X	X	X	X	X	X
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
Z	Z	Z	Z	U	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
G	Z	Z	Z	U	Z	G	U	G	G	G	U	U	G	G	G	U	G	G	G	G	G	G	G
T	Z	Z	Z	U	Z	U	T	T	U	U	T	T	T	T	U	T	T	T	T	T	T	T	T
K	Z	Z	Z	U	Z	G	T	K	G	G	T	T	K	K	G	T	K	K	K	K	K	K	K
Q	0	U	0	U	Z	G	U	G	Q	U	U	U	Q	U	Q	U	Q	U	U	Q	Q	Q	Q
E	U	1	1	U	Z	G	U	G	U	E	U	U	U	E	E	U	U	E	E	U	E	E	E
H	0	U	0	U	Z	U	T	T	U	U	H	U	H	U	U	H	U	H	H	H	U	H	H
J	U	1	1	U	Z	U	T	T	U	U	U	J	U	J	U	J	J	U	J	J	J	U	J
O	0	U	0	U	Z	G	T	K	Q	U	H	U	O	U	Q	H	Q	H	H	O	Q	O	O
I	U	1	1	U	Z	G	T	K	U	E	U	J	U	I	E	J	J	E	I	J	I	E	I
A	0	1	X	U	Z	G	U	G	Q	E	U	U	Q	E	A	U	Q	E	E	Q	A	A	A
B	0	1	X	U	Z	U	T	I	U	U	H	J	H	J	U	B	J	H	B	B	J	H	B
S	0	1	X	U	Z	G	T	K	O	U	U	J	Q	J	Q	J	S	U	J	S	S	Q	S
P	0	1	X	U	Z	G	T	K	U	E	H	U	H	E	E	H	U	P	P	H	E	P	P
C	0	1	X	U	Z	G	T	K	U	E	H	J	H	I	E	B	J	P	C	B	I	P	C
F	0	1	X	U	Z	G	T	K	Q	U	H	J	O	J	Q	B	S	H	B	F	S	O	F
L	0	1	X	U	Z	G	T	K	Q	E	U	J	Q	I	A	J	S	E	I	S	L	A	L
V	0	1	X	U	Z	G	T	K	Q	E	H	U	O	E	A	H	Q	P	P	O	A	V	V
Y	0	1	X	U	Z	G	T	K	Q	E	H	J	O	I	A	B	S	P	C	F	L	V	Y

Таблица 9.7

∪	0	1	X	U	Z	G	T	K	Q	E	H	J	O	I	A	B	S	P	C	F	L	V	Y
0	0	X	X	0	0	Q	H	O	Q	A	H	B	O	Y	A	B	F	V	Y	F	Y	V	Y
1	X	1	X	1	1	E	J	I	A	E	B	J	Y	I	A	B	L	C	C	Y	L	Y	Y
X	X	X	X	X	X	A	B	Y	A	A	B	B	Y	Y	A	B	Y	Y	Y	Y	Y	Y	Y
U	0	1	X	U	Z	G	T	K	Q	E	H	J	O	I	A	B	S	P	C	F	L	V	Y
Z	0	1	X	Z	Z	G	T	K	Q	E	H	J	O	I	A	B	S	P	C	F	L	V	Y
G	Q	F	A	G	G	G	K	K	Q	E	O	I	O	I	A	Y	L	V	Y	Y	L	V	Y
T	H	J	B	T	T	K	T	K	O	I	O	I	O	I	V	B	F	C	C	F	Y	Y	Y
K	O	I	Y	K	K	K	K	K	O	I	O	I	O	I	Y	Y	Y	Y	Y	Y	Y	Y	Y
Q	Q	A	A	Q	Q	Q	O	O	Q	A	O	S	O	L	A	F	S	V	Y	F	L	V	Y
E	A	E	A	E	E	E	I	I	A	E	P	I	V	I	A	C	L	P	C	Y	L	V	Y
H	H	B	B	H	H	O	H	O	O	P	H	B	O	C	V	B	F	P	C	F	Y	V	Y
J	B	J	B	J	J	I	J	I	S	I	B	J	F	I	L	B	S	C	C	F	L	Y	Y
O	O	Y	Y	O	O	O	O	O	O	V	O	F	O	V	V	F	F	V	Y	F	Y	V	Y
I	Y	I	Y	I	I	I	I	I	L	I	C	I	Y	I	L	C	L	C	C	Y	L	Y	Y
A	A	A	A	A	A	A	V	Y	A	A	V	L	V	L	A	V	L	V	Y	Y	L	V	Y
B	B	B	B	B	B	Y	B	Y	F	C	B	B	F	C	Y	B	F	C	C	F	Y	Y	Y
S	F	L	Y	S	S	L	F	Y	S	L	F	S	F	L	L	F	S	Y	Y	F	L	Y	Y
P	V	C	Y	P	P	V	C	Y	V	P	P	C	V	C	V	C	Y	P	C	Y	Y	V	Y
C	Y	C	Y	C	C	Y	C	Y	Y	C	C	C	Y	C	Y	C	Y	C	C	Y	Y	Y	Y
F	F	Y	Y	F	F	Y	F	Y	F	Y	F	F	F	Y	Y	F	F	Y	Y	F	Y	Y	Y
L	Y	L	Y	L	L	L	Y	Y	L	L	Y	L	Y	L	L	Y	L	Y	Y	Y	L	Y	Y
V	V	Y	Y	V	V	V	Y	Y	V	V	V	Y	V	Y	V	Y	Y	V	Y	Y	Y	V	Y
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Таблица 9.8

a_i	0	1	X	U	Z	G	T	K	Q	E	H	J	O	I	A	B	S	P	C	F	L	V	Y
$\sim a_i^1$	I	O	K	Y	Y	B	A	X	C	F	L	V	I	O	B	A	P	S	Q	E	H	J	U

Следствие 9.1. Для А-алфавита действительны упомянутые законы и тождества абстрактной решетки, что создает обобщенную алгебру множеств того же класса решеток.

Следствие 9.2. На двухтактном формате А-алфавита выполняются тождества с операциями конкатенации, объединения и дополнения:

$$a_i^0(t-1,t) * a_i^1(t-1,t) = a_i^0(t-1,t) \cup a_i^1(t-1,t);$$

$$a_i^0(t-1,t) * a_i^1(t-1,t) \neq a_i^1(t-1,t) \cup a_i^0(t-1,t);$$

$$\overline{a_i}(t-1,t) = \overline{a_i^0}(t-1,t) \cup \overline{a_i^1}(t-1,t),$$

где символом инверсии обозначена операция дополнения.

А-алфавит, теоретико-множественные операции, векторные операции (далее просто операторы) пересечения, объединения, поглощения, минимизации определяют двухтактное кубическое исчисление как математическую структуру для описания и анализа моделей цифровых объектов в целях выполнения прямой и/или обратной импликации:

$$M = < A, \cap, \cup, \sim, *, \#, P^{\cup}, P^{\cap}, P^{\sim}, P\wedge >,$$

где $A = \{A^0, A^1, A^X\}$ – алфавит ДКИ;

$\cap$ – координатная операция пересечения (см. табл. 9.6);

$\cup$ – координатная операция объединения (см. табл. 9.7);

~ – координатная операция дополнения (см. табл. 9.8);

* – координатная операция конкатенации (см. табл. 9.9):

$$a_i(t-1,t) \in A = (a_i^0 * a_i^1);$$

– координатная операция разъединения

$$a_i^1(t) \in A^1 = P^{\#}(a_i),$$

представленная в табл.9.10;

$P^{\cap}$ – векторная операция пересечения:

$$C_i \cap C_t = \varnothing, \leftarrow \exists j (C_{ij} \cap C_{tj}) = \varnothing;$$

$$B = C_i \cap C_t \neq \varnothing, \leftarrow \forall j (C_{ij} \cap C_{tj}) = \varnothing;$$

$P^{\cup}$ – векторная операция объединения:

$$B = C_i \cup C_t, \leftarrow \forall j (B = C_{ij} \cup C_{tj}).$$

$P^{\sim}$ – векторная операция поглощения:

$$C_i \subseteq C_t, \leftarrow \forall j (C_{ij} \cap C_{tj} = C_{ij});$$

P^ – векторная операция минимизации:

$$B = C_i \cup C_t, \leftarrow \exists j (C_{ij} \neq C_{tj}) \& \wedge \exists r (C_{ir} \neq C_{tr}),$$

где (i,t=1,m; j,r=1,n).

Оператор пересечения определяет возможность существования общих для C_i и C_j решений в n-мерном векторном пространстве, что реально существует лишь при отсутствии пустых пересечений по всем координатам векторов:

$$[C_i = (01XEZFLY)] \cap [C_j = (OEKP1SFA)] = [B = (01ZEZSSA)].$$

Таблица 9.9

t-1	t				
*	0	1	X	U	Z
0	Q	E	A	G	G
1	H	J	B	T	T
X	O	I	Y	K	K
U	0	1	X	U	Z
Z	0	1	X	N	U

Таблица 9.10

a_i	0	1	X	U	Z	G	T	K	Q	E	H	J	O	I	A	B	S	P	C	F	L	V	Y
a_i^1	0	1	X	U	Z	Z	Z	Z	0	1	0	1	0	1	X	X	X	X	X	X	X	X	X

Используется при выполнении прямой и обратной импликациях на двух автоматных тактах, составляющих основу анализа модели примитивного элемента в целях определения значений соответственно выходных и входных переменных в условиях частичной неопределенности состояний линий. Пустое пересечение свидетельствует о противоречивости входного слова относительно заданной таблицы переходов, что является следствием неполноты последней или подачи на модель устройства запрещенного набора сигналов. Вычисление оператора основывается на использовании бинарной операции пересечения, представленной в табл. 9.6.

Оператор объединения предназначен для получения общего решения относительно двух или более векторов, используемый при определении состояний выходных линий, если входное слово не противоречит нескольким строкам таблицы переходов:

$$R = B1 \cup B2 = (OIASFGT) \cup (EHPVX10) = (VCVYYEH).$$

Определение конечного результата основывается на последовательном использовании бинарной координатной операции объединения, представленной в табл. 9.7.

Оператор поглощения предназначен для сокращения количества векторов (кубов), описывающих n-мерное векторное пространство состояний входных, внутренних и выходных переменных, исключением таких кубов, которые полностью содержатся в других векторах. Набор A = (EHPV01X) должен быть исключен, если существует куб B = (POYVXIY), пересечение которого с A равно A по всем координатам при использовании правил бинарной операции (см. табл. 9.6).

Оператор минимизации (сограней) есть элементарная процедура итеративного процесса минимизации количества векторов благодаря избыточности алфавита, сводящаяся к выполнению оператора объединения двух и только двух кубов, определенных непустыми значениями, при наличии единственной координаты, по которой они различаются:

$$[A = (EHPSI01)] \cup [B=(EHPSO01)] = [C = (EHPSY01)].$$

Тема минимизации таблицы переходов в однотактном алфавите практически обойдена вниманием исследователей из-за отсутствия положительного эффекта, связанного с уменьшением объемов информации для представления фиксированного пространства состояний. Причиной этого следует считать значительное количество выходов, нагруженных на одни и те же входы. При этом каждый выход формирует свою собственную функцию, зависящую от предыдущего состояния переменных, охваченных глобальными обратными связями, и комбинации входных сигналов. Таким образом, минимизация сильносвязных выходных функций если и возможна теоретически, в реальности дает отрицательный (вследствие дополнительных затрат на выполнение процедуры) эффект. Другой недостаток однотактного алфавита связан с минимизацией таблиц истинности комбинационных функций. Даже при рассмотрении одновыходовых схем существует реальная возможность минимизации только нулевых или только единичных строк. Это связано с выполнением условий для применения оператора минимизации. С одной стороны, два вектора могут быть объединены в один при наличии только одной координаты, по которой они различаются. С другой стороны, в силу свойств дискретного конечного комбинационного автомата различные состояния выхода схемы не могут определяться одним и тем же входным словом. На фоне таких ограничений в минимизации кубических покрытий цифровых автоматов в однотактном алфавите уникальными представляются возможности двухтактного кубического исчисления.

Утверждение 9.1. Процедура минимизации строк таблицы переходов, записанных в двухтактном алфавите, не требует дифференцирования координат на входные, внутренние и выходные.

Во-первых, выходные линии автомата являются и входными условиями для определения следующего состояния, поэтому нет смысла в разделении переменных. Во-вторых, входные линии также способны нести нагрузку в двух автоматных тактах, например при записи фронтального перехода синхронизации. В-третьих, формат переменных ОМ-автомата задает равенство всех существенных (влияющих на состояния выходов) линий относительно двух фреймов, формирующих условия или сам переход.

Теорема 9.3. *Минимизация двухтактных векторов возможна относительно множества невходных координат.*

Равенство всех координат перед процедурой минимизации интересно прежде всего фактом возможности объединения пары строк относительно состояний выходов без потери адекватности модели поведению объекта. Естественно, это имеет место лишь при выполнении условия применимости оператора сограней и является следствием тезиса для последовательностного автомата – на одно входное слово может существовать две или более различные реакции при разных исходных состояниях автомата. Такое утверждение ничего не дает для минимизации таблицы переходов в однотактном алфавите, поскольку она в этом случае есть таблица истинности псевдоком-

бинационного эквивалента устройства, для которого минимизация по выходным координатам невозможна. В двухтактном исчислении такое свойство есть результат маскировки одного из двух отличий пары векторов в двухтактном символе, благодаря неявному присутствию модельного времени. Например, два куба 0101 и 0110 невозможно объединить в один по правилам однотактного кубического исчисления, поскольку они отличаются по двум координатам. Но если первые два символа формируют передний фронт на синхровходе триггера, а вторые два есть изменение логического состояния его выхода, то запись соседних пар переходов символами А-алфавита задает уже двухтактные кубы EE и EH. Для последних выполняются условия применимости оператора сограней, а последующее объединение по правилам табл. 9.7 формирует результат EP. Таким образом, различие в двух координатах исходных однотактных векторов не есть препятствие для минимизации их в двухтактном виде. Разумеется, для объединения координат во времени должны существовать весомые аргументы, каковыми и являются автоматные переходы физически существующих линий на соседних тактах.

Другие векторные операции, используемые далее при анализе функций примитивов, будут введены по ходу изложения конкретных алгоритмов. Что же дает на практике введение столь существенной избыточности в традиционный четверичный алфавит кубического исчисления? Эффективность практического использования А-алфавита для описания автоматных моделей может быть продемонстрирована на примере минимизации табл. 9.11 (столбец А) рекуррентно-ретроспективного покрытия D-триггера, синхронизированного передним фронтом.

Таблица 9.11

	A						B			C			D			E		
	t-1	t	t-1	t	t-1	t												
	C	C	D	D	Q	Q	C	D	Q	C	D	Q	C	D	Q	C	D	Q
1	1	0	1	1	X	1	H	J	I	H	J	I	H	J	I	H	J	I
2	0	0	X	X	1	1	Q	Y	J	L	Y	J	L	Y	S	L	Y	S
3	0	1	X	X	1	1	E	Y	J	H	Q	O	H	Q	O	H	Q	O
4	1	1	X	X	1	1	J	Y	J	L	Y	Q				H	P	X
5	1	0	0	0	X	0	H	Q	O									
6	0	0	X	X	0	0	Q	Y	Q									
7	0	1	X	X	0	0	E	Y	Q									
8	1	1	X	X	0	0	J	Y	Q									

Оно представлено на двух тактах существенных автоматных переменных <C,D,Q> восемью кубами, где строки 1 и 5 – установка соответственно в 0 и 1; наборы 2,3,4,6,7,8 задают функции сохранения состояния триггера при отсутствии заднего фронта на входе синхронизации. Такое покрытие не может быть уменьшено в однотактном кубическом исчислении, поскольку любая пара строк имеет различия не менее чем по двум координатам. Выполним

кодирование переходов булевых переменных символами А-алфавита и запишем модель D-триггера в формате двухтактных автоматных переменных. Для этого воспользуемся формальным преобразованием двух однотактных состояний в одно двухтактное с помощью несимметричной *-операции конкатенации (см. табл. 9.9).

Двухтактное КП с уменьшенным числом переменных представлено столбцом в табл. 9.11. Попытка минимизации кубов столбца В полученного двухтактного покрытия дает положительный результат для наборов: 2,3,4 и 6,7,8, что определяется кубами С. Объединяя в последнем покрытии кубы 2 и 3, также отличающиеся одной переменной, получаем конечный результат, представленный группой кубов D. Данная модель триггера не является адекватной по отношению к переходным процессам, поскольку состояние выхода не определено при наличии заднего фронта на синхровходе с одновременным изменением сигнала на D. Устраняет эту неполноту и делает модель адекватной триггеру куб 4 в покрытии Е.

Алгоритм минимизации таблицы переходов с учетом условий выполнения оператора сограней и теоремы 9.3 таков.

1. Формирования однотактной таблицы переходов в алфавите {0,1,X} на множестве входных, псевдовходных и выходных переменных.

2. Конкатенация состояний переменных, определенных в таблице на двух временных автоматных тактах.

3. Итеративная минимизация числа двухтактных векторов путем объединения пары наборов, различающихся одной координатой. В целях получения минимального результата каждый исходный вектор может участвовать в процедуре неограниченное число раз. Итерации заканчиваются при невозможности на очередном шаге уменьшить количество кубов. Представленные пункты, как и любой алгоритм минимизации, не гарантируют получения минимального множества кубов по причине отсутствия полного перебора, требующего больших временных затрат для достижения такого результата.

9.3. Анализ кубических покрытий

В основу методов анализа цифровых устройств положены процедуры прямой и обратной импликации кубического покрытия $C = \{C_1, C_2, \ldots, C_i, \ldots, C_m\}$ примитива, представляющего собой совокупность кубов, каждый из которых $C_i = \{C_{i1}, C_{i2}, \ldots, C_{ij}, \ldots, C_{in}\}$ задает отношения между входными, внутренними и выходными переменными с помощью символов А-алфавита. Содержательно в качестве КП может выступать таблица истинности комбинационного автомата, таблица переходов последовательностного устройства, любая другая система векторов на конечном множестве двух типов переменных (входных и выходных в частности), определенных А-символикой. Однако для выполнения импликаций не явля-

ется существенным дифференцирование переменных кубического покрытия на три типа, вполне достаточно такого его определения:

$$[(C^X_{1,1}, C^X_{1,2}, \ldots, C^X_{1,j}, \ldots, C^X_{1,k}), \quad (C^Y_{1,1}, C^Y_{1,2}, \ldots, C^Y_{1,t}, \ldots, C^Y_{1,n})];$$

$$[(C^X_{2,1}, C^X_{2,2}, \ldots, C^X_{2,j}, \ldots, C^X_{2,k}), \quad (C^Y_{2,1}, C^Y_{2,2}, \ldots, C^Y_{2,t}, \ldots, C^Y_{2,n})];$$

$$[(C^X_{i,1}, C^X_{i,2}, \ldots, C^X_{i,j}, \ldots, C^X_{i,k}), \quad (C^Y_{i,1}, C^Y_{i,2}, \ldots, C^Y_{i,t}, \ldots, C^Y_{i,n})];$$

$$[(C^X_{m,1}, C^X_{m,2}, \ldots, C^X_{m,j}, \ldots, C^X_{m,k}), \quad (C^Y_{m,1}, C^Y_{m,2}, \ldots, C^Y_{m,t}, \ldots, C^Y_{m,n})],$$

задающего систему отношений в одном временном фрейме в виде функционала $F[(X = X_1, X_2, \ldots, X_i, \ldots, X_k)] \rightarrow [(Y = Y_1, Y_2, \ldots, Y_t, \ldots, Y_n)]$,

где X – множество входных слов (X* – фактическое входное воздействие);

Y – совокупность выходных реакций.

Теорема 9.4. *Для решения задачи прямой импликации* $F(X, Y)|_{X^*} = 1$ *в одном временном фрейме необходимо выполнить процедуру объединения пересечений* $\bigcup_i \forall i (X^* \bigcap_i C_i \neq \varnothing)$.

Если вектор X^* – двоичный, существует только одно входное слово, для которого выполняется условие $X_i = X^*$. Этому условию будет отвечать только один куб $C_i = (C^X_i, C^Y_i)$ в покрытии C в соответствии с определением детерминированного комбинационного автомата – не существует двух одинаковых входных воздействий, дающих различные реакции выходов.

Тогда процедура объединения пересечений получит единственное решение, отвечающее условию X^*, которое определится состоянием выходов куба $C_i = (C^X_i, C^Y_i)$. При этом независимо от значений входных координат C^X_i результат по входам не искажается и будет представлен в формате (X^*, C^Y_i), поскольку при любых значениях координат C^X_i непустое пересечение с двоичным вектором X^* есть X^*.

Если вектор X^* – произвольный троичный входной набор, то при выполнении пересечения $X^* \bigcap_i C_i (i = 1, m)$ может существовать более одного непустого результата. Их объединение по входам не может дать ничего другого, кроме X* в силу тождественности выражения

$$X^* \cap C_i \cup X^* \cap C_j = X^* \cap (C_i \cup C_j)$$

и полноты покрытия С. Если КП неполное, то в X^* может существовать такой двоичный вектор, который не отображен в С. В этом случае по входам примитива будет зафиксировано состояние $X^{**} \neq X^*$ при $X^{**} \subset X^*$. При этом состояния выходов примитива в общем случае будут определяться в троичном алфавите, поскольку значения координат $C_i^Y \cup C_j^Y$ в силу их возможного неравенства могут быть равны X. Случай определения $\bigcup_i \forall_i = (X^* \bigcap_i C_i \neq \varnothing) = \varnothing$ получается при наличии символа $\varnothing$ в исходном наборе X^*, что не имеет смысла, или при отсутствии для слова X^* хотя бы одного куба C_i, для которого $X^* \cap C_i \neq \varnothing$, что свидетельствует о неполноте покрытия.

Следствие 9.3. Получение результата $X^{**} \neq X^*$ или $Y^{**} = \varnothing$ есть свидетельство неполноты покрытия.

Следствие 9.4. При троичном алфавите задания входного слова X^* состояния выходов определены символами $\{0,1,X,\varnothing\}$.

Теорема 9.5. *Использование символа пустого множества Z в дополнение к алфавиту {0,1,X} для определения координат покрытия не противоречит процедуре объединения пересечений.*

Дуализм символа Z проявляется, во-первых, его достаточной полнотой по отношению к символу $\varnothing$, чтобы обеспечить выполнение операций $Z \cap U = U$, $Z \cup U = Z$, во-вторых, он является достаточно пустым по отношению к существенным символам, что подтверждается истинностью условий $Z \cap a_i = Z$, $Z \cup a_i = a_i$. Но если это так, то в силу свойств полноты Z, присущих буквам {0,1,X}, он формально может присутствовать на координатах кубов покрытия С. Благодаря своей пустоте, он должен задавать значения тех и только тех невходных координат $C_i^{Y_j} = Z$, которые на входном слове C_i^X не имеют функциональной зависимости: $C_i^{Y_j} \neq g(C_i^X)$, хотя при этом существует функциональная зависимость $Y_j = g(X)$, но она описывается другим кубом C_t, для которого выполняется равенство: $C_t^{Y_j} = g(C_t^{X_j})$. По отношению присутствия Z на входных координатах $C_t^{X_j} = Z$ оно имеет место, если для всех невходных координат C_i^Y на кубе C_i не существует

функциональной зависимости $C_i^Y \neq g(C_i^{X_j})$, хотя при этом существует функциональная зависимость хотя бы для одного r такого, что $Y_r = g(X_j)$, но она описывается другим кубом C_t, для которого выполняется равенство $C_i^{Y_r} = g(C_t^{X_j})$.

Следствие 9.5. Символ Z в кубе C_i определяет несущественность входной C_{ij}^X и/или выходной C_{ij}^Y координаты при формировании отношения входных и выходных переменных.

Следствие 9.6. Символ Z может быть использован для обозначения высокоимпедансного состояния на выходах функционального элемента.

Следствие 9.7. Символ Z может быть использован для записи системы независимых функций

$$[Y_1 = g(X_1), Y_2 = g(X_2),..., Y_j = g(X_j),..., Y_k = g(X_k)],$$
$$(\forall j, t = 1, k, t \neq j)(X_j \cap X_t = \varnothing):$$

на объединенном множестве существенных переменных: $(X_1, X_2,..., X_j,..., X_k, Y_1, Y_2,..., Y_j,..., Y_k)$ в формате одной таблицы.

Следствие 9.8. Символ Z может быть использован для записи системы частично зависимых функций

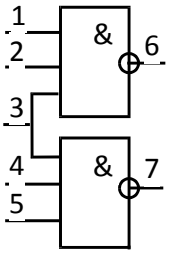
$$Y_1 = g(X_2); Y_2 = g(X_2),..., Y_j = g(X_j),..., Y_k = g(X_k)],$$
$$(\exists j, t, t \neq j)(X_j \cap X_t = \varnothing) \text{ and/or},$$
$$(\exists j, t, t \neq j)(X_j \cap X_t = \varnothing) \text{ and/or},$$
$$(\exists j, t, t \neq j)(X_j \cap X_t = X_j)$$

на объединенном множестве переменных $[(\bigcup_{j=1}^{k} X_j), Y_1, Y_2,..., Y_j,..., Y_k]$ в формате одной таблицы.

Примером записи двух частично зависимых функций может служить кубическое покрытие функциональных элементов, имеющих одну общую входную переменную, что представлено на рис. 9.7.

Процедура объединения пересечений для выполнения прямой импликации в целях доопределения состояний невходных переменных комбинационного автомата в моменте t по известным входным на алфавите {0,1,X,Z,U} состоит из нескольких этапов.

1. Задается исходный вектор моделирования $E = (E_1, E_2, ..., E_j, ..., E_n)$, входные координаты которого принимают значения из {0,1,X}, а все невходные имеют символы X.



КП отношений Yi=F(X)							
Ci	1	2	3	4	5	6	7
1	1	1	1	X	X	0	Z
2	X	X	1	1	1	Z	0
3	0	X	X	X	X	1	Z
4	X	0	X	X	X	1	Z
5	X	X	0	X	X	1	1
6	X	X	X	0	X	Z	1
7	X	X	X	X	0	Z	1

Рис. 9.7. Покрытие частично зависимых функций

Таблица 9.12

A	B	C	S	P
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

2. Выполняется координатное пересечение по правилам табл. 9.6 вектора E с каждым кубом покрытия C.

3. Объединение непустых результатов пересечения по правилам табл.9.7 формирует доопределенный по выходам вектор моделирования.

Примером демонстрации процедуры может служить выполнение прямой импликации для КП одноразрядного сумматора, представленного табл. 9.12.

Пересечение исходного вектора E=X00XX с каждым кубом КП дает положительные результаты: $B_1 = E \cap C_1 = 00000, B_5 = E \cap C_5 = 10010$. Результат будет отрицательным или пустым $C_{ij} \cap E_j = \varnothing$, если хотя бы по одной координате зафиксировано пустое пересечение. Доопределенный вектор моделирования есть объединение непустых пересечений $E = B_1 \cup B_5 = X00X0$.

Для автоматных (не требующих итераций при моделировании) моделей последовательностных элементов процедура объединения пересечений имеет

некоторые отличия, связанные с применением двухтактных символов в КП. С учетом того, что каждый куб C_i покрытия определен на двух тактах $C_i = (C_i^{X(t-1,t)}, C_i^{Y(t-1,t)}) \in C$, задача выполнения прямой импликации уже формулируется в виде

$$F(X(t-1,t), Y(t-1,t))\Big|_{X^*(t-1,t), Y(t-1)} = 1.$$

Отсюда стремление к эквивалентированию систем координат для описания входного воздействия и кубического покрытия. Обычный естественный формат X^* определяется состоянием линий в момент t. Если преобразовывать форму КП к однотактному виду, то возникает несоответствие между физическими и логическими переменными. В противном случае входное воздействие следует представлять в двухтактной форме, путем конкатенации исходного вектора моделирования $(E^{t-1(X)}, E^{t-1(Y)})$ и текущего входного набора $(E^{t(X)}, E^{t(Y^*)})$, где Y^* – поле, априорно задаваемое символами X, которое в процессе импликации подлежит доопределению.

С учетом дополнения процедура объединения пересечений принимает универсальный характер для комбинационных и последовательностных автоматов:

$$P^{\#}[\cup_i \forall_i \{[E^{t-1(X)}, E^{t-1(Y)}) \times (E^{t(X)}, E^{t(Y^*)}] \cap_i C_i \neq \varnothing\}], \qquad (9.7)$$

где оператор $P^{\#}$ – предназначен для разъединения двухтактных символов (табл.9.10) по всей длине вектора E^t в целях определения координат $E^{t(Y)}$, полученных при выполнении импликации: $E_j^t = P^{\#}(E_j^{t-1,t})$.

Таблица 9.13

J	C	K	Y	Q
0	1	0	S	S
0	1	1	0	S
1	1	0	1	S
1	1	1	0	J
1	1	1	1	Q
X	0	X	Q	0
X	0	X	J	1

До начала анализа покрытия выполняется временное сцепление (конкатенация) пары соседних исходных векторов моделирования, а после – их разъединение (операция, обратная конкатенации) в целях получения вектора E^t.

Основные этапы процедуры прямой импликации по двухтактному КП.

1. Выполняется конкатенация по правилам табл.9.9 соседних исходных векторов моделирования $E_j^{t-1,t} = E_j^{t-1} * E_j^t$ при условии, что E^t на невходных координатах имеет символы X, кроме того, первоначальное состояние вектора $E^{t-1}(t=1)$ по всем переменным имеет значения X. Данный пункт позволяет сделать двухтактным формат записи соседних векторов моделирования, что дает возможность выполнять их пересечение с кубами покрытия такой же размерности.

2. Объединение непустых результатов пересечения вектора $E^{t-1,t}$ с каждым кубом покрытия.

3. Выполнение операции разъединения по правилам табл. 9.10 в целях определения однотактного результирующего вектора E^t, невходные координаты которого содержат искомое решение прямой импликации.

Далее приведен пример моделирования JK-триггера, синхронизированного задним фронтом и имеющего две ступени, обозначенные переменными Y, Q, покрытие которого показано в табл. 9.13.

Операция конкатенации $E^0 * E^1$ между начальным состоянием и первым входным набором дает результат:

$$(E0 = XXXXX) * (E^1 = 011XX) = (E^{01} = 011YY).$$

Вектор E^{01} имеет непустое пересечение только с кубом 2:

$$B_2 = E^{01} \cap C_2 = 0110S.$$

Применение процедуры разъединения для B_2 формирует окончательное решение, свидетельствующее об установке первой ступени в нуль: $E^1 = 0110X$. Конкатенация пары $E^1 * E^2$, предшествующая моделированию вектора E^2 и представленная выражением $E^1 = (0110X) * (E^2 = 001XX) =$ $= (E^2 = 001XX) = (E^{12} = QHJAY)$, а также последующее пересечение E^{12} с кубами покрытия дает результат $B^6 = 00100$, что означает установку триггера по переменным Y, Q в нуль: $E^2 = 00100$.

Процедура объединения пересечений для обработки двухтактных кубов покрытия является универсальной в смысле ее применения и для анализа КП комбинационных функционалов, хотя в части выполнения операций конкатенации и разъединения она избыточна для тривиальных автоматов.

Прямая импликация есть процедура одновариантного анализа, поскольку реакция на входное слово в троичном алфавите – всегда единственное решение. Обратная импликация представляет собой процедуру многовариантного

анализа, которая предназначена для определения множества входных последовательностей, удовлетворяющих заданным состояниям выходов. Постановка данной задачи имеет смысл только для явных табличных форм автоматных моделей, когда на основе анализа КП вычисляется совокупность векторов в троичном алфавите. При этом никогда не ставится задача обратной импликации символа X, поскольку решение всегда будет тривиальным и равным полному КП для одновыходовой схемы. Основа процедуры заключается в пересечении исходного вектора импликации с каждым кубом покрытия, после чего на полученных непротиворечивых наборах выполняются операторы минимизации и поглощения в целях уменьшения количества решений, удовлетворяющих исходным состояниям выходов.

9.4. Алгебраическая форма представления графа (АФПГ)

Графовая структура является наиболее удобной формой представления модели сети для пользователя. Естественным представляется сделать такую форму технологичной и для машины. Это означает разработать модель, которая обладала бы следующими свойствами:

– компактность представления информации о графе;

– привязка к распространенному математическому аппарату;

– наличие эффективных методов анализа графовых отношений;

– возможность аналитического описания функций и структур.

Учитывая, что вершины графа и переменные в булевой алгебре связаны между собой системой отношений, воспользуемся аппаратом последней для описания графовых структур [6, 7, 20].

Определение 9.1. Ориентированный граф $G =< H, E >$ есть множество вершин $H = \{H_1, H_2, ..., H_i, ..., H_k\}$, которые связаны между собой отношениями, идентифицируемыми совокупностью дуг $H = \{E_1, E_2, ..., E_r, ..., E_n\}$, где каждая дуга задается конкатенацией двух соединенных между собой вершин и обозначается в виде $E_r = H_i \wedge H_j$ или $H_i \& H_j$, или $H_i \cdot H_j$, или $H_i H_j$.

Определение 9.2. Ориентированная дуга $H_i \& H_j$, связывающая две вершины в графе, где H_i – исток, H_j – сток, называется импликативным отношением.

Определение 9.3. Если α, β – множества вершин или дуг, принадлежащих графу G, то $\alpha \vee \beta$ есть дизъюнкция или объединение этих множеств.

Определение 9.4. Символ вершины $H_i \in H$ есть формула АФПГ.

1) Символ 1 есть обозначение псевдовершины и является также формулой АФПГ.

2) Если α, β – формулы АФПГ, то $\alpha \cdot \beta$ и $\alpha \vee \beta$ также являются формулами.

3) Формулы используют открывающую и закрывающую скобки.

4) Упорядоченная совокупность вершин, соединенная знаками импликации, называется термом АФПГ.

5) Алгебраической формой представления графа являются выражения, построенные в соответствии с указанным выше определением.

Для АФПГ выполняются следующие аксиомы абстрактной математической решетки.

Аксиома 9.1. Коммутативность:

$$\alpha \vee \beta = \beta \vee \alpha. \tag{9.8}$$

Аксиома 9.2. Ассоциативность:

$$\begin{gathered} \alpha \vee (\beta \vee \delta) = (\alpha \vee \beta) \vee \delta; \\ \alpha \wedge (\beta \wedge \delta) = (\alpha \wedge \beta) \wedge \delta. \end{gathered} \tag{9.9}$$

Аксиома 9.3. Дистрибутивность:

$$\begin{gathered} \alpha \wedge (\beta \vee \delta) = (\alpha \wedge \beta) \vee (\alpha \wedge \delta); \\ \alpha \vee (\beta \wedge \delta) = (\alpha \vee \beta) \wedge (\alpha \vee \delta). \end{gathered} \tag{9.10}$$

Аксиома 9.4. Идемпотентность:

$$\alpha \vee \alpha = \alpha. \tag{9.11}$$

Аксиома 9.5. Аксиома о единичном элементе:

$$\begin{gathered} \alpha \vee 1 = \alpha; \\ \alpha \wedge 1 = \alpha. \end{gathered} \tag{9.12}$$

Для целей структурного анализа и синтеза графов полезными являются следующие тождества.

Тождество 9.1. Правило поглощения (минимизации) фрагмента графа. Если есть выражения АФПГ, описывающих два фрагмента графа, то действительными являются следующие уравнения:

$$\begin{gathered} \alpha\beta \vee \beta = \alpha\beta; \\ \alpha\beta \vee \alpha = \alpha\beta. \end{gathered} \tag{9.13}$$

На основании аксиомы 9.3 для первого равенства имеем, что

$$\alpha\beta \vee \beta = (\alpha \vee 1)\beta,$$

а после применения (9.12) получаем конечный и требуемый результат

$$(\alpha \vee 1)\beta = \alpha\beta.$$

Аналогично доказывается и справедливость второго равенства.

Следствие 9.9. Если α, β – вершины графа, то дуга $\alpha\beta$ поглощает любую вершину, входящую в данную дугу.

Тождество 9.2. Правило конкатенации (подстановки). Если α, β, δ есть выражения АФПГ, то действительно следующее уравнения:

$$\alpha\beta \vee \beta\delta = \alpha\beta\delta. \tag{9.14}$$

Пусть есть вершины графа G. Тогда выражение $\alpha\beta \vee \beta\delta$ означает, что существуют направленные дуги, соединяющие вершины α, β и β, δ. Но поскольку вершина β является для первой дуги стоком, а для второй – истоком, то существует единственное решение, когда через вершину β проходит путь $\alpha\beta\delta$.

Тождество 9.3. Правило разложения (декомпозиции). Если α, β, δ есть выражения АФПГ, являющиеся компонентами или буквами терма, то его можно разложить на два терма по любой букве (переменной), входящей исходное выражение:

$$\alpha\beta\delta = \alpha \vee \alpha\beta\delta; \alpha\beta\delta = \alpha\beta \vee \beta\delta; \alpha\beta\delta = \delta \vee \alpha\beta\delta. \tag{9.15}$$

В самом деле, первое и последнее равенства истинны согласно правилу поглощения (9.13). Второе верно вследствие применения правила конкатенации (9.14).

Теорема 9.6. *Если* α, β *есть произвольные формулы АФПГ, то истинным является выражение* $\alpha\beta\alpha = \beta\alpha\beta$.

Д о к а з а т е л ь с т в о. На основании тождества (9.14) имеем $\alpha\beta\alpha = \alpha\beta \vee \beta\alpha$, а с учетом (9.8) получаем $\alpha\beta \vee \beta\alpha = \beta\alpha \vee \alpha\beta$. После этого, используя (9.14), получаем требуемое выражение $\beta\alpha \vee \alpha\beta = \beta\alpha\beta$.

Доказанная теорема определяет необходимые и достаточные условия существования контура в графе, что представляется интересным для решения задач анализа цифровых схем в целях разрыва глобальных обратных связей и последующего ацикливания графовых структур.

Следствие 9.10. Если в терме (пути) существует вершина α, входящая в терм дважды, то все вершины, находящиеся между α и α совместно с вершиной α образуют контур или замкнутый путь.

Определение 9.5. Терм, в котором существуют только две одинаковые вершины α и α и не существует других вершин, называется единичным. Терм, состоящий из вершин, которые образуют контур, называется контурным.

Определение 9.6. Длина контура, задаваемого контурным термом, определяется числом входящих в него букв без одной:

$$L(g_i) = \mathrm{card}(g_i) - 1, \tag{9.16}$$

где $g_i \in G = \{g_1, g_2, ..., g_i, ..., g_n\}$ – контурный терм, входящий а аналитическую запись графовой структуры.

Определение 9.7. Критерий смежности вершин. Неориентированный граф можно представить в виде расширения ориентированного. Основное условие при этом – если две вершины α, β являются смежными, то для ориентированной структуры данное обстоятельство предполагает наличие двух дуг $\alpha\beta \vee \beta\alpha$.

Лемма 9.1. Каждая пара смежных дуг неориентированного графа представлена в АФПГ контуром длины 2.

В самом деле, если вершины α, β смежные, то существует путь (ориентированная дуга) $\alpha\beta$ и наоборот $\beta\alpha$. В этом случае две дуги можно записать в виде одного терма $\alpha\beta \vee \beta\alpha = \beta\alpha \vee \alpha\beta$, что является, по определению 9.7, критерием смежности для неориентированного графа.

Следствие 9.11. Для алгебраической формы представления неориентированного графа выполняется аксиома коммутативности относительно операции конкатенации

$$\alpha \wedge \beta = \beta \wedge \alpha. \tag{9.17}$$

Следствие 9.12. АФПГ есть универсальный математический аппарат для аналитического (компактного) представления графовых структур, включающих фрагменты с ориентированными, неориентированными дугами или без них.

Пример 9.1. Граф, представленный на рис. 9.8 и отображающий топологию звезды, может быть описан с помощью следующего уравнения алгебраической формы представления графа:

$$\begin{aligned}
G &= W_0 \,\&\, M_X \vee W_1 \,\&\, M_X \vee W_2 \,\&\, M_X \vee W_3 \,\&\, M_X \vee \\
&\vee W_4 \,\&\, M_X \vee W_5 \,\&\, M_X \vee W_6 \,\&\, M_X \vee W_7 \,\&\, M_X \vee \\
&\vee M_X \,\&\, W_0 \vee M_X \,\&\, W_1 \vee M_X \,\&\, W_2 \vee M_X \,\&\, W_3 \vee \\
&\vee M_X \,\&\, W_4 \vee M_X \,\&\, W_5 \vee M_X \,\&\, W_6 \vee M_X \,\&\, W_7 = \\
&= (W_0 \vee W_1 \vee W_2 \vee W_3 \vee W_4 \vee W_5 \vee W_6 \vee W_7) \,\&\, M_X \vee \\
&\vee M_X \,\&\, (W_0 \vee W_1 \vee W_2 \vee W_3 \vee W_4 \vee W_5 \vee W_6 \vee W_7).
\end{aligned}$$

Предложенная форма описания графов особенно полезна в случае необходимости описания функций и структур одним математическим аппаратом, для которого разработаны эффективные методы анализа.

АФПГ также представляет удобную форму для решения задачи нахождения минимального покрытия путями всех вершин в ориентированном графе. Например, для структуры, представленной на рис. 9.9, решение такой задачи приводит к следующему конечному результату:

$$G = H_1H_2H_5H_7 \vee H_1H_3H_5H_7 \vee H_1H_3H_6H_7 \vee H_1H_4H_6H_7,$$

который определяет в качестве термов все пути, в данном случае длиной 4, в графе от вершины-истока H_1 к стоку H_7.

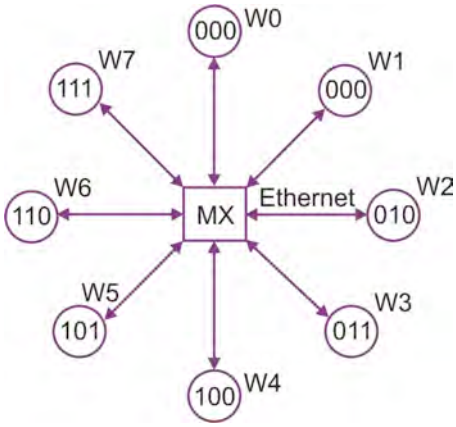


Рис. 9.8. Граф топологии звезды

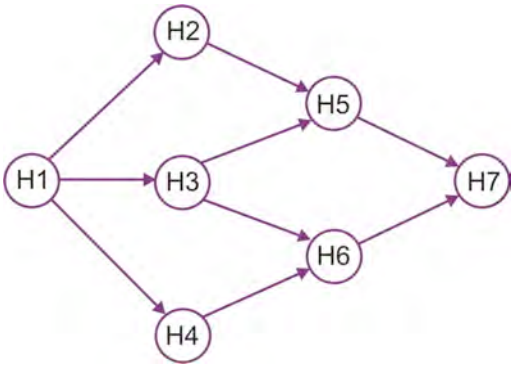


Рис. 9.9. Пример ориентированного графа

9.5. Кубическая форма представления графа

Для задания моделей цифрового устройства (ЦУ) существует три разновидности форм: табличная, аналитическая и графическая [8]. Каждая является универсальной и при наличии соответствующих трансляторов любая их них может трансформироваться в две другие. Какую из них использовать для описания покрытий, функций, графов – вопрос привычки и навыков пользователя. Естественно, табличная форма пригодна не только для описания функций, но и для графовых структур. Традиционные формы задания графов укладываются в матрицы достижимостей, смежностей, в К-формулы, списки, алгебраические выражения [4]. Кубическая форма представления графа (КФПГ) является таблицей, но сокращенной. По сравнению с аналитическими записями анализ кубов более прост для получения решений. Данные об-

стоятельства определяют срединное положение кубического исчисления между крайностями таблиц и аналитических формул, устраняя их недостатки и используя достоинства. Другое важное обстоятельство заключается в применении двухтактного кубического исчисления для описания не только функций, но и структур, поскольку специфика модели объекта диагностического обслуживания основывается на структуре взаимосвязанных функций. Кроме того, для решения отдельных задач анализа существует необходимость преобразования функционального описания в структурный эквивалент, и наоборот. Естественно, что технологичнее всего это выполнять в рамках одного математического аппарата. Помимо компактности КФПГ при записи отдельных типов графов практическая польза ее введения заключается в непосредственном применении автоматных переменных для кодирования вершин-состояний графа. Это, во-первых, соответствует общей концепции уровня детализации описания функций дискретных объектов, во-вторых, позволяет строить тесты для графовых форм задания поведения устройства относительно одиночных константных неисправностей переменных, предназначенных для идентификации вершин, в-третьих, трудно найти более технологичный математический аппарат для прямого преобразования графа переходов в структуру автоматных (булевых) переменных [9].

Любой граф представляет собой совокупность вершин, связанных дугами, иначе, граф – это система бинарных отношений или переходов, которые можно разбить на элементарные связи смежных вершин, одни из которых истоки (исходные состояния), другие – стоки (конечные состояния). В автоматном представлении определение графа есть частный случай, задаваемый выражением

$$W = < X, Y, Z, Z_0, f, g, > \big|_{X=1,\ Y=\varnothing};$$

$$Z(t) = f\,[X(t-1, t), Z(t-1)]\big|_{Z(t)=Hq,\ Z(t-1)=Hp,\ X=1} =$$

$$= [H_q = f\,(H_p) \text{ or } H_p \to H_q].$$

Следуя такому определению, каждый символ двухтактного алфавита (см. табл. 9.5) есть граф, содержащий две вершины, соединенные дугами, число которых в зависимости от символа определяется интервалом значений от 1 до 4. Однако интерес представляет не изоморфизм двухтактных символов элементарным графам, но возможность преобразования любой структуры в кубическое покрытие с последующим анализом ее с помощью методов, ориентированных на обработку ЦУ автоматного уровня.

Процедура проектирования кубической формы представления графа по известной структуре объекта состоит из нескольких этапов.

1. Определение минимального множества разрядов (автоматных переменных) в целях присвоения каждой из N вершин двоичного кода по формуле нахождения целой части числа, округленной в большую сторону: n=]log2 N[.

2. Дуге ставится в соответствие куб КФПГ C_i, получаемый на основе координатной операции конкатенации (см. табл. 9.9) над соответствующими разрядами кодов вершин истока и стока:

$$C_i = C_{pq} = (H_p * H_q)\big|\forall_{p,q\,(Hp \to Hq)}, (i-1, k),$$

где m – число дуг в графе (кубов для обозначения графа).

3. Итеративное применение операторов минимизации

$$C_i^e = C_i^{e-1} \cup C_i^{e-1}\big|\ \exists j\,(C_{ij}^{e-1} \neq C_{tj}^{e-1})\ \&\wedge \exists r\,(C_{ir}^{e-1} \neq C_{tr}^{e-1})$$

(e = 1,s – очередной шаг итерации)

и поглощения

$$C_i^e \subseteq C_t^e\big|\ \forall j\,(C_{ij}^e \cap C_{tj}^e = C_{ij}^e)$$

к исходным кубам покрытия, в котором каждый вектор может участвовать более одного раза.

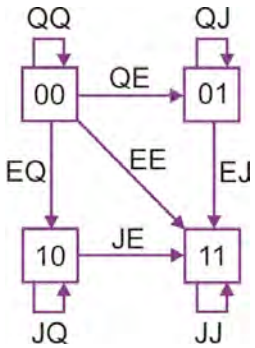
Выполнение процедуры требует затрат, определяемых оценкой:

$$L = k2/2+p^*,$$

где p* – стоимость выполнения операции конкатенации.

Иллюстрацией выполнения приведенных пунктов может служить рис. 9.10, на котором представлен граф и шаги проектирования минимальной КФПГ.

Множество дуг рассматриваемого графа является исходной КФПГ. Последовательное применение трех итераций минимизации кубов дает результат в виде вектора LL. В общем случае число кубов в минимальной форме зависит не только от структуры графа, но и от варианта кодирования вершин. В худшем случае число кубов определяется количеством дуг ориентированного графа. Лучший вариант для конкретной, но не всякой структуры, будет представлен одним двухтактным вектором.



Дуги	Минимизация			
	шаг 1	шаг 2	шаг 3	шаг 4
QQ	QS	AS	AL	LL
QJ				
EJ	ES			
EQ				
QE	AE	AE		
EE				
JQ	JA	JL	JL	
JE				
JJ	JJ			

Рис. 9.10. Проектирование КФПГ

Исходя из условия применения оператора минимизации, обозначение вершин должно подчиняться правилу, обеспечивающему оптимизацию оценки качества кодирования:

$$Q = (\sum_{r-1}^{n} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} M_i^r \# M_j^r / k,$$

где n – число двоичных разрядов для обозначения вершины графа; k- количество переходов; $M_i(M_j)$ – двоичная матрица кодирования i(j)-го перехода, определенная двумя вектор-кодами вершин истока и стока соответственно; $M_i^r(M_j^r)$ – двухразрядный двоичный столбец r матрицы кодирования перехода, который далее будет трансформирован с помощью операции конкатенации в одну букву А-алфавита; # – операция, представленная табл. 9.14 и предназначенная для сравнения двоичных столбцов матрицы кодирования в целях подсчета количества различий в одноименных разрядах для пары матриц.

Таблица 9.14

#	0 0	0 1	1 0	1 1
0 0	0	1	1	1
0 1	1	0	1	1
1 0	1	1	0	1
1 1	1	1	1	0

Каждое совпадение одноименных столбцов различных матриц уменьшает относительную оценку качества кодирования вершин, о чем свидетельствует диагональная строка нулей в #-операции.

Идеальным вариантом является кодирование вершин, приводящее к минимальной оценке кодовых расстояний в обозначении переходов, когда Q=1 свидетельствует об отличии кодов M_i и M_j только в одном разряде, для $i, j = 1, k, i \neq j$. На рис. 9.11 представлены отдельные типы графов с оптимальным кодированием вершин, приводящим к минимальным КФПГ. Для задания переходов неориентированного графа следует иметь в виду наличие двух дуг, определяющих переход от А к В, и наоборот.

Для структуры, обозначенной кубом YE (см. рис.9.11), матрицы кодирования переходов имеют вид:

M1		M2		M3		M4	
0	0	0	0	1	0	1	0
0	1	1	1	0	1	1	1

M1 имеет три различия q=3 от M2,M3,M4 по первому столбцу. Матрица M2 отлична от M3,M4 также на первом столбце, что формирует q2=2, а для M3,M4 получается q3=1. Сумма полученных оценок q=q1+q2+q3=6, деленная на число переходов, определяет качество кодирования Q=1. Следствием этого можно считать получение единственного куба YE, представляющего структуру графа, в результате выполнения пунктов процедуры проектирования КФПГ.

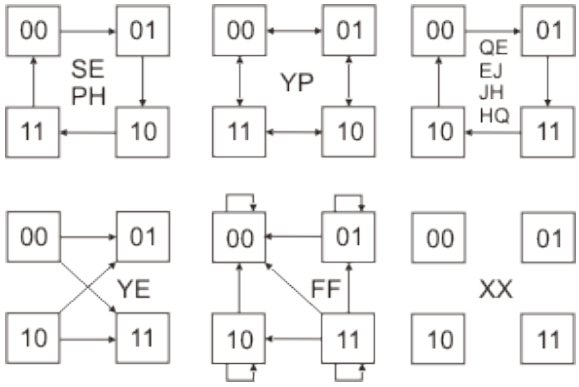


Рис. 9.11. Графы и их КФПГ

При другом обозначении вершин графа, коды которых составляют матрицы

M1		M2		M3		M4	
0	0	0	0	1	1	1	1
0	1	1	0	0	1	1	0

,

оценка качества кодирования переходов Q=2, следствием чего результирующее покрытие не минимизируется и имеет четыре куба:

{QE, EQ, HJ, JH}.

Для экономии временных ресурсов проектирования КФПГ целесообразным представляется априорное вычисление качества кодирования вершин нескольких вариантов с последующим выбором такого, который обеспечивает минимальное значение Q для последующего получения компактного покрытия. Для формализации процедуры проектирования КФПГ полезным представляется доказательство теоремы.

Теорема 9.7. *Подграф, представленный подмножествами истоков*

$$H_p = \{H_{p1}, H_{p2}, ..., H_{pi}, H_{pj}, ..., H_{pn}\}$$

и стоков $H_q = \{H_{q1}, H_{q2}, ..., H_{qt}, H_{q1}, ..., H_{qm}\}$, *в каждом из которых коды вершин отличаются в r-м разряде,* $d(H_{pi}, H_{pj}) = 1, d(H_{qt}, H_{q1}) = 1$, *описывается одним вектором КФПГ.*

Условием применения оператора сограней является различие двухтактных векторов в одном r-м разряде. Поскольку одна координата C_{ir} куба C_i может быть задана четырьмя состояниями, количество наборов или дуг, участвующих в одной итерации минимизации, не может быть больше четырех. С другой стороны, это не противоречит условиям неравенства кодов $H_{p1} \neq H_{q1} \neq H_{p2} \neq H_{q2}$, но только в разряде r, разнообразие которых может сформировать четыре дуги, при этом все остальные разряды вершин должны быть попарно одинаковы: $H_{p1} = H_{p2}, H_{q1} = H_{q2}$. В случае различных кодов H_p и H_q получается четыре вершины, которые при выполнении условий теоремы и при наличии четырех дуг будут представлены одним кубом. Если $H_p = H_q$, то граф содержит всего две вершины, четыре возможные дуги которого сворачиваются также в один двухтактный вектор.

Следствие 9.13. При обозначении вершин графа следует стремиться к выполнению условий минимальности кодовых расстояний между вершинами-истоками и вершинами-стоками.

Двудольная интерпретация графовой структуры ориентирована на представление истоков и стоков множеством кодов. Мощность этого множества кратна степени двойки, среди элементов которого выполняется отметка вершин в соседнем коде.

Следствие 9.14. В ряде случаев более предпочтительной является процедура минимизации или объединения кодов вершин, а затем построение двухтактных кубов, соответствующих дугам. Например, если четыре вершины имеют связь с пятой, то при их соответствующем обозначении кодами 000, 001, 010, 011 и 111 первые четыре объединяются по правилам оператора сограней в одну вершину 0XX. Последующая конкатенация полученного истока 0XX со стоком 111 дает двухтактный куб EII. Аналогично можно получить покрытие сильносвязного графа, число вершин которого кратно степени двойки. Для структуры, имеющей восемь вершин, объединение (последовательное выполнение оператора сограней) всех стоков дает куб XXX, учитывая, что стоки также обозначены вектором XXX, после выполнения операции конкатенации получается двухтактный куб YYY, стоимость получения которого в данном случае значительно ниже традиционной процедуры.

Рекомендации, вытекающие из теоремы 9.7, полезны при идентификации вершин, если их коды не были определены условиями функционирования объекта или модели.

1. Любой сильносвязный граф с числом вершин, кратным степени двойки (N=2n), при любом способе их кодирования имеет КФПГ, содержащую один куб YYY...Y, где количество символов равно n.

2. Последовательное соединение вершин типа "цепь" (даже если она замкнутая) кодируется двоичными числами в порядке их возрастания.

3. Структура типа «звезда», расходящаяся (сходящаяся) и имеющая одну вершину-исток (сток), соединенную со всеми остальными, число которых кратно степени двойки, при обозначении центральной кодом 000...0, а других в произвольном порядке, будет иметь единственный куб покрытия, равный QAA...A (QOO...O). Для девятиэлементного графа таким кубом является QAAA (QOOO).

4. Связные вершины двудольного графа кодируются четными и нечетными двоичными числами, что для пар вершин, кратных степени двойки может привести к кубическому покрытию, состоящему из одного куба. Для четырех пар, представленных системой переходов

000–001, 010–011, 100–101, 110–111,

конкатенация по соответствующим разрядам определяет исходные двухтактные кубы QQE, QJE, JQE, JJE, последующая минимизация которых дает результат SSE.

5. Для произвольной структуры отметка начинается с определения количества разрядов n для кодирования N вершин, удовлетворяющего неравенству $N \le 2^n$. Затем выполняется итеративная процедура определения вершины H_i с максимальной степенью полуисхода $S_0(H_i)$ (полузахода $S_0(H_i)$) еще не отмеченных стоков (истоков). Последние кодируются в порядке возрастания двоичных чисел подмножествами кодов, кратных степени двойки. Такими являются: (0,1); (00,01,10,11); (000, 001, 010 ,..., 111). При равенстве полустепеней следует просчитывать варианты кодирования вершин вычислением оценки качества Q с последующим выбором такого решения, которое имеет минимальный критерий. Процедура не гарантирует построения оптимального покрытия, которое может быть получено с использованием эвристических приемов.

Иллюстрацией обобщенной процедуры проектирования КФПГ, представленной пунктом 5 рекомендаций, может служить построение покрытия для графа, у которого первые четыре вершины, являясь предшественниками, соединены с пятой, имеющей в свою очередь четырех преемников. Для такой структуры на первом шаге итерации предшественники кодируются наборами: 0000, 0001, 0010, 0011. На втором выполняется обозначение преемников кодами: 1000, 1001, 1010, 1011. На третьем шаге обозначается оставшаяся вершина вектором 1111, для которой, например, степени полуисходов и полузаходов еще не обозначенных вершин равны нулю. Полученные пары двоичных векторов, задающие все дуги графа, имеют вид 0000–1111, 0001–1111, 0010–1111, 0011–1111, 1111–1000, 1111–1001, 1111–1010, 1111–1011.

Конкатенация приведенных пар определяет результат: EEEE, EEEJ, EEJE, EEJJ, JHHH, JHHJ, JHJH, JHJJ, минимизация которого приводит к покрытию: {EEII, JHBB}.

Для данного примера аналогичная кубическая форма может быть получена при использовании процедуры, определенной следствием 9.14 теоремы 9.7. Объединение вершин дает два вектора, 00XX и 10XX, конкатенация которых с вершиной 1111 формирует кубы (00XX)*(1111)=(EEII); (1111)*(10XX)=(JHBB).

Таким образом, двухтактное кубическое исчисление является универсальным математическим аппаратом технологичного для пользователя и компьютера описания и анализа функций и структур цифровых и микропроцессорных объектов.

ДКИ ориентировано на адекватное представление поведения широкого класса объектов вентильного, функционального и алгоритмического уровней детализации на множестве автоматных переменных, заданных в двух временных модельных фреймах.

Предлагаемый математический аппарат позволяет решать задачи автоматизированного проектирования моделей цифровых и микропроцессорных структур; создания компактных кубических покрытий комбинационных, последовательностных схем и графов, выполнения прямой и обратной импликаций, являющихся оригинальными и базовыми процедурами при решении задач проектирования диагностической информации.

9.6. Топологическое представление символов А-алфавита

В рамках системы координат, задающей отображение временной оси и уровней потенциалов для задания логических значений, символы алфавита двухтактного кубического исчисления могут иметь планарную геометрическую интерпретацию, которая изображена на рис. 9.12.

Для автоматного задания состояний переходов переменной (линии) рассматриваются такты t-1, t, в которых значения сигналов могут быть определены примитивами {0,1}. Координатная ось "u" разделяет топологический шаблон на форматы изображения символов A^0- и A^1-алфавитов. При этом символ Z (N) есть полуось t (t-1) пустого (вакантного) знакоместа, в котором может быть записан любой существенный символ {0,1,X} ({G,T,K}). Следовательно, можно сказать, что любой символ А-алфавита, кроме U, содержит символы <N,Z>. При условии, что U – подмножество из пяти мнимых вершин, не имеющее полуоси для размещения на ней знака, а символ Z (N) есть линия, то их существенное отличие заключается в результатах выполнения операций пересечения и объединения между точками и линией: $U \cap Z(N) = U, U \cup Z(N) = Z(N)$. Объединение или конкатенация символов $Z \cup (*)N = U$ дает пустое множество, что соответствует ранее приведенным тождествам.

Символы первых двух строчек (см. рис. 9.12) составлены из точечных координат шаблона, что физически определяет состояние линии комбинационной схемы. Остальные четыре строки задают все возможные системы переходов

на шаблоне из четырех вершин, поэтому соответствующие им символы состоят из линий. Пересечение пары символов может дать линию, т. е. существенный символ А-алфавита, или только совокупность точек (от 1 до 5) шаблона, что определяет пустое пересечение U.

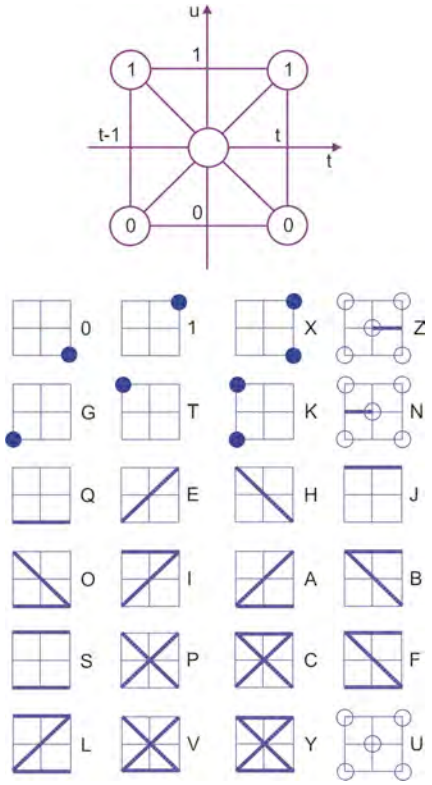


Рис. 9.12. Топология символов А-алфавита

Топологическое или геометрическое толкование символов алфавита двухтактного кубического исчисления (идея бывшего студента С. Рымара) достаточно наглядно иллюстрирует физику статических и переходных состояний, имеющих место в моделях поведения цифровых объектов. Использование показанного на рис. 9.12 шаблона или знакоместа дает возможность создать топологические процессоры моделирования, генерации тестов, контроля и поиска дефектов с непосредственным вычислением состояний линий на экране монитора, применяя для этого команды формирования изображения с помощью видеоконтроллера. Технологичность логического анализа цифровых структур определяется отсутствием знания теоретико-множественных операций. Взамен достаточно лишь помнить, что пересечение геометрических образов есть их общая часть, а объединение – все компоненты, принадлежащие той или другой фигуре. Практическое применение геометрических образов букв А-алфавита ориентировано на построение новых эффективных

средств моделирования вычислительных процессов на основе использования геометрических или видеопроцессоров.

9.7. Выводы

Представлена теория компактного описания моделей цифровых устройств, инвариантных по отношению к функциям и структурам, которые задаются в форме таблиц, универсальных относительно решения задач анализа, включающих выполнение прямой и обратной импликаций, являющихся базовыми процедурами в методах моделирования, синтеза тестов и диагностирования дефектов цифровых систем.

Приведены доказательные примеры, иллюстрирующие эффективность алгебр и структур для аналитического (табличного) описания и логического анализа моделей цифровых систем и компонентов.

Показаны различные области применения теории компактного описания функций и структур в виде кубических покрытий (однотактных и двухтактных), КФПГ и АФПГ для решения задач прямой и обратной импликации на графовых и функциональных моделях.

Представлена оценка адекватности создаваемых моделей функционированию реальных объектов, учитывающая три компонента: структурной, временной и функциональной точности, которая позволяет сравнивать существующие и новые модели для решения задач проектирования и верификации.

9.8. Литература

1. Хаханов В.И. Техническая диагностика цифровых и микропроцессорных структур.- К.: ИСИО, 1995. – 242 с.
2. Чжен Г., Мэннинг Е., Метц Г. Диагностика отказов цифровых вычислительных систем.– М.: Мир, 1972. – 230 с.
3. Миллер Р.Е. Теория переключательных схем. Т.1. Комбинационные схемы/ Под ред. П.П. Пархоменко: Пер с англ.– М.: Наука, 1970. – 416с.
4. Горбатов В.А. Основы дискретной математики.– М.: Высш. шк., 1986. – 311с.
5. Баранов С.И., Майоров С.А., Сахаров Ю.П., Селютин В.А. Автоматизация проектирования цифровых устройств.- Л.: Судостроение, 1979. – 264 с.
6. Элементы анализа моделей дискретных устройств.– Харьков: Харьковский институт радиоэлектроники.– Деп в УкрНИИНТИ 05.02.85, № 219Ук-85Деп.– 1984.– 18с.
7. Воробьев В.Н. Об алгебраическом методе описания графов. Известия АН СССР. –1984. – Т.117. – С.89-93.
8. Закревский А.Д. Графическое отображение комбинационных схем// Автоматика и вычислительная техника.– 1990. – № 6. – С.59-65.
9. А. Д. Закревский, Н. Р. Торопов. Минимизация булевых функций многих переменных в классе ДНФ – итеративный метод и программная реализация // Прикладная дискретная математика. – 2009. – № 1. – С. 5–14.

10. Основи цифрових систем / За ред. М. П. Благодарного, В. С. Харченка. – Підручник. Міносвіти і науки України. –2002. – 672с.
11. Чумаченко С.В. Суммирование избранных рядов при моделировании радиоэлектронных устройств.– Харьков.– ХНУРЭ.– 2005.– 174 с.
12. Хаханов В.И. Двухтактное кубическое исчисление. I. Аппарат математического моделирования вычислительных устройств // АСУ и приборы автоматики. Х.: ХТУРЭ. – 1997. – Вып. 105. – С. 35-43.
13. Хаханов В.И., Кривуля Г.Ф., Монжаренко И.В. Кубическое исчисление для анализа автоматов. Информатика, кибернетика и вычислительная техника. – Донецк.: ДонГУ, 1997. – Вып. 1. – С. 159-164.
14. Hahanov V., Krivoulja G., Monzharenko I. Two-Frames Cubical Calculus for Modeling and Simulation of Digital Circuits. 4-th int. Works of Workshop "Mixed Design of Integrated Circuits and System". Poznan, Poland. 12-14 June 1997. – P.195-199.
15. Hahanov V., Krivoulja G., Al-Matarnah R. J. Multivalued Models of Digital Circuits. Theses of Conf. "Mathematical Modeling and Information Technologies MMIT-97". Belgorod, Russija. 6-9 October 1997. – P. 15-16.
16. Хаханов В.И. Техническая диагностика элементов и узлов персональных компьютеров. – К.: ІЗМН. 1997. – 308 с.
17. Хаханов В.И., Монжаренко И.В., Бережная М.А. Двухтактное кубическое исчисление. II. Анализ моделей цифровых устройств // АСУ и приборы автоматики. – Х.: ХТУРЭ, 1997. – Вып. 106. – С. 93-105.
18. Контроль и диагностика вычислительных устройств и систем / Хаханов В.И., Кривуля Г.Ф., Рысованый А.Н., Монжаренко И.В./ Под ред. В.И. Хаханова. – Харьков: ХВУ, 1997. – 304 с.
19. Бондаренко М.Ф., Кривуля Г.Ф., Рябцев В.Г., Фрадков С.А., Хаханов В.И. Проектирование и диагностика компьютерных систем и сетей. – К.: НМЦ ВО. – 2000. – 306 с.
20. Хаханов В.И., Абу Занунех И.М. Халиль, Егоров А.А. Булева форма графовых моделей для описания цифровых систем и сетей // АСУ и приборы автоматики.– 2000.– Вып. 113.– С. 142-148.
21. Хаханов В.И., Чумаченко С.В. Модели пространств в научных исследованиях// Радиоэлектроника и информатика.– 2002.– №1.– С. 124-132.
22. Семенец В.В., Хаханова И.В., Хаханов В.И. Проектирование цифровых систем с использованием языка VHDL.– Харьков: ХНУРЭ.– 2003.– 492 с.
23. Хаханов В.И., Чумаченко С.В. Метод суммирования рядов в гильбертовом пространстве с воспроизводящим ядром для анализа моделей радиоэлектронных устройств //Электрон. моделирование. – 2006. – № 4. – С. 1-15.
24. Хаханов В.И., Хаханова И.В. VHDL+Verilog = синтез за минуты. – Харьков: ХНУРЭ.– 2006.– 264с.

10. СИНТЕЗ ТЕСТОВ ДЛЯ КОМПОНЕНТОВ ЦИФРОВЫХ СИСТЕМ

Проблема генерации тестов для цифровых систем всегда находится в зоне повышенного внимания ученых и практиков в области создания эффективных вычислительных устройств. Современные исследования, касающиеся синтеза и верификации тестов, представлены работами [1 – 20].

Предлагается стратегия генерации тестов для структурно-функциональных моделей цифровых устройств на основе активизации одномерных логических путей при использовании кубических покрытий (КП) функциональных элементов. Алгоритмы, основанные на стратегии, позволяют проектировать проверяющие тесты для всех неисправностей существенных линий комбинационных и последовательностных схем, полученных из систем булевых уравнений, описанных на HDL-языках.

10.1. Объект тестирования

Объект тестирования представлен в виде булевых уравнений, записанных в BNF-формате, который, в свою очередь, является подмножеством языков VHDL, Verilog. Примеры описания последовательностных примитивов представлены в виде элементов (Latch, Flip-Flop).

Пример 10.1. Templates for Latch elements:

— latch (c_low, D)

S=D*!C; R=!C*!D; Q(t)=S+(!R*Q(t-1));

— latch (c_high, D)

S=D*C; R=C*!D; Q(t)=S+(!R*Q(t-1));

— latch (R,S,c_high,D)

S1=S+(D*C); R1=R+(C*!D); Q(t)=S1+(!R1*Q(t-1));

— latch (R,S, CE, c_low,D)

C=CLK*CE; S1=S+(D*!C); R1=R+(!C*!D); Q(t)=S1+(!R1*Q(t-1));

— latch (R,S, CE, c_high,D)

C=CLK*CE; S1=S+(D*C); R1=R+(C*!D); Q(t)=S1+(!R1*Q(t-1));

Пример 10.2. Templates for FF elements:

—flip-flop(C_low,D)

S=!C*Q1(t-1); R=!C*!Q1(t-1); Q(t)=S+(!R*Q(t-1)); S1=D*C; R1=C*!D; Q1(t)=S1+(!R1*Q1(t-1));

—flip-flop (C_high,D)

S=C*Q1(t-1); R=C*!Q1(t-1); Q(t)=S+(!R*Q(t-1)); S1=D*!C; R1=!C*!D; Q1(t)=S1+(!R1*Q1(t-1));

—flip-flop(CE, C_low, D)

C=CLK*CE; S=!C*Q1(t-1); R=!C*!Q1(t-1); Q(t)=S+(!R*Q(t-1)); S1=D*C; R1=C*!D; Q1(t)=S1+(!R1*Q1(t-1));

—flip-flop (CE, C_high, D)

C=CLK*CE; S=C*Q1(t-1); R=C*!Q1(t-1); Q(t)=S+(!R*Q(t-1)); S1=D*!C; R1=!C*!D; Q1(t)=S1+(!R1*Q1(t-1));

—flip-flop(CE, S,C_high,R,D)

C=CLK*CE; S1=S+(C*Q1(t-1)); R1=C*!Q1(t-1); Q(t)=S1+(!R1*Q(t-1)); S2=S+(D*!C); R2=(!C*R)+(!D*!C);

Q1(t)=S2+(!R2*Q1(t-1));

Здесь применяется примитивный синтаксис описания булевых выражений, использующий знаки “=”, “(”, “)”, а также следующие конструкции:

$$(* \approx \wedge);(+ \approx \vee);(! \approx \neg).$$

Булевы уравнения преобразуются в структурно-функциональные модели примитивных комбинационных и последовательностных элементов, описанных кубическими покрытиями. Триггерные структуры формализуются к виду одно- и двухступенчатых примитивных автоматов, показанных на рис. 10.1.

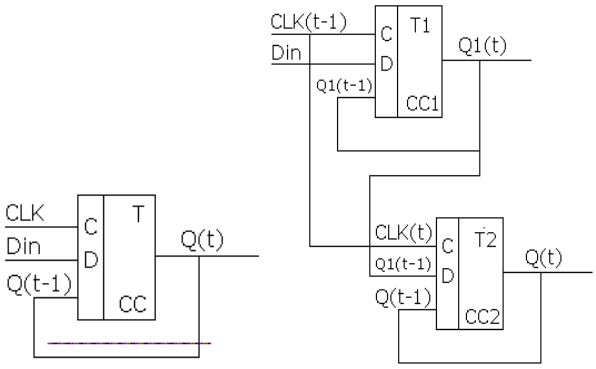
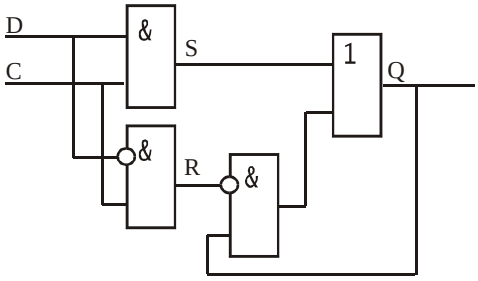


Рис. 10.1. Одно- и двухступенчатая триггерные структуры

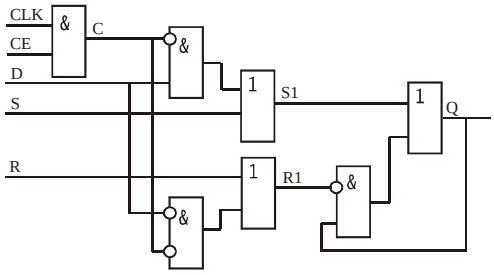
Для примитивных последовательностных элементов строятся кубические покрытия, необходимые для построения проверяющих тестов и моделирования неисправностей [21-23]. Примеры моделей примитивов в последовательности (структура, двухтактное, однотактное покрытие) представлены на рис. 10.2 - 10.5.



C	D	Q
1	0	O
1	1	I
0	X	S

C	D	Q_{t-1}	Q_t
1	0	X	0
1	1	X	1
0	X	0	0
0	X	1	1

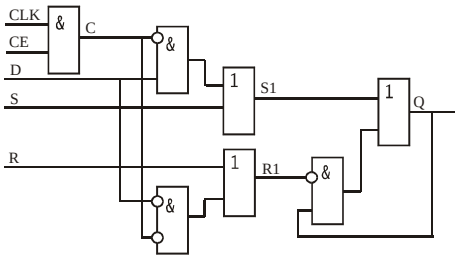
Рис. 10.2. Модель защелки — latch (c_high, D)



S	CE	CLK	R	D	Q
1	X	X	X	X	I
0	0	X	X	0	O
0	0	X	X	1	I
0	1	0	X	0	O
0	1	0	X	1	I
0	1	1	1	X	O
0	1	1	0	X	S

S	CE	CLK	R	D	Q_{t-1}	Q_t
1	X	X	X	X	X	1
0	0	X	X	0	X	0
0	0	X	X	1	X	1
0	1	0	X	0	X	0
0	1	0	X	1	X	1
0	1	1	1	X	X	0
0	1	1	0	X	0	0
0	1	1	0	X	1	1

Рис. 10.3. Модель защелки – latch (R,S, CE, c_low,D)



S	CE	CLK	R	D	Q
1	X	X	X	X	I
0	0	X	X	0	O
0	0	X	X	1	I
0	1	0	X	0	O
0	1	0	X	1	I
0	1	1	1	X	O
0	1	1	0	X	S

S	CE	CLK	R	D	Q_{t-1}	Q_t
1	X	X	X	X	X	1
0	0	X	X	0	X	0
0	0	X	X	1	X	1
0	1	0	X	0	X	0
0	1	0	X	1	X	1
0	1	1	1	X	X	0
0	1	1	0	X	0	0
0	1	1	0	X	1	1

Рис. 10.4. Модель триггера – flip-flop(CE, C_low, D)

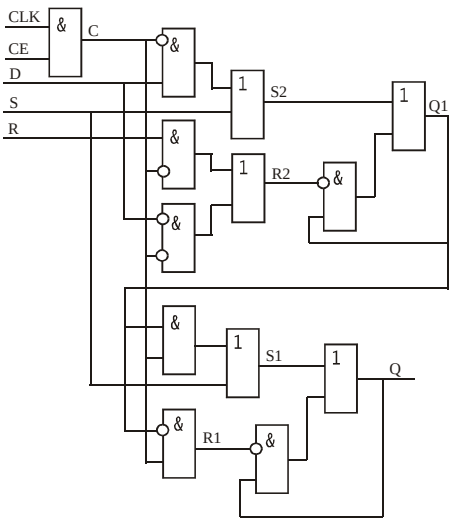
Двухтактное покрытие есть сокращенная запись таблицы переходов на векторе всех физически существующих неизбыточных (входных, внутренних, выходных) переменных при использовании символов алфавита кубического исчисления, задающих поведение автоматных линий в двух временных фреймах.

Что касается триггеров и защелок, то использование двухтактных символов предназначено для обозначения переходов (сохранения состояния) выходной (внутренней) переменной.

Однотактное покрытие есть сокращенная запись таблицы истинности (переходов) на множестве существенных (входных, псевдовходных, внутренних, выходных) переменных при использовании алфавита

$$A^1 = \{0,1,X,\varnothing\}.$$

Однотактные покрытия триггеров имеют большее число переменных, но при описании их состояний алфавитом меньшей мощности.



CLK	CE	S	R	D	Q1	Q
0	X	0	X	0	O	S
X	0	0	X	0	O	S
0	X	0	X	1	I	S
X	0	0	X	1	I	S
1	1	0	X	X	Q	O
1	1	0	X	X	J	I
X	X	1	X	X	I	I

CE	CLK_{t-1}	CLK_t	S	R	D	Q_{t-1}	Q_t
0	X	X	0	X	X	0	0
0	X	X	0	X	X	1	1
X	X	X	1	X	X	X	1
1	0	1	0	X	0	X	0
1	0	1	0	X	1	X	1
1	X	0	0	X	X	0	0
1	X	0	0	X	X	1	1
1	1	X	0	X	X	0	0
1	1	X	0	X	X	1	1

Рис. 10.5. Модель защелки – flip-flop(CE, S,C_high,R,D)

При проектировании цифровых систем на основе программируемых логических интегральных схем используется номенклатура компонентов, включающая около 40 типов триггеров и защелок, применяемых в схемах в качестве элементов памяти.

10.2. Формализация задач тестирования цифровых систем

Регулярные методы генерации тестов ориентированы на проверку неисправностей наперед заданного класса. Чаще всего это – одиночные константные неисправности (ОКН). Известные алгоритмы построения тестов [24,25] имеют определенные недостатки, связанные с формализацией выполнения процедур прямой и обратной импликаций, что в общем случае не гарантировало получение полного проверяющего теста относительно ОКН для произвольных цифровых структур. Многочисленные научные исследования, обобщенные в [24-26], посвящены решению задач проектирования проверяющих последовательностей для схем со сходящимися разветвлениями, анализ которых традиционно считается трудноформализуемым. Предлагается метод двухфреймовой активизации одномерных логических путей и его теоретическое обоснование в двухтактном алфавите кубического исчисления с последующим применением алгебры Кантора для метода активизации. Ниже представлена задача генерации тестов как одна из проблем логического анализа цифровых схем.

Моделирование – процесс определения неизвестных компонентов тетрады <M,D,T,R>: <модель, дефекты, стимулы, реакция> на основе применения правил логического вывода.

Обобщенное уравнение анализа модели цифрового изделия при наличии или отсутствии неисправностей

$$R^*(T,M,D) \cap R(T,M) = \varnothing$$

формализует постановку трех следующих задач.

1. Определение модели исправного поведения устройства в виде совокупности всех его возможных реакций на функциональные входные наборы:

$$M = f^1(T,D,R)\big|_{D=\varnothing,T=\varnothing}.$$

2. Проектирование теста (проверяющего, диагностирования) цифрового изделия по его модели для заданных неисправностей:

$$T = f^2(M,D,R)\big|_{R=\varnothing}.$$

3. Моделирование одиночных константных неисправностей цифрового изделия на входных наборах в целях определения качества теста:

$$D = f^3(M,T,R).$$

Далее рассматриваются задачи, сформулированные в пунктах 1 и 2. Решение проблемы, связанной с пунктом 3, необходимо лишь для верификации генерируемых тестов цифровой системы.

10.3. Аппарат двухтактного кубического исчисления

Для задания моделей функциональных элементов и их анализа будем использовать их табличное представление, как наиболее технологичную форму восприятия информации пользователем и компьютером, которая основывается на кубическом исчислении – математическом аппарате компактного описания цифровых структур.

Основной недостаток табличного представления конечных автоматов – размерность – может быть сведен к минимуму введением избыточности в алфавит кодирования состояний булевых переменных. Алгебра Кантора есть алфавит описания автоматных переменных в одном временном фрейме:

$$A^1 = \{0,1, X = \{0,1\}, \varnothing\}.$$

Основным компонентом двухтактного кубического исчисления (ДКИ) является многозначный алфавит [3]: Ax={Q=00, E=01, H=10, J=11, O={Q,H}, I={E,J}, A={Q,E}, B={H,J}, S={Q,J}, P={E,H}, C={E,H,J}, F={Q,H,J}, L={Q,E,J}, V={Q,E,H}, Y={Q,E,H,J}, U}, который задает все возможные переходы двухтактной автоматной переменной в моменты (t-1,t) универсумом примитивов Y={00,01,10,11}. Символ U обозначает пустое множество $\varnothing$ и служит для замыкания алфавита относительно теоретико-множественных операций, а также для его связи с математическими структурами более высокой иерархии и построения систем доказательств.

Обьединяющим форматом для алфавитов A^X и A^1 служат два временных фрейма <t-1,t>. Приведение однотактных символов $a_i^1 \in A^1$ к двухтактной форме выполняется с помощью символа Z, который эквивалентен пустому множеству U в одном автоматном такте, но полнее последнего, и задает описание переменной в состоянии высокого импеданса или высокого выходного сопротивления. С помощью буквы Z формат однотактных символов принимает вид: $a_i^1 < Z, a >$, где a – однотактные булевы состояния из множества

A ={0, 1, X={0,1}, Z}, абстрагированные от времени.

По определению на каждом множестве A^1, A^X выполняются все аксиомы и тождества абстрактной математической решетки с дополнениями, такие как ассоциативность, коммутативность, дистрибутивность, идемпотентность, действия с универсумом и пустым множеством, двойного дополнения, правила де-Моргана, поглощения.

10.4. Проектирование двухтактных КП активизации

Основной недостаток D-алгоритма и других методов активизации логических путей заключается в итеративном характере получения многомерного пути активизации при наличии сходящихся разветвлений в целях очувствления некоторого входа относительно наблюдаемых выходов, если такое возможно. Другое решение данной задачи связано с использованием ДКИ. Очувствление входной переменной с помощью построения одно- или многомерных путей активизации осуществляется за одну итерацию, которая включает прямую импликацию символов активизации и обратную импликацию символов обеспечения. Рассмотрим теоретическое обоснование кубического (К)-алгоритма.

Лемма 10.1. Покрытие активизации примитивного элемента (ПЭ) из базиса W={И, ИЛИ, И-НЕ, ИЛИ-НЕ} записывается на алфавите $A^9 \subset A^X$.

Поскольку кубическое покрытие любой комбинационной схемы с учетом возможной минимизации представляется значащими символами однотактного кубического исчисления A^1={0,1,X}, то конкатенация всех возможных упорядоченных пар символов создает подмножество в виде:

$a * b =$

*	0	1	X
0	Q	E	A
1	H	J	B
X	O	I	Y

. (10.1)

Следовательно, другие символы для упомянутых логических элементов появиться не могут, поэтому определим усеченные операции пересечения и объединения, где используются 9 символов, полученных в (10.1):

a ∩ b =

∩	Q	E	H	J	O	I	A	B	Y
Q	Q	U	U	U	Q	U	Q	U	Q
E	U	E	U	U	U	E	E	U	E
H	U	U	H	U	H	U	U	H	H
J	U	U	U	J	U	J	U	J	J
O	Q	U	H	U	O	U	Q	H	O
I	U	E	U	J	U	I	E	J	I
A	Q	E	U	U	Q	E	A	U	A
B	U	U	H	J	H	J	U	B	B
Y	Q	E	H	J	O	I	A	B	Y

; a ∪ b =

∪	Q	E	H	J	O	I	A	B	Y
Q	Q	A	O	S	O	L	A	F	Y
E	A	E	P	I	V	I	A	C	Y
H	O	P	H	B	O	C	V	B	Y
J	S	I	B	J	F	I	L	B	Y
O	O	V	O	F	O	Y	V	F	Y
I	L	I	C	I	Y	I	L	C	Y
A	A	A	V	L	V	L	A	Y	Y
B	F	C	B	B	F	C	Y	B	Y
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

.

Определение 10.1. Покрытие активизации называется полным, если оно содержит все возможные решения, формирующие на выходе ПЭ символы E, H. E(H)-покрытие обеспечивает на выходе изменения из 0(1) в 1(0). Покрытие обеспечения называется полным, если оно содержит все возможные решения, формирующие на выходе ПЭ символы O, I, A, B, Q, J. Объединение упомянутых полных покрытий называется полным двухтактным покрытием, которое обеспечивает прямую и обратную импликации символов {Q,E,H,J,O,I,A,B}.

Теорема 10.1. *Полное покрытие активизации элемента из базиса W содержит 2n кубов и может быть получено конкатенацией в обоих направлениях нулевых и единичных кубов.*

Д о к а з а т е л ь с т в о. Поскольку элемент из базиса W имеет регулярное покрытие, содержащее один нулевой (единичный) и n единичных (нулевых) кубов, то число сконкатенированных в одном направлении пар всегда будет равно количеству входов. Но является ли полученное таким образом КП полным в том смысле, что оно содержит кубы активизации кратности 1,2,...,n? Для ответа на данный вопрос рассмотрим КП элемента ЗИ-НЕ:

$$C = \left|\begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline 0 & X & X & 1 \\ X & 0 & X & 1 \\ X & X & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array}\right| . \qquad (10.2)$$

Сцепление по правилам введенной выше *-операции (10.1) первых трех кубов с последним дает результат:

$$C = \left|\begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline E & I & I & H \\ I & E & I & H \\ I & I & E & H \end{array}\right| . \qquad (10.3)$$

Известное полное покрытие активизации [23] для приведенного выше примитива имеет 7 кубов, которые предназначены для реализации активизации каждой переменной в отдельности, всех возможных сочетаний по два активных входа на трех линиях, а также одновременной активизации трех переменных:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline D & 1 & 1 & N \\ 1 & D & 1 & N \\ 1 & 1 & D & N \\ D & D & 1 & N \\ D & 1 & D & N \\ 1 & D & D & N \\ D & D & D & N \end{array} \right| . \qquad (10.4)$$

Выполнив замену знаков 1,D,N на соответствующие им символы двухтактного алфавита, получим:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline H & J & J & E \\ J & H & J & E \\ J & J & H & E \\ H & H & J & E \\ H & J & H & E \\ J & H & H & E \\ H & H & H & E \end{array} \right| . \qquad (10.5)$$

Кубы 1,2,3 могут быть склеены с использованием операции объединения со строками 4,6,5 соответственно, а куб 7 можно склеить с векторами 4,5,6. Получается покрытие:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline H & B & J & E \\ J & H & B & E \\ B & J & H & E \\ H & H & B & E \\ H & B & H & E \\ B & H & H & E \end{array} \right| . \qquad (10.6)$$

Вторая итерация минимизации наборов 1,2,3 с кубами 5,4,6 соответственно дает окончательный результат в виде Е-покрытия:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline H & B & B & E \\ B & H & B & E \\ B & B & H & E \end{array} \right| , \qquad (10.7)$$

инверсия во времени которого с использованием одноместной операции

$$\frac{C_{ij}}{\bar{C}_{ij}} = \left| \begin{array}{ccccccccccccccc} Q & E & H & J & O & I & A & B & S & P & C & F & L & V & Y \\ \hline Q & H & E & J & A & B & O & I & S & P & C & F & L & V & Y \end{array} \right| \qquad (10.8)$$

определяет Н-покрытие активизации:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline E & I & I & H \\ I & E & I & H \\ I & I & E & H \end{array} \right| . \qquad (10.9)$$

Оба этих покрытия и составляют полное КП активизации.

По индукции данное доказательство может быть распространено на n-входовые логические элементы, которые будут представлены 2n кубами активизации, включающими векторы мерности 1, 2, 3, ..., n.

Поскольку для комбинационных схем направление активизации является несущественным для проверки неисправностей линий, то в целях экономии памяти можно иметь покрытие очувствления линий в одном направлении, которое при необходимости следует трансформировать в покрытие активизации обратного знака с помощью операции временной инверсии (10.8).

Определение 10.2. Тривиальным называется куб, имеющий по входным и выходным координатам символы X.

Теорема 10.2. *Полное двухтактное покрытие обеспечения содержит 3(n+1) кубов для n-входового элемента базиса W.*

Д о к а з а т е л ь с т в о. Рассматриваемое покрытие должно обеспечивать обратную импликацию всех девяти символов, получаемых при выполнении конкатенации (1): {Q, J, E, H, O, I, A, B, Y}. Символ Y не нуждается в обратной импликации, а для обеспечения {E,H} следует использовать ранее определенные покрытия активизации. Остаются еще пять символов. Но поскольку пары символов (O, I), (A, B) являются взаимно инверсными во времени, то для каждого логического примитива проведем доказательство получения одной из пар символов, например, (O, I). Символы можно представить упорядоченной парой: O=(X0), I=(X1). Значения 0 и 1 в момент t определяются состояниями выходов однотактного кубического покрытия (2.2). Состояние выхода ПЭ в момент t-1 равно X, которое может быть получено следующими комбинациями символов X и 1 (0 – для элемента ИЛИ), число которых определяется выражением 2n-1:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline 1 & X & X & X \\ X & 1 & X & X \\ J & X & 1 & X \\ 1 & 1 & X & X \\ 1 & X & 1 & X \\ X & 1 & 1 & X \\ X & X & X & X \end{array} \right| . \qquad (10.10)$$

Приведенные 7 кубов минимизируются, превращаясь в один тривиальный куб XXXX, для которого любое решение при выполнении прямой или обратной импликации в силу своего определения является непротиворечивым. Отсюда следует, что двухтактное покрытие обеспечения для символов {O, I} составляется конкатенацией всех кубов однотактного КП с тривиальным кубом, который содержит все возможные значения входов для X-состояния выхода, с получением для элемента ЗИ-НЕ строк:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline O & Y & Y & I \\ Y & O & Y & I \\ Y & Y & O & I \\ I & I & I & O \end{array} \right| . \qquad (10.11)$$

Кубы A, B-покрытия получаются применением операции временной инверсии к координатам приведенного выше КП, что дает результат:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline A & Y & Y & B \\ Y & A & Y & B \\ Y & Y & A & B \\ B & B & B & A \end{array} \right| . \qquad (10.12)$$

Кубы обеспечения символов {Q,J} получаются из КП ПЭ путем конкатенации каждого куба с самим собой, что дает результат:

$$C = \left| \begin{array}{ccc|c} X1 & X2 & X3 & Y \\ \hline Q & Y & Y & J \\ Y & Q & Y & J \\ Y & Y & Q & J \\ J & J & J & Q \end{array} \right| . \qquad (10.13)$$

По индукции для любого примитивного элемента базиса W всегда можно выполнить аналогичные рассуждения. Поскольку тривиальный куб всегда один, а мощность однотактного покрытия элемента базиса W равна n+1, то число кубов полного покрытия обеспечения символов {O, I, A, B, Q, J} равно 3(n+1), что и требовалось доказать.

10.5. Процедуры кубического алгоритма

Лемма 10.2. Выполнение прямой импликации в комбинационной схеме не порождает новых символов, отличных от {O, I, A, B} или их подмножеств, нуждающихся в доопределении.

Д о к а з а т е л ь с т в о. Покрытия активизации элементов базиса W не имеют других символов обеспечения, кроме {O, I, A, B}. В результате выполнения прямой импликации осуществляется лишь операция пересечения по координатам кубов активизации, которая не может увеличить мощность символа, а может только уменьшить, порождая значения {E,H,Q,J}.

Следствие 10.1. При выполнении прямой импликации не могут появиться символы алфавита A^X мощности 3, определяемые подмножеством: {C, F, L, V}.

Лемма 10.3. Обратная импликация не увеличивает количество неопределенностей при ее выполнении в целях доопределения символов обеспечения.

Д о к а з а т е л ь с т в о. При обработке очередного элемента, выход которого определен символами подмножества {O, I, A, B}, возможны два случая. В

первом входы ПЭ не являются линиями разветвления, а каждый из них соединен только с одним выходом другого примитива. Тогда состояния линий, соответствующих входам рассматриваемого ПЭ, будут определяться значениями координат КП обеспечения, которые могут быть заданы символами {O, I, A, B}. Во втором случае один или несколько входов элемента принадлежат линиям разветвления. Тогда общее непротиворечивое решение для них будет определяться пересечением. Поскольку описанная координатная операция может только уменьшить мощность результирующего символа, то символы подмножества {C,F,L,V} не могут появиться при выполнении обратной импликации.

Следствие 10.2. Реализация кубического алгоритма для комбинационных схем вентильного базиса W выполняется на подмножестве двухтактного алфавита A9, порожденного сокращенной операцией конкатенации (10.1) при использовании полных двухтактных покрытий активизации и обеспечения.

Следствие 10.3. Для выполнения обратной импликации двухуровневых схем необходимо и достаточно использования {O,I,A,B}-покрытий.

Следствие 10.4. Для выполнения обратной импликации неизбыточных и двухуровневых схем покрытия активизации не используются.

Определение 10.3. Выбор одного из возможных решений при выполнении обратной импликации через ПЭ, удовлетворяющих исходному состоянию выхода, будем называть одновариантным анализом. Использование всех возможных решений входных комбинаций, не противоречащих требуемому состоянию выхода, есть многовариантная обратная импликация.

Теорема 10.3. *Кубический алгоритм, использующий полные двухтактные покрытия активизации и обеспечения, выполняет построение пути активизации минимальной мерности, итерацию процедур прямой и обратной многовариантной импликации или определяет невозможность получения такого пути.*

Доказательство. Построение одномерного пути активизации при выполнении прямой фазы определяет активные линии и условия очувствления на входах ПЭ, которые формируют единственный вектор активности, поскольку прямая импликация есть одновариантная задача. Противоречия в данной процедуре возникнуть не могут, если элементы имеют полные покрытия активизации, а это действительно так по условию теоремы 10.1. Обратная фаза основывается на использовании полных двухтактных покрытий обеспечения согласно теореме 10.2. Значит, при обработке очередного ПЭ, что является многовариантной задачей (поскольку в общем случае существует более одного куба для получения некоторого двухтактного состояния выхода), решение обязательно будет получено по входам примитива с учетом, что полное покрытие содержит все возможные кубы для определения состояний {0, 1, X} на выходах элемента. Если в силу структуры гальванических связей, образующих разветвления на входах ПЭ, возникнет пустое

пересечение на одной или нескольких входных линиях между всеми кубами покрытия обеспечения и текущими состояниями вектора активизации D_i, делается вывод о противоречивости текущего решения D_i. Если для всех D_i отсутствует положительное пересечение с кубами анализируемого покрытия, тест активизации для выбранного логического пути не существует.

При построении тестов для комбинационной схемы (КС) будем использовать стратегию построения минимального множества одномерных путей активизации, покрывающих все существенные эквипотенциальные линии устройства. Любой активный путь всегда начинается с постановки символа очувствления на очередном внешнем входе. Далее выбирается один из элементов, которому принадлежит данный вход, что обеспечивает тенденцию к желанию построить одномерный путь активизации. Первое пересечение символа E или H с покрытием активности выбранного ПЭ формирует вектор активизации, который после обработки примитива будет иметь на линии, соответствующей выходу ПЭ, символ из множества {E,H}. Далее вектор активизации пересекается с кубами покрытия элемента, входы которого соединены с уже активным выходом. Такая процедура заканчивается при достижении внешних выходов схемы.

Лемма 10.4. Для выполнения прямой или обратной импликации используются только операции пересечения и временной инверсии.

Доказательство. В соответствии со стратегией проектирования одномерных путей после определения на очередном входе (линии) вектора активизации символа из множества {E,H} все операции прямой фазы сводятся к пересечению вектора D_i с кубами очередного ПЭ. В случае пустого решения выполняется временная инверсия кубов активности, и пересечение повторяется. В обратной фазе наблюдается аналогичная ситуация с той лишь разницей, что вектор D_i может размножаться, благодаря многовариантности обратной импликации.

Лемма 10.5. Двухтактные покрытия активизации используются для выполнения прямой и обратной импликации.

Доказательство. В соответствии со стратегией проектирования одномерных путей вначале всегда определяется символ активизации на входе очередного элемента, после чего выполняется пересечение кубов покрытия с вектором активизации в целях определения очередной активной линии, относящейся к выходу ПЭ. Кроме того, при выполнении обратной фазы на линии сходящегося разветвления возможно возникновение символов активизации H, E при пересечении символов O и B; A и I, что является штатной ситуацией (рис.10.6). При построении активного пути a-c-d вход b элементов 1 и 2 требует условий I, A соответственно, что не является противоречием и при пересечении дает символ E.

Теорема 10.4. *Двухтактное полное покрытие n-входового элемента из множества W для выполнения кубического алгоритма содержит 5n+3 куба.*

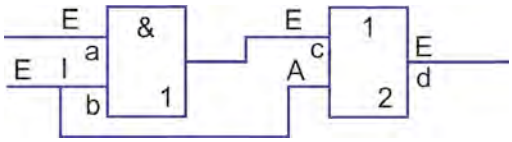


Рис.10.6. Порождение активизации на обратной фазе

Д о к а з а т е л ь с т в о. Мощность КП для выполнения процедуры обеспечения символов {O,I} равна n+1. Аналогичную оценку имеют покрытия, создающие на выходе ПЭ символы {A,B} и {Q,J} (теорема 10.2). Мощность покрытия активизации одного направления (теорема 10.1), например, для обеспечения на выходе символа E равна n. Дополняющее покрытие типа H также содержит n кубов. Суммируя, получаем оценку 5n+3, что и требовалось доказать.

Теорема 10.5. *Для схемы в форме СДНФ существует множество одномерных путей активизации, покрывающих все существенные переменные.*

Д о к а з а т е л ь с т в о. Совершенная дизъюнктивная нормальная форма содержит количество конъюнктивных термов, равное числу единиц таблицы истинности. Каждый терм может содержать существенные и несущественные переменные. Последние из них определяются невозможностью активизировать значение выхода примитива, соответствующего рассматриваемому терму. Это означает, что два соседних входных состояния формируют единичные (нулевые) значения выходов рассматриваемой функции. Кроме того, наличие в карте состояний функции (Карно) двух соседних единиц есть не только условие невозможности активизации выхода при изменении входной переменной, которая определяет координатное различие единичного состояния функции, но и условие ее несущественности. Но если две соседние по Хеммингу координаты в карте Карно имеют различные значения, то изменение входных слов, соответствующих этим координатам, есть необходимое и достаточное условие для активизации функции в целом, поскольку не существует другого простого терма СДНФ, который на заданной паре входных наборов имел бы активность. Верно также и то, что не существует такого терма, который смог бы компенсировать выходную активность примитива в силу детерминизма конечного комбинационного автомата, который всегда имеет нулевую или единичную реакцию на двоичное входное слово. Таким образом, если переменная в терме существенна, ее активизация выполнится построением одномерного логического пути, который всегда существует. Следовательно, для всех существенных переменных термов СДНФ существуют одномерные пути активизации.

10.6. Особенности построения тестов для избыточных схем

Следствие 10.5. Переменная является несущественной, но может быть активизирована многомерно, если не существует пары входных наборов, активизирующих ее в терме СДНФ одномерным путем.

В самом деле, если изменение на одном входе не приводит к изменению выхода функции, то соседние клетки в карте Карно имеют значение Y (нуль или единица), что означает несущественность рассматриваемой входной переменной X_i. Однако в карте состояний функции могут существовать координаты, имеющие нулевое и единичное значение для переменной X_i соответственно с противоположным значением функции $\overline{Y}$, что доказывает возможность многомерной активизации несущественной переменной. Таким образом, проектирование одномерного пути активизации есть способ определения избыточности переменных и элементов в схеме, соответствующей СДНФ.

Следствие 10.6. Если все переменные в примитиве избыточны или несущественны, он является несущественным. Естественно, что такого элемента не может быть в СДНФ, за исключением, когда функция тождественно равна нулю или единице. В противном случае речь идет о продукте минимизации булевой функции, о ДНФ, скобочных формах.

Определение 10.4. Дизъюнктивная нормальная форма называется простой (ПДНФ), если в процессе минимизации каждый терм, соответствующий конституенте единицы, был включен в одно склеивание. В противном случае ДНФ назовем избыточной.

Теорема 10.6. *Простая дизъюнктивная нормальная форма не содержит несущественных переменных в термах, полученных в процессе минимизации.*

Доказательство. Минимизация есть процесс исключения из термов несущественных переменных. Это значит, что после этой процедуры все соседние конституенты единицы, число которых определяется степенью двойки, будут включены в простой терм. Кроме того, изменение любой существенной для склеенного поля переменной X_i должно приводить к изменению функции, в противном случае склеивание осуществлено не полностью и его нужно продолжить, чтобы убрать несущественность переменной X_i.

Следствие 10.7. Минимизация булевой функции в целях получения ПДНФ позволяет построить полный минимальный проверяющий тест для ОКН только одномерными путями активизации.

В самом деле, если все переменные в термах существенны, то каждая из них создает собственное изменение функции, в котором не участвуют другие термы, поскольку все они формируют неперекрывающиеся состояния выходов на карте Карно. Если бы на паре входных наборов существовала другая

активная переменная, то это свидетельствовало бы о несущественности переменной, что невозможно в силу условий теоремы 10.6.

Следствие 10.8. Если для входной переменной примитива в схеме, соответствующей ДНФ, невозможно построить одномерный путь активизации, она является несущественной (рис. 10.7).

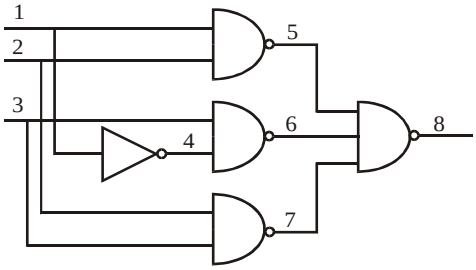


Рис. 10.7. Схема, соответствующая ДНФ

Схема без элемента с выходом 7 соответствует простой ДНФ. Отсюда следует, что все переменные в простых термах являются существенными. Подтверждением тому могут служить пары наборов: 111-101, 110-010 и 000-001, 010-110, активизирующие входы первого и второго примитивов соответственно.

Использование избыточного терма приводит к появлению избыточного элемента, заданного на рис. 10.7 выходом 7. Данный ПЭ имеет несущественные переменные, которые не могут быть активизированы без одновременного очувствления входов других примитивов. Поскольку все входы последнего элемента несущественны, его следует убрать из схемы, что не только минимизирует аппаратурные затраты, но и повышает тестопригодность цифрового устройства.

Другой пример избыточности, заданный функцией $Y = X_1X_2 \vee X_2$, свидетельствует о несущественности переменной X_1, которую невозможно активизировать без очувствления X_2. Однако неизбыточная линия X_2 активизируется вполне самостоятельно в масштабе избыточной схемы или после удаления переменной X_1, когда булева функция превращается в выражение $Y = X_2$.

Теорема 10.7. *Для неизбыточной КС произвольной структуры существует одномерный путь активизации, проходящий через любой вход примитива.*

Д о к а з а т е л ь с т в о. Возможность построения в прямой фазе активного одномерного пути обеспечивается существенностью переменных в примитивах схемы, поэтому всегда может быть построен путь от X_i до Y. При выполнении обратного хода, в случае несущественности X_i, наблюдается фактический разрыв между Y и X_i. Однако при наличии хотя бы одной су-

щественной переменной X_j для Y обратная импликация должна обеспечить изменение выхода Y на паре входных последовательностей. Если этого не произойдет, все входы схемы являются несущественными, что возможно при функции, тождественно равной единице или нулю на всех входных наборах. Получение положительного решения – построения пути активизации к другому входу X_j, не равному X_i, обусловлено обратной импликацией символов обеспечения {O, I, A, B}, которые содержат в том числе символы активности {H, E, E, H} соответственно. Последние могут быть получены на линии разветвления, принадлежащей сходящемуся разветвлению. Это связано с тем, что получение {H, E} из символов {O, I, A, B} есть результат пересечения пар A, I и O, B и ничего более. Поэтому отсутствие разветвлений в схеме – это гарантия ее неизбыточности, и наоборот, если схема избыточна (присутствие несущественных переменных или дублирующих элементов), то она имеет разветвления. Несущественность переменной или избыточность элемента есть свойство структуры, поэтому ее выявление возможно только на обратном ходе, где проверяется непротиворечивость гальванических соединений требуемому условию активизации, полученной при выполнении прямой фазы. Однако наличие сходящихся разветвлений не является условием достаточности для определения факта схемной избыточности, поскольку при этом возможна ситуация существенности всех переменных и отсутствия избыточных ПЭ.

Любая неизбыточная структура при наличии изменения в одном разряде на паре входных векторов может создавать изменение на выходе схемы. Для схемы, соответствующей ПДНФ, все переменные термов являются существенными, что означает возможность активизации любой из них с помощью одномерного логического пути. Отсюда следует, что и через входы элемента, не являющиеся внешними линиями схемы, можно построить одномерные пути активизации до внешнего выхода объекта.

Избыточность есть свойство структурной реализации функции, она всегда устранима и определяется несущественностью переменной, которая проявляется в виде активизации другой входной переменной при выполнении обратной фазы или избыточными термами. Для последнего случая характерна симметричная или несимметричная избыточность. В первом случае более двух одинаковых термов дублируют друг друга при формировании значения функции, что создает для каждой переменной мерность активизации, равную общему числу термов. Примером может служить схемная реализация функци $Y = X_1X_2 \vee X_1X_2$, где каждая переменная в обоих термах активизируется одновременно. Во втором случае при существовании одного или нескольких избыточных термов, не равных неизбыточным, получается одномерная активизация X_i в неизбыточном терме и многомерная (двумерная) активи-

зация указанной переменной при первоначальной обработке X_i в избыточном терме. Например, для схемной реализации функции

$$Y = X_1X_3 \vee \overline{X}_1X_2 \vee X_2X_3$$

переменные в третьем терме активизируются совместно с аналогами в первых двух.

Теорема 10.8. *Для избыточной КС произвольной структуры существует путь активизации, который может быть построен с помощью К-алгоритма, проходящий через любой вход примитива.*

Д о к а з а т е л ь с т в о. Поскольку в примитивах схемы отсутствует избыточность, то через любой его вход можно построить путь активизации при выполнении прямой фазы. Структурная реализация функции может иметь несущественность терма или переменной. Если вход примитива является несущественным, то на обратной фазе согласно доказательству теоремы 10.7 будет построен дополнительный путь активизации для фактического очувствления выхода функции. Формально работа К-алгоритма завершится построением двумерного для выбранного входа терма пути активизации. Если вход примитива принадлежит избыточному элементу, то на обратной фазе будет подтверждена активизация эквипотенциальной линии, которая идентифицирует выбранный вход терма. Однако фактическое очувствление линии осуществится через терм, отличный от избыточного. В совокупности результатом работы К-алгоритма будет двумерный путь активизации, проходящий через избыточный и неизбыточный примитивы. Для симметричной избыточности схемной реализации функции мерность пути активизации определяется количеством ПЭ симметричной избыточности.

10.7. К-алгоритм генерации тестов для комбинационных схем

Речь идет о практическом применении кубического исчисления на основе приведенных выше теоретических обоснований для построения проверяющих тестов методом очувствления одномерного логического пути эквипотенциальных линий, в котором присутствуют прямая и обратная фазы для выполнения процедур активизации и обеспечения соответственно.

Вместо двухтактных символов активизации обеспечения используются их однотактные составляющие, реализующие активность, или обеспечение в двух соседних временных фреймах. Такое упрощение позволяет значительно повысить технологичность алгоритма генерации тестов, сведя все процедуры к выполнению операций над символами алгебры Кантора $\{0, 1, X, \varnothing\}$.

В качестве исходной информации используется структурно-функциональная модель устройства, где для каждого примитива имеется кубическое покрытие $C = \{C_1, C_2, \ldots, C_i, \ldots, C_n\}$, по которому генерируются пары кубов покрытия активизации. Учитывая, что символы алфавита в парах кубов активизации – однотактные, для выполнения обратной фазы – обеспечения –

используются только кубические покрытия примитивов. Результат выполнения алгоритма оформляется во множество пар векторов активизации

$$D = \{D_1, D_2, ..., D_i, ..., D_n\},\ D_i = (D_i^1, D_i^2).$$

Для исключения итеративности процедур прямой и обратной импликации до начала работы алгоритма следует выполнить ранжирование линий и элементов. Первыми нумеруются в порядке возрастания входные линии схемы, после чего очередные номера присваиваются выходам тех элементов, входы которых уже занумерованы. Последними отмечаются выходы ПЭ, относящиеся к внешним выходам схемы. Аналогично осуществляется ранжирование ПЭ, с той лишь разницей, что на первом шаге нумеруются элементы, входы которых являются внешними входами объекта, а затем отмечается очередной ПЭ, имеющий уже отмеченных предшественников. Последними нумеруются элементы, выходы которых принадлежат к внешнему разъему устройства.

С учетом приведенных определений и доказательств основные моменты К-алгоритма представлены следующими пунктами:

1. Выбор очередной входной эквипотенциальной линии j и установка в соответствующей координате нового вектора D активизации символа активности (имеется в виду пара символов однотактного алфавита) D_j=01(10) (на первом шаге i-1, j=1). Все другие координаты вектора D, за исключением D_j, к началу построения активного логического одномерного пути должны иметь значения неопределенностей (XX):

$$D_j^{1(2)} = \{(01) \vee (10)\} \Leftarrow (D_j^{1(2)} = X).$$

2. Определение очередного примитива Pt из множества всех элементов схемы цифрового устройства для выполнения прямой импликации символа активизации по правилу: линия, соответствующая входу ПЭ, должна иметь символ активности в текущем векторе D_i, выходная координата элемента по возможности не должна быть активизирована во всех предшествующих векторах D. Анализ примитивов заканчивается при достижении активности на внешних выходах схемы.

$$D_j(P_t^Y) = \{(01) \vee (10)\} \Leftarrow D_r(P_t^X) = \{(01) \vee (10)\} \vee D_j(P_t^Y) = X.$$

Стратегия выбора очередного пути активизации использует критерий получения минимального множества одномерных путей, покрывающих все эквипотенциальные линии объекта.

3. Обратная фаза заключается в обработке каждого примитива схемы, на выходе которого после прямой фазы существуют символы, отличные от (XX), за исключением элементов, которые стали активными при выполнении прямой фазы. Расширение поля деятельности обратной импликации в целях до-

определения символов активизации 01(10) связано с построением фактического пути, очувствляющего выход схемы, если первоначально при выполнении прямой фазы была сделана попытка активизировать несущественную переменную. Для устранения итеративного характера обратной импликации очередным следует выбирать элемент, имеющий уже обработанных преемников. Суть анализа заключается в пересечении каждого куба C_l покрытия обеспечения с текущим вектором активности D_i. При непустом пересечении D_i с n кубами выполняется размножение строки D_i на n векторов (j=1,n), что является следствием многовариантности решения задачи обратной импликации:

$$D_{ij} = [D_i \cap C_j(P_t^Y)] \Leftarrow \forall r[D_{ir} \cap C_{jr}(P_t^Y) \neq \varnothing].$$

4. Объединение текущих векторов, являющихся претендентами для активизации очередного входа устройства, после выполнения обратной импликации на очередном примитиве. Данная процедура имеет смысл при наличии сходящихся разветвлений. Она способна значительно уменьшить количество промежуточных решений, а значит и время работы алгоритма. Объединение выполняется в случае различия пары строк только по одной координате – основной принцип минимизации, или при получении в результате выполнения координатной операции пересечения двух соседних векторов одного из них – правило поглощения:

$$D_i \cup D_j \Leftarrow [\exists! r(D_{ir} \cap D_{jr}) = \varnothing] \vee (D_i \cap D_j = D_j).$$

5. После формирования пар векторов активизации, покрывающих все линии в схеме, выполняется доопределение символа X на входных координатах:

$$D_{ij}^{1(2)} = \begin{cases} D_{ij}^1 \cap D_{ij}^2 \Leftarrow (D_{ij}^1 \vee D_{ij}^2 = X) \wedge (D_{ij}^1 \neq D_{ij}^2); \\ 0 \vee 1 \Leftarrow (D_{ij}^1 \cap D_{ij}^2 = X). \end{cases}$$

Координаты заменяются нулем или единицей в целях получения минимального кодового расстояния между соседними векторами теста. Возможны и другие критерии доопределения, например, с учетом максимизации проверяющих свойств двоичных наборов или минимизации входных векторов вчеркиванием одинаковых.

6. Для определения двоичных значений линий на всех тестовых последовательностях выполняется их моделирование, которое уточняет состояния невходных линий, оставшихся неопределенными даже после выполнения обратной фазы. Необходимость выполнения данного пункта иллюстрируется состояниями линий фрагмента схемы (рис. 10.8).

Полученный в результате выполнения К-алгоритма тест является полным относительно ОКН, если удалось построить только одномерные пути активизации, покрывающие все линии цифрового устройства. В противном слу-

чае тест будет полным относительно всех ОКН всех неизбыточных или существенных линий. Минимальность полученного двоичного теста не гарантирована, поэтому есть смысл в оценке его качества с помощью программ моделирования неисправностей, что дает возможность получить минимальный набор входных проверяющих последовательностей.

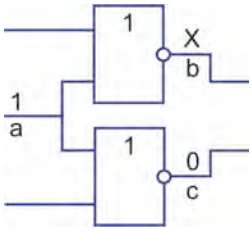


Рис. 10.8. Фрагмент схемы

Иллюстрацию основных шагов алгоритма выполним на примере обработки схемы, представленной на рис. 10.8, покрытия элементов которой записаны по входным и выходным номерам эквипотенциальных линий.

В цифровой схеме формат записи существенных переменных для элемента, типа 2И-НЕ определяется векторами: C^1 = {(1,2,5); (3,4,6); (3,2,7)}. Для инвертора – C^2 = (1,4). Для 3И-НЕ – C^3 = (5,6,7,8). Покрытия соответствующих типов элементов цифрового устройства, записанные по входным и выходным номерам эквипотенциальных линий, имеют вид:

$$C^1 = \begin{vmatrix} 0 & X & 1 \\ X & 0 & 1 \\ 1 & 1 & 0 \end{vmatrix};\ C^2 = \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix};\ C^3 = \begin{vmatrix} 0 & X & X & 1 \\ X & 0 & X & 1 \\ X & X & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix}.$$

Покрытия активизации упомянутых типов примитивов представлены парами кубов:

$$C^1 = \begin{vmatrix} 0 & X & 1 \\ 1 & 1 & 0 \end{vmatrix};\ \begin{vmatrix} X & 0 & 1 \\ 1 & 1 & 0 \end{vmatrix};\ C^2 = \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix};$$

$$C^3 = \begin{vmatrix} 0 & X & X & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix};\ \begin{vmatrix} X & 0 & X & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix};\ \begin{vmatrix} X & X & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix}.$$

Для схемы (см. рис.10.8) выполнение прямой фазы определяет в формате переменных (1,2,3,4,5,6,7,8) следующие векторы активизации:

$$\begin{vmatrix} 1 & 1 & - & - & 0 & X & X & 1 \\ 0 & X & - & - & 1 & 1 & 1 & 0 \end{vmatrix};\ \begin{vmatrix} 1 & - & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & - & X & 1 & X & 0 & X & 1 \end{vmatrix};$$

$$\begin{vmatrix} 1 & 1 & - & - & 0 & X & X & 1 \\ X & 0 & - & - & 1 & 1 & 1 & 0 \end{vmatrix};\ \begin{vmatrix} - & 1 & 1 & - & X & X & 0 & 1 \\ - & 0 & X & - & 1 & 1 & 1 & 0 \end{vmatrix};$$

$$\begin{vmatrix} - & - & 1 & 1 & X & 0 & X & 1 \\ - & - & 0 & X & 1 & 1 & 1 & 0 \end{vmatrix};\ \begin{vmatrix} - & 1 & 1 & - & X & X & 0 & 1 \\ - & X & 0 & - & 1 & 1 & 1 & 0 \end{vmatrix}$$

где знак “-” соответствует символу X. Выполнение обратной фазы для доопределения символов $D^1_{ij}(D^2_{ij})$ при выполнении условий

$$D^2_{ij}(D^1_{ij}) = X \wedge (D^1_{ij} \neq D^2_{ij})$$

на невходных линиях путем использования кубических покрытий примитивных элементов создает тест активизации:

$$\begin{vmatrix} 1 & 1 & - & - & 0 & X & X & 1 \\ 0 & X & - & - & 1 & 1 & 1 & 0 \end{vmatrix}; \begin{vmatrix} 1 & - & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & - & X & 1 & X & 0 & X & 1 \end{vmatrix};$$

$$\begin{vmatrix} 1 & 1 & - & - & 0 & X & X & 1 \\ X & 0 & - & - & 1 & 1 & 1 & 0 \end{vmatrix}; \begin{vmatrix} - & 1 & 1 & - & X & X & 0 & 1 \\ - & 0 & X & - & 1 & 1 & 1 & 0 \end{vmatrix};$$

$$\begin{vmatrix} - & - & 1 & 1 & X & 0 & X & 1 \\ - & - & 0 & X & 1 & 1 & 1 & 0 \end{vmatrix}; \begin{vmatrix} - & 1 & 1 & - & X & X & 0 & 1 \\ - & X & 0 & - & 1 & 1 & 1 & 0 \end{vmatrix}.$$

Доопределение символов X на входных координатах в формате переменных (1,2,3) полученного теста в соответствии с правилами п. 5 К-алгоритма задает наборы D^X, которые после минимизации путем вычеркивания одинаковых приобретают вид D^X_m:

$$D^X = (111,010,101,001,110,100,011,000,001,100,011,010).$$

$$D^X_m = (110,010,101,011,000).$$

Полученный минимизированный тест D^X_m содержит семь входных наборов, исправное моделирование которых представлено правой частью, а проверяющие способности теста – левой частью (в виде векторов проверяемых дефектов) матрицы по формату переменных (1,2,3,4,5,6,7,8):

$$\left|\begin{array}{cccccccc|cccccccc} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & . & . & 1 & . & . & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & . & 1 & . & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & . & 1 & . & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & . & 0 & 0 & . & 1 & . & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & . & 1 & . & . & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & . & . & 0 & . & . & . & . & . \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & . & . & . & . & 0 & 0 & 0 & 1 \end{array}\right|$$

Моделирование ОКН для наборов позволяет сделать вывод об избыточности последних трех, которые дублируют проверку неисправностей, выполненную предыдущими векторами. Поэтому минимальным двоичным тестом являются последовательности:

$$D^X_{min} = (110,010,101,001).$$

Моделирование дефектов также подтверждает невозможность проверки дефекта 7^1 на полученных векторах, поскольку выход 7 относится к избыточному элементу. Выполнение К-алгоритма, основанного на активизации минимального множества путей в схеме, от внешних входов до наблюдаемых выходов, покрывающих все линии, гарантирует построение полного проверяющего теста для тех ОКН, для которых он существует. Дальнейшая минимизация теста связана с применением средств моделирования неисправностей, что дает точную характеристику проверяющих свойств каждого вектора и теста.

Моделирование неисправностей также подтверждает невозможность проверки дефекта 7^1 на полученных векторах, поскольку выход 7 относится к избыточному элементу.

Более существенным примером для верификации К-алгоритма является схема арифметико-логического устройства, представленная на рис. 10.9. Для нее выполнено построение теста активизации всех переменных парами двоичных наборов. В результате получилось 76 двоичных векторов, которые активизируют все линии в схеме. С помощью методики минимизации входных тестовых наборов на основе использования результатов логического моделирования неисправностей путем решения задачи покрытия полного множества дефектов минимальным числом векторов был получен минимизированный тест, включающий 11 двоичных векторов, который представлен на рис. 10.10. Кроме теста, здесь имеется дополнительная информация о качестве тест-вектора в виде процента проверяемых одиночных константных неисправностей. Вектор проверяемых дефектов определен в двоичном алфавите {0,1}. Для последовательностных схем цифровых устройств координаты вектора могут иметь еще и символы X. Также представлена строка, определяющая процент проверенных ОКН всеми предыдущими наборами, включая текущий.

Вектор проверенных неисправностей задается в алфавите {0,1,X}. Если все символы последовательности равны X, то тест проверяет все ОКН схемы. Для данного примера (структурно-функциональная схема коммутатора) тест является полным относительно ОКН всех существенных линий схемы, что подтверждается его качеством, равным 100%.

Таким образом, предложенные процедуры и алгоритмы используют ДКИ для проектирования проверяющих тестов путем очувствления логического пути от внешнего входа до наблюдаемых выходов цифрового устройства.

Показаны преимущества A^2 -алфавита для компактной записи кубов активизации и обеспечения, необходимых для работы предложенного кубического алгоритма, способного построить тест для произвольной структуры объекта, в том числе и избыточной, если он существует.

Выполнена модификация К-алгоритма к троичному алфавиту кубического исчисления, что позволило упростить процедуры прямой и обратной импликаций, являющиеся основой проектирования проверяющих тестов.

На примерах генерации тестов для фрагментов цифровых схем и АЛУ показана эффективность получения полного и минимизированного теста проверки ОКН на основе использования прямой и обратной фаз К-алгоритма.

Что касается применения полученных результатов, то с появлением научного и практического интереса к языку VHDL [27-29] предлагаемый метод может использоваться в качестве базового для обработки семейства цифровых систем, представленных структурно-функциональным уровнем описания.

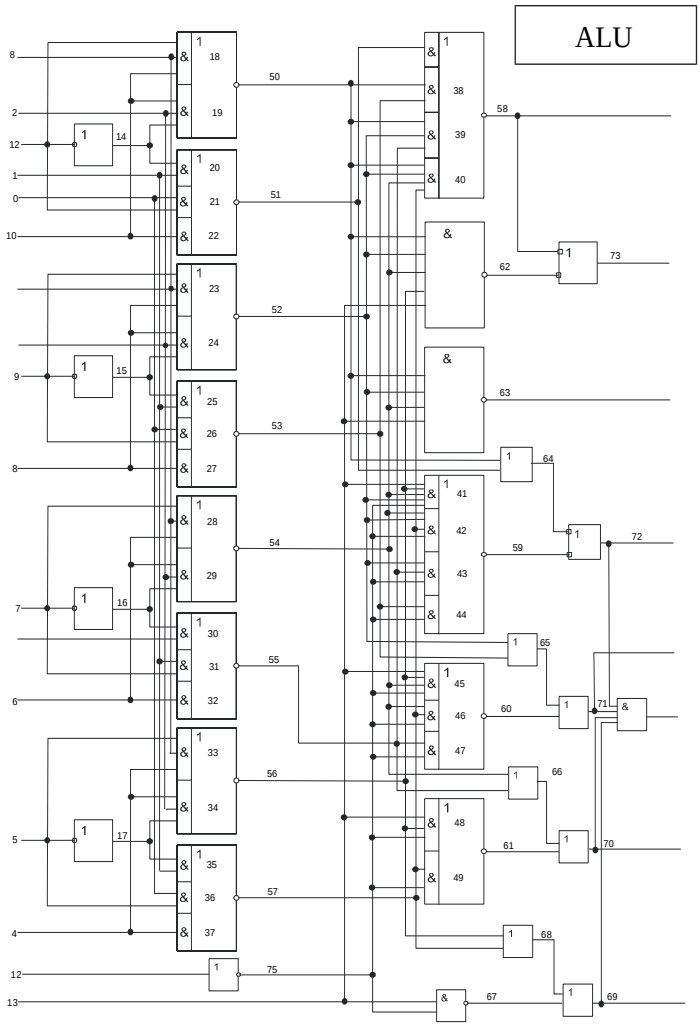


Рис. 10.9. Структурная схема АЛУ

Test:

```
0123456789012345678901234567890123456789012345678901234567890123456
0: 1011000111100010010100100100010000000010110001000001001110010011011011111011
01001110000.110....0..0..011.01111111110100111011111.0..00011011001001000000100  41%
01001110000.110....0..0..011.01111111110100111011111.0..00011011001001000000100 =41%
  1: 0110000100110001010000001000000001010101110010100100010000011110110111001100111 10
10.11110110011101011..110.1111110.01010001.0.0..0...0111110.00100100011100001  41%
XX0X1110XX0011X0101X..X100111X11X1X1X10X0X1X1011X1.001XXX1X0XXX0XX0X0XXX00
X0X =68%
  2: 0111010110011101011000011011000010010001100000000000010011011011101100011 0110
1000101.0110001.100.11.00...11110.10111001...111111110.1001.01000100111001001  41%
XX0X1X10XXX0XXX010XX11XX00111X11X1XXX1XX0X1X1X11X11XXXXXXXX0XXX0XX0X
XXXX0XX0X =79%
  3: 0111100111000010010010100000000110100111001001000100100000111101011110010111 1
10000110001.110110110.0.111111.00.011000110..0...0..0111110.00101000011010000  41%
XX0XXX10XXX0XXXX10XXX1XXXX111X1XX1XXXXXXXXXX1X11XX1XXXXXXXX0XXX0X
X0XXXXXXXX0X =87%
  4: 1110011100110001000001001001010001101000100001000000000111111101101001111001 0
0001.0.01..0.110..11..110..0.011.001011101111.1111111110000000100101100001101  40%
XX0XXX10XXX0XXXX10XXX1XXXX1X1X1XXXXXXXXXXXXXX1X11XX1XXXXXXXX0XXX0
XX0XXXXXXXX0X =88%
  5: 1101011100110001001001001010010001001000100001000000000011111101011010111111010
00.010.0...0.1.0..0...110.0..011.011011101111.1111111110000010100101000000101  38%
XX0XXX10XXX0XXXX10XXX1XXXXXX1X1XXXXXXXXXXXXXX1X11XX1XXXXXXXX0XXX0
XX0XXXXXXXX0X =88%
  6: 1001000000010001111001000000000000001111110001011101101100010001010001111000 1
01.0..11.1.011....0..0111111111111110000001110.....0011101110101110000111 0  37%
XX0XXX1XXXX0XXXX10XXXXXXXXXX1X1XXXXXXXXXXXXXX1X11XX1XXXXXXXX0XXX
0XXXXXXXXXXXXX =91%
  7: 0100001000110001100000001000100000101010110011010001001000011111011110000111 1
10.1..01.10.11.00111..110.110.111101010100..00.0...0..01111.00001000011110000  37%
XX0XXXXXXXX0XXXXXXXXXXXXXXXXXXX1XXXXXXXXXXXXXXXX1XXX1XXXXXXXX0X
XX0XXXXXXXXXXXXX =95%
  8: 0001101001110100101000000000001000001111010001011101101100010000110001111000 1
11.00.0.1.0010....0...111111110.11110000101110.....0011101111001110000111 0  37%
XX0XXXXXXXX0XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX1XXX1XXXXXXXX0X
XXXXXXXXXXXXXXXX =96%
  9: 1101000000000111110010001000100010101010100001100010010100001110111111000111 1
.0.........1100000110.110.110.110.0101010111100...0..0.011110001000000111000 0  37%
XX0XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX =99%
 10: 0000000001100010110000000000000000101111111111011101101000011001010100011001 1
111.......0.11....11..11111111111101000000...0........01111.01101011011100110  33%
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX =100%

76 Path activization pairs have been used
Test contain 11 vectors
```

Рис.10.10. Полный тест для схемы АЛУ

Ориентация на развитие и совершенствование языковых и программных средств VHDL является следствием: 1) создания замкнутого технологического цикла проектирования и производства интегральных схем на основе VHDL; 2) появления мощных компиляторов с языка VHDL и программных средств создания и верификации проектов; 3) стандартизации языков описания аппаратуры VHDL, Verilog на уровне науки и производства развитых стран; 4) открытости и доступности структур данных среды VHDL-Active

для улучшения и модификации программных компонентов; 5) популяризации на рынке стран Европы и Америки новых технологий HDL-Active [30].

Подтверждением сказанному могут служить выводы [31], направленные на обоснование долговременного прогноза перспективности развития новых технологий проектирования под эгидой HDL языков и среды на их основе.

10.8. Проектирование тестов для последовательностных схем

Рассмотрим необходимость построения тестов для функциональных элементов и последовательностных схем в целях выполнения верификации цифровой системы на стадии описания с помощью языка VHDL. Для этого следует генерировать тесты проверки соответствия исходной спецификации и модели, представленной в VHDL-коде. Учитывая, что сложность реализации цифровой системы, которая достигает тысяч и миллионов вентилей, вполне понятной является сложность проблемы генерации представительной выборки входных последовательностей, обеспечивающей верификацию исходного проекта и его конкретной реализации на кристалле ПЛИС.

Для реализации цифрового проекта используются ПЛИС, которые в массовом количестве применяются на рынке микроэлектронных технологий наряду с базовыми матричными кристаллами, сигнальными процессорами. Такой успех определяется: использованием субмикронных технологий изготовления кристаллов, применением Hardware-Software Cooperation-Design; минимальным временем проектирования цифровой системы (4-5 месяцев); высоким быстродействием выполнения операций (до 500МГц); большой степенью интеграции элементов на кристалле (до 3,5 млн); низкой стоимостью чипа, составляющей несколько долларов.

Вместе с преимуществами ПЛИС существуют и проблемы создания диагностического обеспечения для них, связанные с: тестированием цифровых проектов большой размерности, вентильного, функционального алгоритмического уровней описания, заданных в форме графов переходов конечных автоматов, булевых уравнений, многоуровневых иерархических структур; генерацией тестов проверки одиночных константных неисправностей или исправного поведения; получением приемлемого быстродействия алгоритмов моделирования неисправностей для оценки качества тестов.

Теоретические источники раздела: многозначная алгебра [32] – Hayes J.P., двухтактная кубическая алгебра [2], дедуктивный метод моделирования неисправностей [33-38] – Ермилов В.А., Armstrong D.B., Биргер А.Г., Levendel Y.H., методы генерации тестов [41,42] – Breuer M.A., Abramovici M.

10.9. Общая модель функционального элемента

В общем случае автоматная модель Мура модифицируется в целях компактного описания функционального примитива в виде:

$$M=\langle X, Y, Z, f, g\rangle,$$

где X=(X1,X2,...,Xi,...,Xm), Y=(Y1,Y2,...,Yi,...,Xh), Z=(Z1,Z2,...,Zi,...,Zk) – множества входных, внутренних и выходных автоматных переменных, отношения между которыми описываются уравнениями:

Y(t)=f[X(t-1),X(t),Y(t-1),Z(t-1)]; Z(t)=g[X(t-1),X(t), Y(t-1), Y(t), Z(t-1)]. (10.14)

Переменные Z(t) отличаются от Y(t) тем, что первые наблюдаемы по выходным линиям, а Y(t) в этом смысле есть внутренние. Кроме того, между ними существует отношение в виде функциональной зависимости Z(t)=g*[Y(t)]. Характеристическим уравнениям (10.14) соответствует обобщенная, автоматная, двухступенчатая, двухтактная структурная модель, изображенная на рис. 10.11.

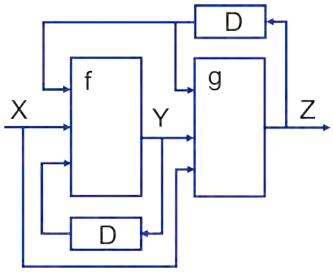


Рис. 10.11. Общая структурная модель автомата

Как следствие анализа структурной модели, можно определить функциональный последовательностный примитив в табличном виде, который задается следующими компонентами:

$$F2 = <(t\text{-}1,t),(X,Z,Y),\{A2\}>, \qquad (10.14)$$

где (t-1,t) – два автоматных соседних такта в описании функции; (X,Z,Y) – векторы входных, внутренних и выходных переменных; $\{A^2\}$ – двухтактный алфавит описания состояний (переходов) автоматных переменных [2].

В качестве основной формы описания примитивного элемента (ПЭ) выступает кубическое покрытие в общем случае троичных векторов:

$$C = (C_1, C_2, ..., C_i, ..., C_n), \qquad (10.15)$$

здесь $C_i = (C_{i1}, C_{i2}, ...C_{ij}, ..., C_{iq})$ – куб, включающий входные, внутренние, выходные координаты $C_i = (C_i^X, C_i^Y, C_i^Z), q = m + h + k$. Для комбинационного автомата формат описания кубического покрытия $F^1 = <(t),(X,Z),\{A^1\}>$ определяется отношениями на (q=m+k)-мерном векторе переменных $C_i = (C_i^X, C_i^Z)$. Формат задает многовыходовой комбинационный примитивный элемент с m входами и k выходами. Булева функция от m переменных $Z = f(X_1, X_2, ..., X_m)$ определяется при k=1.

Однако структура автомата (см. рис.10.11) является универсальной и потому избыточной для задания поведения произвольного вида триггеров или защелок. Кроме того, для повышения эффективности работы алгоритмов генерации тестов целесообразным представляется переход к троичному алфавиту описания автоматных переменных. Это влечет относительное повышение размерности таблиц переходов для триггерных схем, а также увеличение числа линий в виде псевдовходов. Но преимуществом такого отступления является инвариантность большинства процедур алгоритма генерации тестов активизации к обработке любых последовательностных примитивов. Поэтому структура универсального автомата модифицируется к двум эквивалентным, с позиции генерации тестов, формам последовательностных примитивов, определенных на рис. 10.12.

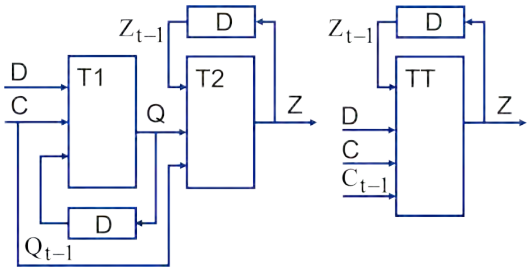


Рис. 10.12. Структурная модель двух типов триггеров

Первая (на рис. справа) ориентирована на задание поведения триггера-защелки, задержки, простого одноступенчатого D-триггера. Она имеет в качестве входов две линии синхронизации C_{t-1}, C_t, информационный вход D, и псевдовход Z_{t-1}. Выходная линия Z отвечает за состояние триггера. Вторая ориентирована на задание поведения двухступенчатого триггера, где состояние первой ступени определяется значением переменной Q, а второй – Z. При этом запись информации в первую ступень осуществляется при C=0, а во вторую – при значении сигнала C=1. Обе формы инвариантны по отношению к любому типу триггерной структуры в смысле, что каждый примитивный автомат может быть представлен одной из упомянутых форм. Для этого в правую структуру вводится переменная C_{t-1} и исключается линия Q. Это дает возможность осуществить переход от двухступенчатой модели триггера к одноступенчатой в целях ее упрощения для решения задачи генерации тестов.

В качестве примера описания функций двух- и одноступенчатого триггеров ниже представлены кубические покрытия C^F и C^{FF} в формате заданных переменных. Все кубы в покрытиях определены в троичном алфавите. Первые два куба в покрытии C^F формируют состояние выхода в зависимости от внешних входов, остальные определяют его значение в

функции от предыдущего сигнала на триггере. Для покрытия C^{FF} первые 4 куба задают состояние первой ступени Q_t в зависимости от входа D при сохранении значения второй ступени. Следующие 4 куба формируют состояние второй ступени Z_t в зависимости от внутренней переменной Q при сохранении значения первой ступени:

$$C^{FF} = \left|\begin{array}{cccc|cc} D & C & Q_{t-1} & Z_{t-1} & Q_t & Z_t \\ \hline 1 & 1 & X & 1 & 1 & 1 \\ 0 & 1 & X & 1 & 0 & 1 \\ 1 & 1 & X & 0 & 1 & 0 \\ 0 & 1 & X & 0 & 0 & 0 \\ X & 0 & 1 & X & 1 & 1 \\ X & 0 & 1 & X & 1 & 1 \\ X & 0 & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 & 0 \end{array}\right|; \quad C^{F} = \left|\begin{array}{cccc|c} D & C & C_{t-1} & Z_{t-1} & Z \\ \hline 1 & 0 & 1 & X & 1 \\ 0 & 0 & 1 & X & 0 \\ X & X & 0 & 1 & 1 \\ X & X & 0 & 0 & 0 \\ X & 1 & X & 1 & 1 \\ X & 1 & X & 0 & 0 \end{array}\right|.$$

По сравнению с покрытием C^{FF}, имеющим 2 выхода, более компактной представляется запись функции двухступенчатого триггера в виде двух последовательно соединенных примитивов, имеющих следующие описания:

$$C_1^{FF} = \left|\begin{array}{ccc|c} D & C & Q_{t-1} & Q \\ \hline 1 & 1 & X & 1 \\ 0 & 1 & X & 0 \\ X & 0 & 1 & 1 \\ X & 0 & 0 & 0 \end{array}\right|; \quad C_2^{FF} = \left|\begin{array}{ccc|c} Q & C & Z_{t-1} & Z \\ \hline 1 & 0 & X & 1 \\ 0 & 0 & X & 0 \\ X & 1 & 1 & 1 \\ X & 1 & 0 & 0 \end{array}\right|.$$

На основе информационной оценки [2], учитывающей объем памяти

$$H = \sum_{i=1}^{p} K_j + \sum_{j=1}^{q} K_j N_j + 3q, \qquad (10.16)$$

где p – общее число примитивов в схеме; q – количество типов ПЭ; $K_i(K_j)$, N_j – число соответственно переменных и кубов в текущем элементе, временного критерия, определяющего быстродействие анализа схемы

$$T = \sum_{i=1}^{p} K_i N_i + 3p, \qquad (10.17)$$

интегрального, учитывающего среднее значение компактности и быстродействия

$$Q = \frac{1}{2}(H + T), \qquad (10.18)$$

достаточно просто определить, что вторая модель триггера в виде структуры двух примитивов более предпочтительна. Это подтверждается оценками, вычисленными для C^{FF} и для структуры, представленной двумя элементами (C_1^F, C_2^F):

$$Q(C_1^F, C_2^F) = \frac{1}{2}[(8+32+6)+(32+6)] = \frac{81}{2} = 42;$$

$$Q(C^{FF}) = \frac{1}{2}[(6+48+3)+(48+3)] = \frac{107}{2} = 54.$$

Очевидно, что модель, состоящая из двух последовательно соединенных примитивов, более компактна, а значит, имеет более высокое быстродействие при выполнении моделирования. Поэтому ей отдается предпочтение.

10.10. Модель генерации тестов для функциональных элементов

Проблема генерации тестов для цифровой системы формулируются в условиях неопределенности компонента T на модели M, где M – модель, представленная кубическим покрытием C; L – кубическое покрытие списков неисправностей (КПСН), T – тест проверки неисправностей или исправного поведения.

Покрытие неисправностей для ПЭ (цифровой схемы) задается в виде:

$$L = (L_1, L_2, ..., L_i, ..., L_n),$$

где $L_i = (L_{i1}, L_{i2}, ..., L_{ij}, ..., L_{iq})$ – куб, включающий входные, внутренние, выходные координаты: $L_i = Li = (L_i^X, L_i^Y, L_i^Z)$, q=m+h+k;

$(L_{ij}^Y, L_{ij}^Z) = \{0,1,X\}$; 0 – определяет вычитание (дополнение) списка L_{ij};1 – задает пересечение L_{ij}; X={0,1} – идентифицирует несущественность списка неисправностей L_{ij}. Если L_{ij}^Z – выходная наблюдаемая переменная, то (0) 1 – есть идентификатор (не-) проверки неисправностей куба L_i на выходе r, X={0,1} – задает неопределенное состояние выходной координаты L_{ij}^Z, которое можно интерпретировать как 0, так и 1.

Кубическое покрытие списков неисправностей L для вектора T и покрытия примитива C вычисляется по линейному уравнению

$$T \oplus C = L, \tag{10.19}$$

где $\oplus$ – бинарная координатная операция XOR, определяющая взаимодействие компонентов T, C (L) в троичном алфавите:

$$T_j \oplus C_{ij} = (0 \oplus 0 \vee 1 \oplus 1 = 0); (0 \oplus 1 \vee 1 \oplus 0 = 1); \\ (X \oplus 0 \vee X \oplus 1 \vee 0 \oplus X \oplus 1 \vee X = X). \tag{10.20}$$

Универсальная формула анализа КПСН, полученного в результате применения (10.20) к тест-вектору T и покрытию многовыходового

примитива C для определения по выходу r списка проверяемых неисправностей L_r, имеет вид:

$$L_r = \bigcup_{\forall i(T_r \oplus C_{ir}=1)} \bigcap_{j=1}^{k} \bar{L}_{ij}^{T_j \oplus C_{ij}}, \qquad (10.21)$$

где $\bar{L}_{ij}$ – рассматривается как список дефектов, относящийся к линии j, который следует вычитать из неисправностей, проверяемых по невыходным линиям примитива; L_{ij}– неисправности, которые необходимо пересекать с невыходными списками. Если тест-вектор определен в троичном алфавите $T_j = \{0,1,X\}$, то после получения покрытия списков неисправностей каждый куб L_i следует верифицировать по правилу

$$L = L - L_i \Leftarrow [f(C_i^{(X)}) = X], i = \overline{1,n}. \qquad (10.22)$$

Здесь знак “–” эквивалентен теоретико-множественному вычитанию; $C_{ij}^{(X)} \in C_i^{(X)}$ – входные координаты куба покрытия ($j = \overline{1,m}$), модифицированные для выполнения прямой импликации $f[C_i^{(X)}]$ в целях определения существенности координат вектора $C_{ij}^{(X)} = X$ на состояние выхода r примитива, по правилу $C_{ij}^{(X)} = \begin{cases} C_{ij} \Leftarrow T_j \neq X; \\ X \Leftarrow T_j = X. \end{cases}$

Для вычисления собственных неисправностей примитива применяется F-процедура: 1) терм дизъюнктивной нормальной формы, имеющий более одной переменной без инверсии или все переменные с инверсией, вычеркивается; 2) оставшиеся термы должны иметь одну переменную без инверсии, которая формирует собственную проверяемую неисправность $j^{\bar{T}_j}$, инверсную состоянию координаты тест-вектора T_j.

Тест проверки дефектов примитива, задаваемых L_i^1 -кубом покрытия списка неисправностей $L = (L_i^0, L_i^1)$, определяется по уравнению:

$$L_i^1 \oplus C = T^k, \qquad (10.23)$$

где $L = (L_i^0, L_i^1)$ – кубы, имеющие нулевые или единичные значения на выходной координате r. При этом определяются векторы-кандидаты в тест $T_t^k \in T^k$. Из множества T^k формируется тест T, где в качестве элемен-

та рассматривается набор $T_t^k \in T^k$. Каждый $T_t \in T$ должен удовлетворять условиям:

$$T_t = C_i \cap T_t^k \Leftarrow \exists i(C_i \cap T_t^k \neq \varnothing). \qquad (10.24)$$

Пример 10.3. Для функции AND-NOT построить тест, проверяющий списки неисправностей, заданные кубом L(011).

Ниже приведена процедура получения теста на основе последовательного применения формул (10.23) и (10.24):

$$L(011) \oplus C_{\overline{AND}} \left| \begin{array}{c|c|c} a & b & c \\ \hline 0 & X & 1 \\ X & 0 & 1 \\ 1 & 1 & 0 \end{array} \right| = T^k \left| \begin{array}{c|c|c} a & b & c \\ \hline 0 & X & 0 \\ X & 1 & 0 \\ 1 & 0 & 1 \end{array} \right| = T \left| \begin{array}{c|c|c} a & b & c \\ \hline 1 & 1 & 0 \\ 1 & 0 & 1 \end{array} \right|. \qquad (10.25)$$

В результате получены 2 набора, которые одномерно активизируют входную переменную b. Следовательно, тест является полным относительно проверки одиночных константных дефектов на линиях b, c.

Тест проверки одиночных константных неисправностей одновыходового примитива, задаваемых L^1 -покрытием списка дефектов

$$L^1 = \left| \begin{array}{ccccc|c} X_1 & X_2 & ...X_j... & X_m & & Z \\ \hline 1 & 0 & ...0... & 0 & & 1 \\ 0 & 1 & ...0... & 0 & & 1 \\ ... & ... & ... & ... & & ... \\ 0 & 0 & ...1... & 1 & & 1 \\ ... & ... & ... & ... & & ... \\ 0 & 0 & ...0... & 1 & & 1 \end{array} \right|, \qquad (10.26)$$

определяется Т-процедурой, которая включает три следующих пункта:

1. Формирование кандидатов в тест

$$T_t^k \in T^k$$

путем выполнения операций над векторами из L^1, C:

$$L^1 \oplus C = T^k. \qquad (10.27)$$

2. Получение теста

$$T\,(T_t \in T)$$

из множества кандидатов T^k на основе применения выражения (10.24).

3. Минимизация множества тест-векторов T путем выполнения операции поглощения над парами двоичных последовательностей:

$$T = T - T_r \Leftarrow T_i \cap T_r = T_r .$$

Пример 10.4. Построить тест проверки одиночных константных дефектов для функции, заданной покрытием C.

Генерируется L^1 -покрытие списка дефектов по аналогии с (10.23) и для строки L_i^1 выполняется $\oplus$ -операция по (10.20):

$$L^1 \left| \begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array} \right| \oplus C \left| \begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ X & 0 & 1 & 0 \\ X & 1 & 0 & 0 \\ 0 & 1 & X & 0 \end{array} \right| = T^k(L_i^1), i = \overline{1,3}. \qquad (10.28)$$

К полученным подмножествам кандидатов в тест:

$$T^k(L_1^1) \qquad T^k(L_2^1) \qquad T^k(L_3^1)$$

$$\left| \begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ X & 0 & 1 & 1 \\ X & 1 & 0 & 1 \\ 1 & 1 & X & 1 \end{array} \right|; \left| \begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline X & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ X & 1 & 1 & 1 \\ X & 0 & 0 & 1 \\ 0 & 0 & X & 1 \end{array} \right|; \left| \begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ X & 0 & 0 & 1 \\ X & 1 & 1 & 1 \\ 0 & 1 & X & 1 \end{array} \right|$$

применяется процедура (10.24), которая определяет в данном случае уже минимальный тест, гарантированно проверяющий схемную структуру, которая соответствует двухуровневой реализации булевой функции:

$$T = \left| \begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline 0 & 1 & 1 & 0 \\ 1 & 1 & X & 1 \\ \hline X & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ X & 1 & 1 & 1 \\ X & 0 & 0 & 1 \\ 0 & 0 & X & 1 \\ \hline X & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ X & 0 & 0 & 1 \\ X & 1 & 1 & 1 \end{array} \right| . \qquad (10.29)$$

Процедура верификации результатов, полученных при решении первых двух проблем, формулируется в виде

$$T \oplus L = C . \qquad (10.30)$$

Ее необходимость связана с ошибками, возникающими в процессе проектирования тестов. Процедура обязательна для верификации покрытия неис-

правностей при наличии в тесте символов X. Для полученных наборов (10.29) верификация путем выполнения операций $L_i^1 \oplus T\ (L_i^1), i = \overline{1,3}$, дает положительный результат, свидетельствующий о валидности теста для функционала:

$$L^1 \left|\begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right| \oplus T \left|\begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline 0 & 1 & 1 & 0 \\ 1 & 1 & X & 1 \\ \hline X & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ X & 1 & 1 & 1 \\ X & 0 & 0 & 1 \\ 0 & 0 & X & 1 \\ \hline X & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ X & 0 & 0 & 1 \\ X & 1 & 1 & 1 \end{array}\right| = C \left|\begin{array}{ccc|c} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ X & 0 & 1 & 0 \\ X & 1 & 0 & 0 \\ 0 & 1 & X & 0 \end{array}\right|.$$

Это означает, что любой вектор $L_i^1 \oplus T\ (L_i^1) \in C$ является кубом исходного покрытия функции. Следовательно, тест проверки неисправностей является валидным.

10.11. Модели элементарных триггерных схем

В общем случае, согласно теории автоматов, последовательностная схема (ПС) может иметь множество запрещенных (невозможных) A_δ, достижимых и устойчивых A_φ, неустойчивых A_γ состояний $A = \{A_\delta, A_\varphi, A_\gamma\}$, мощность которого определяется выражением $A = 2^n$, где n – число эквипотенциальных линий, охваченных глобальными обратными связями [2]. Поскольку неустойчивым называется состояние, из которого существует неоднозначный переход, то не совсем корректным является не только определение, но и его сущность. Поэтому правильнее будет дифференцировать все состояния автомата на два подмножества $A = \{A_\delta, A_\varphi\}$. Но при этом следует ввести классификацию переходов $T = \{T_\delta, T_\varphi\}$ на устойчивые и неустойчивые соответственно.

Определение 10.5. Начальное T_i и конечное T_j состояния перехода T_{ij} в общем случае определены в алфавите {0,1,X}, что в двоичном алфавите можно интерпретировать как совокупность вершин двоичных состояний.

Определение 10.6. Состояние перехода T_{ij} называется неустойчивым, если за сколь угодно большое, но конечное число проходов синхронного моделирования не зафиксировано двух одинаковых состояний автоматных переменных на соседних итерациях. (Проход синхронного моделирования – сквозная процедура выполнения прямой импликации всех примитивов

устройства без учета их временных параметров в целях определения троичного состояния всех линий относительно исходного.)

Определение 10.7. Состояние перехода T_{ij} называется устойчивым, если за конечное число проходов синхронного моделирования зафиксированы два одинаковых состояния автоматных переменных в соседних итерациях.

Теорема 10.9. *Кубическое покрытие автомата есть система устойчивых переходов, записанная в алфавите {0, 1, X}.*

Если относительно исходного состояния T_i синхронное моделирование за конечное число шагов анализа входного набора приводит к двум одинаковым конечным состояниям на паре соседних итераций, то согласно определению 10.7 формируется куб покрытия устойчивого перехода. Иначе (физически под воздействием неодинаковых задержек, влияния внешней среды), вместо P_{ij} может присутствовать любой другой переход T_{ir}, ($r \neq j$). Как в этом случае поступать? Согласно определению 10.6, синхронное моделирование выявляет автоматные переменные, принимающие различные значения на паре соседних итераций, которые представляют собой неэквивалентные конечные состояния переходов T_{ir}, T_{ij}. Далее необходимо применение оператора объединения к соседним векторам, которое определяет единственное, уже устойчивое, согласно определению 10.7, троичное состояние, включающее не менее двух двоичных. Для проверки устойчивости необходимо повторить процедуру моделирования. Количество повторений есть всегда конечное число с верхней границей итераций, определяемой наличием символов X на всех линиях, охваченных глобальными обратными связями. Но такой троичный вектор, по определению 10.5, принадлежит к классу устойчивых, что и следовало доказать.

Следствие 10.9. Для анализа множества кубов покрытия автомата не требуется итераций.

Такое заключение можно сделать, исходя из условий теоремы 10.9 построения КП автомата, в каждом кубе которого уже содержится ранее выполненная работа итеративного определения конечного всегда устойчивого троичного состояния ЦУ. Благодаря использованию кубов устойчивых переходов, другие процедуры анализа цифровых объектов, по сравнению с итеративными моделями, имеют значительную экономию не только памяти, но и времени проектирования диагностической информации. Платой за эти преимущества есть моделирование входного слова каждого куба покрытия относительно исходного состояния, что есть достаточно трудоемкая, но разовая работа. Задача проектирования адекватной и полной автоматной модели сводится к построению кубического покрытия, задающего все возможные и устойчивые переходы на множестве достижимых состояний, описываемых в алфавите {0, 1, X}, при любых входных наборах. Существуют особенности создания моделей основных типов триггеров, как самых распространенных и

сложных базовых функционалов, исходная информация о которых может быть представлена вентильной структурой, таблицей переходов, слабоформализованным словесным описанием.

Для исследования первого случая воспользуемся моделями асинхронных триггеров, представленных на рис.10.13.

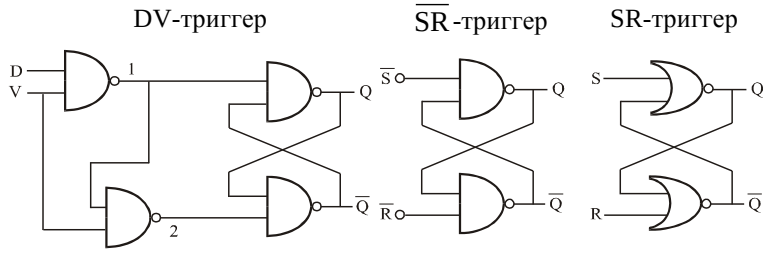


Рис.10.13. Асинхронные триггеры

Построение КП сводится к анализу результатов моделирования всех возможных входных наборов относительно достижимых и устойчивых состояний в целях получения матрицы переходов. Такая операция возможна для вентильных структур цифровых автоматов, имеющих не более двух-трех десятков входных, внутренних (охваченных глобальными обратными связями) и выходных переменных. Например, для $\overline{S}\overline{R}$-триггера такой анализ позволяет построить таблицу или граф переходов (рис.10.14).

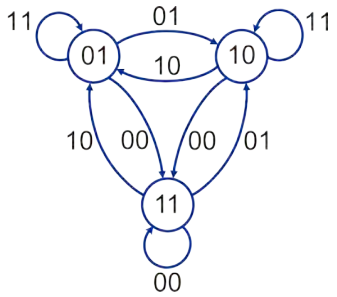


Рис. 10.14. Граф переходов триггера

Триггер имеет три устойчивых состояния и девять переходов. Состояние 00 является недостижимым, а переход из 11 при наличии входного слова 11 является недетерминированным и, в зависимости от соотношения задержек в цепях обратных связей, может выполняться в состояние 01 или 10. Поэтому будем считать конечным значением такого перехода вектор XX. Представленному графу соответствует однотактное покрытие А в табл. 10.1. Первый куб, например, описывает безусловный переход из состояния XX (00, 01, 10, 11) в 11. Поскольку значение 00 недостижимо для данного автомата, нет ошибки в записи такого куба для целей прямой импликации. С другой стороны, отсутствие полного перебора исходных состояний значительно уменьшает число строк. Последующая конкатенация состояний соответ-

ствующих переменных, определенных в моменты (t-1,t), дает двухтактное КП (B) в табл. 10.1.

Таблица 10.1

Двухтактные покрытия триггерных структур

$\bar{S}$	$\bar{R}$	$t-1$ Q	$t-1$ $\bar{Q}$	t Q	t $\bar{Q}$	$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$	$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$	S	R	Q	$\bar{Q}$
0	0	X	X	1	1	0	0	I	I	0	0	1	1	1	1	0	0
0	1	X	X	1	0	0	1	I	O	0	1	1	0	0	1	1	0
1	0	X	X	0	1	1	0	O	I	1	0	0	1	1	0	0	1
1	1	0	1	0	1	1	1	Q	J	1	1	Q	J	0	0	Q	J
1	1	1	0	1	0	1	1	J	Q	1	1	J	Q	0	0	J	Q
1	1	1	1	X	X	1	1	B	B	1	1	B	B	0	0	A	A
					A				B				C				D

Так как буквы O, I обозначают переход из произвольного состояния X в 0 и 1, что тождественно установке автоматной переменной в нуль и единицу в такте t, описываемой символами 0 и 1, последнее покрытие можно привести к виду C, предложенному в табл. 10.1. Здесь в качестве иллюстрации взаимодействия разрешенных переходов и состояний представлено проектирование графа переходов. Формализм построения модели содержит звенья: исходное описание – таблица переходов – двухтактное КП – минимизированное двухтактное покрытие.

Применяя аналогичную процедуру к SR-триггеру, мы можем записать его поведение кубами D в табл.10.1. На рис.10.13 представлен DV-триггер, имеющий всего два устойчивых и достижимых состояния, относительно которых сокращенная таблица переходов имеет строки, обозначенные столбцом A табл. 10.2.

Таблица 10.2

Сокращенная таблица переходов триггеров

D	V	$t-1$ Q	$t-1$ $\bar{Q}$	t Q	t $\bar{Q}$	D	V	Q	$\bar{Q}$	D	V	Q	$\bar{Q}$
X	0	0	1	0	1	X	0	Q	J	X	0	S	S
X	0	1	0	1	0	X	0	J	Q	1	1	1	0
1	1	X	X	1	0	1	1	I	O	0	1	0	1
0	1	X	X	0	1	0	1	O	I				
					A				B				C

Произведя конкатенацию соответствующих разрядов выходных переменных в моменты t-1, t, получим кубическое покрытие, содержащее четыре куба столбца B в табл.10.2. Принимая во внимание недостижимость автоматом

состояний 00, 11 при наличии входного слова X0, можно говорить только о сохранении значений 01, 10, если записать куб X0SS. В двух последних строках символы O, I заменяются на 0, 1, в силу приведенных выше рассуждений. После таких модификаций покрытие триггера принимает вид C в табл. 10.2.

Структурная сложность вентильного эквивалента схемы двухступенчатого, синхронизированного задним фронтом JK-триггера не позволяет построить его покрытие на основе анализа результатов моделирования. Таким образом, исчерпывающее словесное описание поведения объекта и элементарные знания A-алфавита есть необходимая информация для проектирования непосредственно минимального двухтактного КП, что и представлено рис.10.15.

J	C	K	Q1	Q
0	1	0	S	S
0	1	1	0	S
1	1	0	1	S
1	1	1	0	J
1	1	1	1	Q
X	0	X	Q	0
X	0	X	J	1

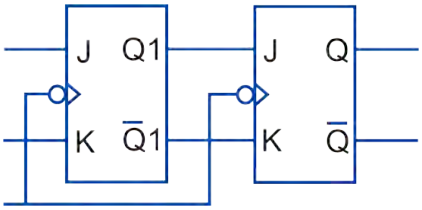


Рис. 10.15. Модель JK-триггера

Начальное описание триггера: при наличии на синхровходе (C) значения 1 формируется состояние первой ступени, которая обозначена переменной Q1; при этом два нуля на входах J, K запрещают запись информации как в первую, так и во вторую ступень (обозначенную символом Q), которые сохраняют свое значение (куб 1); сигналы 01(10) при C=1 формируют 0(1) – состояние Q1 в момент t при неизменном Q (кубы 2,3); если JCK=111, сохраняется состояние второй ступени, по отношению к которой инвертируется первая в момент t (кубы 4,5); при отрицательном потенциале на синхровходе сохраняется состояние Q1, содержимое которого переписывается в Q на автоматном такте t (кубы 6,7). Пользователь может иметь ошибочные знания об объекте и/или может допустить неточность в формировании покрытия. Необходима их проверка на заведомо верных эталонах. Такими могут быть модели существующих и признанных систем моделирования или исправные микросхемы. Любая верификация не дает уверенности в полноте и адекватности моделей. Поэтому формирование библиотеки примитивов и микросхем всегда есть самая ответственная и интеллектуальная работа, где допущенная ошибка может стоить больших временных и материальных затрат. На рис.10.16 представлены модели других триггеров. В отличие от физического устройства T-триггера, его модель имеет смысл только при наличии дополнительного входа установки в нуль (единицу).

D	C	Q	$\overline{Q}$
0	E	0	1
1	E	1	0
X	F	S	S

R	T	C	Q	$\overline{Q}$
0	X	X	0	1
1	0	E	S	S
1	1	E	P	P
1	1	F	S	S

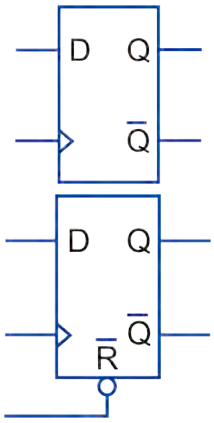


Рис. 10.16. Модели синхронных триггеров

Представленные модели триггеров не учитывают возникновение состязаний при одновременном изменении сигналов на входах. Это связано с применением покрытий для решения таких задач, для которых достаточно таблицы устойчивых переходов. Для реальных схем существует потребность анализа переходных процессов в целях выявления критических состязаний. Здесь КП последовательностного примитива должно учитывать переходные процессы. Например, при изменении сигнала на входе D и наличии переднего фронта на C в триггере (рис. 10.16) возникают состязания, переводящие его в неопределенное состояние по выходам. Данное обстоятельство может быть описано дополнительным кубом PEXX, который следует включить в покрытие. А-алфавит позволяет не только компактно задавать устойчивые переходы, но в дополнение к ним адекватно описывать все входные изменения, приводящие к неопределенным состояниям выходов. При проектировании КП всех основных типов триггеров (D-, T-, SR-, JK-) использовался эвристический подход, позволяющий квалифицированному проектировщику создавать компактную и адекватную модель последовательностного примитива. Другие триггерные схемы (микросхемы), как правило, представляют собой сочетание перечисленных типов. Поэтому построение покрытий для них сводится к композиции кубов, реализующих функции отдельных триггеров. На рис.10.17 в качестве примера представлено покрытие комбинированного триггера.

10.12. Методы построения кубических покрытий и тестов для структурно-функциональных моделей цифровых устройств

Предлагаются методы построения кубических покрытий для структурно-функциональных моделей цифровых устройств. Методы позволяют строить КП и проектировать проверяющие тесты для всех константных неисправностей существенных линий, полученных для схемы, представленной в виде системы булевых уравнений. Объект диагностирования – триггерные структуры, представленные в виде схемного и функционального (булевые уравне-

ния в формате BNF, являющемся подмножеством языков VHDL, Verilog) описания. Математический аппарат для представления поведения объекта диагностирования – кубическое исчисление. Задача – построение адекватного кубического покрытия для триггерной структуры и проектирование проверяющего теста.

S	D	C	R	Q	$\overline{Q}$
0	X	X	0	1	1
0	X	X	1	1	0
1	X	X	0	0	1
1	Q	E	1	0	1
1	J	E	1	1	0
1	X	F	1	S	S
1	P	E	1	X	X
J	X	F	1	B	B

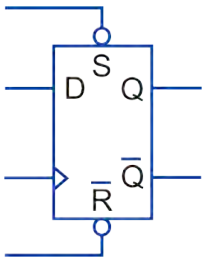


Рис. 10.17. Модель комбинированного триггера

Структура алгоритма построения покрытия триггерной схемы, представленная на рис. 10.18, включает следующие этапы:

1. Производится анализ схемной структуры в целях ее дальнейшей декомпозиции на комбинационные и последовательностные подсхемы. Данный анализ проводится либо по вентильной схеме, либо по булевым уравнениям.

2. Схема проверяется на предмет наличия глобальных обратных связей (ГОС). Данная проверка также осуществляется либо по вентильной структуре, либо по булевым уравнениям.

3. При наличии ГОС в схеме производится их псевдоразрыв, в результате ГОС будет представлена одним псевдовходом и одним псевдовыходом.

4. Для выявленных комбинационных и последовательностных подсхем строятся собственные КП с учетом псевдовходов и псевдовыходов при наличии ГОС. Методы построения КП: по системе булевых уравнений, по таблице состояний, по вентильной схеме. Выбор метода зависит от структуры рассматриваемой схемы (количества и типов подсхем). Так, при наличии в схеме одной триггерной структуры наиболее подходящим является метод построения КП по системе булевых уравнений. При наличии двух и более триггерных структур (ступеней) в схеме данный метод используется для построения КП ступеней.

Они необходимы для получения общего КП схемы по π-алгоритму. Имеется возможность получить общее КП для многоступенчатых триггеров по системе булевых уравнений при условии, что КП ступеней триггера отличаются только по входу синхронизации. В таком случае нет надобности приме-

нять π-алгоритм. Данный метод также подходит для построения КП комбинационных подсхем.

Рис. 10.18. Структура алгоритма построения покрытия триггерной схемы

5. После получения КП для всех подсхем схемы выполняется суперпозиция КП ступеней для построения общего КП по π-алгоритму. В целях обеспечения временной функции автомата для входа синхронизации используется двухтактный алфавит, либо данный вход представляется двумя переменными в соседних временных фреймах.

6. Построенное покрытие минимизируется до получения тупикового КП.

7. Для верификации полученных КП используется среда моделирования Active-HDL, где схема представляется в виде системы логических уравнений либо на уровне потока данных.

Подробное описание, а также преимущества и недостатки каждого метода будут рассмотрены далее.

Триггерные структуры формализуются к виду одноступенчатых и многоступенчатых примитивных автоматов (ПА).

Для ПА необходимо построить кубические покрытия, которые в дальнейшем будут использоваться для построения проверяющих тестов и моделирования неисправностей.

10.12.1. Принцип построения кубических покрытий одноступенчатых триггерных структур.

Алгоритм построения КП для ПА (ступеней триггерной структуры):

1. Булевы уравнения в формате BNF привести к формату аппарата описания булевых уравнений, принятому в дискретной математике (ДМ). Символы формата (*,+,!,=) переводятся в $(\wedge, \vee, \bar{\ }, =)$соответственно.

2. По булевым уравнениям ступеней триггерной структуры, описанных в формате BNF, записать общую функцию структуры в виде булевого уравнения.

3. Минимизировать полученное уравнение по правилам ДМ.

4. По полученному уравнению построить общее КП структуры.

5. Минимизировать КП и проверить его полноту с помощью карт Карно [43].

Работу алгоритма построения КП для ПА проиллюстрируем на примере триггера-защелки Latch (C_high, D) с синхронизацией по переднему фронту, схема которого представлена на рис.10.19.

Формат BNF:

S=D*C; R=C*!D;

Q(t)=S+(!R*Q(t-1)).

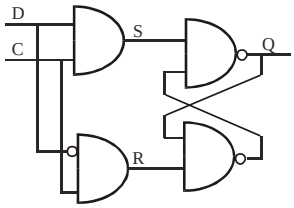


Рис. 10.19. Схемное описание Latch (C_high, D)

Выполняем первый пункт алгоритма, заменяя соответствующие символы:

$$S = D \wedge C;$$

$$R = C \wedge \overline{D};$$

$$Q = S \vee (\overline{R} \wedge Q(t-1)).$$

По булевым уравнениям ступеней, описанным в формате BNF, записываем общую функцию структуры в виде булевого уравнения путем подстановки промежуточных значений в функцию выхода. Заметим, что при подстановке запись символа конъюнкции можно опустить для наглядности.

$$Q = DC \vee \overline{C\overline{D}}Q(t-1).$$

Произведем минимизацию полученного уравнения по правилам ДМ:

$$Q = DC \vee (\overline{C} \vee D)Q(t-1) = DC \vee \overline{C}Q(t-1) \vee DQ(t-1) =$$
$$= DC \vee \overline{C}Q(t-1) \vee DQ(t-1)\overline{C} \vee DQ(t-1)C =$$
$$= DC(1 \vee Q(t-1)) \vee \overline{C}Q(t-1)(1 \vee D) = DC \vee \overline{C}Q(t-1).$$

По полученному уравнению строим кубическое покрытие D-триггера:

C	D	Q(t – 1)	Q
1	1	X	1
0	X	1	1
X	0	0	0
1	0	X	0
0	X	0	0

(10.31)

Минимизируем полученное КП и проверяем его полноту с помощью карт Карно. Видно, что 3-й куб поглощается 4-м и 5-м кубами:

Q(t – 1) CD	00	01
00	0	1
01	0	1
11	1	1
10	0	0

(10.32)

В результате получается полное и неизбыточное КП триггера (см. рис.10.19):

C	D	Q(t – 1)	Q
1	1	X	1
0	X	1	1
1	0	X	0
0	X	0	0

. (10.33)

Аналогичным образом можно построить КП для идентичных триггерных структур, результат построения одной из которых (Latch (R,S,C_high,D) представлен на рис.10.20, где показано схемное описание, двухтактное и однотактное КП соответственно.

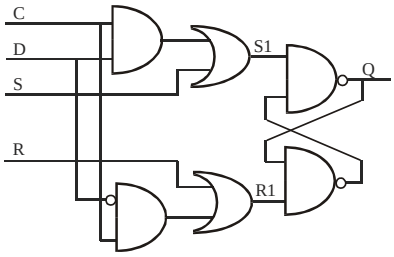
Формат BNF:

S1=S+(D*C);

R1=R+(C*!D);

Q(t)=S1+(!R1*Q(t-1)).

Преимущества данного метода заключаются в относительной простоте его использования, возможности одновременного выполнения процедур получения КП и его минимизации, возможности проверки адекватности модели по карте Карно [44].



C	D	S	R	Q
X	X	1	X	I
0	X	0	1	O
1	0	0	X	O
1	1	0	X	I
0	X	0	0	S

C	D	S	R	Q_{t-1}	Q_t
X	X	1	X	X	1
0	X	0	1	X	0
1	0	0	X	X	0
1	1	0	X	X	1
0	X	0	0	0	0
0	X	0	0	1	1

Рис. 10.20. Модель Latch (R,S,C_high,D)

Также можно сделать вывод о том, что данная методика идеально подходит для одноступенчатых ПА со сравнительно небольшим количеством входных переменных, так как при их большом количестве становится затруднительным процесс проверки полноты КП по карте Карно.

10.12.2. Особенности построения кубических покрытий для двухступенчатых триггерных структур.

Как было отмечено выше, проблема возникает при применении предложенного алгоритма к двухступенчатым триггерам. Проиллюстрируем его работу на примере D-триггера Flip-flop (C_high,D) с синхронизацией по переднему фронту, схема которого изображена на рис.10.21.

Формат BNF:

S=C*Q1(t-1);

R=C*!Q1(t-1);

Q(t)=S+(!R*Q(t-1));

S1=D*!C;

R1=!C*!D;

Q1(t)=S1+(!R1*Q1(t-1));

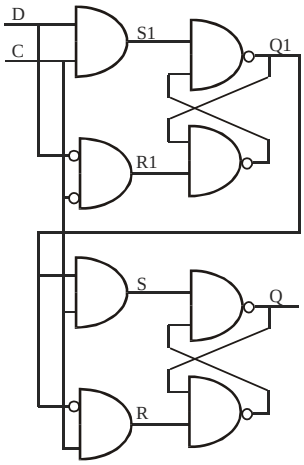


Рис. 10.21. Схемное описание двухступенчатого Flip-flop (C_high,D)

Аналогичным образом производятся преобразования к аппарату ДМ:

$$S = C \wedge Q1(t-1);$$

$$R = C \wedge \overline{Q1}(t-1);$$

$$Q(t) = S \vee (\overline{R} \wedge Q(t-1));$$

$$S1 = D \wedge \overline{C};$$

$$R1 = \overline{C} \wedge \overline{D};$$

$$Q1 = S1 \vee (\overline{R1} \wedge Q1(t-1)).$$

Далее для каждой функции, присутствующей в системе булевых уравнений (ступени триггера), выполняется подстановка промежуточных значений и ее минимизация:

$$Q1 = (D \wedge \overline{C}) \vee (\overline{\overline{C} \wedge \overline{D}} \wedge Q1(t-1)) = D\overline{C} \vee (C \vee D)Q1(t-1) =$$

$$= D\overline{C} \vee CQ1(t-1) \vee DQ1(t-1) = D\overline{C} \vee CQ1(t-1) \vee DQ1(t-1)\overline{C} \vee DQ1(t-1)C =$$

$$= D\overline{C}(1 \vee Q1(t-1)) \vee CQ1(t-1)(1 \vee D) = D\overline{C} \vee CQ1(t-1);$$

$$Q = C \vee Q1 \vee (\overline{C \wedge \overline{Q1}} \wedge Q(t-1)) = CQ1 \vee \overline{C}Q(t-1) \vee Q1Q(t-1) =$$

$$= CQ1 \vee \overline{C}Q(t-1) \vee Q1Q(t-1)C \vee Q1Q(t-1)\overline{C} = CQ1 \vee \overline{C}Q(t-1).$$

Исходя из полученных результатов для ступеней, получим общее кубическое покрытие триггера:

$$Q_{общ} = C(D\overline{C} \vee CQ1(t-1)) \vee \overline{C}Q(t-1) = CD\overline{C} \vee CCQ1(t-1) \vee \overline{C}Q(t-1) =$$

$$= CQ1(t-1) \vee \overline{C}Q(t-1).$$

По последнему уравнению строится КП:

C	Q1(t – 1)	Q(t – 1)	Q(t)
0	X	0	0
1	0	X	0
1	1	X	1
0	X	1	1

. (10.34)

По полученному кубическому покрытию триггера из (10.34) можно определить его принцип функционирования (по переднему фронту синхронизации), однако в процессе получения и минимизации общего булевого уравнения происходит потеря входной переменной.

Можно сделать вывод, что при выполнении п. 2,3 алгоритма невозможно получить общее КП путем подстановки всех значений в общее уравнение, так как при замене Q(t-1) на Q(t) и в процессе минимизации уравнения теряются входные переменные, без которых в дальнейшем невозможно построение теста. Потеря входной переменной в приведенном примере объясняется неявной зависимостью состояния 2-й (выходной) ступени от входа синхронизации и состояния 1-й (внутренней) ступени, что в процессе минимизации

булевого уравнения позволяет элиминировать входную переменную, от которой зависит состояние первой ступени. Булевы уравнения не привязаны к структурно-временной модели триггера, поэтому при их минимизации с применением правил ДМ могут быть утеряны входные переменные, что приведет к бесполезности полученного КП для последующего тестирования. Следовательно, необходим анализ структурно-временной модели.

Для этого рассмотрим следующий алгоритм построения общего КП:

1. Построение КП для всех ступеней триггерной структуры по булевым уравнениям каждой ступени.

2. Построение общего КП путем объединения КП ступеней в одном такте.

Работу алгоритма продемонстрируем на примере триггера на рис.10.21.

Выполнение п.1 алгоритма. Получение двух КП для каждой ступени:

C	D	Q1(t – 1)	Q1
0	1	X	1
1	X	1	1
1	X	0	0
0	0	X	0
X	0	0	0

; (10.35)

C	Q1	Q(t – 1)	Q(t)
1	1	X	1
0	X	1	1
0	X	0	0
1	0	X	0
X	0	0	0

. (10.36)

В (10.35), (10.36) приведены кубические покрытия для первой и второй ступеней соответственно. Анализируя полученные КП ступеней, можно заметить, что они отличаются только по входу синхронизации. Следовательно, можно построить общее КП, объединив эти два КП в одно, и представив триггерную структуру в одном автоматном такте. При этом происходит исключение несущественных внутренних переменных, что соответствует работе алгоритма построения КП для триггерных структур [2].

Аксиома 10.1. Объединение КП ступеней производится только в том случае, когда приведенные КП отличаются только по входу синхронизации.

В результате получается общее КП:

C(t – 1)	C(t)	D	Q(t – 1)	Q(t)
0	1	1	X	1
0	1	0	X	0
X	0	X	0	0
X	0	X	1	1
1	1	X	0	0
1	1	X	1	1

. (10.37)

Можно сделать вывод, что предложенная методика применима для частного случая – многоступенчатых триггерных структур, в которых все ступени имеют одинаковую структуру ПА. Большинство триггерных структур именно такого типа, поэтому для них может быть применен данный алгоритм.

Предложенный алгоритм эффективно обрабатывает двухступенчатые триггерные структуры, имеющие одинаковые типы ПА.

10.12.3. Развитие π-алгоритма.

Проблема возникает при наличии трех и более ступеней и различных типов ПА. Для обработки таких структур необходимо произвести их разбиение на подсхемы, состоящие из последовательностных и комбинационных элементов, независимых друг от друга, при этом может наблюдаться сокращение входов для комбинационных подсхем из-за разрыва связей между триггерами. В результате некоторые вентили могут быть удалены из схемы. Для такой декомпозиции необходимо выделить функциональные элементы структуры, являющиеся примитивными автоматами на два состояния – синхронные и асинхронныеи триггеры (ступени триггерной структуры), и примитивные элементы, являющиеся фрагментами вентильной логики. Построение общего КП произведем с помощью π – алгоритма, относящегося к прямым методам анализа комбинационных схем. Однако данный метод необходимо модифицировать в целях его адаптации к последовательностным схемам с точки зрения их временнуй зависимости. В качестве примитивных элементов (ПЭ) будут выступать ПА (ступени триггерной структуры) и элементы вентильной логики. На первом этапе необходимо построить КП для всех ступеней триггерной структуры. Второй этап – суперпозиция – получение общего КП по известным КП ступеней с применением π-алгоритма. Комбинационные подсхемы, не относящиеся к триггерам в схеме, будут рассматриваться отдельно. Для них необходимо привести покрытия функций, реализуемых всеми типами логических элементов (в дальнейшем будем называть их примитивами). Далее рассматриваются двухвходовые примитивы, которые представлены таблицами истинности 10.3. Входы примитивов обозначены символом I, выходы – O. Пунктирной линией в таблице отделены единичные и нулевые кубы покрытия. Например, для примитива И-НЕ единичные кубы покрытия представлены в строках 3,4, они задают все возможные наборы входов (01, 10, 00), порождающие на выходе примитива значение 1. Нулевое покрытие содержит один куб, показывающий, что при входной комбинации (11) на выходе примитива появится значение 0. Аналогично интерпретируются остальные кубы покрытий примитивов в таблице. Таким образом, любой нулевой или единичный кубы покрытия могут быть построены для примитива, реализующего произвольную логическую функцию от произвольного числа элементов.

Проиллюстрировать основные пункты π-алгоритма для последовательностных схем можно на примере трехступенчатого триггера Flip-Flop(C_high,D),

схема которого представлена на рис.10.22. В качестве ПЭ рассматриваются ПА, являющиеся ступенями триггерной структуры.

Таблица 10.3

Таблицы истинности двухвходовых примитивов

НЕ	
I	O
0	1
1	0

ИЛИ		
I	I	O
1	X	1
X	1	1
0	0	0

И		
I	I	O
0	X	0
X	0	0
1	1	1

ИЛИ – НЕ		
I	I	0
1	X	0
X	1	0
0	0	1

И – НЕ		
I	I	O
0	X	1
X	0	1
1	1	0

Формат BNF:

```
S2_tmp0=C_tmp0*Q3_tmp0(t-1);
Q2_tmp0(t)=!S2_tmp0+(D_tmp0*Q2_tmp0(t-1));                    (10.38)
R3_tmp0=!(D_tmp0*Q2_tmp0(t-1));
Q3_tmp0(t)=!C_tmp0+(R3_tmp0*Q3_tmp0(t-1));
Q(t,0)=!Q3_tmp0(t)+(Q2_tmp0(t)*Q(t-1));
```

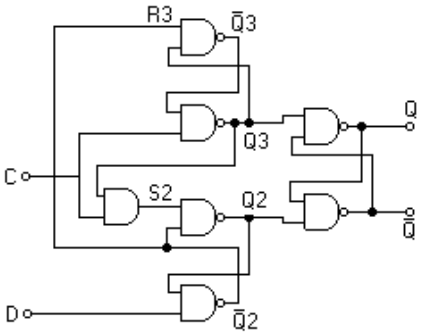


Рис. 10.22. Схемное описание трехступенчатого Flip-Flop(C_high,D)

Несмотря на то, что в данной схеме имеется три триггерные структуры, этот триггер срабатывает за 2 такта за счет параллельного взаиморасположения ступеней первого ранга (с выходами Q2 и Q3).

1. Производится разбиение триггерной структуры на независимые подсхемы (декомпозиция схемы), для этого выделяем части подсхемы с обратными связями и заменяем их на функциональные элементы (ПА) с покрытиями С, являющиеся ступенями этой структуры (рис.10.23). Символами Т1, Т2, Т3 обозначаются ступени триггерной структуры (ПА).

2. Производится ранжирование и нумерация ПА и линий. Сначала отмечаются внешние входы, затем выходы тех ПА, входы которых занумерованы. Аналогичным образом выполняется ранжирование ступеней триггерной структуры. На первом шаге нумеруются ступени, выходы которых являются

внешними. Процедура заканчивается полной нумерацией ПА и линий схемы, как показано на рис. 10.23, где все линии и ПА соответственно пронумерованы.

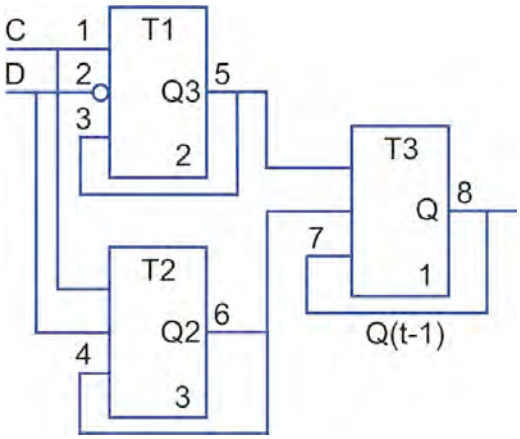


Рис. 10.23. Модель трехступенчатой триггерной структуры

3. Следующим шагом является построение кубических покрытий для каждой из ступеней триггерной структуры, представленной на рис. 10.23, что показано в выражениях (10.39)-(10.41) для первой, второй и третьей ступеней соответственно.

C(t – 1)	C(t)	D	Q3(t – 1)	Q3
X	0	X	X	1
0	1	0	1	1
0	1	1	X	0
1	1	X	0	0
1	1	X	1	1

; (10.39)

C(t – 1)	C(t)	D	Q2(t – 1)	Q2
X	0	X	X	1
0	1	1	1	1
0	1	0	X	0
1	1	X	0	0
1	1	X	1	1

; (10.40)

Q3	Q2	Q(t – 1)	Q
0	1	X	1
1	0	X	0
1	1	0	0
1	1	1	1

. (10.41)

4. Производится заполнение исходной таблицы покрытиями ПА, в которых точки имеют значение символа X, как показано в табл.10.4. Здесь вход синхронизации представлен в двухтактном алфавите, однако использования

этого алфавита можно избежать путем разложения входа на две входные переменные в соседних временных тактах.

Таблица 10.4

C_i	1	2	3	4	5	6	7	8
C1	.	.	.	.	0	1	X	1
	.	.	.	.	1	0	X	0
	.	.	.	.	1	1	0	0
	.	.	.	.	1	1	1	1
C2	O	X	.	X	.	1	.	.
	E	0	.	1	.	1	.	.
	E	1	.	X	.	0	.	.
	J	X	.	0	.	0	.	.
	J	X	.	1	.	1	.	.
C3	O	X	X	.	1	.	.	.
	E	1	1	.	1	.	.	.
	E	0	X	.	0	.	.	.
	J	X	0	.	0	.	.	.
	J	X	1	.	1	.	.	.

5. Строится вторая таблица 10.4, где исходный вектор решений E0, равный X на всех координатах, пересекается с каждым кубом t покрытия ПА $E0 \cap C_t^i$, который имеет только внешние выходы. В данном примере таким является примитив C1 из табл.10.3. Пересечение множества векторов Ei, полученного после обработки покрытия Ci, с кубами ПА $E^i \cap C_t^i$, связанного с уже обработанными ПА. Пересечение строки решения с кубом покрытия пусто, если есть пустое пересечение хотя бы по одной координате. Если вектор решения имеет символы X по выходным координатам анализируемого КП C^{i+1}, то он освобождается от пересечения и переносится в таблицу формируемых кубов E^{i+1}.

6. Выполнение операторов минимизации и поглощения на множестве полученных векторов после обработки очередного примитива дает существенное уменьшение числа промежуточных решений.

7. После этого из векторов решения удаляются столбцы, соответствующие внутренним переменным, к строкам покрытия E на оставшемся множестве переменных применяется оператор минимизации и поглощения (табл. 10.5).

Входные переменные, определенные на всех кубах символами X, также исключаются из результирующего покрытия, как несущественные. Полученная совокупность векторов на множестве входных и выходных существенных переменных есть искомое кубическое покрытие триггерной структуры, которое в общем случае является не минимальным, но тупиковым решением.

Полученное КП должно быть отверифицировано, для чего в САПР Active-HDL вводится модель триггера и моделирующие воздействия и результаты моделирования (временная диаграмма) сравниваются с упомянутым КП.

Таблица 10.5

E^i	1	2	3	4	5	6	7	8
E0	X	X	X	X	X	X	X	X
E1					0	1	X	1
					1	0	X	0
					1	1	0	0
					1	1	1	1
E2	E	1	X		0	1	X	1
	J	X	0		0	1	X	1
	O	X	X		1	0	X	0
	E	0	1		1	0	X	0
	J	X	1		1	0	X	0
	O	X	X		1	1	0	0
	E	0	1		1	1	0	0
	J	X	1		1	1	0	0
	O	X	X		1	1	1	1
	E	0	1		1	1	1	1
	J	X	1		1	1	1	1

E^i	1	2	3	4	5	6	7	8
E3	A	1	X	X	0	1	X	1
	E	1	X	1	0	1	X	1
	I	1	X	1	0	1	X	1
	B	X	0	X	0	1	X	1
	I	1	X	1	0	1	X	1
	J	X	0	1	0	1	X	1
	A	0	X	X	1	0	X	0
	B	X	X	0	1	0	X	0
	E	0	1	X	1	0	X	0
	I	0	1	0	1	0	X	0
	I	0	1	X	1	0	X	0
	J	X	1	0	1	0	X	0
	O	X	X	X	1	1	0	0
	A	1	X	1	1	1	0	0
	B	X	X	1	1	1	0	0
	A	0	1	X	1	1	0	0
	E	X	1	1	1	1	0	0
	I	0	1	1	1	1	0	0
	B	X	1	X	1	1	0	0
	I	1	1	1	1	1	0	0
	J	X	1	1	1	1	0	0
	O	X	X	X	1	1	1	1
	A	1	X	1	1	1	1	1
	B	X	X	1	1	1	1	1
	A	0	1	X	1	1	1	1
	E	X	1	1	1	1	1	1
	I	0	1	1	1	1	1	1
		X	1	X	1	1	1	1

Результаты верификации показали несоответствие КП, полученного по π-алгоритму, модели триггера (рис. 10.24):

C(t – 1)	C(t)	D	Q(t – 1)	Q(t)
0	X	1	X	1
X	1	1	X	1
0	X	0	X	0
X	1	0	X	0
X	0	X	0	0
X	0	X	1	1
0	X	X	0	0
0	1	X	1	1

. (10.42)

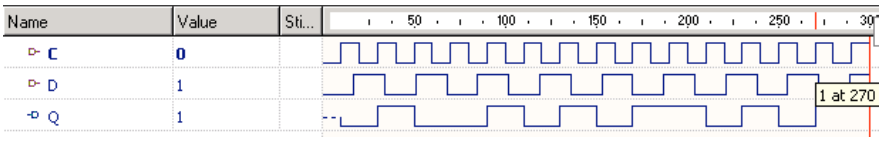


Рис. 10.24. Временная диаграмма модели Flip-Flop(C_high, D)

Таким образом, можно сделать вывод, что π-алгоритм нуждается в дополнительной модификации в целях его корректного применения к последовательностным схемам, имеющим как установочные, так и внутренние состояния, наличие которых, скорее всего, и является источником проблемы синтеза тестов верификации.

Так как последовательностные схемы имеют временную зависимость, т.е., в отличие от комбинационных схем, для состояния схемы небезразлично, в каком временном фрейме она находится, то продвижение назад, выполняемое π-алгоритмом для комбинационных структур, заменяется на продвижение вперед для триггерных структур. В остальном работа π-алгоритма аналогична описанной ранее процедуре.

При продвижении вперед (от кубических покрытий Q2 и Q3 к кубическим покрытиям выходной ступени Q) будем пересекать только непротиворечивые кубы, имеющие одинаковые значения по входным координатам ступеней первого ранга, а для получения выходного значения Q входными координатами будем считать значения функций Q2 и Q3, представленные как внутренние состояния:

C_{t-1}	C_t	D	$Q2_{t-1}$	$Q2_t$	$Q3_{t-1}$	$Q3_t$	Q_t	Q_{t-1}
X	0	X	X	1	X	1	0	0
X	0	X	X	1	X	1	1	1
0	1	0	X	0	1	1	X	0
0	1	1	1	1	X	0	X	1
1	1	X	0	0	0	0	unreached	state
1	1	X	1	1	1	1	0	0
1	1	X	1	1	1	1	1	1

. (10.43)

Путем исключения несущественных внутренних переменных получим окончательное КП триггерной структуры:

C_{t-1}	C_t	D	Q_{t-1}	Q_t
X	0	X	0	0
X	0	X	1	1
0	1	0	X	0
0	1	1	X	1
1	1	X	0	0
1	1	X	1	1

. (10.44)

Данное кубическое покрытие совпадает с результатами моделирования схемы в САПР Active HDL.

С помощью приведенного алгоритма была обработана триггерная структура Flip-Flop (RA,SA,C_low,R,S,D).

Формат BNF:

S2_tmp0=SA_tmp0+(!C_tmp0*Q1_tmp0(t-1));

R2_tmp0=RA_tmp0+(!C_tmp0*!Q1_tmp0(t-1));

Q(t)=S2_tmp0+(!R2_tmp0*Q(t-1));

S1_tmp0=(((S_tmp0*C_tmp0)+(D_tmp0*C_tmp0))*!RA_tmp0)+SA_tmp0;

R1_tmp0=(((R_tmp0*C_tmp0)+(!D_tmp0*C_tmp0))*!SA_tmp0)+RA_tmp0;

Q1_tmp0(t)=S1_tmp0+(!R1_tmp0*Q1_tmp0(t-1)).

Схема структуры представлена на рис. 10.25.

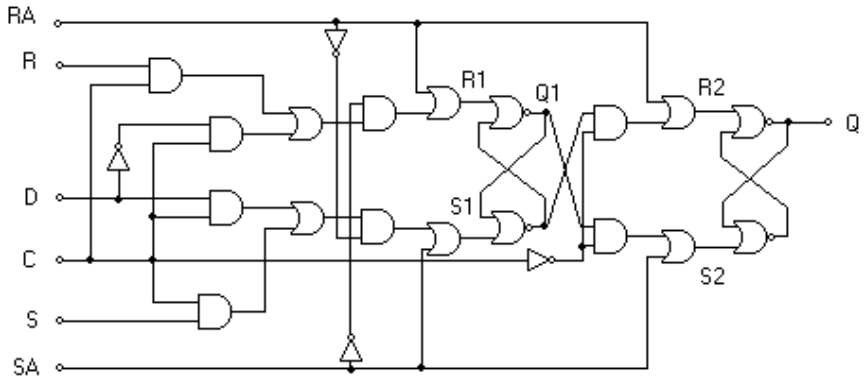


Рис. 10.25. Схемное описание Flip-Flop (RA,SA,C_high,R,S,D)

Пользуясь предложенным модифицированным π-алгоритмом, получили кубические покрытия для первой и второй ступеней соответственно (10.45), (10.46) и общее КП (10.47).

C	SA	RA	S	R	D	$Q1_{t-1}$	$Q1_t$
X	1	X	X	X	X	X	1
1	0	0	X	X	1	X	1
1	0	0	1	X	1	X	1
1	0	X	0	X	1	X	0
X	0	1	X	X	X	X	0
0	0	0	X	X	X	0	0
0	0	0	X	X	X	1	1

; (10.45)

C	SA	RA	Q1	Q_{t-1}	Q_t
0	0	X	0	0	0
X	0	X	X	1	1
0	X	X	1	1	1
1	0	1	X	0	0
1	0	0	X	0	0
1	0	0	X	1	1

; (10.46)

C_{t-1}	C_t	SA	RA	S	R	D	Q_{t-1}	Q_t
1	0	0	0	0	X	0	X	0
1	0	0	0	X	X	1	X	1
1	0	0	0	1	X	X	X	1
X	X	1	X	X	X	X	X	1
X	X	0	1	X	X	X	X	0
X	1	0	0	X	X	X	1	1
X	1	0	0	X	X	X	0	0
0	0	0	0	X	X	X	1	1
0	0	0	0	X	X	X	0	0

. (10.47)

Предложенный модифицированный π-алгоритм имеет преимущество перед классическим, заключающееся в возможности обработки сильно последовательностных схем.

Построение таблицы состояний по системам булевых уравнений вручную– трудоемкий и длительный процесс: количество строк в таблице состояний вычисляется по формуле $N = 2 * 2^n$, где n – количество входных линий. При таком подходе для получения достоверной модели необходимо вычислять значения внутренних линий. При построении таблицы состояний по структурной (схемной) модели триггера отпадает необходимость рассмотрения внутренних линий, что приводит к некоторой оптимизации процедуры получения КП.

Однако эту процедуру можно автоматизировать путем построения модели триггера в среде Active-HDL, подачи на полученную модель верифицирующей последовательности, являющейся множеством входных наборов для таблицы состояний и последующего анализа выходных значений триггера по волновой диаграмме либо по текстовому файлу с результатами моделирования (создание такого файла возможно при подключении библиотеки TEXTIO) [30].

Немаловажным преимуществом использования среды моделирования является возможность визуального выявления синхронной / асинхронной природы входов триггера, что при построении таблицы состояний вручную оказывается затруднительным.

Процедуру моделирования в VHDL продемонстрируем на примере трехступенчатой триггерной структуры Flip-Flop(C_high, D).

В первую очередь описывается архитектура триггера в формате .vhdl по имеющимся булевым уравнениям, представленным ранее в (10.38):

Листинг 10.1.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity C_highD is
        port (
                C: in STD_LOGIC;
```

```
                D: in STD_LOGIC;
                Q: out STD_LOGIC
        );
end C_highD;
—}} End of automatically maintained section
architecture C_highD of C_highD is
signal S2: std_logic;
signal Q3: std_logic;
signal Q2: std_logic;
signal R3: std_logic;
signal Q1: std_logic;
begin
        S2<=C and Q3;
        Q2<=not(S2) or (D and Q2);
        R3<=not(D and Q2);
        Q3<=not (C) or (R3 and Q3);
        Q1<=not(Q3) or (Q2 and Q1);
        Q<=Q1;
end C_highD;
```

Строим тесты для моделирования всех состояний триггера, представленные в листинге 10.2.

Листинг 10.2.

```
process
        begin
        C<= ‘0’,’1' after 10 ns;
        wait for 20 ns;
end process;
process
        begin
        D<=’0',’1' after 17 ns;
        wait for 34 ns;
end process;
```

По результатам моделирования в виде временных диаграмм получается КП ступеней триггерной структуры (рис. 10.26, 10.27, 10.28) соответственно. Общее КП триггера было получено ранее (см. рис. 10.24).

Анализируя полученную временную диаграмму, можно записать все состояния установки и сохранения триггера и, в конечном счете, записать общее кубическое покрытие для триггера, исходя из анализа его поведения, а также меняя список наблюдаемых сигналов во временной диаграмме, КП для всех ступеней триггера.

Полученные покрытия совпадают с покрытиями, полученными ранее (10.39)-(10.41) для этой триггерной структуры, а общее покрытие (10.44) совпадает с покрытием на рис. 10.24. Таким образом, мы провели верификацию полученных ранее кубических покрытий.

Для КП ступеней получена следующая временная диаграмма:

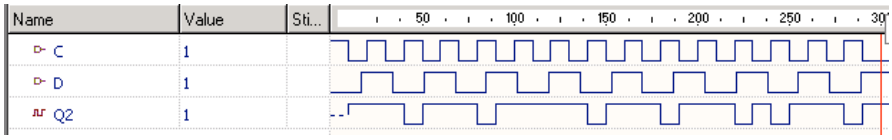


Рис. 10.26. Временная диаграмма КП первой ступени Flip-Flop(C_high, D)

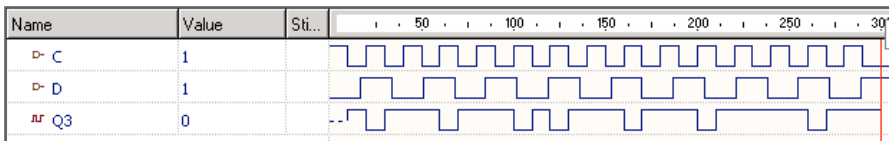


Рис. 10.27. Временная диаграмма КП второй ступени Flip-Flop(C_high, D)

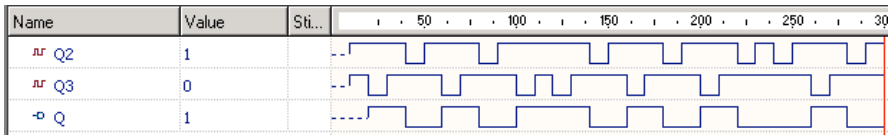


Рис. 10.28. Временная диаграмма КП третьей ступени Flip-Flop(C_high, D)

Исходя из анализа результатов тестирования, можно заметить, что наиболее полный тест для триггерных структур строится детерминированным методом. В табл. 10.6, 10.7 и 10.8 представлены тесты для триггерных структур, рассмотренных ранее: Latch (C_high, D), Latch (R,S,C_high,D) и Flip-Flop(C_high,D) соответственно.

Во втором столбце таблиц приведены построенные тесты, в третьем – проверяемые неисправности, четвертый столбец показывает процент проверенных неисправностей каждым тестовым набором, а именно – полноту теста, полученную с помощью дедуктивного метода моделирования неисправностей. Для всех рассмотренных ранее триггерных структур были получены 100% покрытия неисправностей.

10.13. Выводы

Предложены таблично-ориентированные алгоритмы генерации тестов для структурно-функциональных моделей цифровых устройств на основе активизации одномерных логических путей при использовании кубических покрытий (КП) функциональных элементов. Алгоритмы позволяют проектировать проверяющие тесты для всех неисправностей существенных линий схемы, полученной из системы булевых уравнений.

Таблица 10.6

Тесты для триггерной структуры Latch (C_high, D)

	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	FC,%
0	0	1	1	0	X	0	1	0	.	.	.	1	.	1	.	1	25.00
									.	.	.	1	.	1	.	1	25.00
1	0	1	0	1	0	1	0	0	.	.	1	1	1	1	1	1	50.00
									.	.	1	1	1	1	1	1	50.00
2	0	1	1	1	0	1	1	1	.	.	0	0	.	0	0	0	41.67
									.	.	X	X	1	X	X	X	91.67
3	0	1	0	0	1	0	0	1	.	.	X	0	0	0	X	0	66.67
									.	.	X	X	X	X	X	X	100.00

Таблица 10.7

Тесты для триггерной структуры Latch (R,S,C_high,D)

	0	1	2	3	4	5	6	7	8	9	0	1	0	1	2	3	4	5	6	7	8	9	0	1	FC,%
0	0	1	1	0	0	0	X	0	0	0	1	0	.	.	.	.	1	1	.	1	1	.	.	1	25.00
													.	.	.	.	1	1	.	1	1	.	.	1	25.00
1	0	1	0	0	0	1	0	1	0	0	0	0	.	.	1	.	1	1	1	1	1	.	1	1	40.00
													.	.	1	.	1	1	1	1	1	.	1	1	40.00
2	0	1	1	0	0	1	0	1	0	0	1	1	.	.	0	.	.	0	.	0	.	.	0	0	25.00
													.	.	X	.	1	X	1	X	1	.	X	X	65.00
3	0	1	0	0	0	0	1	0	0	0	0	1	.	.	X	1	.	0	0	0	.	1	X	0	50.00
													.	.	X	1	1	X	X	X	1	1	X	X	80.00
4	0	1	0	1	0	0	1	0	0	1	0	0	.	.	.	0	1	.	.	.	1	0.	.	1	25.00
													.	.	X	X	1	X	X	X	1	X	X	X	90.00
5	0	1	0	0	1	0	0	0	1	0	0	1	.	.	.	.	0	.	.	.	0	.	.	0	15.00
													.	.	X	X	X	X	X	X	X	X	X	X	100.00

Таблица 10.8

Тесты для триггерной структуры Flip-Flop(C_high,D)

	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	FC,%
0	0	1	1	0	0	X	1	0	0	0	.	.	.	1	.	.	.	1	.	1	18.75
											.	.	.	1	.	.	.	1	.	1	18.75
1	0	1	0	1	0	0	0	1	0	0	.	.	1	1	.	1	1	1	.	1	37.50
											.	.	1	1	.	1	1	1	.	1	37.50
2	0	1	1	1	0	0	1	1	0	1	.	.	0	0	1	.	0	0	1	0	43.75
											.	.	X	X	1	1	X	X	1	X	81.25
3	0	1	1	0	1	1	1	0	1	1	.	.	0	0	X	0	0	0	X	0	62.50
											.	.	X	X	X	X	X	X	X	X	100.00

Алгоритмы ориентированы на обработку комбинационных и последовательностных примитивов, а также на синтез тестов для цифровых схем, содержащих упомянутые компоненты.
Представлены многочисленные примеры схем, для которых построены полные проверяющие тесты, верифицированные с помощью программных продуктов моделирования дефектов, которые создают таблицы неисправностей, используемые для поиска дефектов в цифровых системах.
Алгоритмы ориентированы на обработку цифровых компонентов невысокой структурной и функциональной сложности, содержащих не более 10 000 эквивалентных вентилей.

10.14. Литература

1.Pawel Tomaszewicz, Mariusz Rawski Self-Testing of User-Programmed FPGAs Based on the Concept of Linear Segments // IEEE Digital System Design. – 2001. – P. 236-242.

2. Хаханов В.И. Техническая диагностика элементов и узлов персональных компьютеров. – К.: ІЗМН, 1997. – 308 с.

3. Eduardas Bareiša, Vacius Jusas, Kęstutis Motiejūnas, Rimantas Šeinauskas. Functional Digital Systems Testing. – Kaunas:Technologija, 2006. – 281 p.

4. Ondřej Novák, Elena Gramatova, Raimund Ubar. Handbook of Testing Electronic Systems. –Czech Technical University Publishing House, 2005.–395 p.

5. Gizopoulos Dimitris, Paschalis A., Zorian Yervant. Embedded Processor-Based Self-Test.– Series: Frontiers in Electronic Testing, Vol. 28.–2004.– 217 p.

6. Nicolaidis Michael, Zorian Yervant, Pradhan Dhiraj K. On Line-Testing for VLSI.– Series: Frontiers in Electronic Testing, Vol. 11.– 1998.– 164 p.

7. Zorian Yervant. Multi-Chip Module Test Strategies.– Series: Frontiers in Electronic Testing . 1997.– Vol. 7.– 168 p.

8. Leena Singh, Peter Paterson. System-On-A-Chip Verification: Methodology and Techniques.– Springer.– 2001.– 372 p.

9. Kilts S. Advanced FPGA Design: Architecture, Implementation, and Optimization.– Wiley-IEEE Press.– 2007.– 352 p.

10. Samiha Mourad, Yervant Zorian. Principles of Testing Electronic Systems.– John Wiley & Sons, Inc. – 2000.– 411 p.

11. Abramovici M., Breuer M. A., Friedman A. D. Digital Systems Testing and Testable Design.– Wiley-IEEE Press.– 1994.– 672 p.

12. Xijiang Lin, Irith Pomeranz, Sudhakar M. Reddy. Techniques for Improving the Efficiency of Sequential Circuit Test Generation // International Conference on Computer-Aided Design Proceedings. – 2000. – P. 23-26.

13. Marinissen E.J., Zorian Y. IEEE Std 1500 Enables Modular SoC Testing // Design & Test of Computers, IEEE . – Jan.-Feb., 2009. –Volume: 26, Issue: 1. – P. 8 - 17.

14. Benso A., Bosio A., Di Carlo S., Di Natale G., Prinetto P. March Test Generation Revealed // Computers, IEEE Transactions on. – Dec., 2008. – Volume: 57 , Issue: 12. – P. 1704 – 1713.

15. Bose S., Agrawal P., Agrawal V.D. Deriving logic systems for path delay test generation // Computers, IEEE Transactions on. – Aug., 1998. – Volume: 47, Issue: 8. – P. 829 – 846.

16. Heon-Mo Koo, Prabhat Mishra. Functional Test Generation using Property Decompositions for Validation of Pipelined Processors // Design, Automation and Test in Europe, 2006. DATE '06. Proceedings. – March, 2006. – Volume: 1. – P. 1 – 6.

17. Wen C.H.-P., Li-C Wang, Kwang-Ting Cheng. Simulation-Based Functional Test Generation for Embedded Processors // Computers, IEEE Transactions on. – Nov., 2006. – Volume: 55 , Issue: 11. – P. 1335 – 1343.

18. Lingappan L., Jha N.K. Satisfiability-Based Automatic Test Program Generation and Design for Testability for Microprocessors // Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. – May, 2007. – Volume: 15 , Issue: 5. – P. 518 – 530.

19. Hosokawa T., Inoue R., Fujiwara, H. Fault-dependent/independent Test Generation Methods for State Observable FSMs //Asian Test Symposium. ATS '07. 16th. – Oct., 2007. – P. 275 – 280.

20. Hari S.K.S., Konda V.V.R., Kamakoti V., Vedula V.M., Maneperambil, K.S. Automatic Constraint Based Test Generation for Behavioral HDL Models // Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. – April, 2008. – Volume: 16 , Issue: 4. – P. 408 – 421.

21. Хаханов В.И., Скворцова О.Б., Пудов В.А., Масуд М.Д. Мехеди, Чамян А. Генерация тестов для последовательностных схем, имеющих триггерные структуры // Радиоэлектроника и информатика. – 2001. – №1. – С.96-99.

22. Хаханов В.И., Скворцова О.Б., Бабич А.В. Алгоритм построения кубических покрытий для схем с триггерными структурами (модифицированный П-алгоритм). Метод генетических алгоритмов для синтеза тестов верификации цифровых систем // АСУ и приборы автоматики. – 2001. – № 115. – С. 105-111.

23. Скворцова О.Б., Пудов В.А., Хак Х.М. Джахирул, Масуд М.Д. Мехеди. Генерация тестов для последовательностных схем, имеющих триггерные структуры // IV городская научно-техническая конференция “Актуальные проблемы современной науки в исследованиях молодых ученых Харьковщины”. – Харьков: ХГУ. – 2001. – С. 86-89.

24. Скобцов Ю.А., Сперанский Д.В. Структурно-аналитический подход в задачах диагностики синхронных последовательностных схем // Электронное моделирование. – 1980. - №4. – С.32-38.

25. Pomeranz I., Reddy S.M. The multiple observation time strategy // IEEE Transactions on Computers. - 1992. - №5. - P.627-637.

26. David R. Random testing of digital circuits // Marcel Dekker Inc. New York. – 1998. – 234 p.

27. Brown S., Vranesic Z. Fundamentals of Digital Logic with VHDL Design. The McGraw Hill. – 2000. – 530 p.

28. Бибило П.Н. Основы языка VHDL. – М.: Солон-Р, 2000. – 200 с.

29. Ashenden, Peter. Designer's Guide to VHDL. – Morgan Kaufman. – 1994. – 500 p.

30. Active-VHDL Series. Book 1. Reference Guide. – ALDEC Inc. – 1998. – P. 205.

31. Brown S.D., Vranesic Z.G. Fundamentals of digital logic with VHDL design. – USA: McGraw-Hill Companies.– 2000. – 840 p.

32. Hayes J. P. A systematic approach to multivalued digital simulation // ICCD-84: Proc. IEEE Int. Conf. Comput. – 1984. – N 4. – P. 177-182.

33. Ермилов В.А. Метод отбора существенных неисправностей для диагностики цифровых схем. Общие выражения для неисправностей, возможных при эксперименте // Автоматика и телемеханика.–1971.–№1.–С. 159-167.

34. Armstrong D. B. A deductive method of simulating faults in logic circuits // IEEE Trans. on Computers. –1972. – Vol. C-21, N 5. – P. 464-471.

35. Биргер А.Г. Многозначное дедуктивное моделирование цифровых устройств // Автоматика и вычислительная техника. – 1982. – №4. – С. 77-82.

36. Биргер А.Г., Гурвич Е.Т., Кузнецов С.С. Проверка кратных неисправностей комбинационных устройств // Автоматика и телемеханика. – 1975. – № 8. – С. 113-122.

37. Биргер А.Г., Гурвич Е.Т. Построение тестов для древовидных сетей автоматов // Автоматика и телемеханика. – 1981. – №8. – С.142-151; №9 – С. 151-161.

38. Levendel Y. H., Menon P. R. Comparison of fault simulation methods . Treatment of unknown signal values // Journal of Digital System. – 1980. – Vol. 4. – P. 443-459.

39. Levendel Y.H., Menon P.R. Test generation algorithms for nonprocedural computer description languages // FTCS-11: The 11th Ann. Int.: Simp. On Fault-Tolerant Computers. – 1981. – P.200-205.

40. Levendel Y.H., Menon P.R. Test generation algorithms for computer hardware description languages // IEEE Trans. Computer. – 1982. – Vol. C-31, № 7. – P. 557-588.

41. Breuer M.A., Friedman A.D. Functional level primitives in test generation // IEEE Trans. Computers. – 1980. – Vol. C-29, № 3. – P.223-234.

42. Abramovich M., Breuer M. A., Friedman A. D. Digital system testing and testable design. – Computer Science Press. – 1998.

43. Баранов С.И., Майоров С.А., Сахаров Ю.П., Селютин В.А. Автоматизация проектирования цифровых устройств. – Л.:Судостроение, 1979.– С.– 264.

44. Майоров С.А. Проектирование цифровых вычислительных машин. – М.:Изд-во Высш. шк., 1972. – 344 с.

11. AD HOC МЕТОДЫ ГЕНЕРАЦИИ ТЕСТОВ ДЛЯ ЦИФРОВЫХ СИСТЕМ

Представлены специальные (ad hoc) методы генерации тестов для компонентов цифровых систем, основанные на использовании эволюционных технологий и теории двухтактного кубического исчисления, позволяющих решать задачи синтеза тестов с заданной полнотой относительно константных неисправностей. Возможности предложенных методов продемонстрированы на валидных примерах обработки цифровых схем. Описаны процедуры и алгоритмы генерации тестов для последовательностных схем и цифровых систем, ориентированные на имплементацию в программные продукты.

11.1. Модификация метода генетических алгоритмов для синтеза тестов верификации цифровых систем

Проблема построения тестов для последовательностных устройств является одной из самых сложных в технической диагностике. За последние 5 лет разработано множество методов построения тестов для комбинационных и последовательностных схем, позволяющих добиться высокой полноты тестов [1-28]. Методы ветвей и границ сильно зависят от эвристик, используемых при направлении поиска, что делает их сложными и практически неприемлемыми для схем большого размера [29-34]. Методы символьного моделирования также дают хорошие результаты для небольших схем и являются практически непригодными для схем, содержащих несколько десятков триггеров [35-38]. Псевдослучайная генерация тестов дает хорошие результаты для больших комбинационных схем, однако применительно к последовательностным схемам не всегда дает положительные результаты [39-40]. Генетические алгоритмы показывают также хорошие результаты при генерации тестов для больших комбинационных схем, однако для последовательностных существуют проблемы. Они связаны с большим временем их работы. На основе сказанного выше, а также анализа работ в 1 разд. предлагается использовать метод генетических алгоритмов для уменьшения времени генерации тестов класса последовательностных устройств.

В данном подразделе описывается применение генетического алгоритма к проблеме построения тестов для последовательностных устройств. Этот подход относится к псевдослучайным методам и позволяет добиться сопоставимых с детерминированными методами результатов при структурной простоте реализации. Предлагаются модели и процедуры построения тестовых последовательностей для верификации цифровых систем, основанные на методе генетических алгоритмов и позволяющие получать тесты приемлемой полноты при высоком быстродействии для больших последовательностных схем. Программная реализация метода генерации тестов ориентирована на обработку цифровых схем большой размерности и дает

возможность существенно повысить скорость работы генетического алгоритма.

Классический генетический алгоритм, предложенный Голдбергом в [41], хорошо зарекомендовал себя при решении различных переборных задач типа NP полных. Часто генетический алгоритм рассматривают как алгоритм поиска максимума или минимума некоторой функции. В широком смысле термин «генетический алгоритм» применяется для обозначения любой математической модели, основанной на популяциях и использующей операции селекции и рекомбинации для построения новой выборки точек в пространстве поиска. Его суть заключается в попытке повторить природный путь улучшения характеристик живых организмов.

Опишем классическую реализацию генетического алгоритма. Пусть задана произвольная задача и ее пространство поиска, в котором будет производиться поиск. Решение задачи предполагает, что любая точка в пространстве решений может быть закодирована как последовательность $(x_1, x_2, \ldots, x_n)$, называемая особью, где переменные x_i, $i = \overline{1,n}$ являются генами. Произвольный набор особей называется популяцией. Каждой особи ставится в соответствие значение оценочной функции $f(x_1, x_2, \ldots, x_n)$, которая показывает, насколько близко данная особь приблизилась к решению поставленной задачи. Таким образом, решение задачи предполагает поиск экстремума некоторой многомерной функции f. На основе оценочных функций вычисляется фитнесс-функция, показывающая качество данной особи относительно других особей популяции. Процесс поиска решения обычно начинается со случайно сгенерированной популяции. Поиск решения задачи с помощью генетического алгоритма состоит из итеративного процесса порождения новой популяции на основании текущей. Для этого строится популяция следующего поколения. Это производится с помощью некоторых трансформирующих операций, под действием которых особи могут получить новые свойства и тем самым улучшить свою оценочную функцию. Обычно это операции скрещивания и мутации. Скрещивание – это операция объединения хромосом двух (или более) особей-родителей, в результате которой получаются особи-потомки с новыми свойствами. Например, пусть имеются особи 00100100 и XXZZXXZX (для удобства восприятия заменяем во второй особи 0 на X, 1 на Z). В качестве точки пересечения выбираем четвертую позицию. В результате получаем потомков 0010XXZX и XXZZ0100. Цель операции скрещивания – направить процесс поиска в область предположительного нахождения задачи. После скрещивания применяется операция мутации. Она реализуется как случайное изменение генов в особях всей популяции. Ее цель – направить поиск в неисследованные регионы пространства поиска и предотвратить сходимость процесса поиска к локальным минимумам.

Таким образом, процесс эволюции реализуется как замена (полная или частичная) особей популяции на новые, после чего снова происходит вычисление их оценочных и фитнесс-функций. Порождение новых популяций прекращается, когда найдено решение задачи. Следовательно, чтобы задать генетический алгоритм, необходимо определить понятия особи, популяции, операции скрещивания и мутации, задать оценочную функцию применительно к задачам технической диагностики генерации тестов: ген – один элемент (бит) в тест-векторе, принимающий значения (0,1,X); хромосома – тест-вектор; особь – тест; популяция – набор тестов. Очевидно, что эффективность генетического алгоритма зависит от размера популяции, метода выбора особей, скрещивания и мутации и вида оценочной функции.

Таким образом, генетические алгоритмы ориентированы для нахождения экстремума функции [41]. В зависимости от ее свойств: выпуклость, линейность, дифференцируемость выбирается соответствующий алгоритм поиска минимума или максимума. Если функция сложная, имеет много точек разрывов, то найти эффективный алгоритм поиска экстремума – трудная задача. Что касается дискретных функций, здесь можно найти экстремум, не делая никаких предположений о свойствах функции (линейность, самодвойственность) $T = (T^1, T^2, ..., T^r, ..., T^p)$, где T – тест, полученный в результате выполнения p итераций методом генетических алгоритмов, который удовлетворяет заданному критерию полноты проверки неисправностей.

Известно, что генетические алгоритмы (ГА) – алгоритмы оптимизации, основанные на использовании механизмов естественного отбора и селекции. Существует 4 основных отличия ГА от других алгоритмов оптимизации и поиска:

1) В ГА задаются векторы или коды множества параметров.

2) В каждой итерации ГА осуществляется поиск экстремума в популяции.

3) ГА использует только значение функции в точке дискретного пространства, не учитывая ее другие свойства.

4) ГА используют вероятностные правила отбора и селекции.

Опишем теперь применение генетических алгоритмов непосредственно к проблеме генерации тестов для последовательностных устройств. Введем основные понятия и определения, адаптированные к решению задач построения тестов для цифровых систем, такие как: ген, хромосома, особь, популяция, тест.

Определение 11.1. Координата $T_{ij}^{rt} \in T_i^{rt}$ тест-вектора или хромосомы, определенная в алфавите A^1={0,1,X}, называется геном, где T_{ij}^{rt} – координата тест-вектора, обнаруживающего неисправности класса ОКН.

Определение 11.2. Тест-вектор $T_i^{rt} = (T_{i1}^{rt}, T_{i2}^{rt}, ..., T_{ij}^{rt}, ..., T_{ik}^{rt})$, определенный на множестве входных, внутренних и выходных линий схемы, называется хромосомой, которая представляет собой геометрическую фигуру (точка, линия, плоскость, куб, ...) в k-мерном пространстве.

Определение 11.3. Упорядоченная совокупность n тестовых векторов или хромосом, объединенная в понятие сегмента, называется особью:

$$T^{rt} = (T_1^{rt}, T_2^{rt}, ..., T_i^{rt}, ..., T_n^{rt}), \tag{11.1}$$

где T^{rt} – тест-вектор, обнаруживающий неисправности класса ОКН.

Определение 11.4. Конечное и фиксированное множество m независимых тестовых сегментов – особей называется популяцией:

$$T^{r} = \{T^{r1}, T^{r2}, ..., T^{rt}, ..., T^{rm}\}. \tag{11.2}$$

Определение 11.5. Тест – совокупность конечного множества популяций всех хромосом, вычисляемый по формуле:

$$Q^{rt} = \bigcup_{i=1}^{n} Q_{ij}^{rt}, \; j = \overline{1,k}, \tag{11.3}$$

которому соответствует количественный, определяемый как отношение проверенных на множестве возможных неисправностей к полному множеству одиночных константных неисправностей:

$$q(Q^{rt}) = \frac{100}{2 \times k} \times \sum_{j=1}^{k} q(Q_j^{rt})(\%). \tag{11.4}$$

Определение 11.6. Оценочная функция для гена $T_{ij}^{rt} \in T_i^{rt}$ есть качественный показатель, характеризуемый состоянием координаты $Q_{ij}^{rt} = \{0,1,X\}$, которому соответствует количественный, определяемый как число проверяемых неисправностей:

$$q(Q_{ij}^{rt}) = \begin{cases} 1 \Leftarrow Q_{ij}^{rt} = \{0,1\}; \\ 2 \Leftarrow Q_{ij}^{rt} = X. \end{cases} \tag{11.5}$$

Определение 11.7. Оценочная функция для хромосомы $T_i^{rt} \in T^{rt}$ есть качественный показатель в виде вектора проверяемых дефектов:

$$Q_i^{rt} = (Q_{i1}^{rt}, Q_{i2}^{rt}, ..., Q_{ij}^{rt}, ..., Q_{ik}^{rt}), Q_{ij}^{rt} \in \{0,1,X\}, \qquad (11.6)$$

которому соответствует количественный, определяемый как отношение проверяемых дефектов к полному множеству одиночных константных неисправностей (11.4).

Определение 11.8. Оценочная функция для особи $T^{rt} \in T^r$ есть вектор проверенных дефектов – качественный показатель (11.6) (дополняемость). При этом для его определения нужны специальные детерминированные алгоритмы, которые используют значения функции в той или иной точке булева пространства.

Определение 11.9. Оценочная функция для популяции $T^r \in T$ есть вектор проверенных неисправностей – качественный показатель $Q^r = (Q_1^r, Q_2^r, ..., Q_j^r, ..., Q_k^r)$ всех особей, вычисляемый по формуле (11.3), которому соответствует количественный, определяемый как отношение проверенных на популяции дефектов к полному множеству одиночных константных неисправностей (11.4).

Определение 11.10. Оценочная функция для теста T есть вектор проверенных неисправностей – качественный показатель $Q = (Q_1, Q_2, ..., Q_j, ..., Q_k)$ всех популяций, вычисляемый по формуле (11.3), которому соответствует количественный, определяемый как отношение проверенных на тесте дефектов к полному множеству одиночных константных неисправностей (11.4).

Метод генетических алгоритмов для поиска рационального решения в виде входных тестовых последовательностей – хромосом (kx – количество входных линий цифровой схемы): $T_i^{rt} = (T_{i1}^{rt}, T_{i2}^{rt}, ..., T_{ij}^{rt}, ..., T_{ik_x}^{rt})$ использует следующие четыре процедуры:

1. Задание начальной популяции.

2. Скрещивание хромосом популяции выполнением операции кроссинговера (crossover).

3. Мутация хромосом на заданных условиях.

4. Естественный отбор наиболее перспективных хромосом.

Процедуры 2, 3 и 4 повторяются до тех пор, пока тест не будет удовлетворять условию $q(Q) = \frac{100}{2 \times k} \times \sum_{j=1}^{k} q(Q_j) \geq q_{min}$ или пока не выполнится предельное число итераций, равное p.

Преимущества генетических алгоритмов – это на порядок более высокое быстродействие синтеза теста заданной полноты по сравнению с детерминированными методами. Так, для больших схем время построения теста детерминированным методом в десятки раз может превышать генетический метод при одинаковой полноте теста. Недостаток – вероятностные процедуры, которые генерируют часто плохой тест, что является нежелательным при верификации цифровых изделий, так как на его модернизацию уходит много времени. Поэтому фактор случайности, присутствующий в процедурах 1-4, следует максимально исключить.

Для этого выполняется модификация перечисленных процедур с заменой вероятностных компонентов на детерминированные:

1. Определение размерности теста (количества особей) – m (m – четное, необходимое условие для выполнения операции кроссинговера) популяции, в которую на начальной стадии включаются алгоритмически создаваемые тесты: $T^{AG} = (T_1^{r0}, T_2^{r1}, T_3^{ch}, T_4^{g0}, T_5^{g1}, T_6^{gr}, T_7^{lg}, T_8^{gm})$ – бегущий 0, бегущая 1, шахматный код, галлопирующий 0, галлопирующая 1, код Грэя, логарифмический генератор M последовательностей. Число алгоритмических тестов, вкючаемых в начальную популяцию, в данном случае ограничено значением параметра m=8. Количество особей в популяции может быть увеличено за счет использования других алгоритмических, а также детерминированных методов генерации тестов. Для выбора особей-кандидатов в начальную популяцию, необходим выбор критериев, исключающих фактор случайности. При достаточно большом числе особей-кандидатов используется эвристический подбор критериев на основе структурно-фунционального анализа обрабатываемой схемы.

Классическим и фоновым решением задачи выбора начальной популяции может служить генерирование m особей размерностью n, используя код, например, генератор M последовательностей с m различными начальными условиями.

2. Каждая особь в популяции должна иметь одинаковую оптимальную длину n (n – количество хромосом в особи). Поэтому весьма существенной представляется задача определения оптимального значения n, которое является функционально зависимым от числа входных k_x, внутренних k_y и выходных k_z линий схемы ($k_x+k_y+k_z=k$). Для учета структурной сложности схемы при генерации тестов вводится коэффициент структурной глубины:

$$k_s = \frac{k_x}{k_y + k_z}. \qquad (11.7)$$

Он показывает отношение числа входов к невходным линиям устройства. С учетом данного коэффициента размерность особи в популяции определяется следующим выражением:

$$n = \begin{cases} k_x \Leftarrow k_s \le 1; \\ k_x \times k_z \Leftarrow k_s > 1. \end{cases} \tag{11.8}$$

Кроме того, необходимо учитывать реализуемость алгоритмических генераторов в булевом пространстве, содержащем n хромосом. Поэтому параметр n может быть скорректирован, если выполняется условие:

$$n < \min \operatorname{card}(T_i^{AG}), i = \overline{1, m}, \tag{11.9}$$

путем учета мощности (качества) векторов, образуемых каждым из выбранных генераторов:

$$n = \begin{cases} \min \operatorname{card}(T_i^{AG}) \Leftarrow k_s < 1, i = \overline{1, m}; \\ \frac{1}{m} \sum_{i=1}^{m} \operatorname{card}(T_i^{AG}) \Leftarrow k_s = 1; \\ \max \operatorname{card}(T_i^{AG}) \Leftarrow k_s > 1, i = \overline{1, m}. \end{cases} \tag{11.10}$$

Существенным представляется выполнение анализа схемной структуры в целях получения сведений о типах примитивов и переменных: управления, сброса, адресации, информационные, синхровходы. Последние не должны учитываться в качестве переменных, формирующих гены популяций. Это связано с тем, что на синхровход всегда следует подавать последовательность – чередование нуля и единицы: 0101...01, или наоборот.

3. Модернизация процедуры получения популяции. Оценочная функция популяции может выступать в качестве фитнесс-функции. Она является основным критерием окончания итеративного процесса при генерации тестов для проверки неисправностей.

Анализ фитнесс-функций для предыдущей и текущей популяций – итераций служат условием их продолжения или окончания:

$$Q^r = \begin{cases} Q^{r-1} \Leftarrow Q^{r-1} \cap Q^r = Q^r; \\ Q^{r-1} \cup Q^r \Leftarrow [q(Q^r) \ne 100\%] \& (Q^{r-1} \cap Q^r \ne Q^r); \\ Q^r \Leftarrow [q(Q^r) = 100\%] \vee (Q^{r-1} \cap Q^r = Q^{r-1}). \end{cases} \tag{11.11}$$

Таким образом, фитнесс для каждой итерации складывается с оценкой предыдущей популяции, за исключением, когда q(Qr) = 100%. В этом случае итеративный процесс поиска теста считается успешно завершенным.

4. Модернизация естественного отбора особей в каждой популяции основывается на решении задачи покрытия:

– определение суммарного фитнесса для особей популяции q(Qr);

– упорядочение особей по мере убывания их фитнесс-функций:

$$T^r = \{T^{rq_1}, T^{rq_2}, ..., T^{rq_t}, ..., T^{rq_m}\}; \qquad (11.12)$$

– минимизация количества особей в популяции путем последовательного исключения каждой, начиная с особи, имеющей минимальное значение фитнесса, если при этом выполняется условие

$$T^r = T^r - T^{rt} \Leftarrow q(Q^r) = q(\bigcup_{i \neq t}^{i=\overline{1,m}} Q^{ri}).$$

Таким образом, стратегия построения теста методом генетических алгоритмов основывается на конкатенации (последовательном объединении) популяций, получаемых на каждой итерации. При этом внутри популяции выполняется минимизация количества особей решением задачи покрытия в целях исключения несущественных тест-сегментов для формирования фитнесса.

5. Модернизация операции кроссинговера. Все особи из популяции, упорядоченной по убыванию фитнесс-функции, разбиваются на пары по следующей системе:

$$T^r = \{(T^{rq_1}, T^{rq_m}), (T^{rq_2}, T^{rq_{m-1}}), ..., (T^{rq_{t-1}}, T^{rq_t})\}. \qquad (11.13)$$

Замечание. Для упрощения изложения последующего материала введем обозначение особей, участвующих в операции кроссинговера:

$$T^{rs} = T^{rq_{t-1}}; T^{rt} = T^{rq_t}. \qquad (11.14)$$

Кроме того, реализация процедур кроссинговера и мутации осуществляется только на входных координатах хромосом, число которых равно k_x<k.

В результате разбиения по парам получаются примерно равнозначные, относительно фитнесса, особи-родители, которые далее участвуют в операции кроссинговера для образования особей-потомков:

1) Для каждой соответствующей пары хромосом особей-родителей $T_i^{rs} \in T^{rs}, T_i^{rt} \in T^{rt}$ выбираются сегменты обмена данными путем определения точек разрыва, количество которых может быть более одной.

2) Выполнение операции кроссовера $\overset{\times}{\underset{d}{\cap}}$ в точке разрыва d над парой хромосом родительских особей путем обмена между собой левых и правых, отно-

сительно точки разрыва, частей хромосом с их последующей конкатенацией (*) формально записывается в виде:

$$T_i^{rs} \overset{\times}{\underset{d}{\bigcap}} T_i^{rt} = \{(T^{rs}_{i(\overline{1,d})} * T^{rt}_{i(\overline{d+1,k_x})}), (T^{rt}_{i(\overline{1,d})} * T^{rs}_{i(\overline{d+1,k_x})})\}. \qquad (11.15)$$

Для двух точек разрыва и более замена сегмента в каждой хромосоме родительской пары осуществляется после нечетной точки разрыва (1,3,5...). Например, имея два вектора: (XYZ), (ABC), при двух точках разрыва, получаются следующие хромосомы-потомки:

$$(XYZ) \overset{\times}{\underset{1,2}{\bigcap}} (ABC) = \{(X*B*Z), (A*Y*C)\} = \{(XBZ), (AYC)\}. \qquad (11.16)$$

По окончании процедуры кроссинговера над всеми парами особей $(T^{rs}, T^{rt}) \in T^r$ получается новая популяция $T^{r+1}, \mathrm{card}(T^{r+1}) = m$.

6. Определение интервалов для реализации процедуры кроссинговера в каждой паре особей связано с анализом их фитнесс-функций путем выполнения операции объединения между соответствующими координатами двух векторов:

$$Q^{rs} \cup Q^{rt} = Q^{rst}, Q_j^{rst} = Q_j^{rs} \cup Q_j^{rt}. \qquad (11.17)$$

Полученный вектор объединенных фитнессов, определенный в алфавите $A = \{0,1,X,\varnothing\}$, в меньшей степени напоминает об интервалах для кроссинговера. Тем не менее, он представляет маскированный символами $\varnothing$ вектор, необходимый для эффективной реализации кроссинговера по тем генам (битам), которые отмечены знаком :

$$T_i^{rs} \overset{\times}{\underset{\forall j(Q_j^{rst}=\varnothing)}{\bigcap}} T_i^{rt} = (T_{ij}^{rs} = T_{ij}^{rt}) \,\&\, (T_{ij}^{rt} = T_{ij}^{rs}). \qquad (11.18)$$

Здесь операции присвоения, указанные в скобках и реализующие кроссинговер, выполняются параллельно и одновременно.

Таким образом, операция кроссинговера дает возможность скрестить хромосомы пары особей, одна из которых лучшая, другая худшая, в целях модификации только тех генов, которые в обеих особях составляли минимальное значение фитнесс-функции.

7. Мутация генов направлена на качественное, но незначительное изменение каждой хромосомы: $T_i^{rt} \in T^{rt}$, входящей в состав особи текущей популяции.

Классическая процедура мутации носит случайный характер. В хромосоме, соответствующей каждой особи из популяции, с наперед заданной вероятно-

стью, происходит изменение гена на противоположный. Это дает возможность выйти из зоны локального экстремума дискретной функции путем случайного изменения гена хромосомы особи, когда кросинговер и естественный отбор не дают удовлетворительного решения. Число мутируемых генов не должно быть значительным, чтобы не подменять операцию кроссинговера. На практике – это величина, определяемая как $\mathrm{card}(T^{rt}_{i_{mut}}) = \mathrm{int}(0{,}001 \times k_X)$.

Модификация мутации хромосом связана с исключением случайного характера при выборе генов. При этом формулируются дискретные условия:

1) Необходимо изменять гены, относящиеся к входным линиям с непроверенными дефектами на уже построенном тесте, включающем все популяции. Здесь осуществляется анализ координат $Q_j \in Q$ фитнесс-функции теста T для мутации генов каждой хромосомы по правилу:

$$T^{rt}_{ij} = \overline{T}^{rt}_{ij} \Leftarrow Q_j \neq X.$$

Также следует модифицировать входные переменные, функционально связанные с такими невходными линиями, для которых выполняется условие непроверки дефектов.

2) Учитывая, что упомянутых генов может быть значительное количество, достаточным условием мутации является псевдослучайный выбор координат из множества, удовлетворяющего первому условию. Мощность такой выборки определяется выражением

$$\mathrm{card}(T^{rt}_{i_{mut}}) = \mathrm{int}[0{,}01 \times \mathrm{card}(\forall j(Q_j \neq X))].$$

Все процедуры, описанные выше, сводятся к модели построения теста, представленной на рис. 11.1.

Программная реализация метода подтвердила его эффективность с точки зрения быстродействия и качества синтезируемых тестов. Стратегия обработки схемы цифрового устройства: выбор алгоритмических тестов для начальной популяции в зависимости от структурной и функциональной сложности объекта; определение фитнесса популяции – качества теста путем моделирования неисправностей; выполнение процедур кроссинговера и мутации на основе использования детерминированных процедур; формирование особей-потомков для следующей популяции, которая не заменяет предыдущую, а всегда является дополнением к ней.

Верификация программы осуществлялась на 70 последовательностных и комбинационных схемах из бибилиотеки ISCAS, а также из библиотеки тестов фирмы Aldec.

Пример 11.1. Схема c17 (рис.11.2). Параметры: количество входов – 5; линий – 11; примитивных элементов – 6. Параметры работы программы: количество особей – 4; хромосом в особи – 5; вероятность кроссинговера – 1,0; мутации – 0,2; количество итераций – 1.

В первой популяции лучшая особь покрывает 72 % неисправностей (формат следующего ниже теста: векторы исправного поведения цифрового устройства, проверяемых и проверенных дефектов, показатели качества вектора и теста, в %):

1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6	7	8	9	10	11	FC,%
0	0	1	1	1	1	0	1	1	0	0	.	.	0	0	.	0	1	0	0	1	1	36
											.	.	0	0	.	0	1	0	0	1	1	36
0	0	1	1	0	1	1	1	0	0	1	1	.	.	1	0	0	0	0	1	1	X	40
											1	.	0	X	0	0	X	0	X	1	X	63
0	1	1	1	0	1	0	1	1	0	0	1	.	0	.	.	0	1	0	0	1	1	36
											1	.	0	X	0	0	X	0	X	1	X	63
0	0	0	0	0	1	1	1	1	0	0	.	.	.	1	1	0	.	0	0	1	1	31
											1	.	0	X	X	0	X	0	X	1	X	68
1	0	0	0	0	1	1	1	1	0	0	.	1	.	1	1	0	.	0	0	1	1	36
											1	1	0	X	X	0	X	0	X	1	X	72

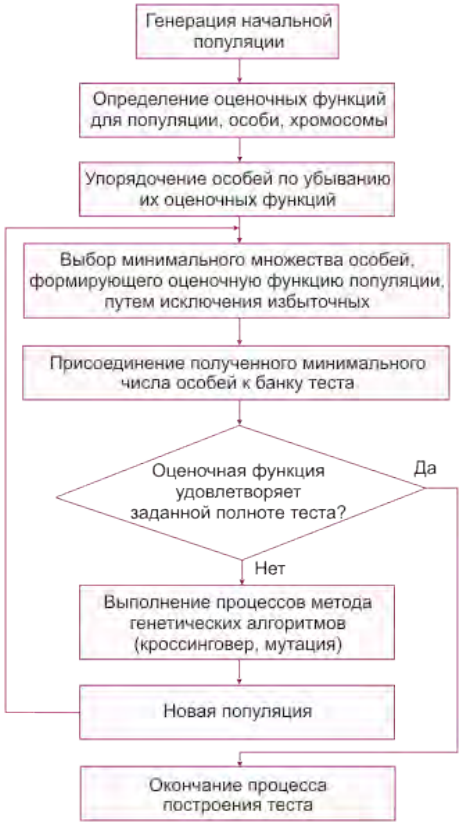


Рис. 11.1. Модель процесса синтеза теста методом генетических алгоритмов

Три итерации синтеза векторов дают тест проверки одиночных константных неисправностей для цифрового устройства, имеющий полноту 100 %:

1	2	3	4	5	6	7	8	9	10	11	1	2	3	4	5	6	7	8	9	10	11	FC,%
0	1	1	1	1	1	0	1	1	0	0	1	.	0	0	.	0	1	0	0	1	1	40
											1	.	0	0	.	0	1	0	0	1	1	40
1	0	1	1	1	1	0	1	1	0	0	.	1	0	0	.	0	1	0	0	1	1	40
											1	1	0	0	.	0	1	0	0	1	1	45
1	1	0	0	1	0	1	1	0	1	1	0	0	.	.	0	1	0	.	1	0	0	36
											X	X	0	0	0	X	X	0	X	X	X	81
1	0	0	1	0	1	1	0	1	1	1	.	.	1	0	.	.	0	1	.	0	0	27
											X	X	X	0	0	X	X	X	X	X	X	90
1	0	0	0	0	1	1	1	1	0	0	.	1	.	1	1	0	.	0	0	1	1	36
											X	X	X	X	X	X	X	X	X	X	X	100

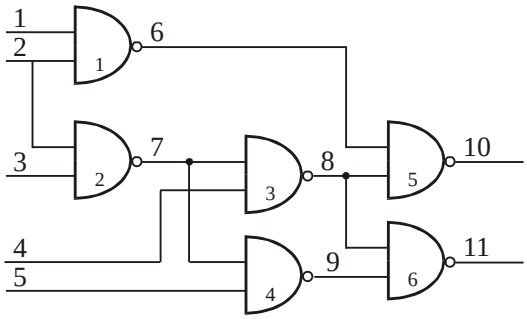


Рис. 11.2. Тест-схема c17

Пример 11.2. Схема comp_01 (рис. 11.3), взятая из библиотеки Aldec. Параметры: количество входов – 5; линий – 7; примитивных элементов – 2; триггерных элементов – 1. Параметры программы (перенастраиваемые): количество особей – 4; хромосом в особи – 5; вероятность кроссинговера – 1,0; мутации – 0,2; количество итераций – 3.

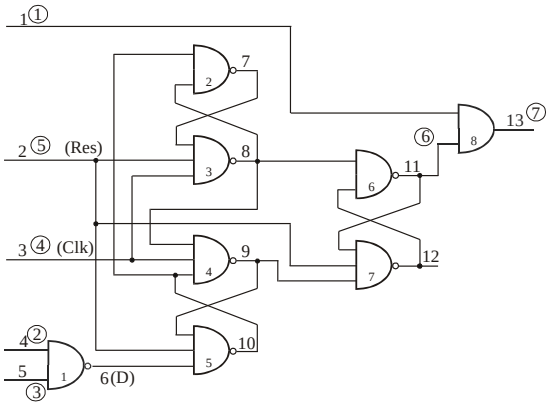


Рис. 11.3. Схема comp_01

В первой популяции лучшая особь покрывает 12,67% одиночных константных неисправностей.

При анализе схемы триггерная структура, описанная в вентильном виде, представлена одним примитивным автоматом, в соответствии с этим произведена нумерация линий. Линия 4 схемы является синхровходом, и представлена в двух автоматных тактах (t, (t-1)).

Для эффективного использования программных средств реализации МГА необходимо квалифицированное задание параметров обработки схемы, функционально зависимых от структуры тестируемого цифрового устройства. В результате выполнения трех итераций получен тест, покрывающий 100% одиночных неисправностей константного типа, что подтверждает эффективность предложенных моделей и алгоритмов синтеза тестов, которые по быстродействию занимают среднее положение между методами алгоритмическими и детерминированными:

1	2	3	4(t)	5	4(t – 1)	6	7	1	2	3	4(t)	5	4(t – 1)	6	7	FC,%
1	0	0	1	1	0	0	0	.	.	.	.	.	.	1	1	13,33
								.	.	.	.	.	.	1	1	13,33
0	0	1	0	0	0	1	0	1	.	.	.	.	.	.	1	6,67
								1	.	.	.	.	.	1	1	16,67
1	0	1	0	0	0	1	1	0	1	0	.	.	1	0	0	36,67
								X	1	0	.	.	1	X	X	53,33
1	0	1	0	1	0	1	1	0	1	0	1	.	1	0	0	40,00
								X	1	0	1	.	1	X	X	56,67
1	0	1	1	1	0	0	0	.	.	.	0	0	.	1	1	23,33
								X	1	0	X	0	1	X	X	73,33
1	0	1	1	0	0	1	1	0	1	0	.	1	1	0	0	40,00
								X	1	0	X	X	1	X	X	76,67
1	0	1	1	1	0	0	0	.	.	0	.	0	.	1	1	23,33
								X	1	0	X	X	1	X	X	76,67
1	0	0	0	0	0	0	0	.	.	1	0	0	.	1	1	33,33
								X	1	X	X	X	1	X	X	83,33
1	0	1	0	0	0	1	1	0	1	0	.	.	1	0	0	36,67
								X	1	X	X	X	1	X	X	83,33
1	0	1	0	0	0	1	1	0	1	0	.	.	1	0	0	36,67
								X	1	X	X	X	1	X	X	83,33
1	1	1	0	0	0	0	0	.	.	.	.	.	0	1	1	13,33
								X	X	X	X	X	1	X	X	90,00
1	0	1	1	1	0	0	0	.	.	.	0	0	.	1	1	23,33
								X	X	X	X	X	1	X	X	90,00
0	0	1	0	0	0	1	0	1	.	.	.	.	.	.	1	6,67
								X	X	X	X	X	1	X	X	90,00
1	0	1	1	1	1	1	1	0	1	0	.	.	X	0	0	46,67
								X	X	X	X	X	X	X	X	100,00

Таким образом, усовершенствованный структурно-функциональный π-алгоритм [42] позволяет строить адекватные кубические покрытия для триггерных структур, описанных в виде булевых уравнений, и дает возможность уменьшить время их анализа и генерации тестов путем:

1) создания структурно-функциональной модели конечного автомата, необходимого для описания многообразия триггеров, используемых в ПЛИС;

2) декомпозиции схемы, благодаря учету количества подсхем, а также их типов (комбинационного или последовательностного), необходимой для выбора методики построения КП. Так, метод построения КП по системе булевых уравнений с применением аппарата ДМ является наиболее подходящим для комбинационных схем, а также для последовательностных, имеющих одну обратную связь.

В результате проведенных исследований было доказано, что:

1. Модифицированный π-алгоритм применим для построения кубических покрытий подсхем в многоступенчатых последовательностных структурах, а также для синтеза общего КП при условии, что кубические покрытия подсхем отличаются только по фронту входа синхронизации.

2. Применение π-алгоритма для последовательностных схем нуждается в модификации благодаря наличию временной зависимости итеративных копий функциональных моделей. Предложено получать общее КП схемы путем продвижения вперед от элементов младших рангов к ступеням старших в отличие от π-алгоритма для КС.

3. Наиболее полный тест для триггерных и последовательностных структур строится детерминированным методом.

Для синтеза тестов верификации цифровых систем разработан детерминированный метод генетических алгоритмов, включающий:

1. Модифицированные модели детерминированного процесса генерации тестов верификации цифровых систем. Суть усовершенствования заключается в максимальном исключении из процедур: задания начальной популяции, кроссинговера, мутации и естественного отбора фактора случайности, где это представляется возможным.

2. Практическую реализацию МГА для построения тестов проверки неисправностей. В ней объединены достоинства генетических алгоритмов – высокое быстродействие синтеза теста заданной полноты с преимуществами детерминированных – качество тестов, близкое к 100%.

3. Программная реализация метода подтвердила эффективность предложенных моделей и алгоритмов синтеза тестов, которые по быстродействию занимают среднее положение между алгоритмическими и детерминированными генераторами. Для эффективного использования программных средств реализации МГА необходимо квалифицированное задание параметров обработки схемы, функционально зависимых от структуры тестируемого цифрового устройства.

11.2. Функциональный К-алгоритм синтеза тестов

Он инвариантен по отношению к сходящимся разветвлениям, поскольку последние находятся внутри функционального элемента, представленного ку-

бическим покрытием. Основная идея синтеза тестов заключается в использовании уравнения тестирования [43, 44]:

$$T \oplus F = L\,.$$

Другими словами, кубическое покрытие списков неисправностей L для вектора T и покрытия примитива C вычисляется на основе использования бинарной координатной операции XOR ($\oplus$), определяющей взаимодействие компонентов T, C, L, заданных в троичном алфавите описания состояний переменных:

$T_j \oplus F_{ij} =$

$\oplus$	0	1	X
0	0	1	X
1	1	0	X
X	X	X	X

Алгоритм синтеза теста проверки одиночных константных неисправностей одновыходового примитива, задаваемых регулярным диагональным L^1-покрытием списка дефектов всех переменных функционального элемента (единица на входной и выходной координате каждого куба обозначает активизацию выхода Z при наличии неисправности любого знака на входе):

$$L^1 = \left|\begin{array}{ccccc} X_1 & X_2 & \ldots X_j \ldots & X_m & Z \\ \hline 1 & 0 & \ldots 0 \ldots & 0 & 1 \\ 0 & 1 & \ldots 0 \ldots & 0 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots 1 \ldots & 1 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots 0 \ldots & 1 & 1 \end{array}\right|,$$

содержит следующие пункты:

1. Формирование кандидатов в тест $T_t^k \in T^k$ путем выполнения операций над векторами из L^1, F:

$$T^k = L^1 \oplus F. \qquad (11.19)$$

2. Получение теста T ($T_t \in T$) из множества кандидатов T^k на основе анализа выражения

$$T = T \cup T_t = F_i \cap T_t^k \leftarrow \exists i (F_i \cap T_t^k \neq \varnothing). \qquad (11.20)$$

3. Минимизация множества тест-векторов T путем применения операции поглощения:

$$T = T \setminus T_r \leftarrow T_i \cap T_r = T_r. \qquad (11.21)$$

4. Для получения еще более минимального теста необходимо выполнить процедуру моделирования дефектов на функциональной или структурной

модели устройства и на полученной многозначной таблице неисправностей [45] решить задачу получения минимального покрытия неисправностей построенным тестом.

Следующий пример иллюстрирует основные шаги описанного алгоритма.

Пример. 11.3. Построить минимальный тест для функционального трехвходового элемента, представленного кубическим покрытием:

$$F = \left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ X & 0 & 1 & 0 \\ X & 1 & 0 & 0 \\ 0 & 1 & X & 0 \end{array}\right| \qquad (11.22)$$

На первом шаге выполняется $\oplus$-операция между кубическим покрытием и векторами из множества L^1 :

$$F\left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ X & 0 & 1 & 0 \\ X & 1 & 0 & 0 \\ 0 & 1 & X & 0 \end{array}\right| \oplus L\left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right| = \{F \oplus L_1, F \oplus L_2, F \oplus L_3\}.$$

Данная операция приводит к получению трех подмножеств кубов, которые являются кандидатами в тест:

$$T^k = \left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ X & 0 & 1 & 1 \\ X & 1 & 0 & 1 \\ 1 & 1 & X & 1 \end{array}\right| \cup \left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline X & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ X & 1 & 1 & 1 \\ X & 0 & 0 & 1 \\ 0 & 0 & X & 1 \end{array}\right| \cup \left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ X & 0 & 0 & 1 \\ X & 1 & 1 & 1 \\ 0 & 1 & X & 1 \end{array}\right|$$

Каждый куб из полученных подмножеств проверяется на его соответствие правилу (11.20). Кубы, удовлетворяющие упомянутому критерию, формируют тест:

$$T = \left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}\right| \cup \left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline X & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ X & 1 & 1 & 1 \\ X & 0 & 0 & 1 \end{array}\right| \cup \left|\begin{array}{cccc} X_1 & X_2 & X_3 & Y \\ \hline X & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ X & 0 & 0 & 1 \\ X & 1 & 1 & 1 \end{array}\right|$$

Первая стадия минимизации заключается в исключении дублирующих векторов в соответствии с (11.21). (Следует заметить, что минимизация теста корректна только для комбинационных схем, где порядок следования векторов не столь существенен.) Затем выполняется доопределение символов X путем их замены таким двоичным значением, которое позволяет: 1) исклю-

чить полученный вектор из теста по правилу избыточности, представленном в выражении (11.21); 2) минимизировать кодовое расстояние между соседними наборами; 3) повысить качество теста – количество неисправностей, проверяемых рассматриваемым вектором.

Описанная фаза минимизации дает результат:

T_{min} =

X_1	X_2	X_3	Y
0	1	1	0
1	1	1	1
X	1	0	0
1	0	1	0
X	1	1	1
X	0	0	1
X	0	1	0
1	1	0	0

=

X_1	X_2	X_3	Y
0	1	1	0
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1

Далее можно уменьшить длину теста Tmin путем построения многозначной таблицы неисправностей [46] цифрового устройства на основе использования программы моделирования дефектов и решения задачи получения минимального покрытия неисправностей тестовыми наборами [47]. Для этого процесс моделирования выполняется на вентильной реализации кубического покрытия (рис.11.4), представленной в виде дизъюнктивной нормальной формы, которая записана по конституентам единицы таблицы (11.22).

Рис. 11.4. Структурная реализация функции

Для рассматриваемого примера описанные выше действия приводят к следующему результату, представленному в форме таблицы неисправностей (левая колонка – двоичные тестовые наборы, средняя колонка – таблица проверки дефектов для линий устройства, правая – список дефектов, обнаруживаемых каждым тестовым набором, который является сокращенной формой записи проверяемых неисправностей):

1	2	3	4	5	6	1	2	3	4	5	6	Fault list
0	1	1	0	0	0	1	–	–	1	1	1	$L_6=\{4^1 5^1 6^1\}, L_4=\{4^1\}, L_5=\{1^1 5^1\}$
1	1	1	0	1	1	0	0	0	–	0	0	$L_6=\{5^0 6^0\}, L_4=\{4^1\}, L_5=\{1^0 2^0 3^0 5^0\}$
1	1	0	0	0	0	–	1	1	1	1	1	$L_6=\{4^1 5^1 6^1\}, L_4=\{2^1 3^1 4^1\}, L_5=\{3^1 5^1\}$
1	0	1	0	0	0	–	1	0	1	1	1	$L_6=\{4^1 5^1 6^1\}, L_4=\{3^0 4^1\}, L_5=\{2^1 5^1\}$
1	0	0	1	0	1	–	1	1	0	–	0	$L_6=\{4^0 6^0\}, L_4=\{2^1 3^1 4^0\}, L_5=\{5^1\}$

В правой части таблицы представлены результаты моделирования неисправностей отдельных примитивов. В среднем столбце показаны проверяемые тест-векторами неисправности константного типа, которые получены с помощью процедуры обратной суперпозиции.

Далее получение минимального теста основывается на выборе таких наборов, которые в совокупности проверят на каждой линии неисправности обоих знаков. Таким тестом здесь является подмножество наборов: Tmin = {1,2,5}.

В общем случае существует формальная процедура синтеза минимального теста [47], основанная на преобразовании конъюнктивной формы записи покрытия неисправностей по столбцам в дизъюнктивную форму в целях последующего получения минимальных термов минимальной функции.

Пусть имеется тест, содержащий 6 наборов, покрывающий все неисправности, представленные в схеме :

Test	F_1	F_2	F_3	F_4	F_5	F_6
t_1	1	1	.	.	1	.
t_2	.	1	1	.	.	.
t_3	1	.	.	1	.	.
t_4	.	.	.	.	1	1
t_5	.	.	1	.	.	1
t_6	1	.	1	.	.	.

Необходимо получить минимальное множество наборов, не уменьшающее покрытие неисправностей. Для этого записывается произведение дизъюнкций в соответствии со столбцами таблицы и выполняется преобразование к минимальной дизъюнктивной форме на основе правила поглощения ($ab \vee a = a$):

$$T = (t_1 \vee t_3 \vee t_6)(t_1 \vee t_2)(t_2 \vee t_5 \vee t_6)(t_3)(t_1 \vee t_4)(t_4 \vee t_5) =$$
$$= (t_1 \vee t_1t_3 \vee t_1t_6 \vee t_1t_2 \vee t_2t_3 \vee t_2t_6)(t_2t_3 \vee t_3t_5 \vee t_3t_6) \wedge$$
$$\wedge (t_1t_4 \vee t_1t_5 \vee t_4 \vee t_4t_5) =$$
$$= (t_1 \vee t_2t_3 \vee t_2t_6)(t_2t_3 \vee t_3t_5 \vee t_3t_6)(t_1t_5 \vee t_4) =$$
$$= (t_1t_2t_3 \vee t_1t_3t_5 \vee t_1t_3t_6 \vee t_2t_3 \vee t_2t_3t_5 \vee t_2t_3t_6 \vee$$
$$\vee t_2t_3t_6 \vee t_2t_3t_5 \vee t_2t_3t_6)(t_1t_5 \vee t_4) =$$
$$= (t_1t_3t_5 \vee t_1t_3t_6 \vee t_2t_3)(t_1t_5 \vee t_4) =$$
$$= (t_1t_3t_5 \vee t_1t_3t_5t_6 \vee t_1t_2t_3t_5 \vee t_1t_3t_4t_5 \vee t_1t_3t_4t_6 \vee t_2t_3t_4) =$$
$$= (t_1t_3t_5 \vee t_1t_3t_4t_6 \vee t_2t_3t_4).$$

Полученный результат формирует все возможные минимальные покрытия в виде трех конъюнктивных термов. Лучшим решением является терм с минимальным числом букв, здесь таких два: $T = t_1t_3t_5 \vee t_2t_3t_4$.

11.3. Совместный К-алгоритм синтеза тестов

Ориентирован на генерацию тестов для комбинационных и последовательностных схем, где фазы активизации и обеспечения выполняются совместно и в направлении от выходов схемы к ее входам. Стратегия алгоритма заключается в построении множества одномерных путей активизации, покрывающих все линии схемы. Основные шаги алгоритма [48]:

1. Выбор очередного пути активизации от внешнего выхода ко входу, который покрывает еще не обработанные линии.

2. Выполнение фазы продвижения активности от выходов ко входам путем определения для каждого примитива пары кубов, обеспечивающих изменение выхода при наличии активного входа.

3. Доопределение неактивных координат примитивов путем выполнения обратной импликации на всех элементах, участвующих в формировании условий активизации на внешних входах схемы.

4. Минимизация полученного теста путем применения процедур при описании функционального К-алгоритма.

Шаги 2 и 3 выполняются совместно во времени, что возможно благодаря предварительному ранжированию линий и элементов схемы.

Рассмотрим предложенный алгоритм на примере построения теста для функционального модуля анализа дефектов, представленного на рис. 11.5.

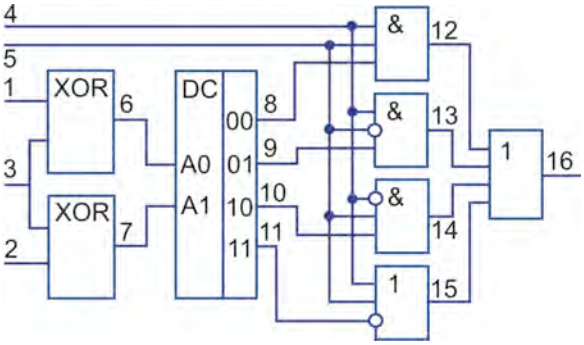


Рис. 11.5. Структурная схема симулятора

Результат выполнения активизации входа 4 до внешнего выхода представлен в форме пары векторов, определенных в троичном алфавите на всех 16 линиях схемы функционального примитива:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	X	0	0	X	0	0	0	0	0	0	0	0
X	X	X	1	1	X	X	1	X	X	X	1	X	X	X	1

Доопределение символов X по входным координатам дает пару двоичных наборов, которые необходимо верифицировать путем выполнения исправного моделирования, что дает следующий результат:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	0	0	0	1	0	0	0	1

Аналогично выполнив активизацию остальных линий, получим следующий результат в виде пар наборов:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	X	0	0	0	X	0	0	0	0	0	0	0	0
X	X	X	1	1	X	X	1	X	X	X	1	X	X	X	1
0	0	0	X	X	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	X	1	X	X	X	1	X	1
0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0
0	1	1	X	X	1	0	X	X	0	1	X	X	X	1	1
1	0	0	X	X	1	0	0	0	1	0	0	0	0	0	0
1	0	1	1	0	0	1	0	1	0	0	X	1	X	X	1

Выполним доопределение входных символов X по описанным правилам, затем – моделирование двоичных векторов, которые объединим с ранее полученной парой наборов для активизации переменной 4. В результате имеем тест:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	0	0	0	1	0	0	0	1
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	1
0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	1	0	0	0	1	1
1	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0
1	0	1	1	0	0	1	0	1	0	0	0	1	0	0	1
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	0	0	0	1	0	0	0	1

Моделирование неисправностей подтвердило 100% качество сгенерированного теста, но обладающего некоторой избыточностью (векторы 9, 10).

Предложенные методы и алгоритмы реализованы в программе синтеза тестов и интегрированы в автоматическую систему тестирования SIGETEST.

Таким образом, представленная стратегия тестирования сложных проектов цифровых систем основана на использовании технологии граничного сканирования сходящихся разветвлений, доставляющих существенные неудобства в процессе синтеза тестов и моделирования неисправностей константного типа. Стратегия позволяет на порядок уменьшить затраты времени при генерации детерминированных тестов и анализе их качества. В целях повышения быстродействия алгоритмов тестирования предложены решения, существенные для автоматизации проектирования тестового обеспечения и обладающие научной новизной:

1. Процедура обратной суперпозиции для древовидных структур, позволяющая в десятки раз повысить быстродействие моделирования неисправностей для сложных цифровых проектов.

2. Анализ структурных коллизий, связанных с обработкой сходящихся разветвлений и обратных связей в целях локализации области их негативного действия на процесс моделирования неисправностей и синтеза тестов.

3. Реконфигурирование модели цифрового устройства для моделирования и синтеза тестов на основе расширения длины регистра сканирования данных, что позволяет в несколько раз уменьшить время синтеза детерминированных тестов методом одномерной активизации.

4. Функциональный К-алгоритм синтеза тестов, эффективно обрабатывающий цифровые устройства, представленные в форме кубического покрытия. Вычислительная сложность алгоритма пропорциональна квадрату числа переменных тестируемой схемы.

5. Совместный К-алгоритм синтеза тестов для цифровых структур, отличающийся от алгоритма Рота меньшим числом переборов за счет введения неопределенности в условия активизации одномерных путей.

11.4. Стратегия генерации тестов для цифровых систем

Объект тестирования – цифровые системы, представленные на языке описания аппаратуры VHDL в форме булевых уравнений, задающих как комбинационные схемы, так и триггерные структуры. Типы переменных определяются структурами языка VHDL: bit, bit-vector, std logic, boolean, integer.

Базовые методы генерации тестов:

1) Детерминированный. Основан на применении алгоритмов активизации одномерных путей на паре входных наборов для комбинационных схем и на упорядоченной последовательности входных векторов конечной длины – для последовательностных. Преимущества определяются возможностью целенаправленного построения тестов для отдельных заказных неисправностей. Недостатки: вычислительная сложность получения решения, если оно существует, определяется NP-полнотой; невозможность за приемлемое время получить полный тест для схем большой размерности (содержащих миллионы вентилей).

2) Алгоритмический. Основан на генерации регулярных алгоритмических входных последовательностей (бегущие 0 и 1, галоп 0 и 1, шахматный код, логарифмический), на требующих решения задачи обратной импликации для установки схемы в наперед заданное состояние. Преимущества заключаются в высоком быстродействии и проверке неисправностей для регулярных структур (сумматоры, счетчики, регистры). Недостатки определяются низкой полнотой проверки неисправностей для последовательностных схем и нерегулярных комбинационных структур.

3) Псевдослучайный. Основан на использовании программных или аппаратурных перенастраиваемых или построенных генераторов псевдослучайных последовательностей для дальнейшей подачи двоичных кодов на входы цифровой системы. Преимущества: высокое быстродействие, ориентация на полное пространство тестируемых состояний; инвариантность к структурной организации цифрового устройства. Недостатки: практическая недостижимость 100% полноты теста схемы относительно заданных неисправностей за приемлемое время. Можно говорить о том, что алгоритмический метод есть частный случай псевдослучайного.

Для оценки качества генерируемых тестов необходимо иметь быстродействующую программу моделирования неисправностей, от которой зависит время построения тестов для цифровой системы. Она должна иметь высокое абсолютное быстродействие (тысячи наборов в секунду для схем, имеющих тысячи эквивалентных вентилей) и инвариантность по отношению к уровням описания цифровых систем (вентильный, алгоритмический, системный).

Учитывая, что все три метода генерации тестов имеют преимущества и недостатки, далее предлагаются стратегии, максимально использующие преимущества каждого из них.

1) Параллельная стратегия. Применение к модели цифрового устройства полного набора псевдослучайных тестов. Выбор из него минимального множества, покрывающего все ОКН, путем решения задачи определения покрытия всех неисправностей минимальным набором тестов. Если тест не полный, то задача покрытия рассматривается относительно полученного процента проверенных неисправностей, который не может быть уменьшен в процессе ее решения. Достоинством такого подхода является возможность распараллеливания времяемкого процесса генерации тестов путем использования нескольких процессоров или компьютеров в сети. По окончании самой длительной процедуры генерации тестов выполняется минимизация совокупного теста уже на одном мощном компьютере. Недостатки связаны с обработкой несложных (до 1000 эквивалентных вентилей) цифровых структур, для которых предлагаемая стратегия является весьма дорогостоящей.

2) Последовательная стратегия. Выполняется на компьютере путем последовательного генерирования псевдослучайных тестов, имеющихся в наличии:

$$T = (T^1, T^2, ..., T^i, ..., T^n). \qquad (11.23)$$

После выполнения генерирования очередного теста T^i выполняется анализ покрытия неисправностей полученными наборами. Если тест неполный – генерируется очередная совокупность T^{i+1}, иначе – окончание последовательной стратегии. В случае фиксации результата, удовлетворяющего следующему условию:

$$Q(T^1 \cup T^2 \cup ... \cup T^i \cup ... \cup T^n) < 100\%, \qquad (11.24)$$

по окончании генерирования всех тестов используется детерминированный метод, ориентированный только на генерацию тестов для еще не проверенных неисправностей цифровой системы. Преимущества определяются возможностью остановки процесса генерирования тестов при достижении требуемой полноты (100%), что предпочтительно для регулярных и несложных структур. Недостатки связаны с последовательной генерацией тестов, что предполагает значительные временные затраты при обработке схем большой размерности (тысячи эквивалентных вентилей).

3) Стратегия остаточного генерирования тестов. Учитывая высокую вычислительную сложность и точность детерминированного метода, его можно применить для построения тестов к фрагментам цифровых систем или для проверки отдельных неисправностей, которые не были обнаружены тестами, сгенерированными с помощью стратегий 1 и 2. Для выбранной неисправности строится путь ее активизации до наблюдаемых выходов. Затем выполняется процедура обеспечения условий активизации, создающая входную последовательность, которая проверяет заказанный дефект. Преимущества определяются алгоритмической сходимостью при решении задачи построения теста для конкретной неисправности, если такой существует. Недостатки связаны с полным перебором решений для худшего варианта решения задачи обеспечения.

4) Использование перенастраиваемых генераторов псевдослучайных последовательностей. Генерация ni наборов с равновероятным распределением по каждому входу нулей и единиц. Если полнота теста FC<100%, то выполняется подсчет количества переключений по всем линиям для целей перенастройки генератора по правилам прямо или обратно пропорциональным частоте появления единиц с последующей генерацией ni+1 наборов с измененными параметрами. Иначе – конец процедуры генерации тестов.

5) Использование множества алгоритмических генераторов, которые дают возможность получать полные, относительно ОКН, тесты для регулярных структур (логические схемы, коммутаторы, дешифраторы, сумматоры, счетчики, регистры, память).

11.5. Анализ схемной структуры

Большая размерность цифровых проектов, определяемая миллионами эквивалентных вентилей, требует значительного времени генерации тестов в соответствии с вычислительной сложностью, задаваемой отношением

$$W(T) = [\sum_{i=1}^{M}(m_i \times q_i)]^2. \qquad (11.25)$$

где m_i и q_i – число линий и кубов в покрытии i-го примитива цифровой схемы.

Для уменьшения данной оценки необходимо понижать размерность обрабатываемых объектов путем их декомпозиции на составляющие подсхемы с использованием следующих принципов:

1. Выделение входов синхронизации. Путем анализа кубических покрытий или кодов языка VHDL можно определить направление синхронизации (передний или задний фронт). Аналогично можно выполнить классификацию остальных линий путем выделения адресных, информационных, управляющих входов.

2. Декомпозиция цифрового проекта на подсхемы по входу синхронизации. Для каждого синхровхода находятся функционально зависимые выходы схемы, для которых определяются в качестве существенных аргументов внешние входы схемной структуры

$$X_C \to \forall i[Y_i = f_i(X_C)] \to \forall j[X_j = f_i^{-1}(Y_i)]. \qquad (11.26)$$

Это дает возможность разбить схему на n+1 часть (n=0,1,2,3,...), где n – число независимых внешних синхровходов. Подсхема n+1 – не имеет входов синхронизации. Взаимодействие подсхем определяется следующими отношениями:

$$S_j \bigcap_{j \neq r}^{j,r=\overline{0,n-1}} S_r \neq \varnothing; S_j \cap S_{n+1} = \varnothing; \bigcup_{j=0}^{n} S_j = S. \qquad (11.27)$$

3. Определение подсхем, соответствующих триггерам. Осуществляется поиск всех триггеров в цифровом устройстве, для входов которого вычисляются все внешние входы, влияющие на состояние каждого триггера:

$$X^{F_i} = \{X_j | \forall j[X_j = f_i^{-1}(Y^{F_i})]\}.$$

Это дает возможность целенаправленно генерировать входные последовательности для проверки каждого триггера или использовать заранее построенные тесты для каждого типа триггера. После определения для триггера внешних входов активизации тестом $X_F^{F_i}$ вычисляется подсхема, соответствующая триггеру F_i. После этого находятся внешние входы, отвечающие за фазу обеспечения при транспортировании теста для триггера на внешние выходы схемы: $X_B^{F_i} = X^{F_i} / X_F^{F_i}$, где $X^{F_i} = \{X_F^{F_i}, X_B^{F_i}\}$ – полное множество входов, необходимое для тестирования триггерной структуры F_i.

4. Подсхема S_n разбивается на фрагменты, число которых равно количеству выходов в S_n с учетом следующих критериев для входов:

$$2^{|X_n = \{X_{n1}, X_{n2}, ..., X_{nm}|} \geq 2^{\left|X_{n1} = f_i^{-1}(Y_{n1})\right|} + 2^{\left|X_{n2} = f_i^{-1}(Y_{n2})\right|} + ... + 2^{\left|X_{nm} = f_i^{-1}(Y_{nm})\right|}.$$

5. Композиция полученных разбиений путем объединения таких подсхем, которые имеют идентичное множество входных переменных или отличаются по ним не более, чем q переменными:

$$S_i \cup S_j \leftarrow [(2^{|X_i|} + 2^{|X_i|} \geq 2^{|X_i \cup X_j|}) \,\&\, |S_i \cap S_j| \leq q],$$

где $X_i, X_j \in X = \{X_1, \ldots, X_i, \ldots, X_n\}$, а X – полное множество входов схемы (подсхемы), подлежащей разбиению. Параметр q следует выбирать, исходя из структуры схемы. Для двухуровневых схем он может быть соизмерим с числом входов подсхемы. Для многоуровневых и последовательностных его следует уменьшать до нуля.

Разбиение цифровой структуры на подсхемы дает возможность применять параллельную стратегию, но уже к частям цифрового устройства. В этом случае при генерации тестов используется более одного компьютера. В лучшем случае их число равно количеству подсхем, которые одновременно обрабатываются в целях получения полного теста. Далее полученные тесты для подсхем складываются $T = (T^1 + T^2 + \ldots + T^i + \ldots + T^n)$, если все подсхемы отвечают условию. При этом длина теста определяется формулой

$$|T| = |T^1| + |T^2| + \ldots + |T^i| + \ldots + |T^n|. \qquad (11.28)$$

В альтернативном случае, когда имеет место условие пересечения

$$\underset{i,j=1,n}{\overset{i \neq j}{\forall}} i, j(S_i \cap S_j = \varnothing),$$

выполняется минимизация теста по правилу аддитивного объединения

$$T = (T^1 \cup T^2 \cup \ldots \cup T^i \cup \ldots \cup T^n).$$

Длина теста для всей схемы будет равна

$$|T| = \max(|T^1|, |T^2|, \ldots, |T^i|, \ldots, |T^n|).$$

В общем случае, когда выполняются условия

$$\underset{i,j=1,n}{\overset{i \neq j}{\exists}} i, j(S_i \cap S_j \neq \varnothing) \,\&\, \underset{i,j=1,n}{\overset{i \neq j}{\exists}} i, j(S_i \cap S_j = \varnothing), \qquad (11.29)$$

формирование теста происходит путем анализа очередной пары подсхем для построения объединенного теста:

$$T = T^p \,\ddot{\cup}\, T^q = \begin{cases} T^p * T^q \leftarrow S_i \cap S_j \neq \varnothing; \\ T^p \,\ddot{\cap}\, T^q \leftarrow S_i \cap S_j = \varnothing, \end{cases} \qquad (11.30)$$

где * – операция конкатенации или последовательного присоединения тестов для подсхем; $\overset{*}{\cap}$ – матричная операция пересечения тестов, которая определяется в следующем виде:

$$T = T^p \overset{*}{\cap} T^q = \begin{cases} T_{ij}^p = T_{ij}^p \bigcap\limits_{i=\overline{1,n_q}}^{j=\overline{1,m}} T_{ij}^q \leftarrow \left|T^p\right| \geq \left|T^q\right|; \\ \\ T_{ij}^q = T_{ij}^p \bigcap\limits_{i=\overline{1,n_p}}^{j=\overline{1,m}} T_{ij}^q \leftarrow \left|T^p\right| \leq \left|T^q\right|, \end{cases} \qquad (11.31)$$

где n_p и n_q – длина (число векторов) в тестах T^p и T^q, m – общее число линий в схеме цифрового устройства.

Таким образом, общая длина теста после объединения частичных тестов для подсхем определяется интервалом

$$\sum_{i=1}^{n}\left|T^i\right| \leq |T| \leq \max_i\left(\left|T^i\right|\right) \text{ или } \sum_{i=1}^{n} n_i \leq |T| \leq \max_i(n_i). \qquad (11.32)$$

На основе изложенной методики декомпозиции схемы и применения нескольких стратегий для генерации тестов разработана автоматическая система генерации тестов SIGETEST.

Для цифровых объектов большой размерности предложены три стратегии генерации тестов, ориентированные на распараллеливание процессов, использование методов декомпозиции схемы на функционально законченные фрагменты. Даны оценки и процедуры разбиения цифровой системы на подсхемы и маркирования внешних входов. Предложена адаптация К-алгоритма [49] к троичному алфавиту кубического исчисления, что значительно упрощает процедуры прямой и обратной импликаций, являющиеся основой детерминированного проектирования тестов.

11.6. Псевдокомбинационные модели цифровых схем

Ориентированы на их использование в алгоритмах построения тестов активизации для цифровых систем. Логическая схема рассматривается как структура иерархически связанных одновыходовых примитивов, заданных в виде КП в троичном алфавите {0, 1, X}. Если исходная схема содержит многовыходовые примитивы (элементы с более чем одним выходом, заданные единым КП), то для них выполняется декомпозиция исходного покрытия на множества покрытий для каждого выхода элемента. При выполнении такой декомпозиции выполняется минимизация покрытия с учетом только интересуемого выхода и если в минимизированном покрытии некоторые аргументы (входы) являются несущественными, то они удаляются. Такая замена описания многовыходового примитива множеством одновыходовых не сни-

жает адекватности модели, и в зависимости от конкретного элемента, может приводить только к изменению размера его описания (суммарного покрытия).

Определение 11.11. Примитивным элементом (ПЭ) будем называть любой одновыходовой логический элемент с одним и более входов, заданный единым кубическим покрытием в алфавите {0, 1, X}, при условии, что пересечение любой пары кубов из альтернативных множеств (0-кубы, 1-кубы) по входным координатам является пустым [50]:

$$[C_i^0 \in C \leftarrow \forall j(C_i^0 \cap C_j^1) = \varnothing] \,\&\, [C_i^1 \in C \leftarrow \forall j(C_i^1 \cap C_j^0) = \varnothing]. \quad (11.33)$$

Определение 11.12. Если множество кубов покрывает не все возможные входные значения элемента, когда существует такой двоичный вектор E, что

$$[\forall j(E \cap C_j^1) = \varnothing] \,\&\, [\forall j(E \cap C_j^0) = \varnothing], \quad (11.34)$$

то такой элемент будем называть не полностью определенным и для некоторых входных наборов его выход будет равен символу X.

Для минимизации описания элемента в покрытии нет смысла вносить кубы, для которых значение выхода не определено. Они могут быть получены вычитанием определенных кубов из полного множества входных значений. В целях сокращения размера описания примитивных элементов цифровой структуры и повышения скорости его обработки целесообразно КП хранить в минимизированной форме, предполагающей наличие символов X на входных координатах покрытия элемента.

Поскольку логическая схема может содержать множество однотипных ПЭ, описываемых единым (либо эквивалентными) покрытием, то для сокращения объема описания схемы целесообразно составить библиотеку типов, в которую включается уникальное описание каждого типа ПЭ. В этом случае каждый элемент схемы ассоциируется с некоторым типом из указанной таблицы. В предлагаемой модели ассоциация выполняется по порядковому номеру в библиотеке. Библиотека уникальна для каждой схемы. В случае ее использования несколькими схемами целесообразно применять связывание по имени.

Отметим, что преимущество от использования одновыходовых примитивов, с точки зрения ресурсоемкости описания, проявляется в еще большей степени, если разнотипные многовыходовые элементы, составляющие исходную схему, после декомпозиции на одновыходовые примитивы имеют общие (однотипные) ПЭ.

Предлагаемый алгоритм генерации тестов базируется на модифицированном П-алгоритме [51, 52] (К-алгоритм) и поэтому в первую очередь ориентирован на схемы комбинационного типа, однако может эффективно использоваться для обработки большинства синхронных последовательных схем, ко-

торые могут быть представлены псевдокомбинационной моделью. Работа с ней предполагает, что подача входных сигналов и считывание выходных происходит в дискретные моменты времени, при этом считается, что в момент формирования выходов все переходные процессы в схеме закончились. Иначе говоря, все элементы, составляющие схему, имеют нулевую задержку распространения сигналов с входа до выхода.

Для представления последовательностной схемы псевдокомбинационной моделью выполняется предварительный разрыв всех обратных связей. Осуществляется замена парами линий: мастер-линия и псевдовход. Мастер-линия соответствует исходной линии ОС, а псевдовход фактически связывается с ней линией задержки. В процессе обработки такой модели предполагается, что значения мастер-линий текущего такта транспортируются на соответствующие псевдовходы следующего такта.

Выбор множества линий, интерпретируемых как обратные связи, подлежащих разрыву, обычно осуществляется в соответствии с критерием минимизации их количества и повышения установочной способности схемы – управляемости: сокращение минимального количества тактов, необходимых для установки всех линий схемы в наперед заданное состояние.

Множество псевдовходов формирует вектор, однозначно определяющий текущее состояние последовательностной схемы. Если некоторый псевдовход соответствует элементу памяти с асинхронной установкой, то его начальное состояние определяется типом элемента и равно 0 либо 1. В противном случае исходное состояние псевдовхода является неопределенным (X). На рис. 11.6 приведена псевдокомбинационная модель произвольной цифровой схемы.

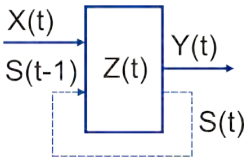


Рис. 11.6. Структура последовательностного примитива

Множество линий схемы разбивается на 4 подмножества: входы (X);

– псевдовходы (S); внутренние линии (Z); выходы (Y).

Структуре соответствуют характеристические функции вида:

$$Y(t)= f[X(t), S(t-1)]; \quad S(t)= g[X(t), S(t-1)]. \qquad (11.35)$$

Отметим, что множество внутренних линий включает те, которые не управляемые и не наблюдаемые через интерфейс схемы. Это множество может отличаться от числа внутренних линий оригинальной схемы. Если не интересует проверка (моделирование) некоторых внутренних линий схемы, то их функциональное описание (кубическое покрытие) может быть поглощено описанием линий, аргументами для которых они являются – построение

обобщенного КП нескольких примитивов. В этом случае может быть сокращено время, необходимое на обработку схемы. При этом увеличивается размерность описания схемы. С другой стороны, в целях сокращения описания схемы может быть выполнена декомпозиция некоторых элементов исходной схемы на множество отдельных примитивных элементов.

Большинство современных синхронных цифровых устройств может быть разделено на две независимые подсхемы: комбинационную и последовательностную (рис. 11.7.). Последняя из них представляет собой цепочку ячеек памяти, синхронизируемую фронтом либо уровнем импульса синхронизации. В этом случае линия синхронизации может быть исключена из логической модели устройства. Характер работы с моделью предполагает синхронное взаимодействие с интерфейсом устройства: подачу входных и съем выходных данных. Наличие ОКН на реальной линии синхронизации приведет к блокированию возможности выполнить переключение выходов элементов памяти.

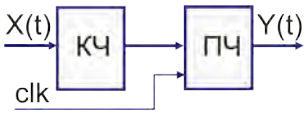


Рис. 11.7. Итеративная структура последовательностного примитива

Если структура синхронной последовательностной схемы является смешанной и не может быть разделена на независимые комбинационную и последовательностную части, а также применяет синхронизацию по фронту и при этом она используется только однотипными элементами памяти, то в модели схемы синхронизация осуществляется по уровню соответствующего синхровхода.

В случае нетривиального использования сигнала синхронизации (по фронту) в качестве аргумента комбинационной части схемы, или по различным уровням, то следует выполнять разбиение по времени линии синхронизации на две с добавлением псевдовхода – предыдущего значения линии синхронизации. В этом случае поведение примитивных элементов памяти с синхронизацией по фронту задается с помощью микромоделей, заданных в виде КП, имеющих аналогичное разбиение синхровхода на два временных фрейма (clk(t) и clk(t-1)). Подобная интерпретация входа синхронизации является наиболее универсальной. Она представлена на рис. 11.7 и может использоваться в рассмотренных выше случаях, но это приводит к некоторому усложнению модели.

В соответствии с синхронной псевдокомбинационной моделью исходная схема представляется в виде множества однотактных переменных в алфавите {U, 0, 1, X}. Им ставятся в соответствие фактические и дополнительные линии, связанные логическими отношениями (функциями), заданными в виде КП. Каждая переменная может быть функцией от некоторого количества аргументов, либо является входной. Кроме того, модель ставит в соответ-

ствие каждому псевдовходу мастер-линию. При формировании описания цифровой структуры переменные подвергаются ранжированию по правилу: вначале нумеруются входные переменные, затем псевдовходы, а далее – все остальные переменные. При этом номер каждой вычисляемой переменной должен быть больше номеров переменных, являющихся ее аргументами.

Для упрощения описания процедуры построения тестов активизации в соответствии с предлагаемым методом определим ряд элементарных моделей, включающих структуры данных и процедуры их обработки. Для описания структур данных и алгоритмов их обработки воспользуемся синтаксисом, близким к языку программирования высокого уровня C++, а также обозначениями стандартных типов этого языка. При этом в ходе описания алгоритмов мы будем опускать технические детали, специфические для конкретного языка и не отражающие особенности описываемого алгоритма, например, работа с памятью, обработка критических ситуаций. Назовем такую нотацию псевдо-C++. Использование синтаксиса позволит выполнить лаконичное описание алгоритма построения тестов, а также всех процедур и структур данных. Согласно синтаксису каждой модели поставим в соответствие объект (класс), определяющий необходимые наборы данных и операций (методов). При описании каждой модели для тривиальных методов дается краткая характеристика, а для методов, отражающих суть процедуры, приводится описание алгоритма на псевдо-C++.

11.7. Процедуры генерации тестов для последовательностных схем

Предлагаемый алгоритм генерации тестов проверки ОКН основан на принципе одномерной активизации логического пути [49] на паре троичных соседних последовательностей. В отличие от подхода, описанного в [53], здесь векторы активизации не ориентируются на проверку конкретных неисправностей. Однако на линиях, принадлежащих одномерному логическому пути активизации, проверяются все одиночные и кратные константные дефекты [49].

Определение 11.13. Путем (логическим) будем называть последовательность эквипотенциальных линий, соединяющую внешний вход схемы с одним или несколькими внешними наблюдаемыми выходами.

Алгоритм активизации последовательностных схем состоит из двух фаз: прямой и обратной импликации. Фаза прямой импликации включает выбор и активизацию одномерного пути от непроверенной линии к любому наблюдаемому выходу схемы, которая заключается в изменении выхода схемы при смене входного сигнала на паре входных соседних последовательностей:

$$\frac{df(X_i,...,X_m)}{dX_i} = f(T_1,\ldots,T_i^0,\ldots,T_m) \cap f(T_1,\ldots,T_i^1,\ldots,T_m) = \varnothing . \qquad (11.36)$$

Обратная фаза включает доопределение сигналов на входах схемы, необходимых для обеспечения активизации выбранного пути, на основе известных значений внутренних Y и выходных Z линий схемы:

$$(X_i,\ldots,X_m)\setminus X_i = f^{-1}(Y_1^*,\ldots,Y_n^*,Z_1^*,\ldots,Z_k^*). \qquad (11.37)$$

Затем осуществляется связывание пар векторов активизации в непрерывную последовательность, которая не может быть изменена.

Схема алгоритма детерминированной генерации тестов при активизации одномерных логических путей приведена на рис. 11.8. Здесь словом «Да» отмечены ветви, соответствующие успешному завершению процедуры, а словом «Нет» – невозможность найти для нее решение.

Процесс генерации тестовой последовательности начинается с выбора начальной точки (линии) пути активизации. Обычно в качестве таковой выбирается первая линия схемы, однако начальной может быть произвольная или заказная линия.

В общем случае выбор пути активизации может выполняться в зависимости от списка уже проверенных ОКН либо последовательно, без дополнительного анализа проверенных путей. В первом случае, если текущая линия имеет ветвление, то в качестве следующей выбирается линия с непроверенной ОКН, а ветви с полностью проверенными ОКН игнорируются.

Для примера (рис. 11.9), если текущий путь 2-5, то следующим будет 2-7. Первый вариант не гарантирует, что будут найдены тестовые векторы для проверки потенциально проверяемых ОКН. Необходимый для их проверки путь может быть не выбран. Подобные ОКН могут быть покрыты тестом в случае применения второго метода, однако он больше подходит для схем со сходящимися разветвлениями, где множество входов формируют несколько выходов и количество ветвлений мало. Этот метод становится практически неприменимым для обработки схем с большим числом ветвлений. Первый метод эффективен для сильносвязных схем, где большинство линий может быть активизировано различными входящими путями.

Эффективность процедуры выбора пути активизации может быть повышена, если при рассмотрении очередного пути использовать информацию о результате последней разметки. Если разметить данный путь не удалось, то новый использует успешно размеченный фрагмент предыдущего пути и дополняется следующей исходящей ветвью из успешно размеченного фрагмента пути.

Поиск очередной ветви начинается с последней размеченной линии. В общем случае определение множества путей P активизации сводится к решению задачи покрытия всех линий в схеме по возможности минимальным множеством:

$$P = \min(P_1,\ldots,P_i,\ldots,P_m)\Big|_{(P_1\cup\ldots\cup P_i\cup\ldots\cup P_m)=\{X,Y,Z\}}. \qquad (11.38)$$

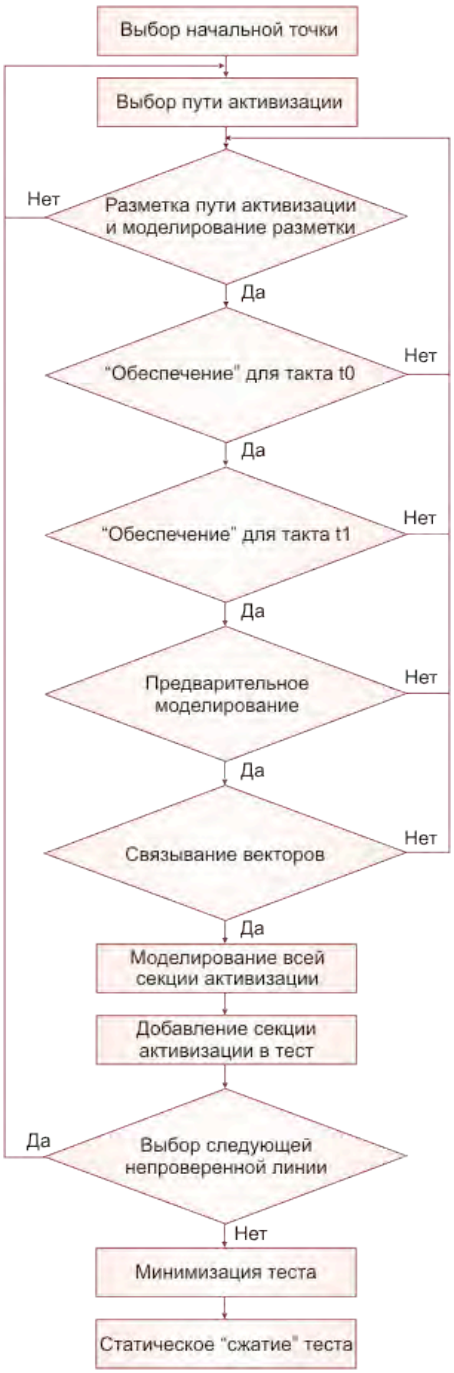


Рис. 11.8. Структура алгоритма детерминированной генерации тестов

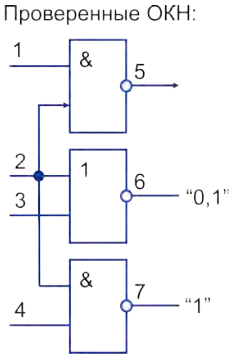


Рис. 11.9. Схема с разветвлениями

При этом лучшим вариантом решения задачи покрытия является выполнение условия $(P_1 \cap ... \cap P_i \cap .. \cap P_m) = \varnothing$.

Алгоритм выбора пути приведен на рис. 11.10. Его применение позволяет минимизировать затраты на перебор путей активизации и их анализ.

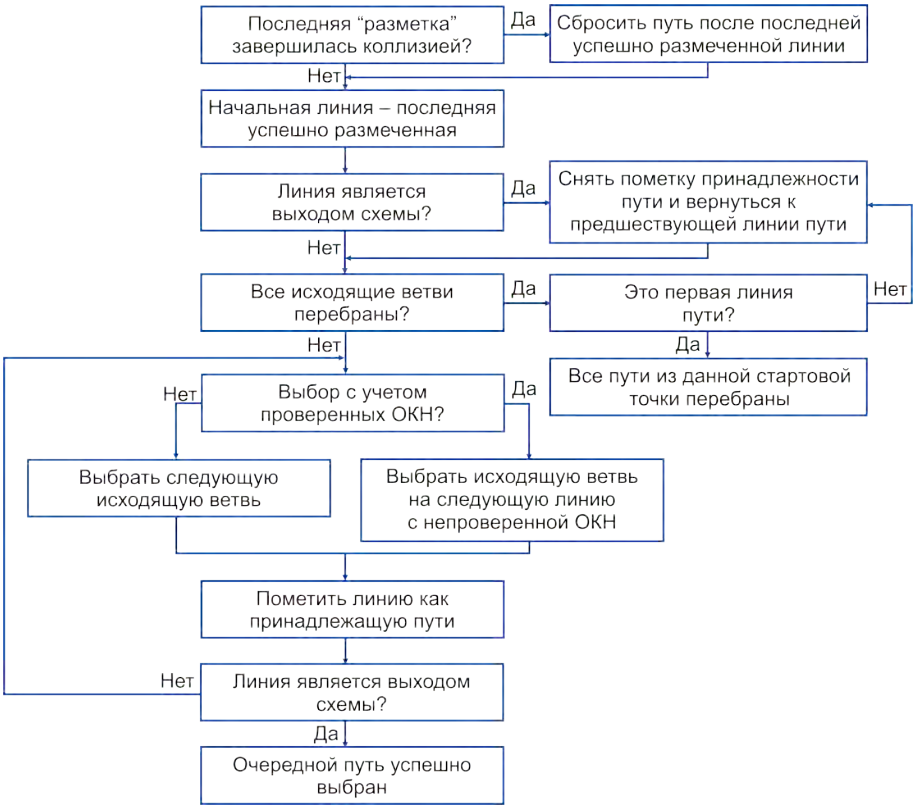


Рис. 11.10. Алгоритм процедуры выбора пути

После выбора очередного пути активизации выполняется его разметка. Суть ее состоит в расстановке на линиях, принадлежащих пути активизации, пары противоположных сигналов. Для каждого примитива, принадлежащего пути активизации, необходимо определить пару входных воздействий, вызывающих изменение (0-1 или 1-0) на его выходе. В схемах со сходящимися разветвлениями возможна ситуация, когда линия является входной для двух или более активных примитивов. Разметка линий потребует установки противоречивых значений (0 и 1) на общих линиях. Поэтому разметка пути активизации проводится от входа схемы к выходу. После выбора пары кубов покрытия примитива для разметки активизации выполняется контроль отсутствия коллизий (противоречий) с уже имеющейся разметкой. Во многих случаях можно выполнить досрочное обнаружение скрытых коллизий разметки без применения обратной фазы. Для этого необходимо параллельно с разметкой выполнять моделирование всех линий схемы, что позволяет оценить распространение активизации по побочным ветвям и проследить продвижение сигналов ее обеспечения. Досрочное выявление подобных коллизий позволяет выбрать только непротиворечивые разметки. В случае невозможности выполнения такой разметки выполняется переход к следующему пути. При этом алгоритм его выбора найдет следующую ветвь из последней непротиворечивой точки.

Предварительное моделирование выявляет коллизии в схемах, содержащих избыточность. На рис. 11.11 и 11.12 даны примеры разметок, приводящих к неразрешимой коллизии вследствие несущественности линии a в первом случае и невозможности (одномерной) активизации пути b-c-e – во втором случае.

В обоих примерах моделирование элемента ИЛИ-НЕ, не принадлежащего пути активизации, позволяет без выполнения фазы обратного прохода устанавливать невозможность обеспечения подобной разметки. Алгоритм процедуры разметки пути активизации приведен на рис.11.13.

В ряде случаев выполнение предварительного моделирования как исправного поведения, так и ОКН позволяет выявить несущественные и/или не активизируемые линии (см. рис. 11.11), однако выполнение моделирования ОКН на фазе разметки значительно усложняет соответствующую процедуру и отрицательно сказывается на общей производительности генератора тестов.

Первая разметка нового пути выполняется от младших линий к старшим. Если разметка пути прошла успешно, а на фазе обратного прохода возникла коллизия, то разметка пути повторяется – выбирается следующий вариант разметки, при этом вначале перебираются варианты для старших линий. В терминах позиционной системы вес линии в разметке возрастает с уменьшением ее номера, а изменение разметки происходит по принципу счетчика,

начиная с меньших весов. В связи с этим, в случае коллизии, разметка возобновляется, начиная с последней линии схемы (см. рис. 11.13).

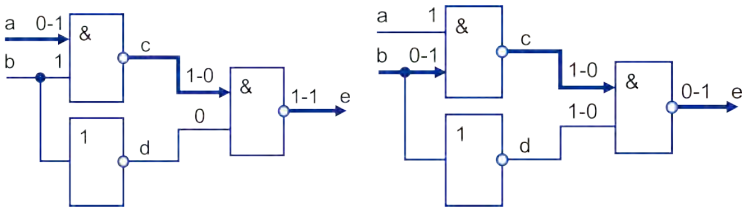


Рис. 11.11. Примеры неразрешимой коллизии по линии a

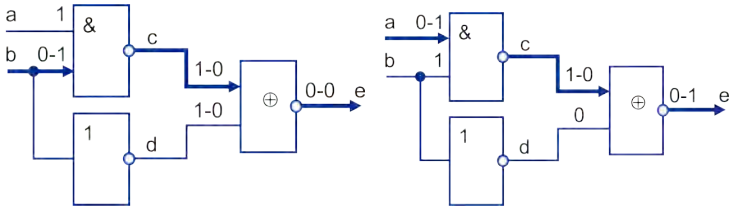


Рис. 11.12. Примеры неразрешимой коллизии по пути b-c-e

После успешного завершения разметки пути активизации выполняется обратный проход – фаза обеспечения сигналов активизации пути необходимыми сигналами на входных и псевдовходных линиях схемы, когда решение находится последовательным выполнением операции пересечения исходного вектора T со всеми (в худшем случае) покрытиями схемы:

$$R = T \bigcap_{1=1}^{p} (C_1,..., C_i,.., C_p). \qquad (11.39)$$

Алгоритм данной процедуры приведен на рис. 11.14. Она выполняет последовательное движение от самой старшей линии к самой младшей функциональной (неаргументной) линии. При этом осуществляется пересечение текущего решения с КП очередной обрабатываемой линии. Исходное заполнение буфера решений соответствует вектору разметки пути. Размер буфера решений является фиксированным, поэтому если в процессе выполнения пересечения происходит переполнение буфера, то проигнорированная ветвь помещается в очередь альтернативных решений. Если в процессе пересечения с очередным КП буфер текущего решения становится пустым, то выполняется восстановление последнего сохраненного в очереди альтернативного решения. Процедура продолжается с той линии, для которой было сохранено это решение. В целях предупреждения зависания генератора на некоторых ветвях максимальное количество возвратов к альтернативному решению ограничивается.

После выполнения фазы обеспечения для каждого из векторов пары активизации получается множество решений. Из каждого массива необходимо

взять только по одному вектору. Метод выбирает пару векторов из всех возможных. В качестве критериев выбора используются свойства векторов [54]:

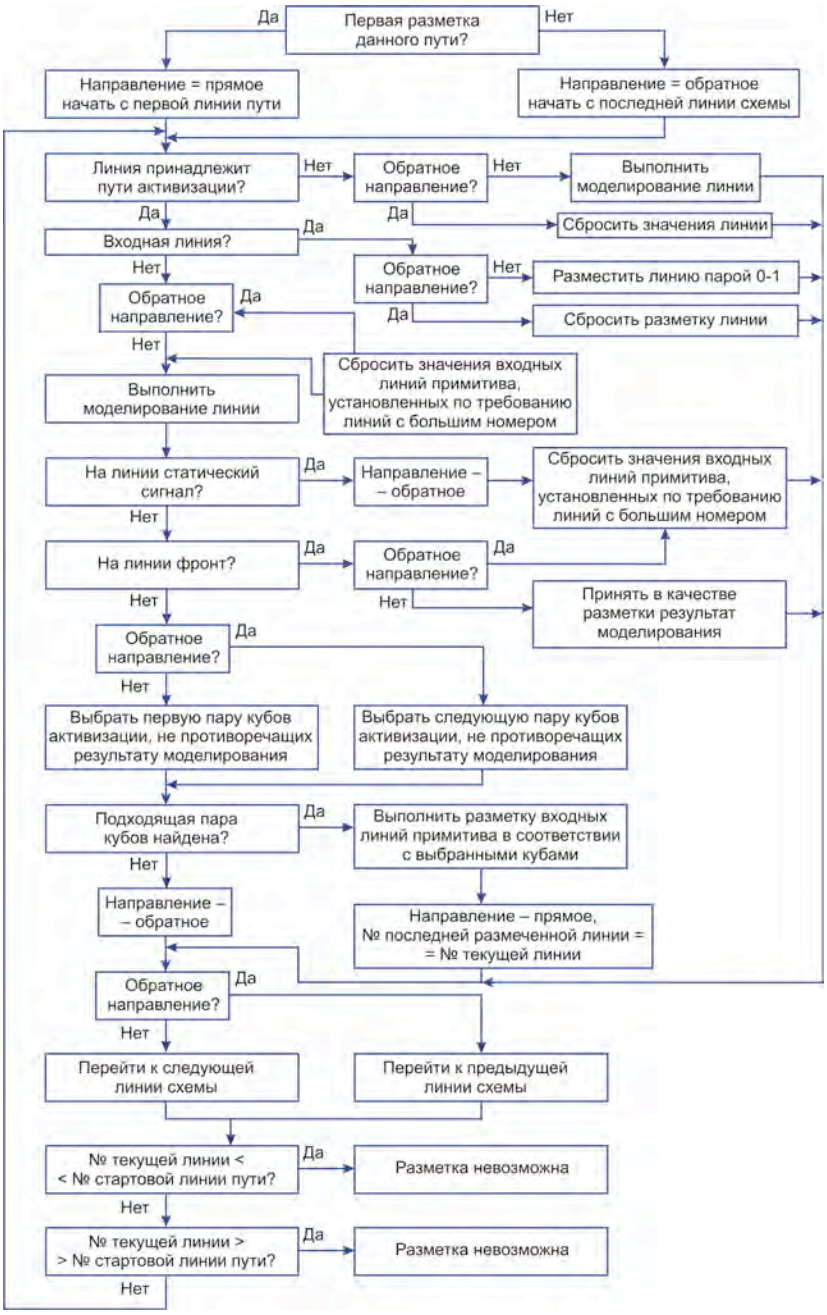


Рис. 11.13. Алгоритм процедуры разметки пути активизации

– количество ОКН, проверяемых вектором;

– совместимость первого вектора пары с последним набором теста;

– совместимость первого и второго векторов пары;

– количество символов X на псевдовходах вектора;

– проверка обеих ОКН на начальной линии пути активизации.

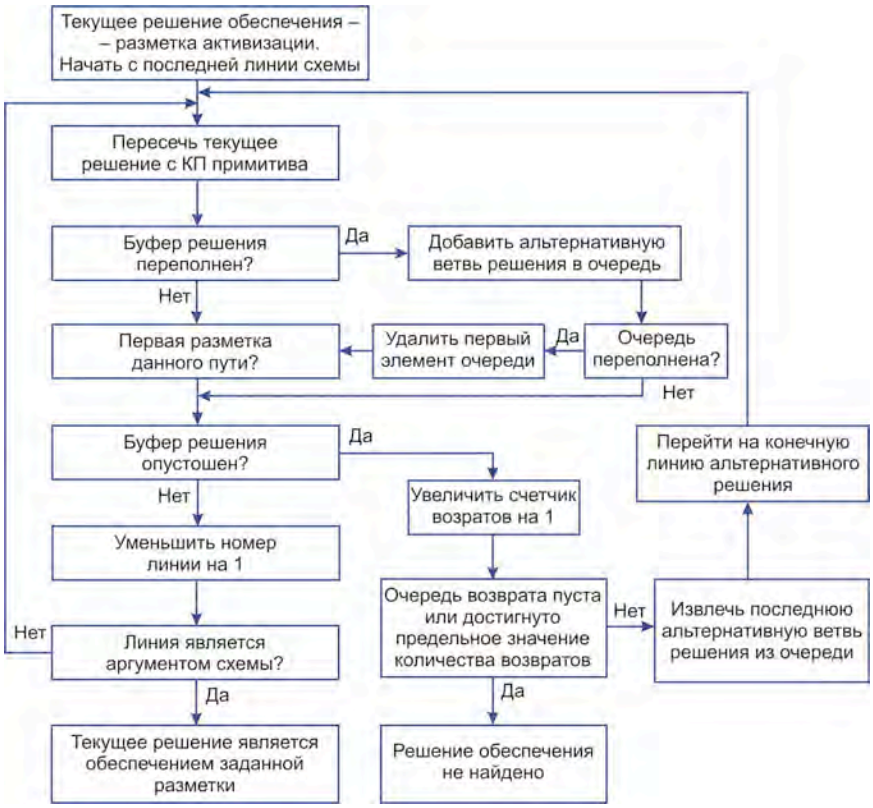


Рис. 11.14. Фаза обеспечения сигналов активизации пути

Приоритетом при сравнении векторов обладают свойства совместимости. Если выполняются оба условия совместимости, дополнительные векторы для пары не требуются, то она получает абсолютный приоритет и выбирается из подобных только по критерию количества проверяемых ОКН. Последний из приведенных выше критериев – проверка обеих ОКН по начальной линии пути активизации – обладает большим приоритетом, чем количество проверяемых ОКН, но меньшим, чем свойство связности линий. Это обусловлено тем, что на следующей итерации активизация начинается с очередной не полностью проверенной линии. Если по данной остаются непроверенные ОКН, то они могут быть идентифицированы только в результате побочного транспортирования ОКН на выход. С другой стороны, если для

векторов не выполняется свойство совместимости, то существует вероятность того, что пара не будет включена в тест. Поэтому проверяющая способность пары имеет меньший приоритет, чем совместимость. Если ни одна проверенная пара не обладает свойством совместимости, тогда учитывается количество символов X по псевдовходам. Чем больше символов X, тем выше вероятность подбора связующего вектора между двумя соседними. Система приоритетов реализуется посредством весовых коэффициентов.

После того, как успешно выполнена фаза обеспечения для каждого из векторов пары активизации, необходимо включить их в тест. Если они требуют обеспечения некоторых сигналов по линиям псевдовходов, то также требуется решение задачи совместимости или связывания.

Определение 11.14. Два вектора будем называть совместимыми, если мастер-линии второго вектора

$$T^{i}(Y_1,\ldots,Y_j,\ldots,Y_k)$$

покрывают своими значениями соответствующие состояния псевдовходов первого вектора

$$T^{i-1}(X_1^Y,\ldots,X_j^Y,\ldots,X_k^Y):$$

$$T^{i-1}(X_1^Y,\ldots,X_j^Y,\ldots,X_k^Y)\bigcap T^{i}(Y_1,\ldots,Y_j,\ldots,Y_k)=T^{i-1}(X_1^Y,\ldots,X_j^Y,\ldots,X_k^Y). \quad (11.40)$$

После того, как из решения обеспечения выбрана пара векторов активизации, необходимо выполнить проверку их совместимости между собой и с последним набором теста. В случае отсутствия совместимости указанной тройки она может быть обеспечена путем модификации исходной пары активизации и/или добавлением промежуточных наборов. За контроль и обеспечение совместимости векторов отвечает модуль их связывания.

Последовательность добавления векторов новой пары активизации существенна для решения задачи совместимости. Если последний вектор теста обозначить как V_0, а наборы новой пары активизации – как V_1 и V_2, то возможные варианты совместимости могут быть представлены двумя графами (рис. 11.15).

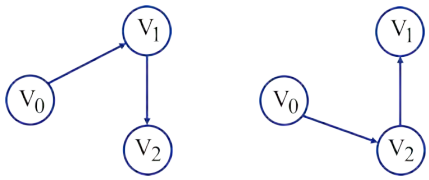


Рис. 11.15. Возможные варианты совместимости

Если ни один из представленных вариантов не выполняется, то для полученной пары активизации решается задача обеспечения совместимости. Для соединения двух несовместимых векторов применяются процедуры:

1) сшивка – модификация первого вектора пары;

2) связывание – вставка дополнительного связующего вектора между векторами исходной пары.

Процедура сшивки двух векторов имеет преимущество перед связыванием, поскольку позволяет обеспечить совместимость пары без введения дополнительных наборов. В предлагаемой реализации алгоритма генерации тестов упомянутая процедура применяется только для решения задачи совместимости пары активизации. Для обеспечения совместимости $V_0 \to V_1$ или $V_0 \to V_2$ метод не применяется. Иначе необходима модификация первого вектора пары V_0. Рассмотрим суть каждой из процедур.

Процедура 11.1. Перед сшивкой двух векторов выполняется пересечение мастер-линий первого вектора с соответствующими псевдовходами второго вектора. Если результат пересечения не пуст, то для результирующего набора (первый вектор пары с модифицированными мастер-линиями) выполняется фаза обеспечения и моделирования. Если на фазе обеспечения было найдено удовлетворительное решение, то первый вектор пары может быть заменен набором из полученного решения, обладающим наибольшей проверяющей способностью. Вектор из найденного решения будет совместим со вторым набором пары активизации путей.

Процедура 11.2. Связывание применяется в случае неприменимости метода сшивки и состоит в поиске вектора, покрывающего своими псевдовходами значения мастер-линий первого вектора. При этом он должен обеспечивать на собственных мастер-линиях значения, покрываемые псевдовходами второго вектора. Поиск такого набора выполняется применением фазы обеспечения к шаблону, содержащему на мастер-линиях соответствующие значения псевдовходов второго вектора, а на псевдовходах – соответствующие значения мастер-линий первого вектора. Если фаза обеспечения дает непустое решение, то в качестве связующего выбирается вектор, обладающий наибольшей проверяющей способностью относительно одиночных константных неисправностей. В этом случае первый набор пары совместим со связующим, а последний – со вторым вектором. Такая тройка может быть представлена непрерывной последовательностью.

В общем случае количество связующих векторов может быть больше одного. Методы сшивки и связывания могут применяться совместно. Практическая реализация подобных алгоритмов становится весьма ресурсоемкой и в большинстве случаев неоправданной. В этом случае затраты на поиск связующих векторов становятся близкими либо даже превосходят затраты на поиск новой пары активизации путей.

В целях минимизации размера решения задачи связывания применяются методы сшивки. В случае неприменимости последних – методы связывания. После решения задачи совместимости в тест будет добавлено от 2-х (обе пары были совместимы либо успешно сшиты) до 4-х (обе пары можно было только связать) векторов. Результат решения задачи связывания может быть представлен диаграммами на рис. 11.16.

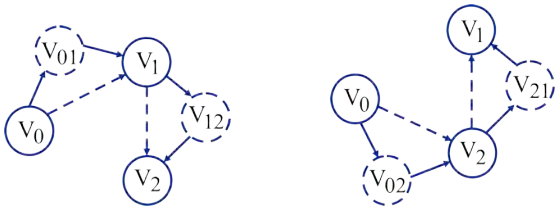


Рис. 11.16. Решение задачи связывания

Если задача связывания решена успешно, то полученная в результате последовательность векторов, называемая секцией активизации, добавляется в конец проверяющего теста. В противном случае текущая пара векторов активизации отбрасывается и выбирается следующий вариант разметки пути активизации эквипотенциальных линий.

После успешного завершения фаз обеспечения и связывания сформированная секция активизации может включать векторы активизации (возможно доопределенные) в порядке, противоположном исходному, а также содержать дополнительные (не полностью определенные) связующие векторы. Поэтому для получения реальной проверяющей способности добавляемой секции теста выполняется ее моделирование (как исправного поведения цифровой схемы, так и ОКН). При этом проводится межтактовое транспортирование значений линий обратных связей и соответствующих списков одиночных константных неисправностей существенных переменных схемы.

После выполнения фазы моделирования секция активизации добавляется в тест и выполняется выбор следующей линии с непроверенными ОКН. Независимо от количества последних на текущей стартовой линии пути активизации выполняется переход к следующей. Такой алгоритм обусловлен необходимостью предупреждения зацикливаний на непроверяемых линиях. Если схема содержит линии с непроверенными ОКН, то ближайшая из них выбирается в качестве стартовой и процедура построения пары активизации повторяется. Если по всем оставшимся линиям проверены все ОКН, то процесс формирования тестовых векторов прекращается и выполняется минимизация и сжатие результирующего теста цифрового устройства.

11.8. Выводы

Представлены специальные методы генерации тестов для компонентов цифровых систем, основанные на использовании технологии генетических алго-

ритмов и теории двухтактного кубического исчисления, позволяющих решать задачи синтеза тестов с заданной полнотой относительно константных неисправностей. Возможности предложенных методов продемонстрированы на валидных примерах обработки цифровых схем. Описаны процедуры и алгоритмы генерации тестов для последовательностных схем и цифровых систем, ориентированные на имплементацию в программные продукты.

Представлена общая модель последовательностного примитивного автомата и его частные случаи для описания триггерных примитивов в одно- и двухступенчатом виде, которые ориентированы на решение задач генерации тестов и моделирования неисправностей соответственно. Их достоинство в троичном функциональном описании автоматной модели в виде КП, которая не требует итераций при построении тестов и моделировании [55].

Предложена математическая модель процесса построения теста на основе совместного анализа кубического покрытия функционального элемента (схемы) и неисправностей, генерируемых по КП. Это дает возможность не выполнять сложные процедуры прямой и обратной импликаций, но взамен необходимо синтезировать покрытие всей схемы [55].

Предложены процедуры анализа схемной структуры для разбиения ее на функциональные фрагменты в целях уменьшения времени генерации тестов и обеспечения возможности обработки устройств большой размерности [56].

Представлены стратегии построения тестов, включающие выбор и использование псевдослучайных генераторов, детерминированных и алгоритмических, дающие возможность уменьшить время проектирования тестов, благодаря распараллеливанию процессов на основе декомпозиции схемы и разбиения множества используемых генераторов тестов.

Предложена модификация алгоритма генерации тестов комбинационных и последовательностных схем, заданных структурно-функциональными моделями с использованием кубических покрытий. Большинство описанных алгоритмических решений направленно на сокращение вычислительных затрат: 1) Представление КП примитивов в виде двух секций входных кубов, устанавливающих на выходе примитива соответственно 0 и 1, что позволяет значительно упростить процедуру формирования пар активизации выхода примитива, а при проведении пересечения вектора T с КП фактически пересекать набор только с теми кубами КП, которые не противоречат T по выходной линии. 2) Реализация предварительного исправного моделирования еще на этапе разметки пути, что позволяет досрочно идентифицировать коллизии разметки. 3) Выполнение процедуры выбора следующей разметки пути по принципу счетчика в направлении от выходных линий ко входным – вначале перебирать разметку старших линий а затем младших – использовать в последующей разметке непротиворечивый фрагмент предшествующей. 4) Реализация процедуры выбора следующего пути активизации после неудачной разметки осуществляется с последней успешно размеченной линии, отбрасывается только та часть пути активизации, которая привела к

коллизии, а не формируется весь путь повторно. 5) Выбор в качестве стартовой линии с непроверенной ОКН. 6) При выполнении обратной фазы обеспечивается сохранение ветви решения, не поместившейся в буфер, и возврат к последнему сохраненному решению при фиксации тупика для текущего решения. 7) В случае переполнения альтернативных решений осуществляется удаление из очереди решения, занесенного первым. 8) Предварительное моделирование всех пар векторов активизации, полученных на фазе обеспечения, в целях выбора пары, имеющей максимальную относительную проверяющую способность. 9) Реализация межтактового транспортирования по линиям ОС списков проверяемых ОКН для повышения адекватности моделирования. 10) При несовместимости новой пары векторов активизации осуществляется безизбыточная сшивка векторов или связывание путем введения дополнительного вектора. 11) Ограничение количества связующих векторов одним для уменьшения затрат на обеспечение отдельной пары активизации. 12) Применение отдельных оптимизированных процедур минимизации и сжатия тестов для комбинационных схем.

Выполнено разбиение модели генератора тестов на ряд элементарных, представляющих независимые сущности: символы, одно- и двухтактные кубы, массивы однотактных кубов, кубические покрытия примитивов, линии схемы как функциональные переменные, тесты проверки ОКН, очереди альтернативных решений задачи обеспечения. Для всех перечисленных моделей определены необходимые структуры данных, наборы процедур для их обработки. Разработаны структуры данных для представления исходной схемы и результатов работы генератора тестов, рациональные с точки зрения быстродействия предложенного алгоритма [57]. Вычислительная сложность алгоритма генерации тестов носит вероятностный характер и зависит от начального приближения, от варианта выбора линий ОС (подлежащих разрыву), результатов ранжирования линий схемы, последовательности расположения кубов КП примитивов, выбора начальной точки решения, а также от существования решения для всех тестируемых линий. Однако для отдельных процедур алгоритма может быть оценено предельное значение вычислительной сложности, измеряемое количеством элементарных операций (пересечение, объединение или поглощение) над двумя кубами. Обозначения: N_{buf} – размер буфера решений задачи обеспечения; N_{cov} – среднее количество кубов в КП примитивов; N_{branch} – среднее количество исходящих ветвлений в расчете на одну линию схемы; N_{back} – максимальное число откатов к альтернативному решению; N_{mrk} – максимальное количество попыток разметить один путь; N_{fun} – количество функциональных (невходных) линий схемы; N_{path} – количество линий, принадлежащих пути активизации; N_{sel} – максимальное число попыток выбрать путь из одной стартовой точки; N_{pair} – среднее количество пар активизации примитива, не противоречащих

текущей разметке. Предельная вычислительная сложность фазы обеспечения может быть оценена максимальным количеством операций пересечения вектора текущего решения с векторами кубического покрытия примитивного элемента устройства:

$$N_{prov} = \frac{1}{3}(N_{fun} \times N_{buf} \times N_{back} \times N_{cov}). \qquad (11.41)$$

Коэффициент 1/3 обусловлен тем, что пересечение вектора текущего решения можно проводить не со всеми векторами КП примитивов, а только с кубами, которые устанавливают на выходе значение, требуемое данным вектором текущего решения. Для схемы с большим числом разнотипных примитивов можно предположить, что среднее количество кубов покрытий, устанавливающих 0 и 1, будет примерно равно; полагая равновероятными события появления в промежуточном решении вектора со значением 0, 1 и X по некоторой координате, получаем значение коэффициента: 1/2 × 1/3 + 1/2 × 1/3 = 1/3. Количество возможных вариантов разметки пути может быть оценено как:

$$N_{mark} = (N_{pair})^{N_{path}}, \qquad (11.42)$$

однако предлагаемый алгоритм вводит ограничение на количество попыток разметки одного пути (N_{mrk}). Тогда предельная вычислительная сложность генерации тестовой последовательности, принудительно активизирующей и проверяющей некоторую линию схемы, может быть оценена как:

$$N_{check} = (N_{prov} + N_{model} + N_{link}) \times N_{mrk} \times N_{sel}, \qquad (11.43)$$

где N_{model} – вычислительная сложность процедуры полного моделирования, N_{link} – вычислительная сложность процедуры связывания.

В указанном выше соотношении полное моделирование включает: предварительное моделирование всех возможных пар активизации, полученных на фазе обеспечения; N_{buf}^2 – число операций моделирования ОКН и исправного поведения схемы; моделирование исправного поведения и ОКН секции активизации.

11.9. Литература

1. Pawel Tomaszewicz, Mariusz Rawski Self-Testing of User-Programmed FPGAs Based on the Concept of Linear Segments // IEEE Digital System Design. – 2001. – P. 236-242.

2. Irith Pomeranz, Sudhakar M. Reddy. On Finding a Minimal Functional Description of a Finite-State Machine for Test Generation for Adjacent Machines // IEEE Transactions on Very Large Scale Integration (VLSI) Systems. – 2000. – № 1. – P. 88-95.

3. Irith Pomeranz, Sudhakar M. Reddy. Functional Test Generation for Full Scan Circuits // Design, Automation and Test in Europe. – 2000.– P. 321-327.

4. Rob Sumners, Jayanta Bhadra, Jacob Abraham. Automatic Validation Test Generation using Extracted Control Models // 13th International Conference on VLSI Design. – 2000.– P. 177-179.

5. Indradeep Ghosh, Masahiro Fujita. Automatic Test Pattern Generation for Functional RTL Circuits Using Assignment Decision Diagrams. // Design and Automation Conference.– 2000. – P. 45-56.

6. F. Corno, M. Sonza Reorda, G. Squillero, A. Manzone, A. Pincetti. Automatic Test Bench Generation for Validation of RT-level Descriptions: an Industrial Experience // Design, Automation and Test in Europe. – 2000.– P.229-239.

7. Хаханов В.И., Сысенко И.Ю., Абу Занунех Халиль И.М. Проектирование тестов для структурно-функциональных моделей цифровых схем // Радиоэлектроника и информатика. – 1999. – №3. – С. 51-59.

8. R. Guo, I. Pomeranz and S. M. Reddy. On Speeding-up Vector Restoration Based Static Compaction of Test Sequences for Synchronous Sequential Circuits // 7th Asian Test Symp. – 1998. – P. 467-471.

9. Khaled Saab, Naim Ben-Hamida, Bozena Kaminska. Parametric Fault Simulation and Test Vector Generation // Design, Automation and Test in Europe. – 2000. – P.41-43.

10. Андрюхин А.И. Алгоритмы параллельного логического моделирования и псевдослучайной генерации тестов для МОП-структур // Микроэлектроника. – 1995. – №5. – С. 331-336.

11. Cheng K.T., Lin C.J. Timing Driven Test Point Insertion for Full-Scan and Partial-Scan BIST // IEEE Int. Test Conf. – 1995. – P. 506-514.

12. Oliver Chiu-sing Choy, Lap-kong Chan, Ray Chan, Cheong F. Chan. Test Generation with Dynamic Probe Points in High Observability Testing Environment // IEEE Transactions on Very Large Scale Integration (VLSI) Systems. – 1996. – № 1. – P. 88-97.

13. Muradali F., Agarwal V.K., Nadeau-Dostie B. A new procedure for Weighted Random Built-In Self-Test // IEEE Int. Test Conf. – 1990. – P. 660-669.

14. Wunderlich H.J. Multiple Distributions for Biased Random Test Patterns // IEEE Trans. on CAD. – 1990. – Vol.9, № 6. – P.584-593.

15. Kanagaris D.B., Tragoudas S. Generating Deterministic Unordered Test Patterns with Counter // IEEE VLSI Test Symp. – 1996. – P. 374-379.

16. Ha D., Lee H. On the Geneation of the Test Patterns for Combinational Circuits // Department of Electrical Engineering, Virginia Polytechnics Institute and State University, technical report . – December. – 1993. – P. 155-160.

17. Fagot C., Gascuel O., Girard P., Landrault C. On Calculating Efficient LFSR Seeds for Built-In Self Test // ETW. – 1999. – P. 7-14.

18. Garbolino T., Hlawiczka A. On the Designing of New Efficient Cellular Automata Structures for Built-In Test Pattern Generators // MIXDES Conference. – 2001. – P. 195-200.

19. Angela Krstic, Kwang-Ting (Tim) Cheng, Srimat T. Chakradhar. Primitive Delay Faults: Identification, Testing, and Design for Testability // IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems. – 1999. – № 6. – P. 669-684.

20. Gramatova E., Cibakova A., Miklos P. Defect-Oriented TPG for combined IDDQ-Voltage Testing of Combinational Circuits // ETW.– 2000. – P. 223-225.

21. Blyzniuk M., Cibakova T., Gramatova E., Kuzmicz W., Pleskacz W., Raik J., Ubar R. Defect Oriented Fault Coverage of 100 % stack-at Fault Test Sets // MIXDES. – 2000. – P. 132-135.

22. Lan Zhao, D. M. H. Walker, Fabrizio Lombardi. IDDQ Testing of Input/Output Resources of SRAM-based FPGAs // Eigth Asian Test Symposium. – 2000. – P. 145-151.

23. Eugeni Isern, Joan Figueras. IDDQ Test and Diagnosis of CMOS Circuits // IEEE Design & Test of Computers. – 1995. – № 4. – P. 60-67.

24. Cibakova T., Fischerova M., Gramatova E., Kuzmicz W., Pleskacz W., Raik J., Ubar R. Defect-Oriented test Generation using Probabilistic Estimation // MIXDES conf. – 2001. – P. 131-136.

25. Ilker Hamzaogluand, Janak H. Patel. New Techniques for Deterministic Test Pattern Generation // 16th IEEE VLSI Test Symposium. – 1998. – P. 148-156.

26. Olfati E., Bengt M. MCM testing by Boundary Scan // MIXDES. – 2000. – 446-449.

27. Olfati E. Built-In Self Test (BIST) in a microprocessor based automotive application // Master thesis. – 2001. – P. 158.

28. Silvia Cataldo, Silvia Chiusano, Paolo Prinetto. Optimal Hardware Pattern Generation for Functional BIST // Design, Automation and Test in Europe. – 2000. – P. 245-248.

29. Cheng W.T. The Back Algorithm for Sequential Test Generation // Int. Conf. of Computer Design. – 1988. – P. 66-69.

30. Cheng W.T. and Davidson S. Sequential Circuit Test Generator (STG) Benchmark Results // Int'l Symp. Circuits & Systems. – 1989. – P. 1938-1941.

31. Cheng W.T. and Chakraborty T. Gentest – An Automatic Test-Generation System for Sequential Circuits // IEEE Computer.–1989. –Vol. 22, № 4.–P. 43-49.

32. Niermann T. and Patel J. HITEC: A Test Generation Package for Sequential Circuits // in European Conf. on Design Automation. – 1991. – P. 214-218.

33. Kelsey T., Saluja K. and Lee S. AnEcient Algorithm for Sequential Circuit Test Generation // IEEE Trans. on Computer. – 1993. – Vol. 42. – P. 1361-1371.

34. Ruifeng Guo, Irith Pomeranz, Sudhakar M. Reddy. A Fault Simulation Based Test Pattern Generator for Synchronous Sequential Circuits // 17th IEEE VLSI Test Symposium. – 1999. – P.123-127.

35. Rudnick E. M., Holm J. G., Saab D. G. and Patel J. H. Application of Simple Genetic Algorithms to Sequential Circuit Test Generation // European Design and Test Conf. – 1994. – P. 40-45.

36. Prinetto P., Rebaudengo M. and Reorda M. S. An Automatic Test Generator for Large Sequential Circuits Based on Genetic Algorithm // Int'l Test Conf. – 1994. – P. 240-249.

37. Hsiao M. S., Rudnick E. M. and Patel J. H. Sequential Circuit Test Generation Using Dynamic State Traversal // Europ. Design & Test Conf. – 1996. – P. 22-28.

38. Xijiang Lin, Irith Pomeranz, Sudhakar M. Reddy. Techniques for Improving the Efficiency of Sequential Circuit Test Generation // International Conference on Computer-Aided Design Proceedings. – 2000. – P. 23-26.

39. Tsai K.H., Marek-Sadowska M., Rajaski J. Scan-Encoded Test Pattern Generation for BIST // Int'l Test Conf. – 1997. – P. 548-556.

40. Pomeranz I. and Reddy S. M. Built-in Test Generation for Synchronous Sequential Circuits // Int’l Conf. on Computer Aided Design. – 1997. – P. 421-426.

41. Goldberg E. Genetic algorithms. – Addison-Wesley USA. – 1991. – 314 p.

42. Хаханов В.И., Скворцова О.Б., Бабич А.В. Алгоритм построения кубических покрытий для схем с триггерными структурами (модифицированный П-алгоритм). Метод генетических алгоритмов для синтеза тестов верификации цифровых систем // АСУ и приборы автоматики. – 2001. – № 115. – С. 105-111.

43. Хаханов В.И., Хак Х.М. Джахирул, Масуд М.Д. Мехеди. Модели анализа неисправностей цифровых систем на основе FPGA, CPLD // Технология и конструирование в электронной аппаратуре.– 2001.– № 2.– С. 3-11.

44. Колесников К.В., Хаханова А.В., Побеженко И.А. Дедуктивный метод обратного моделирования неисправностей цифровых проектов.– Материалы 2-й Международной научно-технической конференции "Проблемы информатики и моделирования.– Харьков.– 2002.– Харьков: НТУ"ХПИ".– 2002.– С. 44.

45. Хаханов В.И., Ханько В.В, Бедратый Р.А. Структурный анализ многозначных таблиц неисправностей для диагностирования цифровых устройств // АСУ и приборы автоматики. Х.: ХТУРЭ.– 1998.– Вып. 107.– С. 35-43.

46. Хаханов В.И. Техническая диагностика элементов и узлов персональных компьютеров.– К.: ІЗМН.– 1997.– 308с.

47. Автоматизация диагностирования электронных устройств/ Ю.В.Малышенко и др./ Под ред.В.П.Чипулиса.– М.: Энергоатомиздат, 1986.– 216с.

48. Хаханов В.И., Сысенко И.Ю., И.М. Абу Занунех Халиль. Проектирование тестов для структурно-функциональных моделей цифровых схем // Радиоэлектроника и информатика.– 1999.– №.3.– С. 51-59.

49. Бондаренко М.Ф., Кривуля Г.Ф., Рябцев В.Г., Фрадков С.А., Хаханов В.И. Проектирование и диагностика компьютерных систем и сетей. К.: НМЦ ВО. – 2000. – 306 с.

50. Основы технической диагностики / Под. ред. П.П.Пархоменко. М.: Энергия. – 1976. – 460с.

51. Баранов С.И.,Майоров С.А., Сахаров Ю.П., Селютин В.А. Автоматизация проектирования цифровых устройств. Л.: Судостроение, 1979. – 264с.

52. Автоматизированное проектирование цифровых устройств / С.С.Бадулин, Ю.М.Барнаулов и др. / Под ред. С.С. Бадулина. М.: Радио и связь, 1981. – 240с.

53. Автоматизация диагностирования электронных устройств / Ю.В.Малышенко и др./ Под ред.В.П.Чипулиса.М.: Энергоатомиздат. – 1986. – 216 с.

54. Масуд М.Д. Мехеди. Стратегии построения тестов для цифровых систем // Радиоэлектроника и информатика.– 2001.– № 3.– С. 50-55.

ЗАКЛЮЧЕНИЕ

Книга представляет собой некоторый итог эволюции субъективных знаний о наиболее динамически развивающейся области, связанной с созданием и тестированием цифровых систем на основе пакетов кристаллов.

Сущность изложенного материала состоит в представлении технологий, моделей, методов и алгоритмов проектирования и тестирования цифровых систем на кристаллах, имплементированных в современные и компактные конструктивы. Большая часть предложенного материала есть оригинальные или усовершенствованные разработки авторов, направленные на создание компонентов встроенной инфраструктуры сервисного обслуживания цифровых изделий в пакетах.

Книга освещает собственные многолетние разработки авторов, дополненные современными технологиями от ведущих компаний и ученых планеты, определяющих политику современного развития компьютерной инженерии. Большое влияние на содержание книги оказали выдающиеся авторитеты в области Design and Test: Ервант Зориан (Yervant Zorian), Павел Пархоменко, Владимир Тарасенко, Раймунд Убар, Дмитрий Сперанский, Аркадий Закревский, Владимир Твердохлебов, Самвел Шукурян (Samvel Shoukourian), Вислав Кузьмич (Wieslaw Kuzmicz).

Выражаем благодарность коллегам и друзьям по диагностическому цеху, безмерно помогающим в формировании правильного и широкого понимания современных и будущих тенденций технической диагностики. Это ученые и организаторы науки: Геннадий Кривуля, Михаил Каравай, Вячеслав Харченко, Александр Дрозд, Юрий Скобцов, Анжела Матросова, Вячеслав Ярмолик, Роман Базилевич, Алексей Романкевич, Леонид Дербунович, Владимир Рябцев, Римантас Шейнаускас, Александр Баркалов, Юрий Малышенко, Сергей Шаршунов, Владимир Рустинов, Александр Шкиль.

Значительную помощь в оформлении оригинал-макета оказали Светлана Чумаченко и Ольга Шевченко. Слова благодарности за прекрасную работу Александре Петровне Гужве, постоянному непревзойденному литературному редактору, с которой мы сотрудничаем на протяжении многих лет.

Следует также выразить теплые слова признательности коллегам (Владимир Елисеев, Светлана Чумаченко, Ирина Хаханова, Евгений Ковалев, Вадим Ханько, Константин Колесников, Ольга Скворцова, Ирина Сысенко, Марина Бережная, Халиль Абу Занунех, Масуд Мехеди, Хак Джахирул, Хассан Ктейман, Ваде Гриби, Александр Егоров, Олеся Гузь, Евгений Каменюка, Марина Каминская, Александр Парфентий), которые на протяжении последних десяти лет принимали активное участие в формировании технологической научной школы в области Design and Test. Новая волна молодых ученых и исследователей (Сергей Зайченко, Владимир Обризан, Андрей Руткас, Кристофер Нгене, Ив Тиекура, Алексей Сушанов, Сергей Михтонюк, Максим Давыдов и Сергей Галаган) ориентирована на создание технологий в области design & testing и их внедрение в мировой рынок микроэлектроники.

СПИСОК СОКРАЩЕНИЙ

AC – Assertion Controller – контроллер ассерций
Ad-Hoc – специализированные и проблемно-ориентированные технологии
ADTG – Adder Test Generator – генератор тестов для сумматора
AL – Algebra-Logical Method – алгебро-логический метод
AMBA – Advanced Microcontroller Bus Architecture
ASIC – Application-Specific Integrated Circuit
ASSP – Application Specific Standard Product
ATE – Automatic Test Equipment – автоматическое тестовое оборудование
ATPG – Automated Test Pattern Generator
BIRA – Built-In Repair Analysis
BSDL – Boundary-Scan Description Language
BISR – Built-In Self Repair – встроенное самовосстановление
BIST – Bilt-in Self Testing – встроенное самотестирование
BSR – Boundary Scan Register
BSTG – Bus Test Generator
CAD – Computer-Aided Design – САПР
CLB – Complex Logic Block
CPLD – Complex PLD
CPU – Central Processor Unit
CTL – Core Test Language
DFM – Design-for-Manufacturability
DFT – Design-for-Testability
DFTG – Design Flow Test Generator
DSP – Digital Signal Processing
EA – Embedded Array – полузаказная ИС
EDA – Electronic Design Automation
ESL – Electronic System Level
FDT – Fault Detection Table – таблица неисправностей
F-IP – Functional Intellectual Property
FPGA – Field Programmable Gate Array
FS – Fault Simulator
FV – Formal Verification – формальная верификация
GA – Gate Array – вентильная матрица
GUI – Graphical User Interface
HDL – Hardware Description Language – язык описания аппаратуры
HDTV – High Definition Television
HES™ – Hardware Embedded Simulator
HFS – Hardware Fault Simulator
HW-SW – Hardware-Software
IEEE –Institute of Electrical and Electronics Engineers

I-IP – Infrastructure Intellectual Property
IP-core – Intellectual Property Core
JTAG – Joint Test Access Group
LCA – Logic Cell Array – массив логических ячеек
LUT – Look-up Table
MEMS – Microelectromechanical System
METG – Memory Test Generator
MCM – Multichip Module – многокристальный модуль
MCP – Multichip Package – многокристальный пакет
MTBF – Mean Time Between Failures – средняя наработка на отказ
NoC – Network on Chip – сеть на кристалле
PAL – Programmable Array Logics
PCI – Peripheral Connect Interconnect
PiP – Package-in-Package
PLA – Programmable Logic Arrays – программируемый логический массив
PLD – Programmable Logic Device– программируемое логическое устройство
PoP – Package-on-Package
PRD – Product Requirement Document
PROM – Programmable Read Only Memory
PRTG – Pseudo-Random Test Generator – генератор псевдослучайных тестов
PSM – Programmable Switch Matrix – программируемая матрица переключателей
RAM – Random Access Memory
RCTG – Register Counter Test Generator
RF – Radio Frequency
RTL – Register-Transfer Level
SAM – System Architectural Model
SATG – Signature Analysis Test Generator
SC – Standard Cell – заказная ИС со встроенными ячейками
SECT – Standard for Embedded Core Test
SIGETEST – SImulation, GEneration of TEST
SiP – System in Package – система в пакете кристаллов
SoC – System on Chip – система на кристалле
SPTG – Single Path Test Generator
STA – Static Timing Analysis
STAR – Self-Testing Area – область самотестирования
TAM – Test Access Mechanism
TAP – Test Access Port Controller
TLM – Transaction-Level Modeling
VHDL – Very high speed integrated circuits Hardware Description Language
VLSI – Very Large Circuit Integrated
АФПГ – алгебраическая форма представления графа
АЦП – аналого-цифровой преобразователь
БИС – большая интегральная схема

ВПН – вектор подозреваемых неисправностей
ВЭП – вектор экспериментальной проверки
ГРП – граф регистровых передач
ДИ – диагностическая информация
ДКИ – двухтактное кубическое исчисление
ДНФ – дизъюнктивная нормальная форма
ДС – древовидная структура
ДП – дедуктивно-параллельный метод
ИС – интегральная схема
КИ – кубическое исчисление
КНФ – конъюнктивная нормальная форма
КП – кубическое покрытие
КПСН – кубическое покрытие списков неисправностей
КФПГ – кубическая форма представления графа
МГА – метод генетических алгоритмов
МДД – матрица действительных дефектов
МП – микропроцессор
МТН – многозначная таблица неисправностей
МЭП – матрица экспериментальной проверки
ОДП – обратное дедуктивно-параллельное моделирование
ОЗУ – оперативное запоминающее устройство
ОКН – одиночная константная неисправность
ОМ – обобщенный модельный автомат
ПЛИС – программируемая логическая интегральная схема
ППЗУ – постоянное запоминающее устройство
ПЭ – примитивный элемент
СБИС – сверхбольшая интегральная схема
СДО – система диагностического обслуживания
СДНФ – совершенная дизъюнктивная нормальная форма
СР – сходящиеся разветвления
ТН – таблица неисправностей
ТОДП – топологическое обратное дедуктивно-параллельное моделирование
ТФН – таблица функций неисправностей
ЦАП – цифро-аналоговый преобразователь
ЦС – цифровая система
ЦУ – цифровое устройство

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

1. ТЕХНОЛОГИЧЕСКИЕ КОНСТРУКТИВЫ ДЛЯ ИМПЛЕМЕНТАЦИИ ЦИФРОВЫХ СИСТЕМ.....5
1.1. Стадии и средства имплементации идеи 5
1.2. Структура и технологии конструктивов цифровых систем 11
1.3. Технологии изготовления и тестирования цифровых систем в пакетах .22
1.4. Тестопригодное проектирование SiP на основе стандарта IEEE 1500....26
1.5. Выводы 35
1.6. Литература.....35

2. АРХИТЕКТУРЫ КОМПОНЕНТОВ ДЛЯ ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ СИСТЕМ.....38
2.1. Состояние рынка электронных технологий 38
2.2. Мобильное специализированное цифровое изделие.....39
2.3. Законы развития рынка электронных технологий 42
2.4. Универсализация или специализация? 45
2.5. Архитектуры систем на кристаллах программируемой логики.....47
2.5.1. Введение 47
2.5.2. Развитие микросхемотехники.....48
2.5.3. Характеристики компонентов программируемой логики 49
2.5.4. Классификация PLD 50
2.5.5. Архитектуры кристаллов программируемой логики 51
2.6. Технологии проектирования цифровых систем на кристаллах 60
2.6.1. Введение 61
2.6.2. Специализированные интегральные схемы – ASIC 62
2.6.3. Технологии и маршруты процесса проектирования 64
2.6.4. Верификация CPLD, FPGA кристаллов.....71
2.6.5. Intellectual Property (IP-) Cores.....72
2.6.6. Проектирование и верификации SoC 75
2.7. Методология системного проектирования TLM 78
2.8. Практическая значимость анализа технологий проектирования 86
2.9. Литература.....91

3. ДЕДУКТИВНО-ПАРАЛЛЕЛЬНОЕ МОДЕЛИРОВАНИЕ НЕИСПРАВНОСТЕЙ ЦИФРОВЫХ СИСТЕМ.....98
3.1. Введение 98
3.2. Обобщенная модель дедуктивно-параллельного анализа дефектов.....99
3.3. Внутренняя модель анализа цифрового устройства.....105
3.4. Дедуктивный метод структурного анализа графа схемы 108
3.5. Метод параллельного дедуктивного моделирования графовых структур цифровых систем 112
3.6. Структурный анализ цифровых последовательностных схем 114

3.7. ОДП-метод моделирования неисправностей цифровых систем 115
3.8. Оценка быстродействия ОДП-метода..... 119
3.9. Топологический ОДП-метод моделирования 120
3.10. Дедуктивный анализ дефектов 128
3.11. Синтез дедуктивных компонентов для функций SoC 132
3.12. Структурные модели примитивов симулятора 139
3.13. Выводы..... 145
3.14. Литература 145

4. ИНФРАСТРУКТУРА ДИАГНОСТИЧЕСКОГО ОБСЛУЖИВАНИЯ SOC..... 149
4.1. Инфраструктура I-IP 149
4.2. Алгебрологический метод диагностирования дефектов..... 153
4.3. Моделирование для уточнения диагноза..... 157
4.4. Структурно-логический метод диагностирования дефектов 161
4.5. Векторно-логический метод диагностирования по таблице неисправностей 163
4.6. Программное средство «Defect Analyzer» 167
4.7. Алгебрологический метод ремонта памяти 174
4.8. Выводы..... 181
4.9. Литература 183

5. ТЕХНОЛОГИИ ТЕСТИРОВАНИЯ SYSTEM-IN-PACKAGE..... 189
5.1. Введение в проблему 190
5.2. Технологии восстановления работоспособности SiP..... 192
5.3. Технологии диагностирования блоков FPGA 199
5.3.1. Алгебро-логический метод диагностирования неисправностей..... 200
5.3.2. Векторно-логический метод диагностирования неисправностей..... 202
5.4. Алгебрологический метод ремонта блоков FPGA..... 204
5.5. Метод обхода матрицы логических блоков для покрытия дефектных компонентов FPGA ремонтными клетками 209
5.6. Программное средство aGalls для покрытия дефектных компонентов FPGA ремонтными клетками..... 215
5.7. Научная новизна и практическая значимость..... 220
5.8. Литература 222

6. ТЕСТИРОВАНИЕ ПРОГРАММНО-АППАРАТНЫХ КОМПОНЕНТОВ SOC..... 224
6.1. Тестопригодность программно-аппаратных продуктов 224
6.2. Модель процесса верификации проекта 228
6.3. Тестирование, диагностирование и ремонт F-IP 233
6.4. Синтез и анализ тестов на граф-модели 241
6.5. Технология диагностирования software модуля 243
6.5.1. Построение графа регистровых передач 245
6.5.2. Синтез и анализ тестов 246
6.5.3. Построение таблицы неисправностей..... 247

6.5.4. Определение диагноза 248
6.6. Выводы 250
6.7. Литература 250

7. ИЕРАРХИЧЕСКИЕ МОДЕЛИ ТЕСТИРОВАНИЯ СЛОЖНЫХ ЦИФРОВЫХ СИСТЕМ 253
7.1. Актуальность проблемы 253
7.2. Выбор модели тестирования сложной цифровой системы 255
7.3. Анализ методов тестирования на основе IEEE стандартов 259
7.4. Анализ архитектур IEEE BS стандартов 267
7.5. Усовершенствование технологий, моделей и методов тестирования сложных цифровых систем 271
7.6. Повышение качества теста на основе технологии IEEE BS 275
7.6.1. Boundary scan архитектура 275
7.6.2. Математическая модель тестирования дефектов 277
7.6.3. Структурный анализ схемы 281
7.7. Выводы 284
7.8. Литература 286

8. ПОИСК ДЕФЕКТОВ НА ОСНОВЕ АНАЛИЗА ТАБЛИЦ НЕИСПРАВНОСТЕЙ 291
8.1. Двузначные таблицы неисправностей 291
8.2. Анализ многозначных таблиц неисправностей 293
8.3. Структурный анализ многозначных таблиц неисправностей 295
8.4. Проектирование условных алгоритмов поиска дефектов 301
8.4.1. Сигнатурный алгоритм обратного прослеживания 301
8.4.2. Алгоритм половинного деления неисправной области 307
8.4.3. Стратегия зондирования дефектов на основе анализа МТН 315
8.4.4. SL-алгоритм диагностирования тестопригодных цифровых структур 320
8.5. Выводы 323
8.6. Литература 324

9. ДВУХТАКТНОЕ КУБИЧЕСКОЕ ИСЧИСЛЕНИЕ 326
9.1. Концепция представления моделей цифровых объектов 326
9.2. Табличное представление моделей цифровых устройств 332
9.3. Анализ кубических покрытий 347
9.4. Алгебраическая форма представления графа 354
9.5. Кубическая форма представления графа 358
9.6. Топологическое представление символов А-алфавита 365
9.7. Выводы 367
9.8. Литература 367

10. СИНТЕЗ ТЕСТОВ ДЛЯ КОМПОНЕНТОВ ЦИФРОВЫХ СИСТЕМ 369
10.1. Объект тестирования 369
10.2. Формализация задач тестирования цифровых систем 372

10.3. Аппарат двухтактного кубического исчисления 373
10.4. Проектирование двухтактных КП активизации 374
10.5. Процедуры кубического алгоритма 378
10.6. Особенности построения тестов для избыточных схем..... 382
10.7. К-алгоритм генерации тестов для комбинационных схем..... 385
10.8. Проектирование тестов для последовательностных схем 393
10.9. Общая модель функционального элемента..... 393
10.10. Модель генерации тестов для функциональных элементов..... 397
10.11. Модели элементарных триггерных схем 401
10.12. Методы построения кубических покрытий и тестов для структурно-функциональных моделей цифровых устройств 406
10.12.1. Принцип построения кубических покрытий одноступенчатых триггерных структур..... 409
10.12.2. Особенности построения кубических покрытий для двухступенчатых триггерных структур..... 411
10.12.3. Развитие π -алгоритма..... 414
10.13. Выводы..... 423
10.14. Литература..... 425

11. AD HOC МЕТОДЫ ГЕНЕРАЦИИ ТЕСТОВ ДЛЯ ЦИФРОВЫХ СИСТЕМ..... 429
11.1. Модификация метода генетических алгоритмов для синтеза тестов верификации цифровых систем..... 429
11.2. Функциональный К-алгоритм синтеза тестов..... 442
11.3. Совместный К-алгоритм синтеза тестов..... 447
11.4. Стратегия генерации тестов для цифровых систем..... 449
11.5. Анализ схемной структуры..... 451
11.6. Псевдокомбинационные модели цифровых схем..... 454
11.7. Процедуры генерации тестов для последовательностных схем 458
11.8. Выводы..... 468
11.9. Литература..... 471

ЗАКЛЮЧЕНИЕ..... 476

СПИСОК СОКРАЩЕНИЙ 477