МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

на правах рукопису

ТАМЕР АБДЕЛЬМАДЖІД САЛЕХ БАНІ-АМЕР

УДК 658: 512.011: 681.326: 519.713

# ХМАРНИЙ СЕРВІС-КОМП'ЮТИНГ ДЛЯ ТЕСТУВАННЯ І МОДЕЛЮВАННЯ SOC-КОМПОНЕНТІВ

05.13.05 - комп'ютерні системи та компоненти

ДИСЕРТАЦІЯ

на здобуття наукового ступеня кандидата технічних наук

Цей примірник дисертації ідентичний за змістом
з іншими примірниками, що подані до спеціалізованої
вченої ради Д 64.052.01

Учений секретар спеціалізованої
вченої ради Д 64.052.01 О. А. Винокурова

Науковий керівник:
Хаханов Володимир Іванович
доктор технічних наук, професор

Харків - 2017

# АНОТАЦІЯ

Тамер Абдельмаджід Салех Бані-Амер. Хмарний сервіс-комп'ютинг для тестування і моделювання SoC-компонентів. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук (доктора філософії) за спеціальністю 05.13.05 «комп'ютерні системи та компоненти». – Харківський національний університет радіоелектроніки, Міністерство освіти і науки України, Харків, 2017.

Дисертаційна робота пов'язана з хмарної реалізацією нової технології проектування і тестування цифрових систем на основі використання векторної (кубітної, квантової) форми опису функцій і структур, орієнтованої на імплементацію комбінаційних схем (reusable logic) в елементах пам'яті. Зазначена форма має істотні переваги перед традиційними таблицями істинності: векторна компактність опису функцій, паралелізм аналізу і синтезу цифрових пристроїв, дискретність переплутування станів, вбудована ремонтопридатність виробів, технологічна суперпозиційність одержуваних рішень, одночасність обчислення булеана, що скорочує час вирішення задачі покриття. Векторна форма опису функциональностей дає можливість істотно підвищити швидкодію синтезу, аналізу, верифікації, тестування, діагностування та ремонту обчислювальних пристроїв шляхом створення відповідних хмарних сервісів.

Науково-практична задача дослідження полягає у приведенні опису функцій і структур до єдиної одновимірної кубітно-векторної метрики в цілях технологічного вирішення задач синтезу й аналізу обчислювальних пристроїв шляхом виконання паралельних логічних операцій в рамках memory-driven computing.

Мета дисертаційного дослідження – істотне підвищення виходу придатної продукції і якості програмно-апаратних виробів за рахунок

створення інфраструктури хмарних сервісів моделювання, тестування і відновлення працездатності на основі використання векторних структур даних адресовних функціональних елементів і підвищення швидкодії кубітних методів синтезу та аналізу.

Сутність дослідження полягає у створенні векторних структур даних і кубітних методів синтезу, тестування і моделювання, інтегрованих в хмарну інфраструктуру сервісного обслуговування компонентів цифрових систем на кристалах з метою підвищення якості виробів і виходу придатної продукції за рахунок адресовних обчислювальних процесів і явищ. Основна інноваційна ідея запропонованої МАТ-моделі обчислень полягає у синтезі й аналізі векторних цифрових структур на основі адресовних елементів пам'яті, що виключають використання reusable or new logic.

Основні результати:

1. Запропоновано нову модель сталого розвитку кіберфізичного комп'ютингу та напрями її використання для прогнозування аттракторів в області ІТ-індустрії на основі просторово-часового аналізу технологічних процесів, які характеризуються трьома фазами розвитку і дають можливість прогнозувати майбутні тренди розвитку кіберкультури планети. Запропоновано кіберкультуру мікро-макро-космо-комп'ютингу, яка формулює, пояснює і прогнозує сучасні технології моніторингу та управління процесами і явищами в фізичному, віртуальному та космологічному просторі. Представлено вербальні і структурні визначення основних типів комп'ютингу, що базуються на сучасних трендах еволюційного розвитку кіберекосистеми планети. Сформульовано універсальну модель МАТ-комп'ютингу: <Memory, Address, Transactions>, яка використовує три компоненти для створення обчислювальної структури в технологічно прийнятному матеріальному середовищі.

2. Удосконалено кубітно-векторні моделі для опису структур і компонентів цифрових систем на базі адресного кодування вхідних сигналів,

які відрізняються від аналогів технологічністю і швидкодією побудованих на їх основі процедур тестування, логічного синтезу та аналізу. Удосконалено методи синтезу тестів і моделювання несправностей, які відрізняються паралельним виконанням логічних регістрових операцій над кубітними покриттями схемних компонентів. Розроблено метод і секвенсор безумовного синтезу тестів для функціональних логічних компонентів, який характеризується паралельним виконанням регістрових логічних операцій (shift, or, not, nxor) над кубітним вектором і його похідними, що дає можливість істотно зменшити час генерування вхідних наборів і тестування пристрою в режимі embedded online. Розроблено метод взяття похідних для генерації тестів функціональних компонентів, який характеризується паралельним виконанням регістрових логічних операцій (shift, or, not, nxor) над кубітним вектором, що дає можливість істотно зменшити час генерування вхідних наборів і тестування пристрою за рахунок апаратної надлишковості. Розроблено дедуктивний метод моделювання несправностей для функціональних компонентів, який характеризується паралельним виконанням регістрових логічних операцій (shift, or, not, nxor) над кубітним вектором і його похідними, що дає можливість істотно зменшити час верифікації та тестування цифрового пристрою в режимі embedded online. Запропоновано процесор кубітного моделювання цифрових пристроїв, імплементований в SoC або Cloud Service, для аналізу справної поведінки і несправностей на основі використання кубітних покриттів функціональних елементів, який відрізняється від відомих реалізацій застосуванням мінімального набору регістрових логічних операцій і високою швидкодією.

3. Запропоновано новий підхід до проектування цифрових пристроїв, який характеризується: 1) удосконаленим методом інтерпретативного паралельного моделювання компонентів цифрових систем на кристалах, який відрізняється застосуванням автоматної МАТ-моделі, що використовує тільки адресовні структури пам'яті і операції транзакції; 2) методами синтезу та

аналізу, що базуються на суперпозиції кубіт-векторних примітивів завдання всіх типів функциональностей, імплементованих в елементи пам'яті, що дає можливість істотно підвищити швидкодію засобів моделювання, тестування і верифікації, а також значно спростити процедури створення реальних і віртуальних комп'ютерних систем; 3) оригінальними структурами даних для моделювання і верифікації цифрових систем, які дають можливість суттєво спростити алгоритми реалізації та підвищити їх швидкодію за рахунок адресовних функціональних квантів і паралельності обробки компонентів; 4) реалізацією всіх обчислювальних структур і процесів на основі використання введеного квантового адресного автомата, який дає можливість безпосередньо використовувати інфраструктуру стандартів тестопридатного проектування для підвищення виходу придатної продукції за рахунок online ремонту функціональних примітивів.

Практична значущість нового підходу синтезу та аналізу цифрових систем полягає в наступному: 1) реалізація процесора тільки на основі використання елементів пам'яті робить його однорідним за структурою і типам функціональних примітивів, що доставляє очевидні технологічні зручності процесам проектування, виробництва і експлуатації, у тому числі верифікації, вбудованого тестування і діагностування, а головне – ремонту в режимі online за рахунок використання на кристалі універсальних адресовних spare-компонентів пам'яті; 2) моделювання в процесі верифікації проектованих обчислювачів на основі адресних моделей компонентів робить дану процедуру технологічно простою за рахунок регулярних структур даних і використання єдиної операції транзакції на елементах пам'яті, а також більш швидкодіючою за рахунок можливості паралельної квантоподібної обробки великих масивів однотипної пам'яті; 3) імплементація квантових only memory-based моделей опису цифрових компонентів і систем безпосередньо пов'язана зі збільшенням виходу придатної продукції, підвищенням надійності обчислювальних виробів, зниженням вартості проектування і виготовлення, а також

автономним відновленням працездатності в режимі remote & online, без участі людини.

4. Створено хмарну інфраструктуру сервісного обслуговування для online проектування, тестування і верифікації цифрових проектів, яка відрізняється візуалізацією цифрових схем і доступністю мікросервісів моделювання для зменшення періоду налагодження HDL-коду.

5. Здійснено тестову верифікацію компонентів хмарної інфраструктури сервісного обслуговування, а також методів тестування, моделювання, синтезу та аналізу на реальних прикладах цифрових схем і елементів. Використано мови програмування: SWIFT, C++, Verilog, Python 2.7 і платформи: Microsoft Windows, X Window і Macintosh OS X. Наукові результати дозволили підвищити швидкодію процедур аналізу структур даних, тестування та інтерпретативного адресного моделювання цифрових схем, на 15% зменшити час налагодження HDL-проектів в процесі проектування SoC.

Ключові слова: кубіт, кубітне покриття, космологічний комп'ютинг, пам'ять-адреса-транзакція, верифікація, діагностування, тестування, моделювання, цифрові системи на кристалах, квантові обчислення.

**Список публікацій здобувача**

в яких опубліковані основні наукові результати дисертації:

1. Хаханов В.И. Метрика для анализа Big Data / В.И. Хаханов, А.С. Мищенко, В.И. Обризан, Tami Bani Amer // Радиоэлектроника и информатика. – 2014. – № 2. – С. 26-29. (Входить до міжнародних наукометричних баз Index Copernicus, Google Scholar, OECSP, OAJI, Scholar Steer, SIS, Cyberleninka, CiteFactor, TIU Hannover, I2OR).

2. Хаханов В.И. Кубитные технологии анализа и диагностирования цифровых устройств / В.И. Хаханов, Т. Бани Амер, С.В. Чумаченко, Е.И. Литвинова // Электронное моделирование. – Том 37, № 3. – 2015. – С. 17-40.

(Входить до міжнародних наукометричних баз Cambridge Scientific Abstracts, Computer and Information Systems Abstracts, INIS Collection, Inspec, ВИНИТИ РАН).

3. Bani Amer T. Синтез Q-тестов по кубитному описанию функциональностей / T. Bani Amer, I. Iemelianov, M. Liubarskyi, V. Hahanov // Радиоэлектроника и информатика. – 2016. – № 2. – С. 38-47. (Входить до міжнародних наукометричних баз Index Copernicus, Google Scholar, OECSP, OAJI, Scholar Steer, SIS, Cyberleninka, CiteFactor, TIU Hannover, I2OR).

4. Хаханов В.И. Процессорные структуры для анализа Big Data / В.И. Хаханов, Е.И. Литвинова, С.В. Чумаченко, И. Емельянов, Т. Bani Amer // Радіоелектронні і комп'ютерні системи. – 2016.– № 6 (80).– С. 163-175.

5. Bani Amer T. Кубитная форма описания вычислительных структур / T. Bani Amer, С.В. Чумаченко, И.В. Емельянов // Радиоэлектроника и информатика.– 2016.– № 1.– С.47-52. (Входить до міжнародних наукометричних баз Index Copernicus, Google Scholar, OECSP, OAJI, Scholar Steer, SIS, Cyberleninka, CiteFactor, TIU Hannover, I2OR).

6. Хаханов В. Облачное управление физическими и кадровыми ресурсами / В. Хаханов, С. Чумаченко, Е. Литвинова, А. Мищенко, И. Емельянов, Т. Бани Амер // Australian Journal of Scientific Reseach.– № 1(5).– 2014.– С. 202-212.

7. Bani Amer T. Компьютинговые модели облачных сервисов / T. Bani Amer, В.И. Хаханов, И.В. Емельянов, М. Любарский // АСУ и приборы автоматики. – 2015. – Вип. 173. – С.48 (Входить до міжнародних наукометричних баз Google Scholar, Cyberleninka).

8. Bani Amer T. Кубитные модели описания цифровых устройств / T. Bani Amer, И.В. Хаханов, Е.И. Литвинова, И.В. Емельянов // АСУ и приборы автоматики. – 2016. – Вып. 174. – С. 24-41. (Входить до міжнародних наукометричних баз Google Scholar, Cyberleninka).

які засвідчують апробацію матеріалів дисертації:

9. Emelyanov I. Qubit Modeling Digital Systems / I. Emelyanov, I. Hahanova, T. Bani Amer // Proc. of IEEE East-West Design and Test Symposium. – Kiev, 26-29 September. – 2014. – P. 246-248. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

10. Hahanov I. Automaton MQT-model for virtual computer design / I. Hahanov, T. Bani Amer, I. Hahanova, S. Dementiev, A. Arefiev // Proceedings of 13th International Conference: The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2015. – Lvov, Ukraine. – P. 161-165. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

11. Hahanov V. MQT-model for Virtual Computer Design / V. Hahanov, T. Bani Amer, I. Hahanov // Proc. of Microtechnology and Thermal Problems in Electronics (Microtherm), 23-25 June 2015. – Poland. – P. 182-185.

12. Gerasimenko K. Method for Functional Testing Critical Control Systems / K. Gerasimenko, V. Hahanov, T. Bani Amer, A. Pryimak // Proceedings of IEEE East-West Design & Test Symposium 2015. – Batumi, Georgia. – P. 149-153. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

13. Hahanov I. QuaSim – Cloud Service for Digital Circuits Simulation / I. Hahanov, W. Gharibi, I. Iemelianov, T. Bani Amer // Proceedings of IEEE East-West Design & Test Symposium. – 2016. – Yerevan, Armenia. – P. 141-158. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

14. Soklakova T. Technological culture of Big Data / T. Soklakova, I. Iemelianov, T. Bani Amer, I. Hahanov // Матеріали XIII Міжнародної конференції TCSET-2016. – 23-26 лютого, 2016. – Львів–Славське. – С.549-554. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

15. Bani Amer T. Кибер-компьютинг – новый бренд IoT-рынка / T. Bani Amer, И. Емельянов // Материалы XX Международного молодежного форума «Радиоэлектроника и молодежь в XXI веке». – 2016. – Часть 5. – С.36-37.

16. Hahanov V. Qubit Test Synthesis of the Functionality / V. Hahanov, T. Bani

Amer, E. Litvinova, T. Soklakova, M. Liubarskyi, N. Shavlak, K. Dziuba // Proceedings of 14th International Conference: The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2017. – Lvov, Ukraine. – P. 161-165. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

які додатково відображають наукові результати дисертації:

17. Hahanov V. Cloud-Driven Traffic Control: Feasibiliti and Advanteges / V. Hahanov, S. Chumachenko, T. Bani Amer, I. Hahanov // Proc. of the 4th Mediterranean Conference on Embedded Computing (MECO). – 2015. – Budva, Montenegro. – P.17-20. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

18. Hahanov V. Smart Resources Control / V. Hahanov, T. Bani Amer, S. Chumachenko, E. Litvinova // The 4th International Academic Congress “Science and Education in the Modern World”. – Vol. II. – New Zealand, Auckland. – 2015. – P. 1021-1034.

# ABSTRACT

Tamer Abdelmajeed Saleh Bani-Amer. Cloud service-computing for testing and modeling SoC components. – Qualification scientific work. Manuscript. PhD thesis (candidate degree of technical sciences) in speciality 05.13.05 – Computer Systems and Components. – Kharkiv National University of Radio Electronics, Ministry of Education and Science of Ukraine, Kharkiv, 2017.

The PhD thesis is devoted to the cloud implementation of a new technology for designing and testing digital systems based on the use of a vector (qubit, quantum) form of the description of functions and structures focused on the implementation of combinational circuits (reusable logic) in memory elements. This form has significant advantages over traditional truth tables: the vector compactness of the function description, the parallelism of analysis and synthesis of digital devices, the discreteness of state entanglement, the built-in maintainability of products, the technological superposition the obtained solutions, the simultaneous calculating the power set that reduces the time of solving the coverage problem. The vector form of describing the functionality makes it possible to significantly improve the speed of synthesis, analysis, verification, testing, diagnosis and repair of computing devices by creating appropriate cloud services.

The scientific and practical problem of the investigation is to transform the description of functions and structures to a single one-dimensional qubit-vector metric in order to solve the problems of synthesis and analysis of computing devices by executing parallel logical operations within the framework of memory-driven computing.

The goal of the investigation is to significantly improve the yield and quality of software and hardware products through the creation of an infrastructure of cloud services for simulation, testing and repair based on the use of vector data

structures of addressable functional elements and increasing the speed of the qubit synthesis and analysis methods.

The essence of the research is the creation of vector data structures and qubit methods for the synthesis, testing and modeling components of digital systems-on-chips, integrated into the cloud IP-infrastructure, in order to improve the quality of products and yield through the addressability of all computing processes and phenomena. The main innovative idea of the proposed MAT-model of computations is the synthesis and analysis of vector digital structures based on addressable memory elements, which exclude the use of reusable or new logic.

The main results are the following:

1. A new model of sustainable development of cyber-physical computing and the directions of its applying for the prediction of attractors in the IT industry based on the space-time analysis of technological processes, characterized by three phases of development and enable to predict future trends in the development of cyberculture of the planet, are proposed. A cyberculture of micro-macro-cosmo-computing is proposed; it defines, explains and predicts modern technologies for monitoring and controlling processes and phenomena in the physical, virtual and cosmological space. Verbal and structural definitions of the main types of computing are presented, based on modern trends in the evolutionary development of the planet's cyber ecosystem. A general model of MAT-computing <Memory, Address, Transactions> that uses three components to create a computational structure in a technologically acceptable material environment, is formulated.

2. Qubit-vector models for describing the structures and components of digital systems based on address coding of input signals are improved; they differ from analogues by the technology effectiveness and high speed of logic synthesis, testing and analysis procedures built on their basis. Methods of test synthesis and fault simulation, which differ in parallel execution of logical register operations with the qubit coverage of circuit components, are improved. A method and sequencer of unconditional test synthesis of functional logic components are

developed; they are characterized by parallel execution of register logical operations (shift, or, not, nxor) with the qubit vector and its derivatives that allows significantly reducing the time of input set generation and embedded online testing a device. A method for taking derivatives to generate test of functional components is proposed; it is characterized by parallel execution of register logical operations (shift, or, not, nxor) with the qubit vector that allows significantly reducing the time of input set generation and testing the device due to the hardware redundancy. A deductive method for simulating faults of functional components is developed; it is characterized by the parallel execution of register logical operations (shift, or, not, nxor) with the qubit vector and its derivatives that allows significantly reducing the time of embedded online verification and testing of a digital device. A processor of qubit simulation of digital-devices, implemented in SoC or Cloud Service, is proposed; it is designed for the analysis of fault-free behavior and faults based on the use of qubit coverage of functional elements and differs from the known implementations by applying a minimum set of register logical operations and high speed.

3. A new approach to the design of digital devices is proposed; it is characterized by the following: 1) an improved method for interpretative parallel simulation of the components of a digital system-on-chips, which differs by leveraging an automaton MAT model, and also only addressable memory structures and transaction operations; 2) methods for synthesis and analysis, based on the superposition of qubit-vector primitives defining all types of functionality and implementing in memory elements, which makes it possible to significantly improve the speed of the simulation, testing and verification, and also significantly simplify the procedures for creating real and virtual computer systems; 3) original data structures for simulation and verification of digital systems, which make it possible to significantly simplify the implementation algorithms and increase their performance through the addressability of functional quanta and parallel processing components; 4) the implementation of all computing structures and

processes based on the use of the introduced quantum address automaton, which makes it possible to directly use the infrastructure of testable design standards to improve the yield through online repair of functional primitives.

The practical significance of the new approach to the synthesis and analysis of digital systems is as follows: 1) the implementation of the processor based only on the use of memory elements that makes it homogeneous in structure and types of functional primitives; this provides obvious technological convenience to the design, manufacturing and operational processes, including verification, embedded test and diagnosis, and most importantly – online repair by using universal addressable spare components of the memory on the chip; 2) simulation in the process of verification of designing calculators based on the address models of components that makes this procedure technologically simple due to leverage regular data structures and a single transaction operation on memory elements, and also faster, due to the opportunity of parallel quantum-like processing large arrays of memory of the same type; 3) the implementation of quantum only memory-based models for the description of digital components and systems is directly related to an increase in the yield and reliability of computing products, reduction in the cost of design and manufacturing, and also autonomous human-free repair in remote & online mode.

4. A cloud service infrastructure has been created for online design, test and verification of digital projects, which differs by the visualization of digital circuits and the availability of simulation micro-services to reduce the debugging time of HDL-code.

5. Test verification of the components of the cloud service infrastructure, and also methods for testing, simulation, synthesis and analysis of real examples of digital circuits and elements are performed. The following programming languages are used: SWIFT, C++, Verilog, Python 2.7 и платформы: Microsoft Windows, X Window и Macintosh OS X. The scientific results allowed to increase the speed of procedures for analyzing data structures, testing and interpretative address

simulation of digital circuits, and also to reduce the time of debugging HDL-projects in the process of designing SoC by 15%.

Key words: qubit, qubit coverage, cosmological computing, memory-address-transaction, verification, diagnosis, testing, simulation, digital system-on-chip, quantum computing.

# АННОТАЦИЯ

Тамер Абдельмаджид Салех Бани-Амер. Облачный сервис-компьютинг для тестирования и моделирования SoC-компонентов. – Квалификационная научная работа на правах рукописи.

Диссертация на соискание ученой степени кандидата технических наук (доктора философии) по специальности 05.13.05 «компьютерные системы и компоненты». - Харьковский национальный университет радиоэлектроники, Министерство образования и науки Украины, Харьков, 2017.

Диссертационная работа связана с облачной реализацией новой технологии проектирования и тестирования цифровых систем на основе использования векторной (кубитной, квантовой) формы описания функций и структур, ориентированной на имплементацию комбинационных схем (reusable logic) в элементах памяти. Указанная форма имеет существенные преимущества перед традиционными таблицами истинности: векторная компактность описания функций, параллелизм анализа и синтеза цифровых устройств, дискретность перепутывания состояний, встроенная ремонтопригодность изделий, технологическая суперпозиционность получаемых решений, одновременность вычисления булеана, сокращающего время решения задачи покрытия. Векторная форма описания функциональностей дает возможность существенно повысить быстродействие синтеза, анализа, верификации, тестирования, диагностирования и ремонта вычислительных устройств путем создания соответствующих облачных сервисов.

Научно-практическая задача исследования заключается в приведении описания функций и структур к единой одномерной кубитно-векторной метрике в целях технологичного решения задач синтеза и анализа

вычислительных устройств путем выполнения параллельных логических операций в рамках memory-driven computing.

Цель исследования – существенное повышение выхода годной продукции и качества программно-аппаратных изделий за счет создания инфраструктуры облачных сервисов моделирования, тестирования и восстановления работоспособности на основе использования векторных структур данных адресуемых функциональных элементов и повышения быстродействия кубитных методов синтеза и анализа.

Сущность исследования заключается в создании векторных структур данных и кубитных методов синтеза, тестирования и моделирования, интегрированных в облачную инфраструктуру сервисного обслуживания компонентов цифровых систем на кристаллах в целях повышения качества изделий и выхода годной продукции за счет адресуемости всех вычислительных процессов и явлений. Основная инновационная идея предложенной МАТ-модели вычислений заключается в синтезе и анализе векторных цифровых структур на основе адресуемых элементов памяти, исключающих использование reusable or new logic.

Основные результаты:

1. Предложена новая модель устойчивого развития киберфизичного компьютинга и направления ее использования для прогнозирования аттракторов в области IT-индустрии на основе пространственно-временного анализа технологических процессов, которые характеризуются тремя фазами развития и дают возможность прогнозировать будущие тренды развития киберкультуры планеты. Предложена киберкультура микро-макро-космо-компьютинга, которая определяет, объясняет и прогнозирует современные технологии мониторинга и управления процессами и явлениями в физическом, виртуальном и космологическом пространстве. Представлены вербальные и структурные определения основных типов компьютинга, основанные на современных трендах эволюционного развития

киберэкосистемы планеты. Сформулирована универсальная модель МАТ-компьютинга <Memory, Address, Transactions>, которая использует три компонента для создания вычислительной структуры в технологически приемлемой материальной среде.

2. Усовершенствованы кубитно-векторные модели описания структур и компонентов цифровых систем на основе адресного кодирования входных сигналов, которые отличаются от аналогов технологичностью и быстродействием построенных на их основе процедур тестирования, логического синтеза и анализа. Усовершенствованы методы синтеза тестов и моделирования неисправностей, которые отличаются параллельным выполнением логических регистровых операций над кубитными покрытиями схемных компонентов. Разработан метод и секвенсор безусловного синтеза тестов для функциональных логических компонентов, который характеризуется параллельным выполнением регистровых логических операций (shift, or, not, nxor) над кубитным вектором и его производными, что позволяет существенно уменьшить время генерирования входных наборов и тестирования устройства в режиме embedded online. Предложен метод взятия производных для генерации тестов функциональных компонентов, который характеризуется параллельным выполнением регистровых логических операций (shift, or, not, nxor) над кубитным вектором, дает возможность существенно уменьшить время генерирования входных наборов и тестирования устройства за счет аппаратной избыточности. Разработан дедуктивный метод моделирования неисправностей для функциональных компонентов, который характеризуется параллельным выполнением регистровых логических операций (shift, or, not, nxor) над кубитним вектором и его производными, что позволяет существенно уменьшить время верификации и тестирования цифрового устройства в режиме embedded online. Предложен процессор кубитного моделирования цифровых устройств, имплементированный в SoC или Cloud

Service, для анализа исправного поведения и неисправностей на основе использования кубитних покрытий функциональных элементов, который отличается от известных реализаций применением минимального набора регистровых логических операций и высоким быстродействием.

3. Предложен новый подход к проектированию цифровых устройств, который характеризуется: 1) усовершенствованным методом интерпретативного параллельного моделирования компонентов цифровых систем на кристаллах, который отличается применением автоматной МАТ-модели, использующей только адресуемые структуры памяти и операции транзакции; 2) методами синтеза и анализа, основанными на суперпозиции кубит-векторных примитивов задания всех типов функциональностей, имплементируемых в элементы памяти, что дает возможность существенно повысить быстродействие средств моделирования, тестирования и верификации, а также значительно упростить процедуры создания реальных и виртуальных компьютерных систем; 3) оригинальными структурами данных для моделирования и верификации цифровых систем, которые дают возможность существенно упростить алгоритмы реализации и повысить их быстродействие за счет адресуемости функциональных квантов и параллельности обработки компонентов; 4) реализацией всех вычислительных структур и процессов на основе использования введенного квантового адресного автомата, который дает возможность непосредственно использовать инфраструктуру стандартов тестопригодного проектирования для повышения выхода годной продукции за счет online ремонта функциональных примитивов.

Практическая значимость нового подхода синтеза и анализа цифровых систем заключается в следующем: 1) Реализация процессора только на основе использования элементов памяти делает его однородным по структуре и типам функциональных примитивов, что доставляет очевидные технологические удобства процессам проектирования, производства и

эксплуатации, включая верификацию, встроенные тестирование и диагностирование, а главное – ремонт в режиме online за счет использования на кристалле универсальных адресуемых spare-компонентов памяти. 2) Моделирование в процессе верификации проектируемых вычислителей на основе адресных моделей компонентов делает данную процедуру технологически простой за счет регулярных структур данных и использования единственной операции транзакции на элементах памяти, а также более быстродействующей, за счет возможности параллельной квантоподобной обработки больших массивов однотипной памяти. 3) Имплементация квантовых only memory-based моделей описания цифровых компонентов и систем непосредственно связана с увеличением выхода годной продукции, повышением надежности вычислительных изделий, снижением стоимости проектирования и изготовления, а также автономным восстановлением работоспособности в режиме remote & online, без участия человека.

4. Создана облачная инфраструктура сервисного обслуживания для online проектирования, тестирования и верификации цифровых проектов, которая отличается визуализацией цифровых схем и доступностью микросервисов моделирования для уменьшения периода отладки HDL-кода.

5. Выполнена тестовая верификация компонентов облачной инфраструктуры сервисного обслуживания, а также методов тестирования, моделирования, синтеза и анализа на реальных примерах цифровых схем и элементов. Использованы языки программирования: SWIFT, C++, Verilog, Python 2.7 и платформы: Microsoft Windows, X Window и Macintosh OS X. Научные результаты позволили повысить быстродействие процедур анализа структур данных, тестирования и интерпретативного адресного моделирования цифровых схем, на 15% уменьшить время отладки HDL-проектов в процессе проектирования SoC.

Ключевые слова: кубит, кубитное покрытие, космологический компьютинг, память-адрес-транзакция, верификация, диагностирование, тестирование, моделирование, цифровые системы на кристаллах, квантовые вычисления.

ЗМІСТ

# ВСТУП

**Актуальність дослідження**. Мотивація пов'язана зі створенням загальної картини розвитку комп'ютингу в просторі та часі для прогнозування інноваційних трендів в даній області. Кожні 20 років змінюється технологічний уклад розвитку людства в планетарному масштабі. Тут комп'ютинг відіграє визначальну роль, масштабуючись в останні 60 років у три просторових рівня: Single, Network and Global. Перший рівень розвивається за сценарієм: Personal Computing, Mobile Computing and Interface Computing (embedded braine-computer interface – убудований мозкоподібний комп’ютерний інтерфейс). Другий рівень визначається компонентами: Network Computing, Cloud Computing and Cyber Physical Computing. Третій рівень містить: Internet Computing, Internet of Things Computing and Internet of Nature Computing (Cyber Nature Computing). Відповідно до запропонованого сценарію, протягом поточних 20 років (2010-2030) вченим необхідно вирішити такі проблеми: 1) Створення вбудованого інтерфейсу для безпосереднього спілкування людського мозку з комп'ютером. 2) Розробка кіберфізичних систем моніторингу та управління всіма соціально-технологічними процесами і явищами, у тому числі транспортом і соціальними групами. 3) Створення глобального мозку людства в рамках культури Cyber Nature Computing для точного моніторингу та оптимального управління всіма природними і штучними процесами та явищами.

Пасивне споглядання дійсності в кіберфізичному просторі еволюційно змінюється на активне актюаторне цифрове управління процесами і явищами шляхом впровадження точного сенсорного моніторингу без участі людини, як найбільш ненадійної ланки в будь-який технічній або соціальній системі. Технологічна модель такого управління покривається автоматним поняттям кіберфізичного або хмарного комп'ютингу. Останній визначається як процес

досягнення поставленої мети шляхом використання механізмів управління та виконання в циклічно замкнутій системі з заданими входами і виходами, сигналами моніторингу та актюації. Інакше, комп'ютинг  це галузь знань, спрямована на дослідження, проектування і застосування інтелектуальних програмно-апаратних систем, мереж і сервісів для моніторингу та управління кіберфізичними процесами і явищами. Карта комп'ютингу покриває: нано-мікро-макро-електроніку, радіотелекомунікації, комп'ютерну, програмну, системну, виробничу, транспортну та соціальну інженерію, штучний інтелект, кібермониторинг і управління, кіберфізичні інтернет технології. Для ринку електронних технологій, що формує індекс NASDAQ, найбільш популярними аттракторами комп'ютингу є: Big Data, Internet of Things, Mobile to Computing, Cyber Security, Brain-Computer Interface, Cloud Service Computing, Internet-Driven Design and Test. Інтенсивно формується кіберфізичний безперервний простір на базі вбудованих мікросистем: wearable's, smart car, smart home, smart city, 3D printing, quantum computing, robotics networks, big data analytics, the maker movement, drones. Фактично до 2020 року людство матиме розумну взаємодію віртуального і реального світів: 50 мільярдів розумних пристроїв і 6 мільярдів смартфонів, 250 мільйонів машин без водіїв, 10,2 мільйона розумного одягу, 20 мільярдів RFID-міток для цифрової ідентифікації об'єктів і процесів, 15 трильйонів доларів, як індекс IoT-капіталізації. Хмарний сервіс буде найбільш цікавою моделлю IT-бізнесу в найближчі 10 років. До кінця 2017 року дві третини від 2000 провідних світових компаній трансформують свою діяльність під цифрові процеси моніторингу та управління. Протягом двох років будуть інстальовані 22 мільярди Internet of Things пристроїв, які матимуть доступ до більш, ніж 200 000 нових Internet of Things сервісів, які забезпечать USA-підприємствам понад 60 мільярдів доларів щорічної економії. До 2018 року 50 відсотків усіх підприємств матимуть хмарні платформи для поширення інновацій та цифрові двері для клієнтів і зовнішніх пропозицій. Звідси існує

простий висновок: інноваційні ідеї в науці, техніці та освіті повинні імплементуватися переважно в хмарні сервіси, які потенційно стають доступними для всіх людей на планеті. Якість таких сервісів забезпечує розробнику високий рівень продажів і матеріальний добробут. Актуальність технології Cloud-Driven Service Computing була неодноразово підкреслена в виступах провідних вчених планети на IEEE East-West Design and Test Symposium 2016, Yerevan, Armenia. Доктор Yervant Zorian, Synopsys: «Сервіси для проектування, тестування, діагностування та ремонту будь-яких SoC і комп'ютерних виробів повинні розміщуватися на хмарах для масового доступу до них користувачів і покриття технологічної культурою всього науково-технічного простору планети».

Тема дисертаційної роботи пов'язана з хмарної реалізацією нової технології проектування і тестування цифрових систем на основі використання векторної (кубітной, квантової) форми опису функцій і структур, орієнтованої на імплементацію комбінаційних схем (reusable logic) в елементах пам'яті. Зазначена форма має істотні переваги перед традиційними таблицями істинності: векторна компактність опису функцій, паралелізм аналізу і синтезу цифрових пристроїв, дискретність переплутування станів, вбудована ремонтопридатність виробів, технологічна суперпозиційність одержуваних рішень, одночасність обчислення булеана, що скорочує час вирішення задачі покриття. Практично векторна форма опису функциональностей дає можливість істотно підвищити швидкодію синтезу, аналізу, верифікації, тестування, діагностування та ремонту обчислювальних пристроїв шляхом створення відповідних хмарних сервісів.

Проблеми проектування, тестування, діагностування та відновлення працездатності цифрових систем розглядаються в публікаціях вчених: Y. Zorian, M. Abramovichi, J. Bergeron, Z. Navabi, A. Jerraya, D.B. Armstrong, P. Prinetto, J. Abraham, H. Fujiwara , T. Nishida, X. Wang, А.Петренко, Р. Убар, A. Ivanov, О. Романкевич, Д. Сперанський, А. Матросова, П. Пархоменко,

J. P. Roth, В. Мелікян, С. Шукурян, В. Тарасенко, М. Коровай, О. Палагін, В. Опанасенко, В. Харченко, Л. Дербуновіч, В. Ярмолік, Р. Шейнаускас, Н. Євтушенко, Р. Базилевич, Г. Кривуля.

Зв'язок роботи з науковими програмами, держбюджетними темами. Розробка теми дисертації здійснювалася відповідно до планів держбюджетних НДР і господарських договорів, виконуваних на кафедрі АПОТ Харківського національного університету радіоелектроніки в період з 2014 року, в тому числі: 1) Договір про дружбу та співробітництво між ХНУРЕ та корпорацією «Aldec Inc.» (USA ) №04 від 01.11.2014; 2) Держбюджетна НДР «Мультипроцесорна система пошуку, розпізнавання та прийняття рішень для інформаційної комп'ютерної екосистеми», д/б № 269 (2011-2013), №ДР 0111U002956; 3) Фундаментальна НДР «Персональний віртуальний кіберкомп'ютер та інфраструктура аналізу кіберпростору», №258 (2012-2014). 4) SEIDA BAITSE "Baltic Academic IT Security Exchange", Blekinge Institute of Technology, Sweden. 5) Curricula Development for New Specialization: Master of Engineering in Microsystems Design 530785-TEMPUS-1-2012-1-PL-TEMPUS-JPCR MastMST (2012-2016). 6) State grant from Ministry of Education and Science of Ukraine "Cyber Physical System  Smart Cloud Traffic Control" № 0115U-000712 (2015-2017).

Автор дисертаційної роботи брав участь у виконанні зазначених договорів і програм як розробник і програміст інфраструктури верифікації цифрових систем при створенні векторних моделей і методів тестування, діагностування та вбудованого ремонту обчислювальних пристроїв. Автор також брав участь у C++ кодуванні програмних модулів системи верифікації і моделювання на основі IEEE стандартів, інтегрованих із сервісами компанії Aldec.

Науково-практична задача дослідження: приведення опису функцій і структур до єдиної одновимірної кубітно-векторної метрики в цілях технологічного вирішення задач синтезу й аналізу обчислювальних

пристроїв шляхом виконання паралельних логічних операцій в рамках memory-driven computing.

Сутність запропонованого науково-технологічного дослідження полягає у створенні векторних структур даних і кубітних методів синтезу, тестування і моделювання, інтегрованих в хмарну інфраструктуру сервісного обслуговування компонентів цифрових систем на кристалах з метою підвищення якості виробів і виходу придатної продукції за рахунок адресовних обчислювальних процесів і явищ. Основна інноваційна ідея запропонованої МАТ-моделі обчислень полягає в синтезі й аналізі векторних цифрових структур на основі адресовних елементів пам'яті, що виключають використання reusable or new logic. Важко створити двовимірний регістр, що відповідає матриці суміжності або таблиці істинності, тому приведення опису функції і структури до єдиного одновимірного формату означає: технологічно вирішувати всі задачі синтезу та аналізу для функциональностей і графів у кубітно-векторній метриці, що створює memory-driven computing на основі виконання паралельних логічних операцій, що представлено на рисунку.

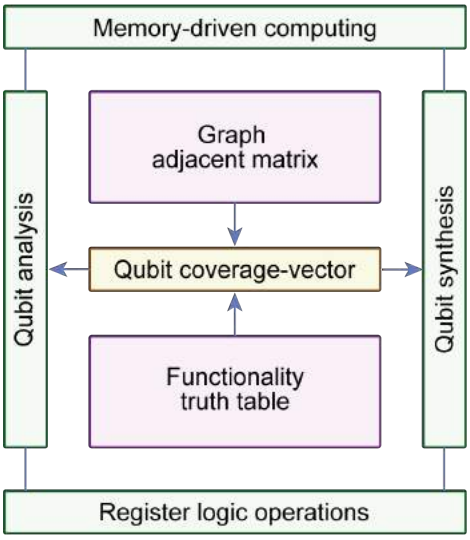



Рисунок. Memory-driven комп’ютинг на основі метрики кубітного покриття

Мета дослідження – істотне підвищення виходу придатної продукції і якості програмно-апаратних виробів за рахунок створення інфраструктури хмарних сервісів моделювання, тестування і відновлення працездатності на

основі використання векторних структур даних адресовних функціональних елементів і підвищення швидкодії кубітних методів синтезу та аналізу.

Задачі дослідження:

1. Розробити модель та напрямки сталого розвитку кіберфізичного комп'ютингу для прогнозування аттракторів в області IT-індустрії на основі просторово-часового аналізу технологічних процесів.

2. Удосконалити векторні моделі кубітного представлення структур і компонентів цифрових систем на основі адресного кодування вхідних сигналів для підвищення технологічності та швидкодії моделювання.

3. Розробити методи синтезу та аналізу векторних описів цифрових схем на основі використання кубітних покриттів для вирішення задач тестування і верифікації.

4. Розробити хмарну інфраструктуру сервісного обслуговування для online проектування і верифікації цифрових проектів з метою зменшення періоду налагодження (time-to-market) HDL-коду.

5. Виконати тестову верифікацію компонентів хмарної інфраструктури сервісного обслуговування, а також методів моделювання, синтезу та аналізу на реальних прикладах цифрових схем і компонентів. Використовувати мови програмування: C++, Verilog, Python 2.7 и платформы: Microsoft Windows, X Window и Macintosh OS X.

Об'єкт дослідження – процеси і явища паралельної обробки комбінаційних і послідовностних елементів для синтезу, аналізу, тестування, діагностування та ремонту цифрових виробів на основі використання векторних структур даних.

Предмет дослідження – кубітно-векторні моделі опису комбінаційних і послідовностних схем, методи та інфраструктури для синтезу, аналізу, тестування і ремонту цифрових систем.

Методи дослідження – архітектури комп'ютерів, булева алгебра, теорія множин, теорія графів, теорія цифрових автоматів, квантово-кубітні методи

обчислень і структури даних – для побудови моделей цифрових пристроїв; векторно-логічний аналіз, теорія алгоритмів, методи, засоби, мови проектування і моделювання цифрових систем – для синтезу та аналізу; методи і критерії якості створення обчислювальних проектів – для оцінювання тестопридатності цифрових виробів; засоби синтезу схем і аналізу кубітних покриттів – для верифікації програмно-апаратної інфраструктури хмарних сервісів.

Наукова новизна:

1. Вперше запропоновано модель та напрями сталого розвитку кіберфізичного комп'ютингу для прогнозування аттракторів в області IT-індустрії, що базуються на просторово-часовому аналізі технологічних процесів, які характеризуються трьома фазами розвитку і дають можливість прогнозувати майбутні тренди розвитку кіберкультури.

2. Удосконалено векторні моделі кубітного представлення структур і компонентів цифрових систем, що відрізняються адресним кодуванням вхідних сигналів і дозволяють істотно підвищити технологічність та швидкодію побудованих на їх основі процедур синтезу й аналізу.

3. Удосконалено метод інтерпретативного паралельного моделювання компонентів цифрових систем на кристалах, який відрізняється застосуванням автоматної МАТ-моделі, що використовує тільки адресовні структури пам'яті та операції транзакції. Це дає можливість істотно підвищити швидкодію засобів моделювання, тестування і верифікації.

4. Удосконалено методи синтезу тестів і моделювання несправностей, які відрізняються паралельним виконанням логічних регістрових операцій над кубітними покриттями схемних компонентів, що дає можливість істотно зменшити час генерування вхідних наборів і тестування пристрою.

Практичне значення отриманих результатів:

1. Створено прототип хмарної інфраструктури сервісного обслуговування для online проектування і верифікації цифрових проектів, яка

відрізняється візуалізацією цифрових схем і доступністю мікросервісів моделювання для зменшення періоду налагодження HDL-коду.

2. Виконано тестову верифікацію хмарної інфраструктури сервісного обслуговування, а також методів тестування, моделювання, синтезу та аналізу на реальних прикладах цифрових схем і компонентів. Використано мови програмування: SWIFT, C++, Verilog, Python 2.7 і платформи: Microsoft Windows, X Window і Macintosh OS X. Доказово представлена спроможність отриманих результатів щодо підвищення швидкодії процедур синтезу структур даних та інтерпретативного адресного моделювання цифрових схем, що дає можливість на 15% зменшити час налагодження HDL-проектів в процесі проектування SoC.

Отримані в процесі виконання досліджень наукові висновки і результати є достовірними, що підтверджується певною кількістю проведених експериментів, тестуванням і моделюванням реальних функціональних модулів з відкритих бібліотек компаній і конференцій. Практична значущість наукових досліджень підтверджується інтеграцією розробленої технології аналізу і верифікації з сервісами проектування компанії Aldec. Результати дисертації в складі моделей, методів та інфраструктури впроваджено в навчальний процес Харківського національного університету радіоелектроніки (довідка про впровадження від 06.03.2017); в науково-дослідну і виробничу діяльність компанії Aldec, USA (довідка про впровадження від 16.12.2016).

**Особистий внесок здобувача.** Всі наукові і практичні результати отримані автором особисто. У роботах, опублікованих зі співавторами, здобувачеві належать: [123] – метрика аналізу Big Data, структури великих даних для їх паралельного аналізу на основі мультіматричного процесора; [124] – векторно-кубітні моделі опису функціональних цифрових схем; [125] – метод синтезу Q-тестів по кубітному покриттю функціональностей; [126] – процесорні структури для паралельного аналізу Q-покриттів з метою

формування оптимального рішення; [127] – кубітні моделі опису графових структур для паралельного аналізу та синтезу обчислювальних архітектур; [128] – хмарні структури даних і модель кіберфізичної системи для вирішення проблем управління ресурсами; [129] – хмарний сервіс квантово-векторного моделювання цифрових пристроїв на основі кубітного опису функцій примітивів; [130] – моделі сталого розвитку кіберфізичних компьютингових систем для імплементації програмно-апаратних мікросервісів проектування цифрових систем на кристалах; [131] – структури даних для опису функціональності у вигляді кубітних векторів; [132] – структури даних і алгоритм моделювання на основі використання memory-qubit-transaction моделі обчислювальних процесів; [133] – MQT модель віртуального хмарного процесора для інтерпретативного моделювання цифрових систем; [134] – структури даних для функціонального тестування та діагностування критичних систем управління; [135] – хмарна структура моніторингу та управління транспортом на основі віртуальної інфраструктури забезпечення дорожнього руху; [136] – інфраструктура розумного управління ресурсами; [137] – модель кіберкомпьютингу для хмарної реалізації мікросервісів моделювання і тестування цифрових пристроїв; [138] – структури даних для хмарного моніторингу та управління освітніми сервісами; [139] – опис кіберкомпьютнгу як нового бренду IoT-ринку; [140] – моделі, метод і алгоритм кубітного моделювання комбінаційних і послідовностних схем.

Апробація результатів дисертації. Результати роботи були представлені та обговорені на наступних конференціях: 1-3) IEEE East-West Design and Test Symposium 2014 (Ukraine), 2015 (Georgia), 2016 (Armenia); 4) XX Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті» 2016 (Україна); 5) the 11-th IEEE International Conference TCSET 2016 (Slavsk, Ukraine); 6) Conference of Microtechnology and Thermal Problems in Electronics (Microtherm), 2015 (Poland); 7) IEEE 4th Mediterranean

Conference on Embedded Computing, 2015 (Budva, Montenegro); 8) the 4th International Academic Congress “Science and Education in the Modern World”, 2015 (New Zealand, Auckland); 9-10) 13th International Conference: The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2015, 2017 (Lviv, Ukraine).

Публікації. Результати дисертаційної роботи відображені у 18 друкованих працях. Серед них 7 статей, опублікованих у наукових журналах, які входять до «Переліків наукових фахових видань України» (з них 6 статей входять до міжнародних наукометричних баз), 1 стаття в міжнародному науковому журналі за кордоном, а також 10 міжнародних наукових конференціях (з них 5 за кордоном та 7 входять до наукометричної бази Scopus).

Структура дисертації представлена 192 сторінками (з них 150 сторінок основного тексту) і містить: 4 розділи, 45 рисунків, список джерел із 140 найменувань (на 16 с.), 2 додатки (на 24 с.).

# РОЗДІЛ 1

# MEMORY-DRIVEN АРХІТЕКТУРИ ТА МЕТОДИ ЇХ АНАЛІЗУ

Представлена нова парадигма обчислень, орієнтована на наближення даних до місць їх обробки. Описано архітектури обчислювальних платформ і реконфігурованого комп'ютингу, алгоритм проектування FPGA, види несправностей FPGA і методи їх виявлення, методи забезпечення відмовостійкості та відновлення працездатності [1-74].

Мета – аналітичний огляд архітектур обчислювальних платформ і реконфігурованого комп'ютингу, що використовують memory-driven проектування процесорних компонентів, спрямованих на підвищення швидкодії програмних і апаратних засобів аналізу цифрових пристроїв, а також істотне збільшення виходу придатної продукції і зменшення часу її виходу на ринок.

Завдання розділу: 1) Архітектури обчислювальних платформ. 2) Архітектури реконфігурованого комп'ютингу. 3) Реалізація програмованої логіки на FPGA. 4) Алгоритм проектування FPGA. 5) Види несправностей FPGA і методи їх виявлення. 6) Методи забезпечення відмовостійкості та відновлення працездатності.

Джерела дослідження: архітектури обчислювальних платформ [1-26]; архітектури реконфігурованого комп'ютингу [27-37]; реалізація програмовної логіки на FPGA [38-68]; алгоритм проектування FPGA [52]; види несправностей FPGA і методи їх виявлення [54-68]; методи забезпечення відмовостійкості та відновлення працездатності [69-74].

## 1.1 Архітектури обчислювальних платформ

Згідно з прогнозом компанії Gartner Inc. кількість підключених до мережі Internet пристроїв IoT до 2020 року досягне 20,8 млрд [1]. Обсяг даних, що формуються цими пристроями (аудіо, відео, електронна пошта, GPS, ДНК) в даний час неможливо обробляти наявними програмними і апаратними засобами. Необхідно розробляти нові підходи до зберігання інформації, організації обчислень, безпеки і передачі даних.

1.1.1 Обчислення в пам'яті. Традиційна обробка даних базується на використанні трьох компонентів: процесора, що виконує обчислення, пам'яті для зберігання даних і команд, пристроїв передачі інформації між зазначеними компонентами. Вузьким місцем архітектури фон Неймана є спільне використання процесором і пам'яттю каналу передачі програм і даних, продуктивність якого обмежує швидкість обчислень [2, 3]. Для вирішення зазначеної проблеми необхідно розмістити дані максимально близько до місця їх обробки, щоб уникнути нераціональної передачі інформації. Для цієї мети можна використовувати такі обчислювальні платформи:

1) обчислення в пам'яті (in-memory computing);

2) процесор в пам'яті (processor-in-memory);

3) обчислення, керовані пам'яттю (memory-driven computing).

Обчислення в пам'яті базуються на використанні проміжного програмного забезпечення, яке дозволяє зберігати дані в оперативній пам'яті комп'ютерного кластера і обробляти їх паралельно [2-5]. Різновидом обчислень в пам'яті є обчислювальна платформа, побудована на принципі зберігання опису функціональності в масивах пам'яті у вигляді функціональних генераторів. При цьому обчислення функції замінюється отриманням значень з таблиць.

Перехід від обчислень з використанням традиційних сховищ даних на жорстких дисках до обчислень в пам'яті має на увазі зменшення проміжних шарів на шляху від вихідних даних до результатів їх обробки. Так, у першому випадку обробка інформації виконується за наступним алгоритмом: вихідні дані зберігаються в пам'яті; частина даних витягується і поміщається в масиви зберігання проміжних даних з метою попередньої обробки; виконання обчислень за заданим алгоритмом; збереження результатів обчислень в проміжних масивах; обробка та візуалізація результатів. У другому випадку необроблені дані розміщуються у сховищі, де і відбуваються обчислення; додатки не можуть звертатися до проміжних результатsd обробки, використовуються тільки остаточні результати обчислень, які виконуються в реальному часі. У цьому випадку шар проміжних даних стає непотрібним. Більш того, обчислення в пам'яті дозволяють обробляти часто оновлювані дані. Так наприклад, додатки з обробки транзакцій можуть безпосередньо отримувати і передавати дані в сховище.

1.1.2 PIM-системи [6-9]. Поділ обчислень і зберігання інформації в архітектурах фон Неймана в даний час залишається актуальним з кількох причин [6]. При розгляді рівнів абстракції від низу до верху, традиційні цифрові логічні компоненти і елементи пам'яті SRAM/DRAM фізично несумісні один з одним; технологічно вони формуються окремо. При розгляді рівнів абстракції зверху вниз, більшість традиційних додатків вимагають виконання інтенсивних і точних логічних/арифметичних операцій, які можуть задіяти повністю ресурси центрального процесора. Прогрес в галузі мікроелектроніки дозволив отримати незалежну резистивную пам'ять ReRAM, яка дозволяє не тільки зберігати інформацію, але й виконувати обчислення за допомогою логічних і арифметичних операцій. Це унікальна властивість дозволяє встановити нові відносини між обчисленнями і пам'яттю, а також знайти компроміс між точністю обчислень

і продуктивністю, забезпечивши певний рівень наближення обчислень, пов'язаних з обробкою медіа інформації.

Процесор в пам'яті або обчислювальне ОЗУ (Processing-in-memory, PIM) – це концепція обчислень, орієнтована на максимальне наближення процесів обробки до даних за рахунок перенесення частини обчислень від центрального процесора безпосередньо в пам'ять з метою забезпечення максимально можливих значень наступних параметрів: швидкість доступу до даних при масовому зверненні до пам'яті та пропускна здатність, а також забезпечення широкого розпаралелювання обчислювального процесу. Логіка обробки може бути інтегрована в різні рівні ієрархії пам'яті (кеш, DRAM, постійна пам'ять Solid State Drive-SSD) [10-16].

З точки зору імплементації PIM можна умовно виділити два етапи: перший – комбінування комірок логіки і DRAM на одному кристалі (недолік – складність реалізації через несумісність технологічних процесів виробництва пристроїв різних типів); другий – поява технології тривимірної упаковки (3D-stacked die) [17], яка дозволила об'єднати на одному кристалі пам'ять і швидку логіку. Технологія 3D-stacked die дозволяє встановлювати напівпровідникові кристали процесора і пам'яті вертикально один на одного, зменшуючи відстань між ними і енергоспоживання, одночасно збільшуючи пропускну здатність.

В даний час розроблено такі архітектури PIM-систем [18]: Active Pages in FlexRAM (активна сторінка) [19], DIVA (Data Intensive Architecture – інтенсивна архітектура даних), архітектура IRAM (Intelligent RAM – інтелектуальна оперативна пам'ять), VIRAM (Vector IRAM – векторна інтелектуальна оперативна пам'ять), архітектура з потоком пам'яті користувальницького рівня (User-Level Memory Thread – ULMT).

Архітектура FlexRAM реалізована на кристалах з об'єднаною логікою і пам'яттю (Merged Logic and DRAM, MLD) і містить множину відносно простих процесорів, кожен з яких пов'язаний з банком пам'яті [20-22].

Перевагою архітектури FlexRAM є висока пропускна здатність і мале споживання енергії у порівнянні з традиційною обчислювальної системою. Приклад організації системи пам'яті на основі FlexRAM представлений на рис. 1.1. До складу системи входить множина простих обчислювальних елементів (P.Arrays), які перемежовуються з макро-комірками пам'яті DRAM. Щоб уникнути ускладнення міжз'єднань кожен елемент P.Array може «бачити» тільки частина пам'яті на кристалі. Суперскалярний RISC процесор на кристалі P.Mem координує роботу елементів P.Arrays і виконує послідовні завдання.

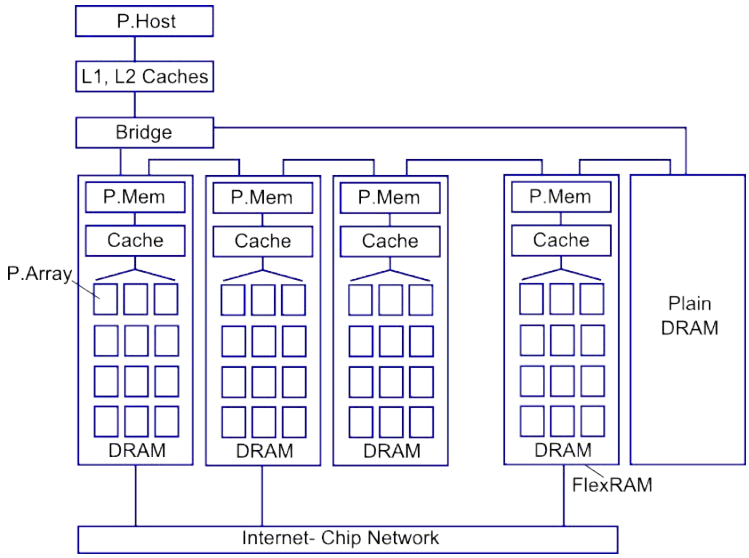



Рис. 1.1 – Організація системи пам'яті на основі FlexRAM

Архітектура Intelligent RAM базується на інтерграції процесора загального призначення і DRAM-пам'яті на одному кристалі з метою вирішення проблеми постійно зростаючого розриву між швидкодією процесора і пам'яті. До переваг архітектури IRAM можна віднести більш високу пропускну здатність пам'яті (до двох порядків), нижчу латентність пам'яті, зменшене енергоспоживання, гнучку деталізацію пам'яті і економію місця на кристалі.

Архітектура VIRAM дозволяє розширити набір команд MIPS за рахунок векторних операцій над даними, представленими у вигляді цілих чисел і чисел з плаваючою точкою, а також операцій з пам'яттю для послідовного,

крокового та індексного доступу [23]. Основними компонентами архітектури є (рис. 1.2): MIPS процесор з кеш пам'яттю, модуль обчислень з плаваючою точкою, DMA механізм для забезпечення доступу до пристроїв за межами кристалу, підсистема пам'яті, планка пам'яті, векторний пристрій. Підсистема пам'яті розбивається на секції. Секція пам'яті визначається як мінімальний розділ пам'яті, який може бути оброблений однією операцією на один такт. Кожна секція складається з контролера секції і набору банків пам'яті. Планка пам'яті забезпечує з'єднання контролера секції з процесором для завантаження / збереження даних. Її розрядність і гнучкість визначають реальну пропускну здатність пам'яті і функціональність міжз'єднань (один або кілька адресних потоків). Процесор має 32 векторних регістри, кожен з яких може містити до 32-х 64-бітних значень. Теоретично, векторна операція може бути виконана на всіх елементах векторного регістра паралельно. В архітектурі, наведеної на рис. 1.2, виділено чотири 64-бітних векторних лінії, тому одна векторна інструкція виконується одночасно чотирма елементами.

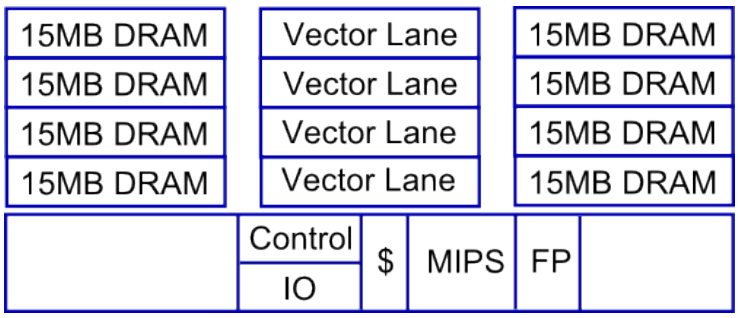



Рис. 1.2 – Блок-схема архітектури VIRAM

Data Intensive Computing є вид додатків, орієнтованих на введення-виведення або обробку великих обсягів даних (big data) [15, 24, 25].

Data-IntensiVe Architecture (DIVA) являє собою систему робочих станцій, побудовану на основі технології вбудованої пам'яті з метою заміни системи пам'яті традиційної робочої станції «розумною» пам'яттю, призначеною для обробки дуже великих обсягів даних [15, 25]. Подолання існуючих обмежень пропускної здатності пам'яті традиційної обчислювальної системи можливе

трьома способами, як показано на рис. 1.3: 1) забезпечення зв'язку єдиного PIM процесора з банком пам'яті на кристалі; 2) формування декількох вузлів процесор-пам'ять на кожному PIM кристалі; 3) використання окремих міжз'єднань типу кристал-кристал для організації безпосереднього зв'язку між вузлами на різних кристалах в обхід головної системної шини.

На рис. 1.3 показана архітектура системи, в якій набір елементів PIM підключений до одного зовнішнього хост-процесору через інтерфейс хост-пам'яті. Зв'язок між елементами PIM забезпечується каналами міжз'єднань [15, 24].

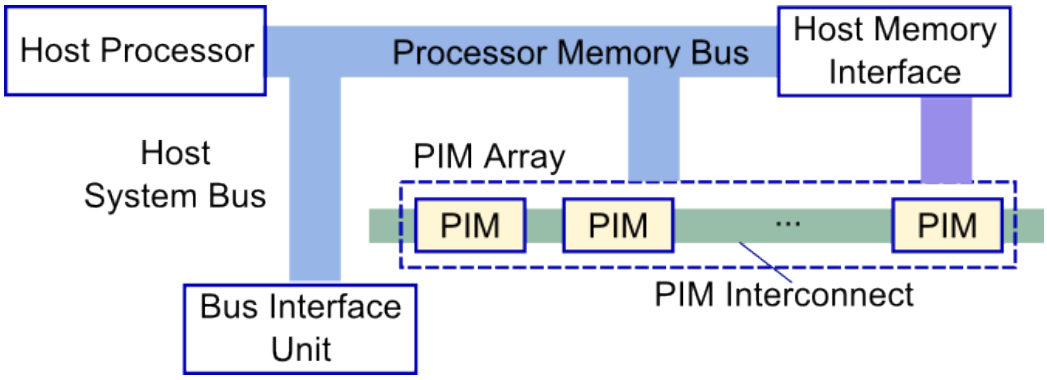



Рис. 1.3 – Організація системи DIVA

В роботі [26] наведено класифікацію обчислювальних систем в залежності від місця розташування робочого набору. На ранніх комп'ютерах (до 1980-х рр.) робочий набір розташований в основній пам'яті, рис. 1.4, а). Розрив між швидкодією ядра (CPU) і пам'яті долається за допомогою кеша, в якому розміщується робочий набір, що дозволяє збільшити продуктивність системи рис. 1.4, б). Сучасні обчислювальні системи, призначені для інтенсивної обробки даних, як і раніше базуються на архітектурі фон Неймана і використовують набір паралельних (міні) ядер з поділюваною кеш-пам'яттю SRAM (паралельні процесори, графічні процесори, SIMD-архітектури VLIW, векторні процесори), див. рис. 1.4, в).

Кластери ядер можуть відтворюватися багаторазово, але формування для кожного з них власної кеш-пам'яті L1 є недоцільним, оскільки при цьому збільшуються розміри і вартість кожного міні-ядра. Крім того, при зменшенні

продуктивності ядра збільшується споживана потужність і з'являються обмеження масштабованості системи. У сучасних додатках з інтенсивним обміном даними, де величезні масиви інформації передаються між процесорними елементами шляхом виконання інструкцій завантаження / збереження, істотно збільшується час простою процесорів при очікуванні даних.

Обчислення, які є основною функцією системи, вимагають менше споживаної енергії і реалізуються на меншій площі кристала у порівнянні з функціями комунікації і доступу до пам'яті [26], особливо це актуально для систем з інтенсивною обробкою даних. Забезпечення програмованості процесора також вимагає значних витрат енергії. Вирішення цієї проблеми можливе шляхом використання нових моделей високопродуктивних обчислювальних систем, орієнтованих на дані (data centric підхід), що дозволяє поліпшити продуктивність і енергоефективність за рахунок скорочення переміщення даних (фактична обробка даних виконується максимально близько до місць їх зберігання в пам'яті). Так, розроблено такі альтернативні архітектури обчислювальних систем:

− процесор-в-пам'яті, рис. 1.4, г), Де додаткові блоки обробки інформації (прискорювачі) розміщуються навколо одного або декількох модулів пам'яті, які і являють собою місце розташування робочого набору; прикладами є FlexRAM, DIVA, TeraSys, EXECUBE, HTMT, обчислювальна RAM, DSP-RAM, архітектура на основі розумної пам'яті (Smart memory based architecture), Gilgamesh, континуумна обчислювальна архітектура (Continuum computer architecture), архітектура MICRON для обробки автоматів;

− пам'ять-в-процесорі, яка є розширенням архітектури, представленої на рис. 1.4, в), де додаткова адресовна пам'ять розміщується максимально близько до ядер; прикладами є архітектура Data Arithmetic SRAM, і машина з'єднань (Connection machine);

– обчислення/СУБД в пам'яті (in memory computing/database), призначена в основному для керування базами даних і дозволяє зберігати робочий набір в основний пам'яті на спеціалізованих серверах, рис. 1.4, д).

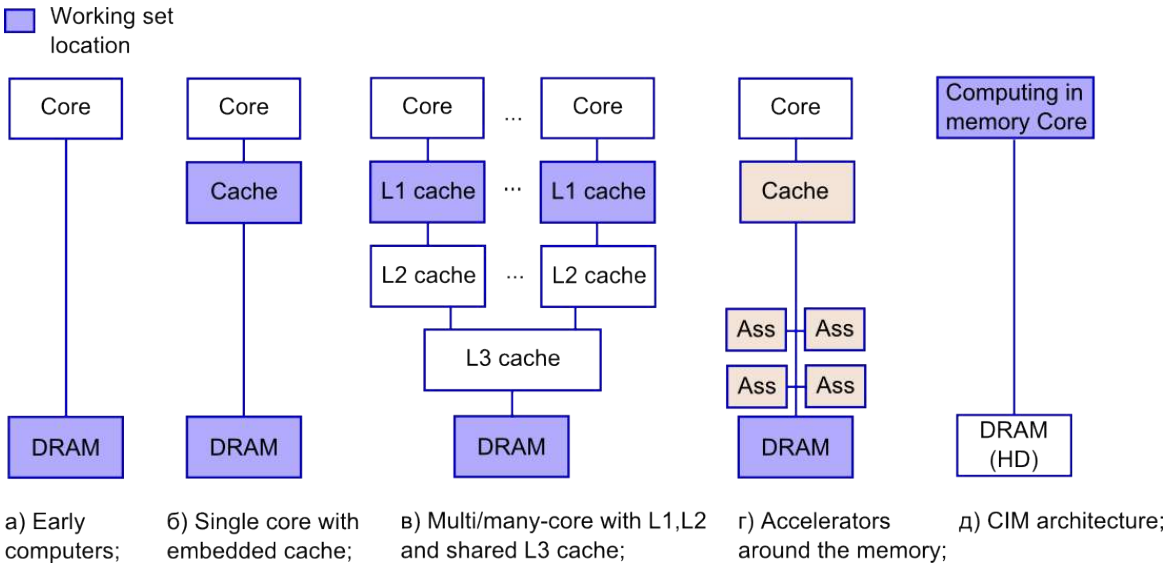



Рис. 1.4 – Класифікація обчислювальних систем на основі розташування робочого набору

1.1.3 Обчислення, керовані пам'яттю. Технологія обчислень, керованих пам'яттю (Memory-Driven Computing), на перше місце висуває пам'ять, яка відіграє ключову роль при виконанні обчислень. Структури обчислювальних систем, які управляються процесором і пам'яттю показані на рис. 1.5.

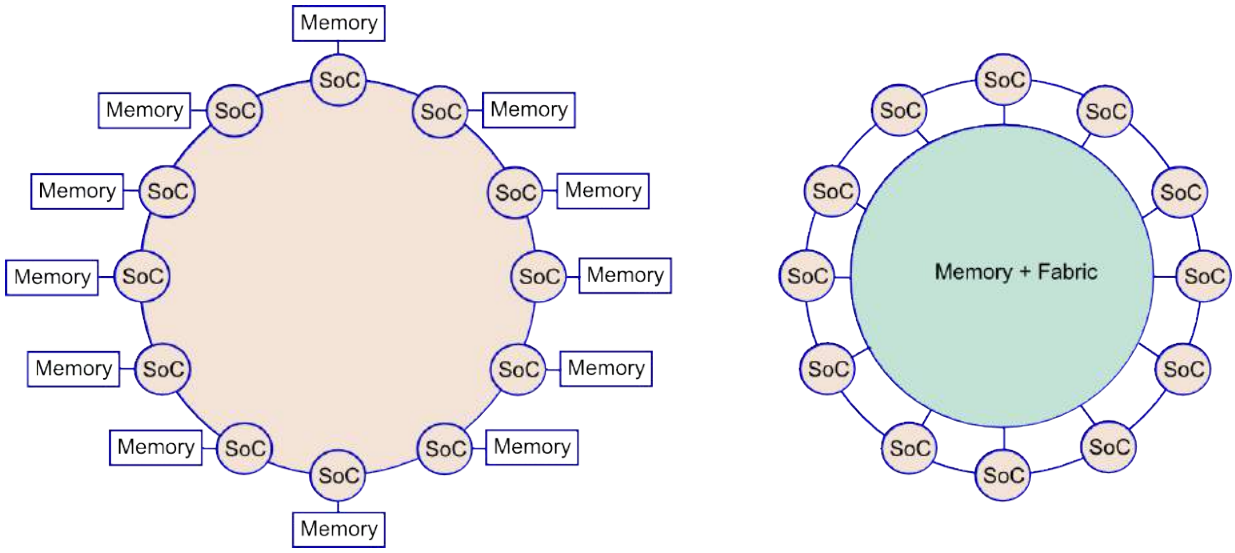
Обчислення, керовані пам'яттю, стали можливими завдяки таким тенденціям комп'ютингу:

– суміщення пам'яті і сховища – постійна пам'ять з адресовним байтом (non-volatile memory, NVM) замінює жорсткі диски і SSD-накопичувачі;

– розукрупнення ресурсів призводить до поділу пулу пам'яті; пул fabric-attached memory доступний для всіх обчислювальних ресурсів; оптичні мережі забезпечують практично рівномірну затримку (латентність); локальна незалежна пам'ять забезпечує менший час очікування і високу продуктивність;

– розподілені гетерогенні обчислювальні ресурси переміщують обчислення ближче до даних;

– програмне забезпечення – постійна швидкодія пам'яті; мала затримка доступу до пам'яті; глобальна адресовність.




а) Processor-centric computing б) Memory-driven computing

Рис. 1.5 – Структури обчислювальних систем, керованих процесором (а) і пам'яттю (б)

Поява нового фундаментального електронного компонента мемрістора, який може не тільки зберігати інформацію, але й робити обчислення, є революцією у компьютингу, що дозволяє істотно збільшити швидкодію і зменшити енерговитрати.

Перший прототип memory-driven комп'ютера “The Machine” розроблений у компанії Hewlett Packard Enterprise [http://www.zdnet.com/article/hpe-demonstrates-memory-driven-computing-prototype/]. Він містить універсальний пул незалежної пам'яті, доступної для великої кількості спеціалізованих ядер, в яких дані не переміщаються між процесорами (серверами), а можуть залишатися на місці в процесі обробки. Метою комп'ютера є не переміщення великих масивів даних через відносно повільні міжз’єднання, а досягнення високої швидкодії обчислень без використання коштовної DRAM. Перехід до квантових обчислень дозволить

здійснити стрибок зростання продуктивності та енергоефективності при збільшенні кількості обчислень.

Можна виділити п'ять технологій, які використовуються в прототипі “The Machine”:

1) гетерогенні спеціалізовані процесорні ядра;

2) пул фотонних міжз’єднань для організації зв'язку між процесорними ядрами і пам'яттю;

3) пул універсальної незалежної пам'яті;

4) нова операційна система;

5) схема програмування.

Процесорні ядра орієнтовані на певні робочі навантаження і можуть являти собою ядра ATOM, ARM або комбіновані. Процесор реалізований у вигляді системи на кристалі (SoC) разом з пам'яттю і фотонними міжз'єднаннями (рис. 1.6).

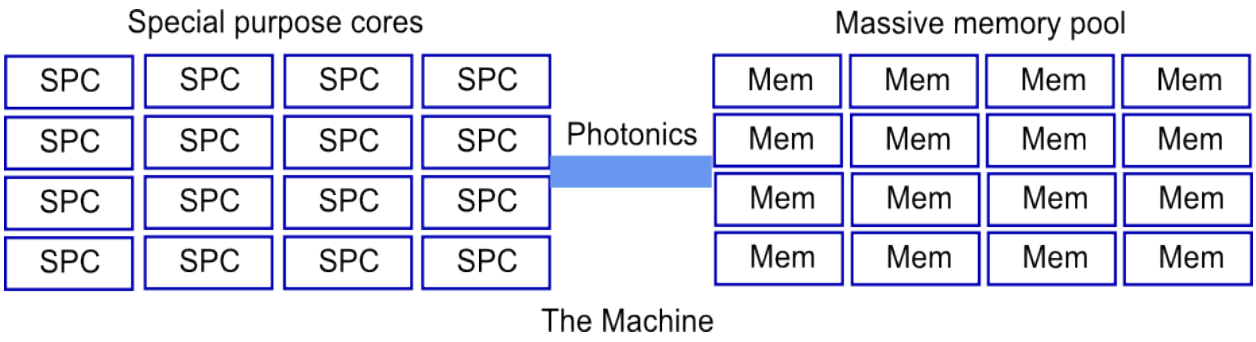



Рис. 1.6 – Схема прототипу memory-driven комп'ютера

## 1.2 Архітектури реконфігурованого комп'ютингу

Реконфігуровані обчислювальні архітектури надають унікальні можливості для вирішення дослідницьких завдань. Вони характеризуються високою продуктивністю, енергоефективністю і гнучкістю програмного забезпечення [27-31]. Реконфігурований комп'ютинг реалізується на просторово програмованих архітектурах і ефективно використовується для проектування замовного апаратного забезпечення, цифрової обробки сигналів

(digital signal processing, DSP), складних обчислень на послідовних і мультипроцесорах.

Вентильні матриці (Field-programmable gate array, FPGA) з'явилися в середині 1980-х років як платформа для нарощування сполучної логіки, яка характеризується більшою продуктивністю, ніж її попередник – програмовна логічна матриця (Programmable Array Logic, PAL) [27]. До початку 1990-х років, з подальшим зростанням продуктивності FPGAs, розширилися і можливості їх використання для логічної емуляції і прототипування, а також персоналізації комп'ютера з метою вирішення спеціальних завдань, пов'язаних з обробкою інформації (відео та аудіо-сигналу) в режимі реального часу, організацією штучного зору, управління, вимірювань, комунікації. При вирішенні завдань оптимізації та просторового моделювання матриці FPGA дозволяють досягти продуктивності рівня суперкомп'ютерів і при цьому характеризуються вартістю рівня робочої станції. Більш того, перепрограмування FPGA дозволяє перенастроювати спеціалізований комп'ютер для вирішення різних завдань і застосування нових алгоритмів. В даний час FPGA широко використовуються в обробці сигналів, криптографії, наукових обчисленнях і телекомунікації.

Три типи елементів входять до складу матриці FPGA: логічні блоки, блоки введення-виведення і програмовні ключі для формування з'єднань між блоками [38-41]. Логічні блоки розташовуються у вузлах решітки вертикальних і горизонтальних провідників, за допомогою яких можна створювати різні зв'язки між блоками. З'єднання є програмовними і здійснюються шляхом налаштування внутрішніх сполучних ключів.

Обчислювальною структурою FPGA являється функціональний генератор Lookup-table (LUT), який по суті являє собою статичну пам'ять [38, 42]. Кожен LUT містить запам'ятовувальні комірки, які використовуються при реалізації невеликої логічної функції. Одинична комірка здатна зберігати значення однієї логічної змінної, 0 або 1. Розмір LUT визначається кількістю входів, яка

залежить від типу мікросхеми. Для програмування LUT необхідно лише зберегти таблицю істинності реалізованої функції в комірках пам'яті блоку. Наявність функціонального генератора LUT і конфігурованих з'єднань в структурі FPGA є необхідною умовою реалізації будь-якої булевої функції.

В даний час використовуються кілька архітектур реконфігурованих обчислень [29], що розрізняються ступенем зв'язку з хост-процесором. Програмовна логіка, як правило, неефективна при реалізації таких операцій, як цикли і розгалужене управління. З метою найбільш ефективного використання обчислювальних ресурсів виконується поділ на програмно і апаратно реалізовані додатки, які виконуються відповідно на хост-процесорі і за допомогою реконфігурованої логіки. Варіанти побудови реконфігурованої системи описані нижче та ілюструються рис. 1.7.

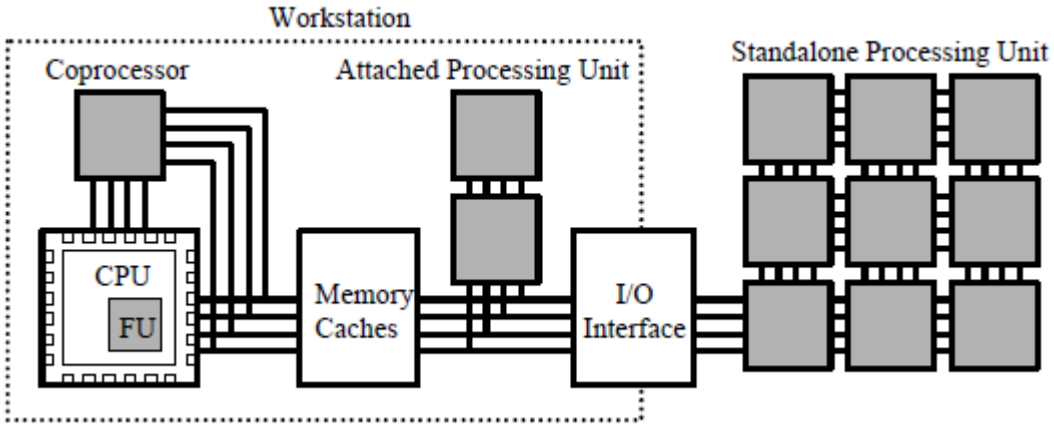



Рис. 1.7 - Різні рівні з'єднань в реконфігурованій системі (реконфігурована логіка має сірий колір)

1. Реалізація реконфігурованих апаратних модулів в складі хост-процесора дозволяє використовувати традиційне середовище програмування і користувальницькі інструкції, які можуть змінюватися з часом. Реконфігуровані модулі використовують головний канал передачі даних мікропроцесора і регістри, призначені для зберігання вхідних і вихідних операндів.

2. Реконфігурований модуль являє собою співпроцесор, що виконує обчислення незалежно від хост-процесора. Останній ініціює реконфігуровані апаратні модулі і передає необхідні дані на елементи логіки (або формує інформацію про те, де ці дані розташовані в пам'яті).

3. Реконфігурований модуль обробки функціонує як додатковий процесор у мультипроцесорній системі. В цьому випадку кеш даних центрального процесора не доступний для реконфігурованого модуля і має місце затримка передачі інформації між хост-процесором і реконфігурованими апаратними модулями при передачі конфігураційної інформації, вхідних даних і результатів обробки.

4.Реконфігурований модуль реалізується у вигляді зовнішнього пристрою обробки інформації. Дана модель конфігурації подібна до мережі робочих станцій, які не вимагають інтенсивного обміну інформацією з хост-процесором.

Кожен з перерахованих вище варіантів має переваги і недоліки. Чим вище щільність інтеграції реконфігурованої системи, тим вище може бути інтенсивність використання апаратних модулів завдяки більш низьким накладним витратам, пов'язаним з комунікацією. Однак обсяг реконфігурованої логіки обмежується необхідністю періодичної взаємодії з хост-процесором. Конфігурації, які характеризуються меншою взаємодією реконфігурованої логіки з хост-процесором, дозволяють реалізувати паралелізм при виконанні програмних додатків, але при цьому зростають комунікаційні витрати.

В даний час вентильні матриці розвиваються в напрямку Coarse Grain Reconfigurable Architectures (CGRA) шляхом додавання компонентів DSP і інших спеціалізованих IP-блоків, використання гетерогенних процесорів, які підтримують паралелізм і містять реконфігуровану логіку [32-37].

## 1.3 CPU і програмована логіка

Вбудовані процесорні пристрої дозволяють істотно розширити можливості FPGA і можуть бути реалізовані як програмні або апаратні ядра (IP-модулі). В даний час розроблено безліч VHDL моделей процесорних пристроїв, які можуть бути багаторазово використані в різних проектах. Перевагою IP-ядра є можливість модифікації архітектури пристрою за рахунок додавання користувальницьких інструкцій та зміни організації кеша. Недоліки – більш низька, ніж у мікропроцесорів, продуктивність, складність реалізації обчислень з плаваючою точкою і необхідність використання великої кількості логічних елементів, що призводить до збільшення площі пристрою та нівелювання наявних переваг [43, 44].

Одним з варіантів вирішення зазначених вище проблем є гібридна архітектура обчислювальної системи, в якій об'єднані жорстке ядро процесора і програмована логіка [45]. Система на кристалі Intel SoCs є апаратною процессорною системою на основі ARM (hard processor system, HPS), до складу якої входять: процесор, периферійні пристрої, інтерфейси пам'яті, що характеризуються високою пропускною спроможністю міжз'єднань. Система характеризується високою продуктивністю і низьким енергоспоживанням у поєднанні з гнучкістю програмовної логіки.

Системи на кристалі на основі ARM, які налаштовуються користувачем, ідеально підходять для вирішення наступних завдань:

− зменшення споживаної потужності, вартості і розмірів за рахунок інтеграції функцій процесора і цифрової обробки сигналів (DSP) в одну FPGA;

− підвищення продуктивності системи за рахунок використання міжз'єднань з високою пропускною спроможністю між процесором і FPGA;

− формування кінцевого продукту шляхом налаштування апаратного і програмного забезпечення;

− адаптивного налагодження програмного забезпечення FPGA.

Кристал FPGA складається з програмовних логічних елементів і міжз'єднань [46-48].

Функціональний генератор Look-up Table є базовим будівельним блоком FPGA, що дозволяє імплементувати будь-яку логічну функцію N булевих змінних [49]. По суті LUT є таблицею істинності, в якій різним комбінаціям вхідних значень відповідають вихідні значення. Граничний розмір таблиці істинності дорівнює N, де N – кількість входів LUT. Для узагальненої N-входової LUT, кількість комірок пам'яті, доступна в таблиці, дорівнює $2^N$, що дозволяє реалізувати $2^{N^N}$ функцій. Типове значення N для Xilinx FPGA дорівнює 6.

Апаратна імлементація LUT може бути розглянута як сукупність елементів пам'яті підключених до набору мультиплексорів. Входи LUT виступають в якості селекторних бітів для мультиплексора, що дозволяють здійснити вибір результату в заданий момент часу. LUT може бути використаний як обчислювач і елемент для зберігання даних. На рис. 1.8 показано функціональне представлення LUT.

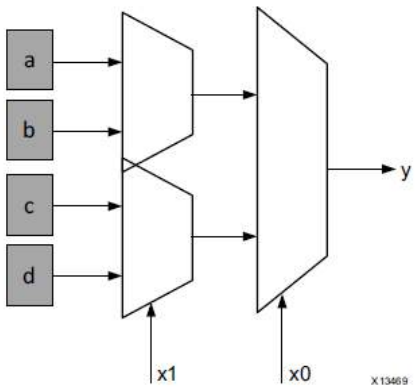



Рис. 1.8 – Функціональне представлення LUT як набору елементів пам'яті

Функціональний k-входовий генератор (K-LUT) являє собою логічний елемент з k входами і одним виходом, що складається з 2k SRAM комірок. K-LUT може реалізувати будь-яку функцію k змінних. В цілому, використання логіки FPGA може бути розширене, але це призводить до зменшення

продуктивності через збільшення глибини схеми. Для вирішення зазначеної проблеми в більшості архітектур FPGA використовується складний програмовний логічний блок (PLB), який складається з декількох LUT, що забезпечує більшу гнучкість.

На рис. 1.9 показано чотири PLB архітектури: (а) - XC4000 CLB; (б-г) - PLB блоки.

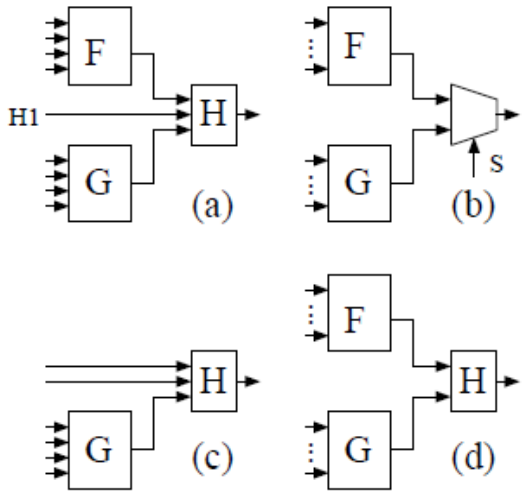



Рис. 1.9 – PLB архітектури: (а) - XC4000 CLB; б) - PLB1; в) - PLB2; г) - PLB3

Функціональні генератори XC4000 CLB, PLB1, і PLB3 містять два вхідних LUT (F і G) і один вихідний MUX або LUT – H. PLB2 містить один вхідний і один вихідний LUT. Блок XC4000 CLB використовується для імплементації будь-якої функції п'яти вхідних змінних або двох незалежних функцій чотирьох вхідних змінних або деяких функцій 9 вхідних змінних (wide functions). PLB1 – блок XC4000 CLB, коли функціональні генератори F і G є 4-входовими, і може імплементувати будь-яку функцію п'яти вхідних змінних або деякі широкі функції 9 змінних. Для зменшення площі кристала LUT генератори F, G або обидва повинні бути трехвходовими. Але при цьому зменшуються їх функціональні можливості, наприклад, не гарантована імплементація будь-якої функції п'яти змінних. Блоки PLB2 і PLB3 спрощені в порівнянні з XC4000 CLB і мають меншу площу, але їх функціональні можливості також зменшені.

Тригер є базовим елементом зберігання даних в кристалі FPGA. Цей елемент завжди використовується в парі з LUT для організації конвейеризації і зберігання. Структура тригера містить входи даних, синхронізації, генератора тактових імпульсів, скидання і виведення даних. Під час нормального функціонування будь-яке значення на вхідному порту даних замикається і передається на вихід на кожному такті. Вихід дозволу синхронізації необхідний для забезпечення можливості передачі значення на більше, ніж одному такті. Входи даних clock і clock enable дозволяють організувати передачу даних на вихідний порт, коли обидва ці входи дорівнюють одиниці. Структура тригера показана на рис. 1.10.

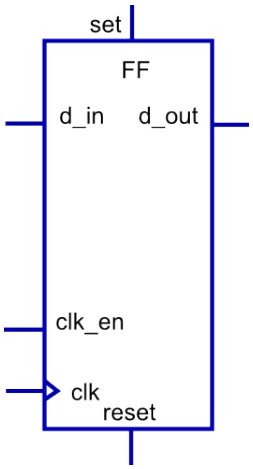



Рис. 1.10 – Структура тригера

Пам'ять на кристалі FPGA представлена BRAM і вбудованими елементами пам'яті, які можна використовувати в якості пам'яті з довільним доступом (RAM), постійної пам'яті (ROM) або регістрів зсуву. BRAM являє собою двухпортовий модуль RAM, вбудований в кристал FPGA для забезпечення зберігання відносно великого набору даних. На кристалі доступні два типи пам'яті BRAM: 18К або 36К біт.

У сімействах Xilinx FPGA Artix-7, Kintex-7 і Virtex-7 використовується уніфікована архітектура нового покоління [50, 51], що базується на масиві конфігурованих логічних блоків (Configurable Logic Block, CLB). Блоки CLB є основними логічними ресурсами для імплементації послідовних і

комбінаційних схем, в які входять такі поліпшені високопродуктивні логічні елементи:

- шести-входові функціональні генератори LUT;
- подвійні п'яти-входові LUT;
- розподілена пам'ять і логіка регістра зсуву;
- спеціалізована логіка прискореного перенесення для арифметичних функцій;
- мультиплексори.

Кожен блок CLB з'єднаний з перемикальної матрицею, яка забезпечує доступ до трасування ресурсів. До складу CLB блоку входять дві секції, які можуть складатися з елементів двох типів SLICEM і SLICEL. Кожен з них включає чотири шестівходові функціональні генератори, з виходами яких пов'язана пара тригерів.

Функціональний генератор LUT Xilinx FPGA серії 7 може бути налаштований як шести-входовий LUT з одним виходом або як 5-входовий LUT з окремими виходами і загальними адресними або логічними входами.

Чотири шести-входові LUT, вісім тригерів, мультиплексори і логіка перенесення утворюють секцію (slice). Дві секції утворюють блок CLB (рис. 1.11). Чотири тригери секції (по одному на LUT) можуть бути опційно налаштовані як засувка. У цьому випадку інші чотири тригери в секції залишаються невикористаними.

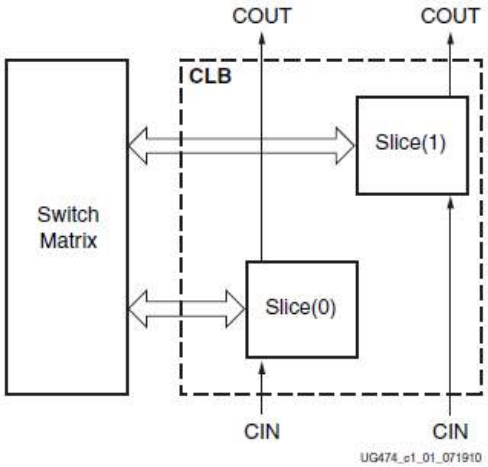



Рис. 1.11 – Схема блоку CLB

Приблизно дві третини секції становлять логічні елементи SLICEL, іншу частину – SLICEM, які можуть використовувати LUT як розподілений 64-розрядний ОЗП, 32-розрядний регістр зсуву (SRL32) або два 16-розрядних регістри SRL16. Сучасні інструменти синтезу дозволяють використовувати переваги високоефективної логіки, арифметичних функцій і пам'яті.

## 1.4 Синтез і моделювання

Алгоритм проектування FPGA представлений на рис. 1.12 і передбачає такі кроки [52, 53]:

1. Введення інформації про проектований пристрій. Найбільш поширеними мовами опису специфікації на рівні RTL є VHDL і Verilog, що дозволяють описати операції на кожному такті. В даний час існує тенденція підвищення рівня абстракції з використанням мов опису поведінки системи, наприклад, SystemC або засобів Matlab, Simulink.

2. Функціональне моделювання проекту або моделювання на рівні RTL для перевірки синтаксису і виконуваних функцій. На даному етапі виконується роздільне моделювання модулів проекту перед тестуванням всієї системи.

3. Введення обмежень проекту (тактова частота, затримки проходження сигналів, прив'язка виводів проекту до входів-виходів кристалу, вибір вихідних рівнів, критичних ланцюгів).

4. Синтез і оптимізація проекту (архітектурно-незалежна оптимізація, меппінг).

5. Оцінка розмірів проекту, продуктивності, стилю кодування і параметрів системи.

6. Розміщення і трасування. Часове моделювання, статичний часовий аналіз.

7. Генерація бітового файлу конфігурації проекту.

8. Створення PROM, ACE або JPAG файлу. Завантаження проекту в кристал FPGA.

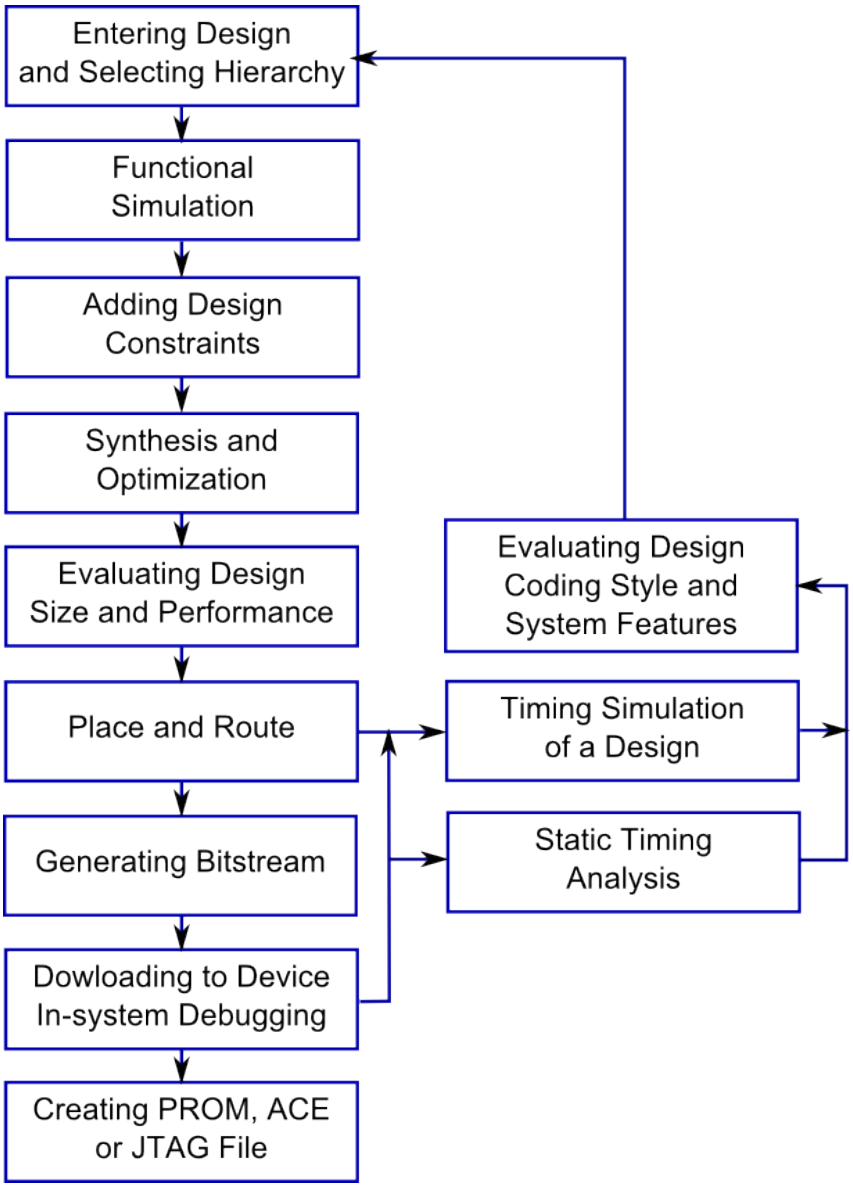



Рис. 1.12 – Алгоритм проектування FPGA

9. Тестування проекту.

## 1.5 Види несправностей FPGA і методи їх виявлення

У матрицях FPGA можна виділити постійні і минущі (тимчасові) несправності [54-65]. Постійні – є результатом фізичних порушень елементів топології схеми, що виникають в процесі старіння або обумовлені дефектами виробництва. Ці несправності моделюються як короткі замикання або розриви ланцюгів, константні несправності вмикання/вимикання, 0/1. Минущі несправності з'являються в результаті короткочасних зовнішніх впливів

(електромагнітні поля, радіактивне випромінювання, джерела живлення), які на деякий час змінюють стани логічних вентилів. Можуть виникати також несправності, що перемежовуються, які мають характер повторних короткочасних збоїв [54-58]. До постійних несправностей матриці FPGA на основі SRAM пам'яті відносять:

1) константні несправності в елементах пам'яті RAM;

2) константні несправності сигнальних ліній CLB;

3) несправний мультиплексор всередині CLB;

4) несправний D-тригер одиничного CLB;

5) константна несправність міжз'єднань CLB.

Діагностування постійних і тимчасових несправностей FPGA виконується так само, як і для класичної фіксованої логічної схеми [66]. За аналогією розглядаються несправності, пов'язані з основними логічними компонентами FPGA (наприклад, CLB, елементами пам'яті). Найбільш критичними є минущі помилки в конфігураційній пам'яті, оскільки вони можуть змінити функції блоків і з'єднань FPGA.

Методи виявлення несправностей можуть бути умовно розділені на три групи [66]: 1) введення надлишковості для одночасного виявлення несправностей – використовується додаткова логіка виявлення в разі, коли функціональна логіка генерує неправильний вихідний сигнал; 2) автономне (off-line) тестування під час, коли система FPGA не функціонує в робочому режимі; 3) «стрибуче» тестування (Roving test) [66, 67] FPGA за допомогою інтерфейсу граничного сканування (Boundary Scan Interface), що полягає у відключенні тестованої області від нормального функціонування на період тестування без порушення штатного режиму роботи решти системи.

Потрійна надлишковість (Triple Modular Redundancy, TMR) широко використовується в якості методу виявлення несправностей в FPGA. Найпростіша форма реалізації даного підходу полягає в двократному або трикратному резервуванні функціонального модуля і порівнянні вихідних

сигналів в процесі функціонування. Будь-яка невідповідність вихідних сигалів вказує на наявність несправності.

Одночасне виявлення несправностей дозволяє використовувати алгоритми кодування для виявлення помилок (наприклад, контроль парності).

Метод характеризується високою швидкістю виявлення помилок, але має недоліки – високі накладні витрати (надлишковість) реалізації, низьку роздільну здатність ідентифікації відмовного компонента (несправність може бути діагностована з точністю до конкретного функціонального блоку).

Метод автономного виявлення несправностей в режимі off-line за допомогою засобів вбудованого самотестування (Built-In Self-Test, BIST) дозволяє виявляти дефекти виготовлення [65]. BIST схеми для FPGA підтримують кілька тестових конфігурацій, які завантажуються окремо від операційної конфігурації. Тестова конфігурація містить генератор тестових наборів, схему, що тестується (набір логічних елементів і міжз'єднань), аналізатор вихідних відгуків. До складу BIST можуть входити також додаткові модулі прискореного перенесення і мультиплексори, призначені для підвищення ефективності та прискорення процесу тестування. На відміну від НВІС (ASIC) FPGA характеризуються регулярністю структури, що є безперечною перевагою, оскільки дозволяє багаторазово використовувати програми тестування в різних проектах. Можливість реконфігурації FPGA зменшує, а в деяких випадках навіть усуває, необхідність формування спеціалізованих тестових структур, вбудованих в кристал. До переваг підсистеми BIST відноситься також те, що вона не впливає на функціональність тестованої системи під час штатного функціонування і забезпечує повне покриття несправностей, в тому числі і таких, які важко або неможливо перевірити в автономному режимі. Виникають лише накладні витрати, пов'язані з необхідністю зберігання тестових конфігурацій. Недоліками BIST є: 1) наявність великої кількості варіантів реалізації логіки на кристалі FPGA, що призводить до необхідності оптимізації тестових

наборів; 2) можливість виявлення несправностей тільки в заданому тестовому режимі, коли FPGA не перебуває у робочому стані.

«Стрибуче» виявлення несправностей використовує run-time реконфігурацію для виконання тестування в області з мінімальними накладними витратами. В процесі тестування FPGA розбивається на області однакового розміру. Одна з них конфігурується для виконання самотестування, в той час як інші продовжують функціонувати в штатному режимі. Після закінчення перевірки обраної області вона переводиться в режим штатного функціонування і вибирається наступна область для самотестування. Процес триває до тих пір, поки всі області не будуть перевірені. Переваги методу полягають в більш низьких накладних витратах (в порівнянні з вищевикладеними методами), які визначаються необхідністю виведення однієї області з режиму штатного функціонування і наявності контролера для управління процесом реконфігурації, а також забезпеченні найбільш повного покриття несправностей і роздільної здатності (глибини пошуку) несправностей. Швидкість виявлення несправностей залежить від періоду повного циклу «стрибучого» тестування.

Тестування міжз'єднань FPGA є складним завданням, оскільки вони займають 80% площі пристрою. Методологія тестування несправностей міжз'єднань, як правило, базується на створенні великої кількості конфігурацій програмовних SRAM комірок, пов'язаних між собою різними проводовими сегментами [56]. Тестування складної системи на кристалі високого ступеня інтеграції є часовитратним процесом, оскільки необхідно сформувати необхідну кількість тестових наборів, виконати тестування і зібрати відгуки системи.

В сучасних FPGA використовуються підсистеми самотестування BIST, що забезпечують максимальне покриття константних несправностей і високу швидкість тестування для різних моделей несправностей FPGA. Основними компонентами BIST є [68]: генератор тестових наборів (Test Pattern Generator,

TPG), тестована схема (BUT - Block Under Test) і аналізатор вихідних відгуків (Output Response Analyzer, ORA), рис. 1.13.

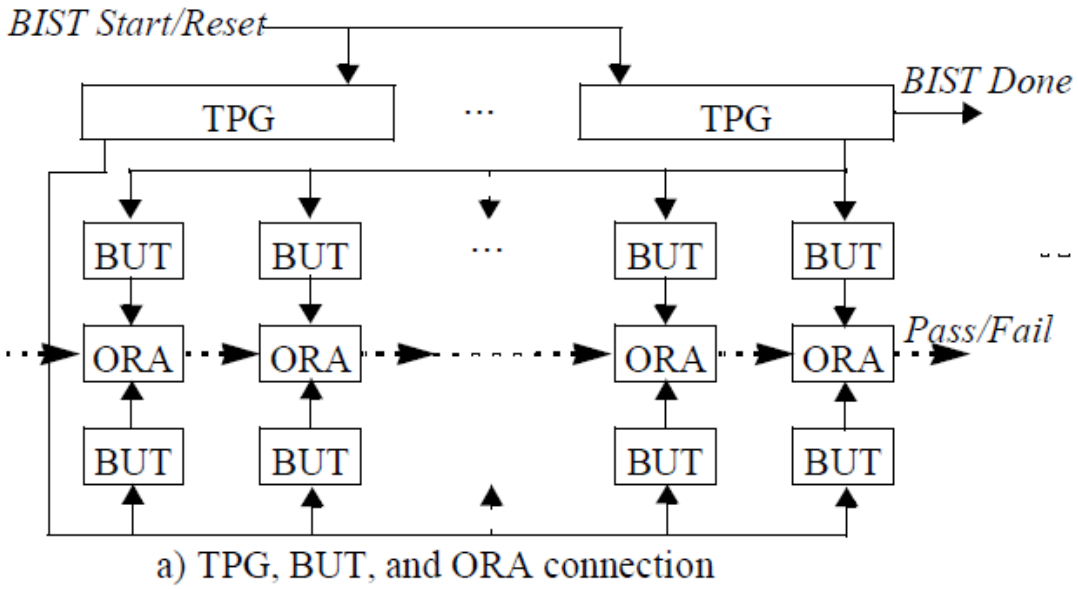



Рис. 1.13 – Архітектура BIST

Кожна фаза тестування складається з наступних етапів: 1) реконфігурація FPGA; 2) ініціалізація тестової послідовності; 3) генерація тестових наборів; 4) аналіз вихідних відгуків; 5) зчитування результатів тестування. На кроці 1 тест-контролер взаємодіє з тестованою FPGA для реконфігурації логіки шляхом вилучення конфігурації BIST з конфігураційної пам'яті і завантаження її в FPGA. Тестовий контролер також ініціалізує модулі TPG, BUT, ORA, ініціює BIST-послідовність і зчитує отримані результати. Кроки 3 і 4 одночасно виконуються логікою BIST всередині пристрою. Після того, як робота BIST системного рівня завершена, тест-контролер переналаштовує FPGA для нормального функціонування (робоча конфігурація пристрою збурігається разом з конфігурацією BIST).

Описаний метод тестування програмовних логічних блоків FPGA на основі пам'яті SRAM за допомогою системи вбудованого тестування BIST може бути використаний на будь-якому рівні проектування і не вносить додаткових накладних витрат або затримок. Кожен PLB піддається псевдовипадковому тестуванню у всіх режимах роботи, тому тести практично

є повними для будь-якої моделі логічної несправності. За допомогою даного методу виявляється будь-який поодинокий несправний програмований логічний блок PLB або будь-яка комбінація несправних блоків. Кількість тестових конфігурацій не залежить від розміру FPGA.

## 1.6 Методи забезпечення відмовостійкості та відновлення працездатності

Відновлення працездатності системи на кристалі може бути виконано на різних рівнях:

1. Апаратний рівень – дозволяє відновити працездатність без зміни конфігурації. У пристрої зберігається початкова кількість і розташування доступних логічних кластерів і міжз'єднань.

2. Конфігураційний рівень – відновлення працездатності забезпечується за рахунок використання додаткових (резервних) ресурсів. При виникненні несправності відмовний модуль замінюється на справний шляхом реконфігурації системи.

3. Системний рівень дозволяє відновити працездатність проекту з високим ступенем модульності. Несправність може бути усунена шляхом використання запасного функціонального блоку або за рахунок зниження продуктивності.

Деякі методи виявлення несправностей дозволяють забезпечити певний рівень відмовостійкості системи. Система голосування при потрійному резервуванні дозволяє ігнорувати одиничну несправність модуля. «Стрибуче» тестування забезпечує відмовостійкість шляхом зупинки процесу заміни тестованої і штатно функціонуючої областей в разі виявлення несправності. Якщо несправність проявляється в межах перевіряльної області, то ця область не буде повернута в режим нормального функціонування. В цьому випадку система працює зі зниженою надійністю, коли поява іншої несправності може привести до порушення відмовостійкості або навіть неможливості її

виявлення. Описаний стан дозволяє системі фукціонувати до відновлення її працездатності.

Регулярна структура FPGA дозволяє здійснювати апаратне відновлення працездатності аналогічно пам'яті. При виникненні несправності частина схеми переноситься на інші ресурси FPGA без зміни її функції [69-74]. Перевагами даного підходу є: простота реалізації, яка визначається відсутністю необхідності передачі інформації про розміщення і трасування контролера відновлення; збереження продуктивності системи і тактової синхронізації на заданому рівні, оскільки несправні елементи замінюються заздалегідь визначеними елементами. Недолік – збільшення кількості несправностей, які можуть бути виявлені та усунуті, призводить до зростання накладних витрат.

Найбільш поширеним способом реалізації апаратного відновлення працездатності є зсув стовпця/рядка. На кінцях рядків/стовпців комірок вводяться мультиплексори, що дозволяють замінювати ряд комірок резервним стовпцем або рядком. Якщо FPGA побудована на основі шин, зсунуті комірки можуть бути підключені до первинних ліній міжз'єднань. У разі наявності сегментованих міжз'єднань, повинні бути передбачені шунтувальні секції для блокування несправного рядка/стовпця. У структурі FPGA також можуть бути виділені підмасиви, які конфігуруються незалежно один від одного і є взаємозамінними. Додавання обхідних з'єднань і мультиплексорів забезпечує більшу гнучкість методу при усуненні кратних несправностей і дозволяє підвищити ефективність використання запасних ресурсів.

Конфігураційний рівень відновлення працездатності базується на використанні ключових особливостей FPGA – реконфігурації і доступності невикористовуваних ресурсів. Стратегії відновлення працездатності на даному рівні можна умовно розділити на три підкласи:

1) альтернативні конфігурації;

2) інкрементний меппінг, розміщення і трасування;

3) еволюційні алгоритми.

Найбільш простим способом відновлення працездатності є попереднє генерування альтернативних конфігурацій. FPGA розбивається на розділи (фрагменти), кожен з яких має власний набір конфігурацій, що визначають функціональність і інтерфейс з сусіднім розділом. Реалізація методу не потребує складних обчислень, оскільки розміщення і трасування проекту виконані раніше. Недоліки методу – співвідношення характеру несправності і розмірів розділу не завжди оптимальне, для охоплення всіх можливих несправностей необхідно формувати велику кількість конфігурацій.

Інкрементний меппінг застосовується для відновлення працездатності логічних кластерів. Метод орієнтований на мінімально можливу зміну результатів розміщення та трасування проекту в процесі відновлення працездатності. При цьому використовуються прості алгоріми дла незначного переміщення окремих блоків і використання звільнених областей для розміщення модифікованих ділянок схеми. Логічні кластери можуть бути переконфігуровані для ізолювання несправності. Конфігурація кластера істотно не впливає на часові характеристики проекту, але можлива ситуація, коли реконфігурація неможлива (наприклад, при переповненні регістра). У разі, коли є запасні кластери, вони можуть бути використані для заміни несправних. З метою мінімізації впливу реконфігурації на часові параметри і трасування проекту може використовуватися галькове зміщення у поєднанні з реконфігурацією кластера. Несправні кластери можуть бути повторно використані іншою функцією, якщо несправність не буде проявлятися. Інкрементний меппінг забезпечує високу ступінь гнучкості для пошуку випадкової несправності. Недолік методу – високі накладні витрати і обчислювальна складність. Для зменшення складності відновлення працездатності та забезпечення гарантованої синхронізації запасні кластери і міжз'єднання можуть розподілятися в процесі проектування.

Еволюційні алгоритми базубться на використанні пулу конкурентоспроможних конфігурацій. Конфігурації конкурують за правильність (коректність), а несправні – мутують. Даний підхід дозволяє забезпечити більшу гнучкість при відновленні працездатності, але вимагає значних обчислювальних і часових ресурсів.

## 1.7 Постановка мети і задач наукового дослідження

Аналітичний огляд архітектур обчислювальних платформ і реконфігурованого комп'ютингу, що використовують memory-driven проектування процесорних компонентів, виконаний в першому розділі і орієнтований на підвищення швидкодії програмних і апаратних засобів аналізу цифрових пристроїв, а також істотне збільшення виходу придатної продукції і зменшення часу її виходу на ринок, дає підстави визначити актуальні напрями в області Design and Test, які є основою для формулювання мети і задач дисертаційної роботи.

Сутність запропонованого науково-технологічного дослідження полягає у створенні векторних структур даних і кубітних методів синтезу, тестування і моделювання, інтегрованих в хмарну інфраструктуру сервісного обслуговування компонентів цифрових систем на кристалах з метою підвищення якості виробів і виходу придатної продукції за рахунок адресовних обчислювальних процесів і явищ. Основна інноваційна ідея запропонованої МАТ-моделі обчислень полягає в синтезі і аналізі векторних цифрових структур на основі адресовних елементів пам'яті, що виключають використання reusable or new logic.

Функція мети Z є підвищення якості цифрових виробів E (виходу придатної продукції Y) за рахунок збільшення розмірності структур даних і пам'яті на основі використання кубітних покриттів функціональних елементів, методів адресного паралельного моделювання, синтезу тестів і аналізу їх якості, інтегрованих в хмарну інфраструктуру сервісного обслуговування цифрових систем, підвищення швидкодії хмарних сервісів

синтезу тестів і моделювання цифрових пристроїв за рахунок використання паралельних регістрових операцій:

$$E = F(L,T,H),$$
$$Z = \min[L,T]\big|_{(H\le 0{,}15F;\,Y\ge 94\%)}\cdot$$
$$Y = (1-P)^{n};\ L = 1 - Y^{(1-k)} = 1-(1-P)^{n(1-k)};$$
$$T = \frac{(1-k)\times H^{s}}{H^{s}+H^{a}};\ H = \frac{H^{a}}{H^{s}+H^{a}}.$$

Тут, L – інверсна змінна виходу придатної продукції Y, що залежить від тестопридатності проекту k, ймовірності P існування дефектних блоків і кількості невиявлених несправностей n. Час тестування, діагностування та відновлення працездатності залежить від тестопридатності архітектури k, помноженої на кількість примітивів, поділеної на загальну кількість логічних елементів проекту. Інфраструктурна надлишковість H для синтезу тестів, оцінки їх якості, розробки таблиць несправностей і ремонту виробу впливає на апаратну складність проекту, будучи платою за надійність, якість проектування та експлуатації.

Мета дослідження – істотне підвищення виходу придатної продукції і якості програмно-апаратних виробів за рахунок створення інфраструктури хмарних сервісів моделювання, тестування і відновлення працездатності на основі використання векторних структур даних адресовних функціональних елементів і підвищення швидкодії кубітних методів синтезу та аналізу.

Задачі дослідження:

1. Розробити модель та напрямки сталого розвитку кіберфізичного комп'ютингу для прогнозування аттракторів в області IT-індустрії на основі просторово-часового аналізу технологічних процесів.

2. Удосконалити векторні моделі кубітного представлення структур і компонентів цифрових систем на основі адресного кодування вхідних сигналів для підвищення технологічності та швидкодії моделювання.

3. Розробити методи синтезу та аналізу векторних описів цифрових схем на основі використання кубітних покриттів для вирішення задач тестування і верифікації.

4. Розробити хмарну інфраструктуру сервісного обслуговування для online проектування і верифікації цифрових проектів з метою зменшення періоду налагодження (time-to-market) HDL-коду.

5. Виконати тестову верифікацію компонентів хмарної інфраструктури сервісного обслуговування, а також методів моделювання, синтезу та аналізу на реальних прикладах цифрових схем і компонентів. Використовувати мови програмування: C++, Verilog, Python 2.7 и платформы: Microsoft Windows, X Window и Macintosh OS X.

# РОЗДІЛ 2

# КОМПЬЮТИНГОВІ МОДЕЛІ ХМАРНИХ СЕРВІСІВ

Пропонується кіберкультура мікро-макро-космо-комп'ютингу, яка формулює, пояснює і прогнозує сучасні технології моніторингу та управління процесами і явищами в фізичному, віртуальному та космологічному просторі. Представляються вербальні та структурні визначення основних типів комп'ютингу, що базуються на сучасних трендах еволюційного розвитку кіберекосистеми планети. Формулюється універсальна модель MAT-комп'ютингу: <Memory, Address, Transactions>, яка використовує три компоненти для створення обчислювальної структури в технологічно прийнятному матеріальному середовищі. Показується інформаційно-квантовий напрям експансії людини в космічний простір, а також можливість аналогічного проникнення неземних біотехнічних об'єктів в екосистему нашої планети. Пропонується модель комп'ютингу, яка описує квазіоптимальні структури моніторингу та управління масштабованими процесами різної природи: технічними, біологічними, соціальними, віртуальними і космологічними.

## 2.1 Введення

Розглядається автоматна модель сталого розвитку людства, задана в метриці зеленої концепції збереження планети і підвищення якості життя людства [75]. Для формування моделі слід визначити трендові аксіоми (The 10 Commitments of Greening), яким необхідно слідувати для сталого розвитку людства:

1) використання автоматної моделі комп'ютингу для моніторингу та управління всіма процесами та явищами, як найнадійнішу і просту для розуміння і виконання. Модель має мінімальну кількість компонентів і сигналів при реалізації замкнутої детермінованої комп'ютингової системи

для досягнення поставленої мети. Життєвість автоматної моделі підтверджується не тільки створенням індустрії комп'ютерів, але й адекватним комп'ютинговим описом процесів і явищ у фізиці, біології, медицині, техніці, соціології, космології;

2) механізми виконання і управління при реалізації моделі комп'ютингу не повинні перетинатися за складом компонентів. Умова регламентує суворий поділ функціональних обов'язків між механізмами управління і виконання, що також означає невтручання одного механізму в справи іншого. Порушення даного пункту є типовою системною помилкою, що породжує корупцію, і властивою управлінню соціальними групами;

3) застосування концепції Бога – все бачить і віддає належне – для моніторингу та управління індивідуумом, соціальними групами і людством в рамках створення хмарних комп'ютингових сервісів. Стійко виключати людину, як ненадійнішу ланку, з моніторингу та управління кіберфізичними, біологічними, технічними, технологічними і соціальними процесами і явищами. Умови дозволяють оптимально управляти кожною людиною, соціальними групами і державами, що виключає всі негативні наслідки від втручання людини, у тому числі війни, соціальні конфлікти, корупцію і несправедливість. Економічний ефект від реалізації концепції вимірюється десятками трильйонів доларів;

4) автоматний детермінізм і передбачуваність реакції комп'ютингу на ініціювальні або актюаторні впливи. Виключення ймовірності з подій і процесів, як «фігового листка на голому тілі нашого невігластва», за визначенням Ейнштейна. Умови орієнтовані на детермінізм комп'ютингових сервісів, що максимально виключає непередбачуваність імовірнісних методів і суттєво зменшує накладні витрати на дублювання та резервування системних компонентів і процесів;

5) тотальне оцифровування всіх просторових, кіберфізичних, біологічних, соціальних процесів, явищ і транспорту, що забезпечують точне

управління на основі цифрового моніторингу та позиціонування. Умови дозволяють в глобальному масштабі вирішити проблему надійного безаварійного, безпілотного управління всіма видами транспорту, зменшити витрати на виготовлення автомобільних номерів (5 млрд доларів), інфраструктуру дорожніх знаків і світлофорів (500 млрд доларів) за рахунок створення віртуальної е-інфраструктури управління транспортом. В даний час на дорогах всіх країн гине в рік 1,2 мільйона чоловік. Цифровий моніторинг природних явищ, ураганів, тайфунів, землетрусів, цунамі, потепління на основі впровадження мережі сенсорів забезпечить актюаторне управління кліматичними і геопатогенними катастрофами. Впровадження технології SlingShot в якості спеціалізованого вирішення проблеми чистої води, від нестачі якої щорічно помирає 2 мільйони жителів планети, шляхом точного моніторингу забруднюючих компонентів і їх подальшого усунення адекватними актюаторнимі впливами;

6) метричне оцінювання всіх процесів і явищ, які формують інтегральний критерій «час-гроші-якість» з метою морально-матеріального стимулювання соціально та екологічно значущих проектів. Метричне оцінювання і подальше адекватне морально-матеріальне стимулювання членів соціальних груп і колективів, що роблять істотний вплив на позитивні процеси підвищення якості життя людей і екосистеми планети. Реалізація глобального ринкового тренду e-infrastructure від програми EU Horizon 2020 стосовно всіх земних процесів і явищ, у тому числі: health care e-infrastructure, smart home e-infrastructure, smart university e-infrastructure, smart city e-infrastructure, social management and government e-infrastructure, traffic control e-infrastructure, internet driven e-infrastructure for diagnosis and repair of technics, computers;

7) повне виключення вторинних ознак ідентифікації людини в кіберфізичному просторі на основі повсюдного впровадження е-інтерфейсів для введення первинних аутентифікаторів (відбитки пальців, райдужна

оболонка ока, ДНК). Умови дозволяють прибрати з обігу мільярди паперових паспортів і електронних карток, які можна підробити, втратити, що дає можливість зберегти ліси і заощадити до 10 мільярдів доларів на виготовлення документів;

8) створення глобальної е-інфраструктури надійного, захищеного доступу до кіберфізичного простору планети і розробка відповідного законодавства, що забезпечують легітимне online виконання функціональних обов'язків, не прив'язаних до робочого місця. Умови дозволяють в масштабах планети на 20 відсотків зменшити транспортний трафік, споживання бензину і шкідливих викидів завдяки виконанню функціональних обов'язків працівниками за місцем проживання;

9) впровадження електронного громадянства для всіх жителів планети і виключення паперових носіїв інформації з усіх сфер людської діяльності. Умови дозволяють зберегти не менше 20 відсотків лісу від вирубки для виготовлення паперу, що підвищить вміст кисню на планеті і поліпшить її екологію. Електронне громадянство скоротить міграцію громадян і зробить все країни привабливими завдяки реальній небезпеці (для керівників відсталих держав) електронного переміщення всіх громадян разом з їх податками;

10) хмарний і big data комп'ютинг на основі планетарної мережі data центрів дозволяє вирішити проблему захисту інформації, сервісів, персональних даних, прибрати мільйони настільних комп'ютерів, локальних серверів і перейти до використання економічних енергозберігаючих планшетів доступу до сервісів та персональним кабінетів. Brain-Computer інтерфейси дають можливість усунути численні пристрої введення даних і перейти до безпосереднього образно-імпульсного спілкування людського мозку з комп'ютинговими терміналами. Сканування мозку на основі зовнішніх або вбудованих сенсорів дозволяє запобігати злочиннам або

нелегітимним діям людини, що істотно вплине на ефективність роботи поліції і спеціальних служб.

Виконання перерахованих аксіом дозволить перемістити всі компоненти механізму управління земними процесами і явищами в хмари комп'ютингових сервісів, що звільнить істотну частину (20%) людства від виконання невластивих йому управлінських непродуктивних процесів. Передати управління хмарному комп'ютингу означає істотне зниження накладних витрат і шлях до створення зеленої планети (якість життя + чиста екологія).

Вартість реалізації перерахованих аксіом порядку 50 мільярдів доларів, економія від їх впровадження – не менше 50 трильйонів доларів, плюс якість життя кожної людини і реінкорнація екології зеленої планети.

Всі технічні проблеми для реалізації десяти трендів вже вирішені в тій чи іншій мірі. Головною перешкодою на шляху створення зеленої планети і щасливого життя є невисокий рівень кіберкультури людини. Тому проблема досяжності поставлених зелених цілей жорстко пов'язана з вихованням довіри у людей до надійності, справедливості і непідкупності комп'ютингових сервісів моніторингу та управління.

Сталий розвиток зеленого комп'ютингу представлено в метриці простору і часу трьома історичними періодами або фазами, наведеними на рис. 2.1:

1) відображення (моніторинг) фізичних процесів і явищ, представлений сингулярним (Single Computing), мережевим (Network Computing) і глобальним комп'ютингом (Global Computing - Internet). Тут також фігурують поняття Desktop, Servers, Data Base;

2) управління фізичними процесами і явищами на основі e-infrastructure, цифрового моніторингу, представлене Cloud Computing, Cyber Physical Networks, Internet of Things. Цикл теперішнього часу, де основними учасниками є Gadget, Laptop, Data Centers, Big Data;

3) створення інтелектуальних кіберфізичних процесів і явищ під керуванням кібермозку планети, представлене Brain Computing, Robotic Networks, Internet of Nature (World). Очікується поява Massive Quantum-Atomic Computing, Brain-Computer Interface, Atomic Data Center Networks, Smart Big Data Networks.

На рис. 2.1 також представлені вектори розвитку зеленого комп'ютингу, які оформлені в наступні фазові трансформації за трьома історичними періодами: 1) Single Computing – Cloud Computing – Brain Computing. 2) Network Computing – Cyberphysical Computing – Network Robotic Computing. 3) Internet Computing – Internet of Things – Internet of Nature. 4) Data based Computing – Big Data Computing – Smart Data Computing.

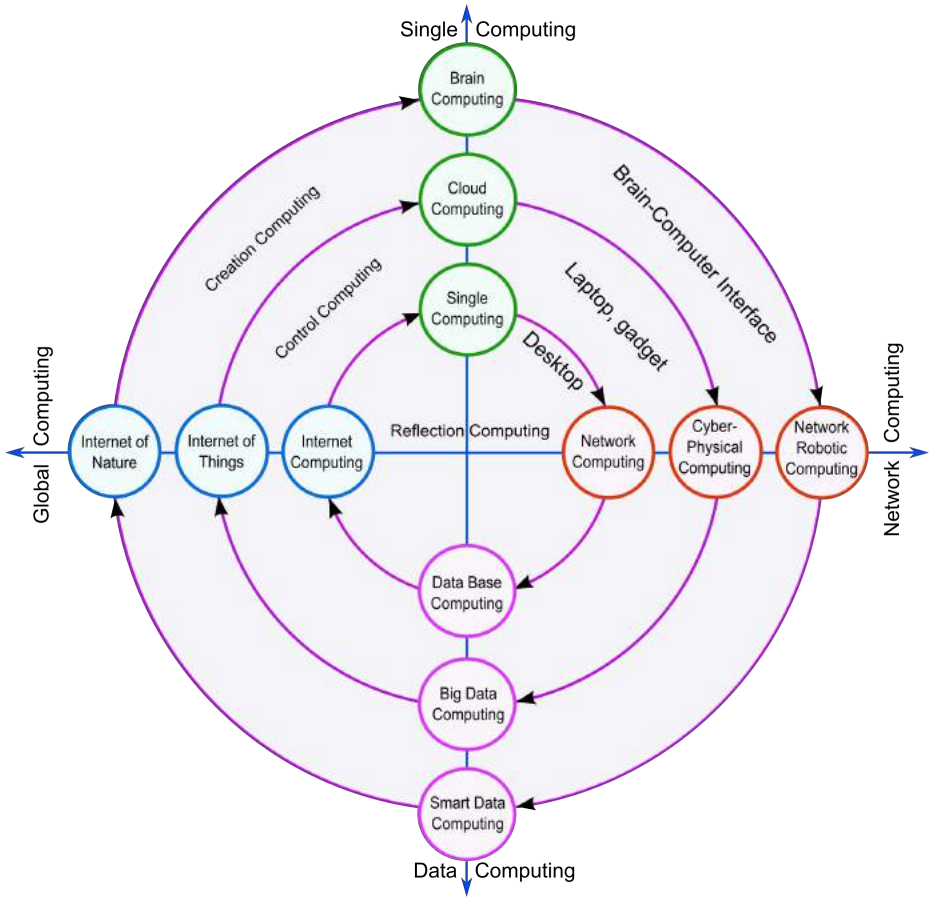



Рис. 2.1 – Комп'ютинг у сталому розвитку

Космологічна модель Всесвіту представлена жорстко пов'язаними між собою взаємодіючими компонентами: Простір і Час, Матерія і Енергія.

Наведену структуру базових понять можна застосувати і до процесу сталого розвитку комп'ютингу. Для цього необхідно виділити такі компоненти: 1) маса матерії (m) для реалізації примітиву-транзистора або комп'ютера; 2) енергія (E) для виконання елементарної операції або функціонування комп'ютера; 3) швидкодія (t), як величина, зворотна часу (t=1/T) виконання однієї елементарної операції або продуктивність комп'ютера (t=I/T, кількість інструкцій за секунду); 4) простір (S) (кіберфізичний), що обслуговується комп'ютингом, шляхом надання користувачам сервісів моніторингу, управління, творення.

Використовуючи введені компоненти в якості аргументів, можна визначити на історично незначному проміжку часу (1970-2020) два досить парадоксальних і дуже оптимістично зелених закони розвитку комп'ютингу (рис. 2.2): 1) зв'язок між енергією і часом; підвищення швидкодії комп'ютера пов'язане зі зменшенням енергоспоживання; 2) зв'язок між матерією і простором; зменшення маси комп'ютера пов'язане з розширенням обслуговуваного простору; обидва закони вірні також у зворотному прочитанні.

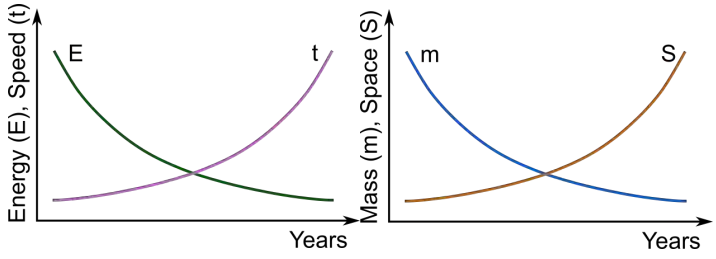



Рис. 2.2 – Взаємодія компонентів комп'ютингу

Якщо виконати суперпозицію двох законів, то можна отримати інтегральний закон комп'ютингу: збільшення швидкодії і простору комп'ютингу жорстко пов'язане зі зменшенням його енергоспоживання і маси. Якщо умовно визначити інтервал зміни відносних значень всіх чотирьох параметрів між 0 і 1, то закон можна записати в наступному вигляді: t+E=1, S+m=1. Це означає, що адитивна оцінка швидкодії (простору)

і енергоспоживання (маси) комп'ютингу на піввіковому відрізку часу є величиною постійною.

Найбільш значущі ринково-орієнтовані відкриття та інновації вчені роблять шляхом суперпозиції міждисциплінарних досліджень, а також імплементації досягнень однієї технологічної культури в суміжні галузі знань. Моделі комп'ютингу знаходять все більше застосування для моніторингу та управління процесами і явищами в усіх областях діяльності людини і природи [76-79]. Інтеграція природної і біосоціальної культури з кібертехнологічними рішеннями моніторингу та управління приводить до оригінальних системних (кіберфізичних, біоінформаційних) наукових результатів та інновацій в традиційно консервативних галузях знань, таких як: природа, біологія, соціологія, екологія, техніка, транспорт і промисловість. Підтвердженням сказаного можуть служити модні глобальні технології, що використовують масштабовані моделі комп'ютингу [77,78]: 1) Cyber-Physical Systems; 2) Internet of Things and Everything; 3) Web- Cloud-, Mobile-, Service-, Network-, Automotive, Big Data and Quantum computing; 4) Internet-Driven Smart Infrastructures: Enterprise, University, City and Government. Так, наприклад, використання автоматної моделі комп'ютингу для опису процесів мозку привело до точного моніторингу порушень в ньому і подальшого ефективного відновлення функціональностей шляхом застосування актюаторних впливів [80]. Вчені роблять успішні спроби в розробці комп'ютерних структур на основі використання природних рішень, які ототожнюються з мозком живих істот [81], що неодмінно приведе до створення кібермозку людства.

Поняття комп'ютингу розвивається з класичної автоматної моделі обчислювача, що об'єднує механізми управління і виконання, сигнали моніторингу та актюації, входи для введення інструкцій і даних, а також виходи стану системи і результатів. Комп'ютинг – процес досягнення поставленої мети шляхом використання механізмів управління та виконання

в циклічно замкнутій системі з заданими входами і виходами, сигналами моніторингу та актюації. У вузькому сенсі, комп'ютинг – це дії, цілеспрямовані на дослідження, проектування і застосування інтелектуальних програмно-апаратних систем і мереж для моніторингу та управління кіберфізичними процесами і явищами. Область комп'ютингу покриває: нано-, мікро- і макро-електроніку, радіотелекомунікації, комп'ютерну, програмну, системну, виробничу, транспортну та соціальну інженерію, штучний інтелект і кіберуправління, комп'ютерні науки та інформаційні технології.

Слід нагадати, що інформаційні технології – це процеси, методи і способи пошуку, збору, зберігання, обробки і поширення інформації шляхом використання засобів обчислювальної техніки [82,83]. З урахуванням наведених вище визначень можна зробити висновок, що комп'ютинг поглинає досить застаріле, але, в силу інерції людського мислення, поширене поняття «інформаційні технології», яке формально ототожнюється з відображенням (моніторингом) фізичних процесів і явищ у віртуальному кіберпросторі.

Істотним видається відмінність комп'ютингу від інформаційних технологій, пов'язана з активним управлінням процесами і явищами в реальному і віртуальному світах. Можна поставити знак відповідності між поняттями змісту і форми: Internet = Information Technology і Internet of Things = Computing. Розвиток комп'ютингу має історично виражені фази (рис. 2.1): 1) сингулярний комп'ютинг; 2) мережевий комп'ютинг; 3) глобальний комп'ютинг; 4) кіберфізичний комп'ютинг; 5) сервіс-комп'ютинг - початок нового циклу. Далі передбачається поява кіберлюдського комп'ютингу, коли human brain буде безпосередньо інтегрований у кіберпростір. Розвиток комп'ютингу, основна функція якого оптимальне і надійне управління всіма процесами та явищами на основі точного цифрового моніторингу без прямої участі людини, слід розглядати тільки у

взаємодії двох світів, реального і віртуального: 1) людина завжди погано керує реальним світом і створює собі в допомогу комп'ютинг; 2) як більш досконалий механізм, комп'ютинг забирає управління технологічними процесами у людей – фаза теперішнього часу; 3) щоб врятувати людство від самоліквідації, комп'ютинг, найближчим часом, повинен забрати і решту управління соціальними процесами під свою юрисдикцію; 4) людина і комп'ютинг об'єднуються в бажанні творчої зміни кіберфізичного континууму в цілях мирного співіснування і взаємного проникнення для створення нового поняття кіберлюдини (Cyber Human – CyMan). Кіберлюдина – персона, яка володіє глобальною технологічною кіберкультурою, безпосередньо (Brain-Computing-Interface) підключена до кіберфізичного простору планети в цілях виконання кіберсоціальної ролі. Комп'ютинг сьогодні перетворюється з ненадійної локальної конструкції в незнищенну глобальну субстанцію. Людина поступово буде трансформуватися з уразливого біологічного суб'єкта в кібербіологічну форму, а далі – в кіберенергетичну, «вічно живу» інформаційно і фізично реінкорновану субстанцію.

В результаті взаємодії людини з кіберфізичним світом формується нове поняття – кіберкультура, як рівень розвитку соціально-технологічних відносин між суспільством, фізичним світом і кіберпростором, який визначається впровадженням інтернет-сервісів точного цифрового моніторингу та надійного метричного управління в усі процеси і сфери людської діяльності, у тому числі освіту, науку, медицину, виробництво і транспорт, в цілях підвищення якості життя людей і збереження екосистеми планети.

Bernard Marr (Forbes) запропонував 9 аттракторів [84], що підтверджують інтерес ринку і світового бізнесу до комп'ютингу: Big Data, Internet of Things, Mobile to computing everywhere, Cyber security, E-Assistants or Brain-Computer Interface, Social Networks, Gamification for Bisiness and

Education, Cloud computing, Video communications. У найближчому майбутньому очікується масове впровадження в життя людей нових технологій, які формують Virical Continuum: Internet of Everything, wearable's, smart car, smart home, smart city, 3D printing, quantum computing, robotics, the cloud, Big Data, the maker movement, drones. Особливу увагу автор приділяє рекламі IoT. Тому, що 87 відсотків людей не чули, що це таке? Фактично – це взаємодія віртуального і реального світу: 2015 рік – 5 мільярдів з'єднаних пристроїв, 1,4 мільярда смартфонів; 2020 рік – 50 мільярдів і 6,1 зазначених виробів відповідно. Це також 250 мільйонів машин у 2020 році, які будуть керуватися без водіїв за допомогою хмарних кіберсервісів дорожнього руху. Вже сьогодні Google-машини проїжджають близько 10 000 миль на тиждень по інноваційним міським дорожнім інфраструктурам. Більше, ніж 10,2 мільйони «розумного» одягу буде вироблено до 2020 року. Також істотно зросте ринок RFID-міток для цифрової ідентифікації об'єктів і процесів з 11,1 до 21,9 мільярда доларів. У підсумку, фінансовий вплив IoT в 2025 році на світовий ринок складе суму в 11 трильйонів доларів, а рівень IoT-капіталізації – 4,6 і 14,4 трильйони в публічному і приватному секторі відповідно.

Susan Galer (Forbes) вважає, що хмарний сервіс буде найбільш цікавою моделлю IT-бізнесу в найближчі 10 років [85]. До кінця 2017 року дві третини від 2000 глобальних компаній трансформують свою діяльність під цифрові процеси моніторингу та управління. Понад 50 відсотків усіх інвестицій компьютингових (IT-) компаній будуть спрямовані на створення технологічних платформ і сервісів, пов'язаних з cloud, mobile, social business and big-data analytics. До 2020 року cloud-based капіталізація IT-компаній досягне 70 відсотків від всіх програмно-технологічних сервісів. У 2018 року будуть інстальовані 22 мільярди internet of things devices, які матимуть доступ до більш, ніж 200 000 нових internet of things додатків і сервісів. До 2018 року більше 50% компаній-розробників будуть додавати до своїх додатків

пізнавальні сервіси (проти 1 відсотка сьогодні), забезпечуючи тільки USA-підприємствам понад 60 мільярдів доларів щорічної економії. До цього часу 50 відсотків усіх підприємств створять хмарні платформи для поширення власних інновацій і споживання зовнішніх пропозицій. До 2018 року 80% компаній в форматі B2B (Busines-to-Business) і 60% B2C (Busines-to-Customer) організацій перебудують свої входи (двері) на цифрові, що забезпечить збільшення споживачів і замовників на 3-4 порядки. Важливим є і те, що до 2020 року більше 30 відсотків успішно існуючих сьогодні компаній – IT-постачальників різних послуг, які не перебудуються під нові цифрові відносини в кіберпросторі, перестануть існувати. Можна резюмувати, всі підприємства та організації, у тому числі університети, міста і країни, виявляться неконкурентоспроможними без інтеграції в кіберпростір заздалегідь підготовленими цифровими правовими відносинами (digital legislations).

## 2.2 MAT-комп'ютинг для фізичного, віртуального і космологічного простору

Мета – створення загальної моделі MAT-комп'ютингу, як універсальної та масштабованої структури моніторингу та управління фізичними, віртуальними і космологічними процесами і явищами, що надає людині можливість освоювати нові життєві простори, у тому числі космос, а також створювати ринково-орієнтовані продукти і кіберсервіси, спрямовані на підвищення якості життя людини і збереження екосистеми планети [77, 78, 86-91].

Задачі: 1) Створення універсальної і масштабованої моделі MAT-комп'ютингу, яка описує фізичні та віртуальні процеси моніторингу та управління. 2) Класифікація існуючих технологій комп'ютингу в реальному і віртуальному просторі для створення загальної картини кіберфізичного континууму (Cyber Physical Continuum). 3) Квантова телепортація процесів, але не об'єктів, як технологія проникнення людини в космос. 4)

Кіберфізичний і квантовий комп'ютинг, як методологія опису процесів розвитку природи і суспільства.

Корисними можуть бути модифікації деяких визначень в бік спрощення. Комп'ютинг – це технологія інтерактивного моніторингу та управління процесами і явищами для досягнення мети за заданою програмою без (прямої) участі людини. Основа комп'ютингу – транзакційна взаємодія адресовних даних (компонентів пам'яті) для досягнення поставленої мети. Модель комп'ютингу: <Пам'ять, Адреса, Транзакція> (рис. 2.3, зліва). Адреса – субстанція, яка визначає структуру координатами компонентів у віртуальному чи реальному просторі. Структура формується різними субстанціями: гравітацією, внутрішньо- і міжатомною взаємодією (базон Пітера Хіггса), електромагнітними полями, електричними зв'язками, свідомістю людини, адресною взаємодією. Транзакція – цілеспрямований процес прийому-передачі даних між адресовними компонентами структурованої пам'яті для реалізації функціональності або сервісу. Пам'ять – будь-яка фізична субстанція, здатна зберігати, приймати і передавати дані, як віртуальну сутність. Оскільки будь-яка матерія складається з атомів і електронів, то кожна з чотирьох форм існування будь-якої матерії має властивості пам'яті.

Рівняння фізичного комп'ютингу задає системна симетрична (xor) взаємодія трьох рівнозначних субстанцій (Memory – Address – Transactions): M⊕A⊕T= 0. Це означає, що будь-який компонент визначається за допомогою взаємодії двох інших: M=A⊕T, A=T⊕M, T=M⊕A. Віртуальний комп'ютинг оперує тріадою: (Data – Address – Transactions). Універсальне характеристичне рівняння комп'ютингу (комп'ютера), оперує єдиною адресною транзакцією (зчитування–запис) даних між джерелом і приймачем [77,78]: M(Yi) = Qi [M(Xi)]. Тут W=<M, Q, X, Y> – структура, яка містить адресовні комопоненти пам'яті і нічого більше: M – пам'ять комп'ютингу; Q – пам'ять функціональних примітивів - квантів [77,78]; X – пам'ять адрес

вхідних змінних примітивів; Y – пам'ять адрес вихідних змінних примітивів. Природно, що всі компоненти пам'яті та комірки в них є адресовними. Це означає, що вони можуть замінюватися і ремонтуватися в режимі remoute and online. Недолік пам'яті один – програш у швидкості виконання транзакцій перед reusable logic або комбінаційними логічними елементами. Таким чином, комп'ютинг, як обчислювальний процес, зводиться до просторово-часової структури транзакцій над адресовними даними.

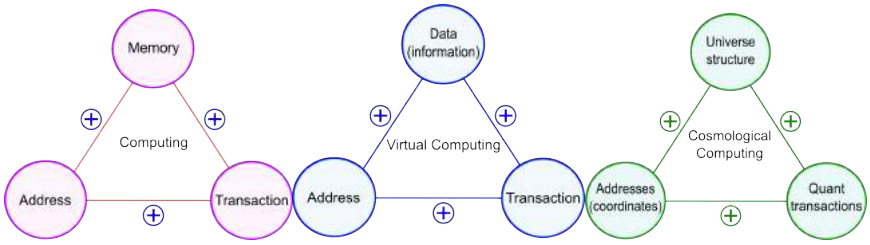



Рис. 2.3 – Модель MAT-комп'ютингу

Далі наведено види комп'ютингу (рис. 2.3) без участі людини, але для людини і збереження планети, із зазначенням пари: тип носія (пам'ять) – сигнатура (транзакції): 1) атомарний, стан або спін електрона - квант, фотон; 2) кристалічний, стан транзистора - електрони; 3) космологічний, стан матерії - гравітаційні і електромагнітні поля; 4) біологічний, стан клітини - електрони; 5) технологічний, стан системи - дані; 6) соціологічний, стан суспільства - законодавство; 7) віртуальний (рис. 2.3, праворуч), стан кіберпростору - інформація.

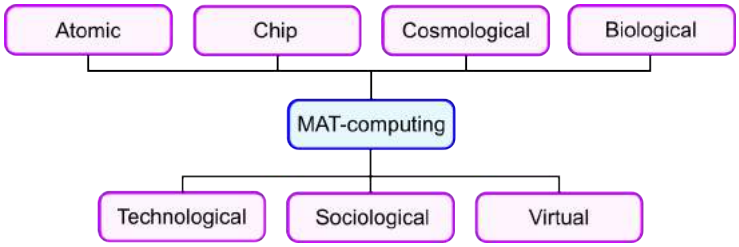



Рис. 2.4 – Види комп'ютингу

Віртуальний або кіберкомпьютинг оперує даними (див. рис. 2.3, праворуч) замість пам'яті, розподілений в (кібер) просторі і сьогодні представлений модними структурними напрямами (рис. 2.4): 1) Cloud Computing; 2) Fog Network Computing; 3) Mobile Computing; 4) Service Computing (Web Services); 5) Social Computing; 6) Automotive Computing; 7) Internet Computing - Smart Everything; 8) Cyber Physical- or Internet of Things (Everything) Computing; 9) Big Data Computing; 10) Quantum Computing (Security).

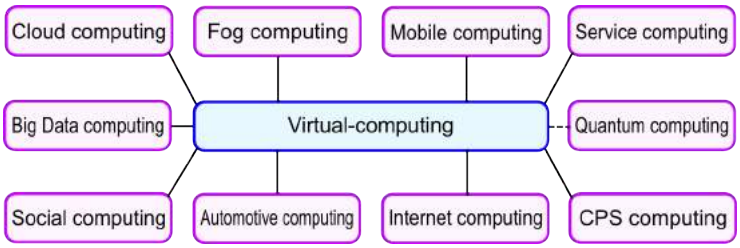



Рис. 2.4 – Віртуальний (кібер) комп'ютинг

Фізичний комп'ютинг (рис. 2.5) сконцентрований у функціональностях, зосереджених у компактному просторі твердотільних спеціалізованих пристроїв: 1) Quantum Computers; 2) Mobile Gadgets and Laptops; 3) Automotive Computers; 4) Smart Sensors and Actuators as MEMS; 5) Robotics; 6) Drones; 7) 3D-Printing; 8) Smart Brain-User Interfaces; 9) Security Computers; 10) Big Data Centers.

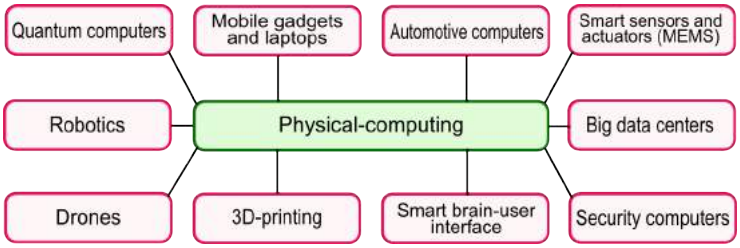



Рис. 2.5 – Фізичний комп'ютинг

Види комп'ютингу, представлені вище, покриваються узагальненою автоматною структурою керованих фізичних і віртуальних процесів, відомих людству (рис. 2.6).

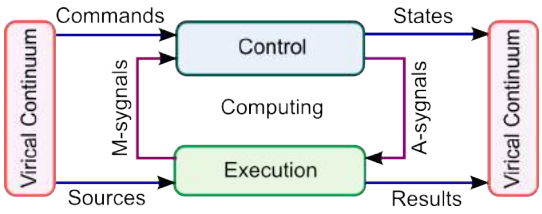



Рис. 2.6 – Автоматна модель комп'ютингу

Тут в якості основного компонента фігурує пам'ять, як будь-яка субстанція, здатна зберігати інформацію, на якій організовані структурні компоненти комп'ютингу (Control and Execution). Функціонування і розвиток системи здійснюється за допомогою моніторингу (M-sygnals) і управління (актюації) (A-sygnals) компонентами обчислювальної системи. Входи і виходи такої системи навантажені на єдиних зовнішній віртуально-фізичний (virical) простір (virtual-physical continuum) [Forbes, Josh Linkner]. Таке континуальне визначення кіберфізичного простору стосується не тільки Землі, але й Всесвіту, який також має програму свого еволюціонування за найпростішою і доступною для розуміння автоматною моделлю комп'ютингу. Модель адекватно, ідеально і просто описує всі процеси функціонування й еволюціонування технічних, біологічних, соціальних, віртуальних і космологічних об'єктів і структур на основі використання сигналів моніторингу та управління. При цьому представлений вище автомат споживає вхідні дані і ресурси із зовнішньої екосистеми, яка також є місцем, куди потрапляють результати або продукти діяльності конкретного комп'ютингу. Тому такий автомат, в залежності від мети, може бути перетворювачем кібер-фізичної екосистеми в Green Planet for Human або в сторону Armageddon деструктуризации.

## 2.3 Комп'ютинг квантової телепортації

Комп'ютинг квантової телепортації передбачає передачу на відстані не об'єктів, а технологічних процесів і відносин, які можна відтворити на приймальній стороні телекомунікаційного каналу. Прихід космічних прибульців на Землю не такий вже неймовірний. Сьогодні дозріває впевненість, що людство зможе продовжити життя на подібних до земних умов планетах у Всесвіті шляхом квантової передачі актюаторних потоків інформації, що описують структури і алгоритми еволюційного синтезу і розвитку земних субстанцій. Потрапивши в сприятливе середовище якої-небудь планети, кванто-кодовані геноми зможуть підготувати інфраструктуру, а потім виростити форсованим або еволюційним шляхом біологічні та технічні об'єкти. Здійснення такої дизрапторної експансії людини у відповідну область космічного простору реально навіть на сучасному рівні розвитку науки і технологій. При цьому космічні польоти з біологічними або технічними речовинами на борту для міжзоряних відстаней – тупиковий шлях фантастів і вчених. Таким чином, розширення життя людства в космосі реально і станеться саме шляхом квантової телепортації процесів синтезу біотехнічних об'єктів в прийнятне планетарне середовище. Для цього вчені повинні в земних умовах вирішити технологічну проблему квантової актюації форсованого або еволюційного синтезу згаданих субстанцій. Неважко припустити, що більш розвинені цивілізації вже вирішили для себе проблему космічної експансії своїх форм життя і зробили квантову інтервенцію біотехнічних об'єктів на Землю шляхом фотонної доставки алгоритмів їх синтезу в благодатному земному середовищі.

Структурно-алгоритмічна сутність квантової телепортації об'єкта полягає в доставці даних про зв'язки компонентів в деяку точку простору, де є комплектуючі елементи, а також про послідовність дій, що становить оптимальну технологію виготовлення певної структури. Наприклад, в Монголії можна побудувати фабрику з виробництва інтегральних схем, для

чого необхідно компанії Інтел найняти фахівців під систему відносин, що телепортується в дану країну, дати їм технології і техніку для виробництва чіпів. Щоб створити університет світового рівня (Кембридж, Стенфорд, Массачусетс), потрібно переміщувати в проблемний вуз тільки систему відносин (статут і традиції) від лідерів науки і освіти. Щоб створити сприятливу для населення країну, слід переміщувати гуманну людино-орієнтовану систему відносин (Конституцію), прийняту, наприклад, в США, Німеччині, Норвегії, Сінгапурі.

Таким чином, при наявності абсолютно однакових умов і компонентів ефективність і якість структури або системи визначаються тільки зв'язками або відносинами, так само як при наявності однакових кадрів, будівель і цілей ефективність університету визначається прийнятою в ньому системою відносин (статут, положення, традиції). Все сказане залишається в силі і може бути застосовано до успішності міст і країн. Можна також додати, що в разі неефективності або помилок в системі міняти потрібно не людей – безглузде заняття – люди скрізь однакові, а структуру відносин між ними. We are not looking for the people who made mistakes, but the situations where the mistakes have been possible (Stanley Hyduke).

Квантові (фотонні) транзакції космологічного комп'ютингу на структурованій гравітацією матерії у формі зірок і планет, що мають адреси в просторі, розширюють горизонти життя у Всесвіті.

Квантова структура біологічного генома проникає в космос і створює нові форми життя в інших світах. Людство не приречене бути вічно на Землі. Гравітація за допомогою бозона Хіггса створює структури, а фотони або кванти – транзакції на них. Структурні елементи Всесвіту мають адреси, виражені через просторово-часові координати завдяки гравітації. Бозон Хіггса вчені сьогодні розглядають як розчин для склеювання цегли світобудови в матеріальні форми і структури. Гравітація – сила тяжіння, що виробляється масами взаємодіючих космічних тіл – створює відносний

структурний порядок у Всесвіті на короткому часовому інтервалі. Гравітації протидіє Великий вибух, сила якого служить причиною всіх змін у циклі еволюції Всесвіту до наступного Великого вибуху.

## 2.4 Висновки

1) Запропоновано модель та напрями сталого розвитку кіберфізичного комп'ютингу для прогнозування аттракторів в області IT-індустрії, що базуються на просторово-часовому аналізі технологічних процесів, які характеризуються трьома фазами розвитку і дають можливість прогнозувати майбутні тренди розвитку кіберкультури.

2) Запропоновано кіберкультуру мікро-макро-космо-комп'ютингу, яка формулює, пояснює і прогнозує сучасні технології моніторингу та управління процесами і явищами в фізичному, віртуальному та космологічному просторі.

2) Представлено вербальні і структурні визначення основних типів комп'ютингу, що базуються на сучасних трендах еволюційного розвитку кіберекосистеми планети.

3) Сформульовано універсальну модель MAT-комп'ютингу: <Memory, Address, Transactions>, яка використовує три компоненти для створення обчислювальної структури в технологічно прийнятному матеріальному середовищі.

4) Показано інформаційно-квантовий напрям експансії людини в космічний простір, а також можливість аналогічного проникнення неземних біотехнічних об'єктів в екосистему нашої планети.

5) Запропоновано модель комп'ютингу, яка визначає квазіоптимальні структури моніторингу та управління масштабованими процесами різної природи: технічними, біологічними, соціальними, віртуальними і космологічними.

# РОЗДІЛ 3

# СИНТЕЗ Q-ТЕСТІВ ПО BLACK BOX КУБІТНОМУ ОПИСУ

Представлено методи і апаратно-програмні реалізації безумовного паралельного синтезу тестів на основі булевих похідних для логічних схем в black box формі, заданих кубітним покриттям. Надано теоретичне обґрунтування застосування методів і оцінки їх ефективностей для широкого класу цифрових схем, що імплементуются в кристали програмовних логічних пристроїв. Запропоновано інноваційні методи взяття булевих похідних і дедуктивного моделювання несправностей для функціональних елементів, описаних кубітними покриттями. Описано структуру процесора моделювання несправностей і справної поведінки для вирішення завдань синтезу тестів і діагностування дефектів, який може бути імплементований в інфраструктуру SoC або хмарний сервіс. Наведено приклади, що ілюструють технологічність реалізації методів в якості компонентів BIST-Infrastructure і хмарних сервісів проектування і верифікації SoC.

## 3.1 Реалізація логічних функциональностей на елементах пам'яті

Поняття адресного виконання логічних операцій, реалізованих на елементах пам'яті LUT в програмованих логічних пристроях (PLD), дає потенційну можливість створювати на кристалі тільки адресний простір, максимально технологічний для вбудованого відновлення працездатності всіх компонентів, що беруть участь у формуванні функціональності [99-102]. Актуальність створення адресного простору для всіх компонентів підтверджується наступним розподілом логіки і пам'яті на кристалі, представленим на рис. 3.1, де після 2020 року на чіпі буде тільки один відсоток логіки і 99 відсотків пам'яті.

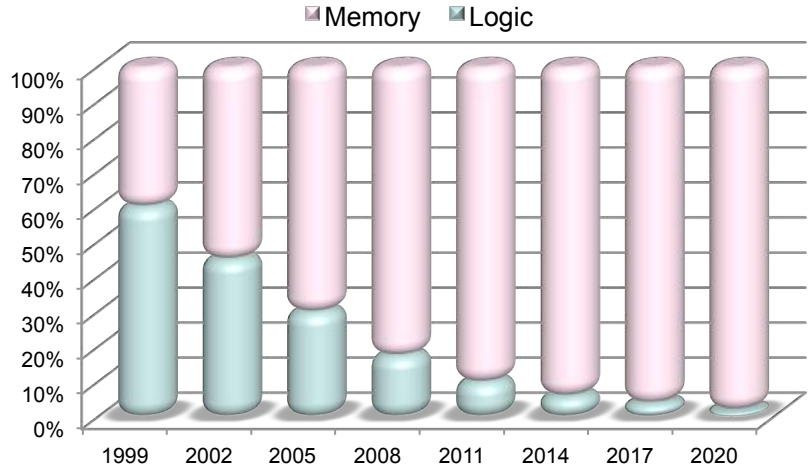



Рис. 3.1 – Розподіл пам'яті і логіки на кристалі

Тенденція до збільшення пам'яті тягне можливість вбудованого відновлення працездатності відмовних комірок за рахунок виділених додаткових ресурсів для їх ремонту (spare logic cells). Проблема автономного усунення дефектів (самовідновлення працездатності) логічних елементів пов'язана з відсутністю у них адрес. Але вирішити її можна, якщо зв'язки між елементами логіки зробити гнучкими за допомогою програми опису структури, розміщеної у пам'яті, яка з'єднає логічні компоненти в схему. Крім структури взаємодії елементів пам'ять повинна містити порядок їх обробки. У разі виникнення дефекту в одному з адресовних логічних елементів, система вбудованого тестування відновить його працездатність шляхом переадресації на завідомо справний аналог з ремонтного запасу. Просто вирішується проблема підвищення якості та надійності цифрових систем на кристалах шляхом створення інфраструктури вбудованого тестування, діагностування, оптимізації та відновлення працездатності за рахунок апаратної надлишковості і зменшення швидкодії виконання функціональних операцій [101-104].

Як наочний приклад, архітектура FPGA Family Xilinx® UltraScaleTM Architecture Configurable Logic Block (the first ASIC-class All Programmable architecture) належить уже до класу ASIC і забезпечує продуктивність реалізованих смарт-функциональностей на рівні сотень гігабіт за секунду.

Вона виконана у вигляді 3D-on-3D об'ємному кристалі по технології 20 nm на основі FinFET транзисторів і працює як мультипроцесорна SoC (MPSoC), забезпечуючи мережеву продуктивність 1+ Tb/s обміну даними. [Www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf]. На початку 2000-х років співвідношення продуктивності процесорної логіки і пам'яті було, як 100:1. В даний час ситуація поступово вирівнюється в сторону зменшення відмінностей, як 10:1, за рахунок підвищення швидкодії транзкцій на пам'яті. Це є істотним аргументом для прискорення процесу «меморизації» – реалізації логіки на пам'яті з метою підвищення надійності цифрових систем на кристалах за рахунок часткової втрати продуктивності. В кінцевому рахунку пам'ять і логіка мають загальну атомарну межа досконалості, коли біт пам'яті буде ототожнюватися зі спіном або орбітою електрона. Швидкодія межатомарних транзакцій на основі обміну квантами стане сумірною зі швидкістю світла. Класична транзисторна логіка, керована потоком заряджених частинок, в межі – одним електроном, має ту ж саму межу досконалості – зміна станів електрона під впливом кванта. Таким чином, перехід електронної індустрії на only-memory-computing (Memory-Address-Transaction) неминучий. Звідси випливає висновок: своєчасне створення теорії синтезу та аналізу цифрових обчислювальних пристроїв на основі МАТ-комп'ютингу може принести ринку принципово нові можливості для вирішення проблем надійності, швидкодії та енергозбереження.

Мета – істотне підвищення швидкодії синтезу тестів і дедуктивної верифікації для black box функциональностей логічних компонентів за рахунок використання компактних описів у формі кубітних покриттів і паралельного виконання мінімальної кількості регістрових логічних операцій (shift, or, not, nxor).

Задачі дослідження пов'язані з розробкою:

1) Методу генерації тестів для black box функциональностей логічних схем на основі використання кубітних покриттів і паралельного виконання регістрових логічних операцій (shift, or, not, nxor).

2) Методу взяття булевих похідних для синтезу тестів на основі використання кубітних покриттів.

3) Методу синтезу тестів на основі використання булевих похідних, представлених векторами в форматі кубітних покриттів.

4) Методу дедуктивного моделювання несправностей для функціональних елементів, представлених векторами кубітних покриттів.

5) Процесору моделювання несправностей і справної поведінки на основі кубітного опису функциональностей, який інтегрує розроблені методи в хмарний або SoC-інфраструктурний сервіс тестування.

6) Тестуванням і верифікацією методів синтезу тестів і моделювання несправностей шляхом паралельного виконання регітрових логічних операцій над кубітними описами прикладів логічних схем.

Сутність дослідження – розробка методів генерування вхідних тестових послідовностей і оцінки їх якості для функціональних логічних компонентів шляхом паралельного виконання регістрових логічних операцій (shift, or, not, nxor) над кубітним покриттям і його похідними в структурі процесора кубітного моделювання.

Тестування і діагностування запам'ятовувальних пристроїв реалізується за допомогою спеціальних алгоритмів, що генерують тести: марш – нуль і одиниця, що біжать, логарифмічного поділу [100-104]. Якщо елементи пам'яті виконують функції логічних схем (reusable logic), то для них слід генерувати тести перевірки функциональностей, не прив'язані до фізики процесів зберігання даних. Тому далі розглядається метод синтезу тестів для функціональних схем, представлених у формі black box, опис яких задається кубітним вектором [105-106].

Ринкова привабливість застосування квантових методів обчислень при створенні компьютингових структур в кіберпросторі визначається використанням кубітних моделей даних, орієнтованих на паралельне вирішення задач проектування, тестування, дискретної оптимізації [102, 103], завдяки збільшенню витрат пам'яті. Не вдаючись в деталі фізичних основ квантової механіки, що стосуються недетермінованої взаємодії атомних часток [92, 94], далі використовується поняття кубіта як двійкового вектора для спільного і одночасного завдання булеана станів в дискретної області кіберпростору на основі лінійної суперпозиції унітарних кодів, орієнтованих на паралельне виконання операцій. У теорії квантових обчислень, що швидко розвивається, вектори станів утворюють квантовий регістр з n кубітів, які формують унітарний або Гільбертів [95] простір H, розмірність якого має ступеневу залежність від кількості кубітів $\mathrm{Dim}\ \mathrm{H} = 2^{\mathrm{n}}$.

Мотивація нового підходу для проектування компьютингових систем обумовлена появою хмарних сервісів в рамках нової кіберкультури Internet of Things, яка являє собою спеціалізовані і розосереджені в просторі системи, реалізовані в апаратурі або в програмному продукті. Будь-який компонент функціональності, так само як і структура системи, може бути представлений векторною формою, впорядкованою за адресами, таблиці істинності, що реалізується за допомогою пам'яті. Логічні функції в традиційному виконанні reusable logic не розглядаються. Від цього частково зменшується швидкодія, але, з огляду на те, що 94% SoC-кристалу становить пам'ять [100, 101], решта 6% будуть імплементуватися в пам'ять, що не критично для більшості хмарних сервісів. Практично для створення ефективних компьютингових структур слід використовувати теорію, що базується на обчислювальних компонентах високого рівня абстракції: адресовна пам'ять і транзакція.

Згідно квантової теорії електрон може бути в двох місцях одночасно. З цього твердження випливає, що точка простору може одночасно містити два електрони або два стійких стани. Квант (кубіт) тут розглядається як

дискретний стан точки в кіберпросторі, який визначається суперпозицією декількох примітивів універсуму.

Особливість організації даних в класичному комп'ютері полягає в адресації біта, байта або іншого компонента. Адресовність створює проблему обробки асоціації неадресовних елементів множини, які не мають порядку за визначенням. Рішенням може бути процесор, де образ універсуму з n унітарно кодованих примітивів використовує суперпозицію для формування булеана $|B(A)| = 2^n$ всіх можливих станів [38, 76].

Існує аналогія векторного представлення булеана з кубітом квантового комп'ютера. Квантовому кубіту зберігання інформації в квантовому комп'ютері [92, 94], що дозволяє суперпозицію, можна поставити у взаємно однозначну відповідність булеан станів, який формує алфавіт Кантора. Тут примітиви алфавіту унітарно кодуються векторами: 0=(10) і 1=(01). Коди інших символів є похідними по операції суперпозиції (10)∨(01) = (11) і перетину (10) ∧ (01) = (00), що формує булеан: {(10),(01),(11),(00)} [38, 95, 76].

Коректність використання прикметника «кубітна» для моделей цифрових пристроїв базується на порівнянні лінійної і булевої алгебри Кантора з алфавітом $A^k = \{0,1,X,\emptyset\}$. Тут перші два символи – примітиви. Третій визначається суперпозицією: $X = 0 \cup 1$. В гільбертовому просторі лінійна алгебра оперує примітивами $\alpha|0\rangle, \beta|1\rangle$ кубіта, третій символ також є суперпозиція двох складових: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. В лінійній алгебрі немає символу, відповідного порожній множині, він є похідним від функції, зворотної по відношенню до суперпозиції. В теорії множин такою операцією є перетин, яке дає порожня множина на примітивах: $\emptyset = 0 \cap 1$. В гільбертовому просторі такою операцією є скалярний добуток або функція Дірака $\langle\alpha|b\rangle$ [110], яка має геометричну інтерпретацію: $\langle\alpha|\beta\rangle = |\alpha|\times|\beta|\times\cos\angle(\alpha,\beta)$. Якщо проекції a і b вектора квантового стану ортогональні, виходить: $\langle\alpha|\beta\rangle = |\alpha|\times|\beta|\times\cos\angle(\alpha,\beta) = |\alpha|\times|\beta|\times\cos\angle 90° = 0$. Скалярний добуток ортогональних векторів дорівнює нулю, що є аналогом символу порожньої

множини в алгебрі Кантора. Таблиця відповідності алгебри множин та лінійної алгебри, наведена нижче, підтверджує властивості ізоморфізму між символами булеана і станами кубіт-вектора:

| Boolean $A^k =$ | 0 | 1 | $X = 0 \cup 1$ | $\varnothing = 0 \cap 1$ |
|---|---|---|---|---|
| Qubit $\lvert\psi\rangle =$ | $\lvert 0\rangle$ | $\lvert 1\rangle$ | $\alpha\lvert 0\rangle + \beta\lvert 1\rangle$ | $\alpha\lvert 0\rangle \lvert \beta\lvert 1\rangle$ |

Отже, структуру даних «булеан» можна розглядати як детермінований образ квантового кубіта в алгебрі логіки, елементи якої унітарно кодуються двійковими векторами і мають властивості суперпозиції, паралелізму і переплутування. Це дає можливість використовувати пропоновані кубітні моделі для підвищення швидкодії аналізу цифрових пристроїв на класичних обчислювачах, а також без модифікації – в квантових комп'ютерах, які з'являться через кілька років на ринку електроніки.

Квантовий опис цифрових функціональних елементів. Кубіт (n-кубіт) є векторною формою унітарного кодування універсуму з n примітивів для завдання булеана станів за допомогою двійкових змінних. Якщо n=2, то 2-кубіт задає 16 станів за допомогою чотирьох змінних, при n=1, кубіт задає чотири стану на универсумі з двох примітивів (10) і (01) за допомогою двох двійкових змінних (00,01,10,11) [76]. При цьому допускається суперпозиція у векторі $2^n$ станів, позначених примітивами. Синонімом кубіта при завданні двійкового вектора логічної функції є Q-покриття (Q-вектор) [77, 78] як уніфікована векторна форма суперпозиційного завдання вихідних станів, відповідних адресним кодами вхідних змінних функціонального елемента. Формат структурного кубітного компонента цифрової схеми $Q^* = (X, Q, Y)$ містить інтерфейс (вхідні та вихідну змінні), а також кубіт-вектор Q, що задає функцію $Y = Q(X)$, розмірність якого визначається ступеневою функцією від кількості вхідних ліній $k = 2^n$. Новизна кубітної форми полягає в заміні невпорядкованих по рядках таблиць істинності функціональних елементів векторами упорядкованих станів виходів. Наприклад, якщо функціональний

примітив має двійкову таблицю, то йому можна поставити у відповідність кубіт або Q-покриття: Q = (1110):

$$C = \begin{array}{ccc} X_1 & X_2 & Y \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array} \rightarrow Q = (1110)$$

Таким чином, дизрапторна ідея, покладена в основу досліджень, полягає в заміні множини вхід-вихідних відповідностей таблиці істинності кубітним вектором адресовних вихідних станів, рис. 3.2.

Примітивізм і компактність кубітної векторної форми (Q-coverage) диктує застосування тільки простих паралельних регістрових операцій над його вмістом: (not, shift, or, and, xor) для вирішення всіх завдань синтезу та аналізу цифрових виробів.

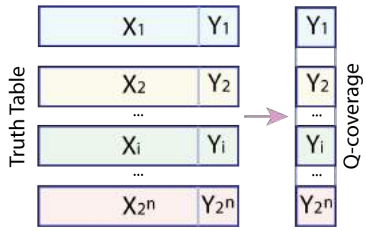



Рис. 3.2 – Таблиця істинності та кубітне покриття

3.2 Кубітний метод синтезу тестів

Пропонується метод синтезу тестів, що використовує кубітні вектори або Q-покриття функціональних примітивів цифрових пристроїв, який характеризується компактністю опису даних і паралелізмом виконання логічних операцій.

Q-покриття є векторна форма опису поведінки цифрового пристрою, де кожен розряд має адресу, що формується двійковими станами його вхідних змінних:

$$Q = (Q_1, Q_2, ..., Q_i, ..., Q_n),\ Q_i = \{0,1\},\ Q_i = Q(i),\ i = (X_1 X_2, ..., X_i, ..., X_n).$$

Q-тест є векторною формою неявного завдання тестових послідовностей цифрового пристрою, де координати вектора формують впорядковану послідовність двійкових наборів, що подаються на вхідні змінні за правилом:

$$Q_i = Q(i) = \begin{cases} 1 \to (X_1 X_2, ..., X_i, ..., X_n) = i, \\ 0 \to (X_1 X_2, ..., X_i, ..., X_n) = \varnothing. \end{cases}$$

Іншими словами, якщо координата вектора Q(i)=1, то тестовий набір, складений з двійкових розрядів, які формують десяткову адресу i, подається на входи пристрою. В іншому випадку, при нульовому значенні координати Q(i)=0, такий тестовий набір відсутній.

Природно, що розмірності Q-покриття і Q-тесту завжди однакові, що робить можливим виконувати паралельний аналіз їх спільної взаємодії при синтезі тестів для функціональних елементів за правилом [38]:

$$F \oplus Q \oplus T = 0, \;\; T = Q \oplus F.$$

Приклад 1. Необхідно згенерувати тест перевірки одиночних константних несправностей для логічного елемента 2and, заданого таблицею істинності. Для цього задається таблиця впливу одиночних константних несправностей вхідних змінних, з числом кубів, що дорівнює кількості входів. Кожен куб несправності має одиницю на координаті вхідної змінної і на виході, а решта координат дорівнюють нулю. Це означає, що існує залежність: зміна вхідної змінної повинна викликати зміну стану виходу.

Алгоритм синтезу тесту по таблиці істинності [38]:

1) На першому етапі виконується покоординатна xor-операція між таблицею істинності і кожним рядком матриці несправностей, яка генерує дві таблиці, по кількості вхідних змінних, що містять рядки-кандидати в тест:

$$\begin{bmatrix} X_1 & X_2 & Y \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \oplus \begin{bmatrix} F_1 & F_2 & F_Y \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} X_1 & X_2 & \overline{Y}_1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \vee \begin{bmatrix} X_1 & X_2 & \overline{Y}_2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

2) Потім рядки двох останніх таблиць упорядковуються за правилом зростання двійкових кодів вхідних змінних:

$$\begin{bmatrix} X_1 & X_2 & \overline{Y}_1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \oplus \begin{bmatrix} X_1 & X_2 & \overline{Y}_2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} X_1 & X_2 & Y_1' \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} X_1 & X_2 & Y_2' \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

3) Після цього виконується порівняння отриманих станів виходів ($Y_1'Y_2'$) з вектором значень виходів вихідної таблиці істинності Y за правилом операції еквівалентності (nxor) – якщо значення однойменних координат двох кубітів рівні, то результат порівняння дорівнює 1, в іншому випадку - 0:

$$\begin{bmatrix} Y \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \overline{\oplus} \begin{bmatrix} Y_1' \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \vee \begin{bmatrix} Y_2' \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} Y_1^t \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} Y_2^t \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

4) На останньому кроці виконується операція логічного додавання отриманих результатів порівняння, яка дає Q-тест або T-вектор, одиничні координати якого ідентифікують тільки ті двійкові вхідні послідовності, які слід подавати на входи цифрового пристрою для його перевірки:

$$\begin{bmatrix} Y_1^t \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \vee \begin{bmatrix} Y_2^t \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} T \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} X_1 & X_2 & Y \\ . & . & . \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

В даному прикладі Q-тест містить три одиниці 0111, тому відповідні йому вхідні набори представлені трьома векторами: 010, 100, 111.

Приклад 2 ілюструє роботу алгоритму [38] синтезу тесту по таблиці істинності для функціонального елемента, що має три вхідних змінних. Тут також фігурує таблиця істинності, в якій останній стовпець Y є кубіт-вектором. Стовпці ($F_1F_2F_3F_Y$) формують матрицю несправностей, сутність кожного рядка якої – одновимірна активізація входу на вихід. Стовпці ($Y_1'Y_2'Y_3'$) – отримані в результаті xor-взаємодії таблиці істинності і матриці несправностей, яка визначає поведінку функціональності при внесенні на кожен вхід несправності, інверсної по відношенню до її справної поведінки.

Стовпці ($Y_1^t Y_2^t Y_3^t$) отримані шляхом покоординатного порівняння вектора вихідних станів вхідної таблиці істинності з наборами ($Y_1' Y_2' Y_3'$), отриманими на попередньому етапі. Вектор Т – логічно об'єднує набори ($Y_1^t Y_2^t Y_3^t$) в тест для заданої функціональності:

| $X_1$ | $X_2$ | $X_3$ | Y | $F_1$ | $F_2$ | $F_3$ | $F_Y$ | $Y_1'$ | $Y_2'$ | $Y_3'$ | $Y_1^t$ | $Y_2^t$ | $Y_3^t$ | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | | | | | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | | | | | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | | | | | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | | | | | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | | | | | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

Останній стовпець являє собою тест перевірки несправностей зовнішніх вхідних і вихідних змінних функціонального елемента Т=(10111111). Тест-вектор задає такі вхідні набори, які слід подати на входи функціонального елемента: 0001 0100, 0111 1000, 1011, 1101, 1110, щоб перевірити всі дефекти заданого класу.

Формальний Q-алгоритм синтезу тестів для константних несправностей функціонального примітиву на основі використання Q-покриття містить такі пункти:

1) Інвертування Q-покриття: $Q_i = \overline{Q}_i, i = \overline{1, 2^n}$.

2) Упорядкування інверсного Q-покриття для кожної з n вхідних змінних $Q_j = S_j(\overline{Q}), j = \overline{1, n}$. Дана процедура зводиться до виконання операцій логічного зсуву (shift) на кубітному векторі, кожна з яких має власний алгоритм для даної вхідної змінної, що ілюструється наступною схемою для трьох входів, рис. 3.3.

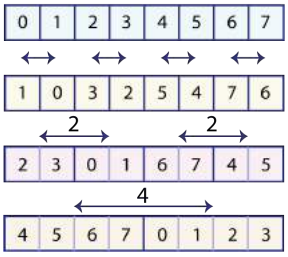



Рис. 3.3 – Операції зустрічного зсуву на регістрах

Тут другий рядок адресними індексами задає процедуру обміну даними між сусідніми координатами кубіт-вектора шляхом зустрічного зсуву вмісту двох сусідніх координат, що формує тест-вектор для першої вхідної змінної. Третій рядок ілюструє обмін даними між сусідніми парами координат кубіт-вектора шляхом зустрічного зсуву їх вмісту, що формує тест-вектор для другої вхідної змінної. Четвертий рядок задає обмін даними між сусідніми тетрадами координат кубіт-вектора шляхом зустрічного зсуву їх вмісту, що формує тест-вектор для третьої вхідної змінної.

3) Отримання Т-векторів для кожної з n вхідних змінних шляхом порівняння з вихідним кубітним покриттям функціонального елемента:

$$T_j = Q \overline{\oplus} \overline{Q}_j, j = \overline{1, n}.$$

4) Отримання Q-тесту функціональності шляхом логічного об'єднання Т-векторів для кожної вхідної змінної:

$$T = \vee_{j=1}^{n} T_j.$$

Інтегрально алгоритм безумовного синтезу тестів для функціональних елементів, заданих кубітнимі покриттями компактно може бути записаний у вигляді такої формули:

$$T = \vee_{j=1}^{n} [Q \overline{\oplus}_{j=1}^{n} S_j(\overline{Q})].$$

Обчислювальна складність безумовного алгоритму синтезу тестів на основі послідовного використання регістрових логічних операцій (not - shift - nxor - or) для функціонального елемента, описаного кубіт-вектором, представлена наступним виразом:

$$q = n + n \times 2^n + n + n = n(2^n + 3).$$

Тут n – кількість змінних; перший доданок визначає обчислювальну складність операції 1) інверсії; другий – 2) логічний зсув для перестановки станів координат кубіт-вектора відносно кожної з n вхідних змінних; третій – 3) nxor-порівняння отриманих тест-векторів з вихідним кубіт-покриттям функціонального елемента; четвертий – 4) об'єднання тест-векторів для вхідних змінних.

Абстрагуючись від поняття таблиці істинності, далі пропонується формальний безумовний алгоритм кубітного синтезу тестів для функціональних примітивів на основі Q-покриття, стосовно раніше розглянутого прикладу:

1) Інвертування всіх розрядів Q-покриття функціонального елемента, що має три вхідних змінних:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Q | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $\overline{Q}$ | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

2) Логічний зсув номерів розрядів інвертованого кубітного вектора відповідно до послідовностів номерів:

$$Q_1(X_1) = 45670123, Q_2(X_2) = 23016745, Q_3(X_3) = 10325476.$$

Тут діє просте логічне правило: для першої вхідної (молодшої) змінної виконується паралельний обмін даними між сусідніми координатами, для другої вхідної змінної реалізується обмін даними але вже між сусідніми парами координат, для третьої змінної виконується обмін даними між сусідніми тетрадами координат вектора адрес.

Узагальнення операцій зсуву для функціональності, що містить 4 змінних, представлено на рис. 3.4. Збільшення кількості змінних принципово не змінює суті зсуву даних: зустрічний зсув двох бітів, пар бітів, тетрад бітів, вісімок, ....

Процедура зустрічного зсуву даних в інверсному кубітному векторі є найбільш часовитратною в алгоритмі синтезу тестів для функціональних

елементів. Тому її швидкодія матиме максимальне значення при апаратній реалізації зсувних операцій.

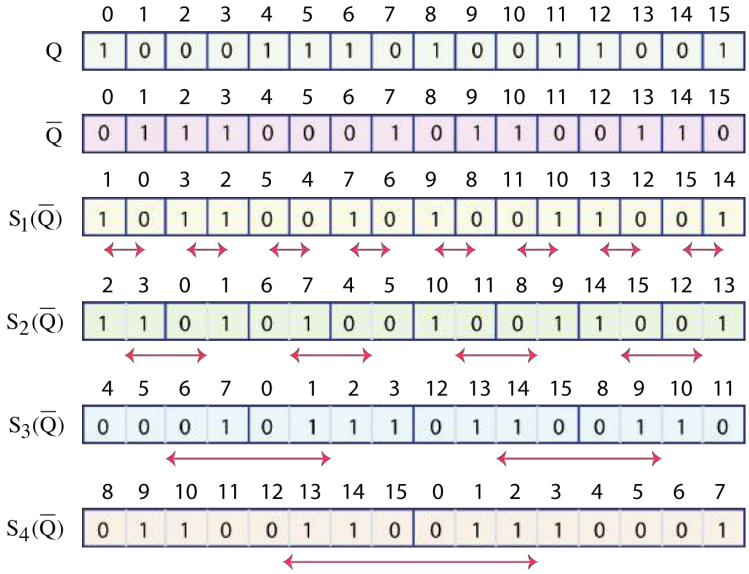



Рис. 3.4 – Операції зсуву для чотирьох-входової функціональності

Процедурна реалізація обміну номерами для формування тест-векторів перевірки несправностей вхідних змінних, стосовно наведеного вище прикладу, має такий вигляд:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Q | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $\overline{Q}_1$ | 1 | 0 | 3 | 2 | 5 | 4 | 7 | 6 |
| $\overline{Q}_2$ | 2 | 3 | 0 | 1 | 6 | 7 | 4 | 5 |
| $\overline{Q}_3$ | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |

Цікаво, що для першої змінної існує проста формула перенумерації комірок кубіт-вектора для синтезу тестів: $j = j + (-1)^j$. Для інших змінних такої простої залежності визначити поки не вдалося.

З урахуванням введених правил зустрічного зсуву комірок кубіт-вектора реалізація даного пункту алгоритму для Q = 11010110 представлена таблицею:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Q = | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| $\overline{Q}$ = | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| $\overline{Q}_1$ = | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $\overline{Q}_2$ = | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| $\overline{Q}_3$ = | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |

Таким чином, кожна змінна визначає розбиття кубіт-вектора на групи упорядкованих послідовностей. Перша (молодша) змінна створює зміни станів за правилом: додати 1 до поточної парної адреси комірки: j=j+1, відняти 1 від поточної непарної адреси комірки: j=j-1. Для другої змінної розглядаються вже пари комірок, які обробляються за правилом: додати 2 до поточної парної адреси пари комірок: j=j+2, відняти 2 від поточної непарної адреси пари комірок: j=j-2. Для третьої змінної розглядаються вже тетради комірок, які обробляються за правилом: додати 4 до поточної парної адреси тетради комірок: j=j+4, відняти 4 від поточної непарної адреси тетради комірок: j=j-4.

3) Порівняння за допомогою операції еквівалентності отриманих, в даному випадку, трьох інвертованих і переупорядкованих Q-векторів з вихідним Q-покриттям функціонального елемента:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $Q$ | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| $\overline{Q}_1$ | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| $S_1(\overline{Q})$ | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $S_2(\overline{Q})$ | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| $S_3(\overline{Q})$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| $Q\overline{\oplus}S_1(\overline{Q})$ | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| $Q\overline{\oplus}S_2(\overline{Q})$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $Q\overline{\oplus}S_3(\overline{Q})$ | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

4) Диз'юнкція отриманих трьох векторів формує Q-тест, одиничні координати яких визначають тестові набори для перевірки всіх одиночних константних несправностей зовнішніх входів і виходів.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $Q$ | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| $\overline{Q}$ | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| $S_1(\overline{Q})$ | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $S_2(\overline{Q})$ | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| $S_3(\overline{Q})$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| $T_1 = Q\overline{\oplus}S_1(\overline{Q})$ | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| $T_2 = Q\overline{\oplus}S_2(\overline{Q})$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $T_3 = Q\overline{\oplus}S_3(\overline{Q})$ | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| $T = T_1 \vee T_2 \vee T_3$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

З огляду на істотність зустрічних регістрових зсувів, нижче пропонується універсальний алгоритм отримання перестановок кубітних фрагментів в залежності від номера вхідної змінної.

Послідовний алгоритм зустрічного зсуву даних в кубітних покриттях для отримання Q-тестів вхідних змінних має три вкладених цикли:

1) Завдання i-номера вхідної змінної або кроку $2^i$ для зустрічного зсуву даних в кубите: $i = \overline{1,n}$.

2) Формування циклу обробки кубіта з уже заданим кроком $2^i$ (2,4,8,16 ...), кратним ступеня двійки: $j = \overline{0,2^n - 1,2^i}$. Залежно від номера вхідної змінної і формуються такі послідовності індексу j=f(i): [(0,2,4,6 ...), (0,4,8,12 ...), (0,8,16,32 ... )].

3) Завдання циклу зустрічного зсуву даних для пари сусідніх груп: $t = j, j + 2^{i-1} - 1$. Значення індексу t=f(i,j): обробка кубіта першої змінної – (0,0; 2,2; 4,4 ...), другої змінної – (0,1; 4,5; 8,9 ...), третьої змінної – (0,3; 8,11; 16,19 ...). Виконання операцій зсуву за допомогою використання буферного регістру B (рис. 3.5):

$$(B_t = Q_{i,t+j}) \rightarrow (Q_{i,t+j} = Q_{i,t}) \rightarrow (Q_{i,t} = B_t)$$

Кінець алгоритму зсуву даних в регістрах.

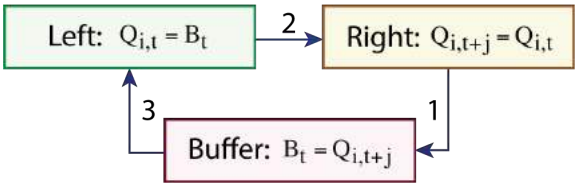



Рис. 3.5 – Зустрічний зсув сусідніх частин кубіта

Обчислювальна складність послідовного алгоритму для обробки кубітного покриття функціональності, що має n вхідних змінних, дорівнює $q = 3n2^n$.

Приклад 3. Побудувати тест для перевірки функціональності, заданої логічним рівнянням: $Y = \overline{X}_1\overline{X}_2 \vee \overline{X}_1 X_2$. Результати виконання алгоритму синтезу тесту по заданому кубіт-вектору представлено в наступному вигляді:

| | |
|---|---|
| Q | 0 0 1 1 |
| $\overline{Q}_1$ | 1 1 0 0 |
| $S_1(\overline{Q})$ | 0 0 1 1 |
| $S_2(\overline{Q})$ | 1 1 0 0 |
| $T_1 = Q \overline{\oplus} S_1(\overline{Q})$ | 1 1 1 1 |
| $T_2 = Q \overline{\oplus} S_2(\overline{Q})$ | 0 0 0 0 |
| $T = T_1 \vee T_2$ | 1 1 1 1 |

Результуючий тест містить 4 вхідних послідовності, кожна з яких перевіряє несправності для вхідної змінної $X_1$. Однак тест для змінної $X_2$ не існує, оскільки $T_2$=0000. Це означає, що змінна не може бути перевірена, а значить вона не є суттєвою. Дійсно, перетворення вихідної функції

$$Y = \overline{X}_1\overline{X}_2 \vee \overline{X}_1X_2 = \overline{X}_1(\overline{X}_2 \vee X_2) = \overline{X}_1$$

в сторону мінімізації виключає змінну $X_2$, як надлишкову або несуттєву. Таким чином, метод генерації тестів на основі кубітного покриття дає додаткову можливість визначати істотність змінних і мінімізувати (зменшувати розмірність) Q-вектор шляхом зменшення кількості змінних.

Приклад 4. Побудувати тест для перевірки функціональності, заданої логічним рівнянням: $Y = \overline{X}_1X_2 \vee X_1\overline{X}_2$. Результати виконання алгоритму синтезу тесту по заданому кубіт-вектору на основі послідовного виконання чотирьох регістрових логічних операцій (not, shift, nxor, or) представлені у вигляді:

| | |
|---|---|
| Q | 0 1 1 0 |
| $\overline{Q}_1$ | 1 0 0 1 |
| $S_1(\overline{Q})$ | 0 1 1 0 |
| $S_2(\overline{Q})$ | 0 1 1 0 |
| $T_1 = Q \overline{\oplus} S_1(\overline{Q})$ | 1 1 1 1 |
| $T_2 = Q \overline{\oplus} S_2(\overline{Q})$ | 1 1 1 1 |
| $T = T_1 \vee T_2$ | 1 1 1 1 |

Отриманий тест містить 4 вхідних послідовності, кожна з яких перевіряє несправності для вхідних змінних $X_1, X_2$. Таким чином, метод генерації тестів на основі кубітного покриття дозволяє згенерувати тест для функціонального компонента, але не надає інструменту зробити його мінімальним.

Приклад 5. Синтезувати тест для перевірки функціональності, заданої логічним рівнянням: $Y = \overline{X}_1 X_2 \overline{X}_3 \vee X_1 \overline{X}_2 \overline{X}_3 \vee X_1 X_2 X_3 \vee \overline{X}_1 X_2 X_3$. Результати виконання алгоритму синтезу тесту по Q-вектору на основі послідовного виконання чотирьох регістрових логічних операцій (not, shift, nxor, or) представлені у вигляді:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $Q$ | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| $\overline{Q}$ | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| $S_1(\overline{Q})$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| $S_2(\overline{Q})$ | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| $S_3(\overline{Q})$ | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| $T_1 = Q \overline{\oplus} S_1(\overline{Q})$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $T_2 = Q \overline{\oplus} S_2(\overline{Q})$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $T_3 = Q \overline{\oplus} S_3(\overline{Q})$ | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $T = T_1 \vee T_2 \vee T_3$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

На рис. 3.6 представлений секвенсор синтезу тестів для функціональних елементів, заданих кубітними покриттями.

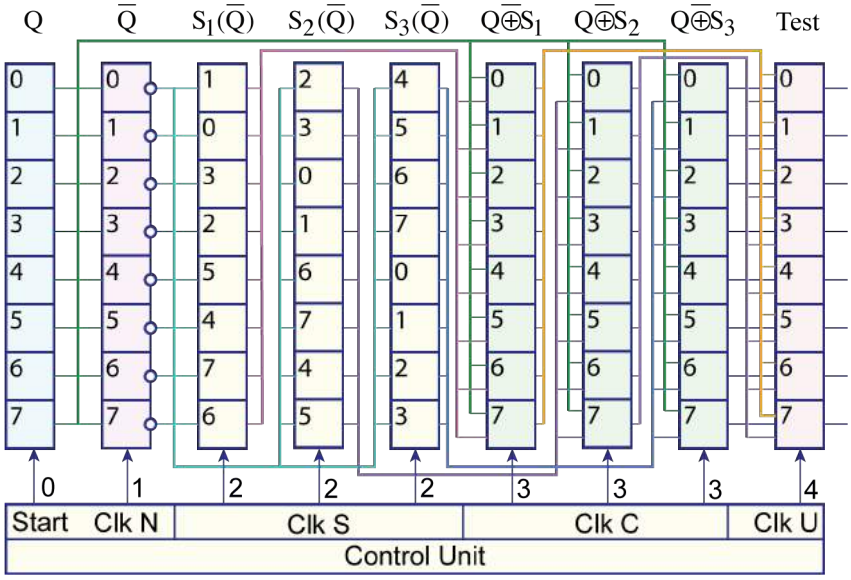



Рис. 3.6 – Секвенсор синтезу тестів

Він містить модуль керування, який розподіляє чотири синхроімпульси, створюючи цикл генерації тесту, що передбачає виконання 4 паралельних операцій послідовно: 0) Початкова стартова операція, що ініціюється сигналом Start, передбачає завантаження в регістр кубітного покриття. 1) Потім синхроімпульс Clk N активує виконання операції інверсії Not над вмістом регістру кубітного покриття. 2) Синхросигнал Clk S активує

виконання операцій зустрічного зсуву над вмістом, в даному випадку трьох, регістрів, і отримання таких результатів: $S_1$ (not Q), $S_2$ (not Q), $S_3$ (not Q). 3) Потім синхросигнал Clk C ініціює виконання паралельних операцій порівняння (в даному випадку для трьох регістрів) отриманих кандидатів в тест з початковим кубітним покриттям: nxor (notQ, $S_1$), nxor (notQ, $S_2$), nxor (notQ, $S_3$). 4) Синхросигнал Clk U активує виконання or-операції над кандидатами в тест, у даному випадку реалізацію логічного об'єднання вмісту трьох регістрів:

$$T = \mathrm{or}[\mathrm{nxor}(\mathrm{notQ}, S_1), \mathrm{nxor}(\mathrm{notQ}, S_2), \mathrm{nxor}(\mathrm{notQ}, S_3)].$$

Якщо не економити на апаратурі, то швидкодію тестового генератора можна довести до чотирьох автоматних тактів q=4 паралельного виконання логічних регістрових операцій, що дає можливість вбудовувати секвенсор у BIST-інфраструктуру цифрових систем на кристалах для online тестування функціональних елементів. При цьому сумарна кількість $2^n$ –розрядних регістрів дорівнюватиме N(n)=1+1+n+n+1=(2n+3), а загальний обсяг регістрової пам'яті в бітах буде дорівнює N(R)=(2n+3) $2^n$, де n – кількість вхідних змінних функціонального елемента.

### 3.3 Обчислення булевих похідних для Q-синтезу тестів

Розглядається метод взяття булевих похідних по кубітному покриттю для створення умов активізації вхідних змінних при синтезі кубітних тестів. Проводиться аналогія між двома формами булевих функцій для взяття похідних: аналітичної та векторної. Дослідження методу виконується на прикладах логічних функцій:

1) $f(x) = x_1 \vee x_1\bar{x}_2$. 2) $f(x) = x_1x_2 \vee \bar{x}_1x_3$. 3) $f(x) = \bar{x}_2\bar{x}_3 \vee x_1x_2x_3$.

Питання, що підлягають вирішенню: 1) Визначення похідних першого порядку за аналітичною і кубітною формами завдання логічної функції. 2) Верифікація отриманих умов активізації шляхом їх моделювання на одній з

форм опису функціональності. 3) Синтез тестів активізації змінних логічної функції на основі обчислення похідних.

Приклад 7. Визначити всі похідні першого порядку по кубітній формі логічної функції $f(x) = x_1 \vee x_1\bar{x}_2$.

Застосування формули обчислення за аналітичним виразом

$$\frac{df(x_1,x_2,...,x_i,...,x_n)}{dx_i} = f(x_1,x_2,...,x_i = 0,...,x_n) \oplus f(x_1,x_2,...,x_i = 1,...,x_n)$$

визначає булеву похідну першого порядку як суму по модулю два нульової та одиничної залишкових функцій.

Для даної функції виходить:

$$\frac{df(x_1,x_2)}{dx_1} = f(0,x_2) \oplus f(1,x_2) =$$
$$= (0 \vee 0\bar{x}_2) \oplus (1 \vee 1\bar{x}_2) = 0 \oplus 1 = 1.$$

$$\frac{df(x_1,x_2)}{dx_2} = f(x_1,0) \oplus f(x_1,1) =$$
$$= (x_1 \vee x_1 \cdot \bar{0}) \oplus (x_1 \vee x_1 \cdot \bar{1}) =$$
$$= (x_1 \vee x_1 \cdot 1) \oplus (x_1 \vee x_1 \cdot 0) =$$
$$= (x_1 \vee x_1) \oplus (x_1 \vee 0) = x_1 \oplus x_1 = 0.$$

Нульове значення похідної означає відсутність умов активізації для змінної $x_2$, що дає підстави вважати її несуттєвою, отже, прибрати з числа змінних, які формують функціональність. Далі пропонуються аналогічні перетворення для кубітного покриття функції, заданого вектором:

| $X_1$ | $X_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

$= (0011).$

Похідна по таблиці істинності може обчислюватися шляхом почергового завдання всіх нулів і одиниць в координатах стовпців, відповідних кожній змінній:

| $X_1$ | $X_2$ | Y | $Y_1^0$ | $Y_1^1$ | $Y_1'$ | $Y_2^0$ | $Y_2^1$ | $Y_2'$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

Таким чином, похідні за першою і другою змінною, записані в форматі кубітного покриття, дорівнюють: 1111 та 0000. Це означає, що похідна за першою змінною дорівнює 1, а за другою – дорівнює 0.

Однак такий результат можна отримати більш формально і технологічно, не розглядаючи вхідні набори таблиці істинності, використовуючи тільки логічні операції зустрічного зсуву і подальшої xor-операції над розрядами кубітного покриття {a,b}=a⊕b, де a,b – сусідні підвектори кубіта Q=(a,b):

| Y | $Y_1'$ | $Y_2'$ |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |
| 1 | 1 | 0 |

Інакше, для першої змінної необхідно xor-скласти, зсунуті один щодо одного дві половинки першого стовпчика, а результат записати в обидві зсувані симетричні області {a,b}=a⊕b, (11,11)=00⊕11. Для другої змінної слід розглядати пари сусідніх координат стовпчика а загальний результат записувати в кожні зсувані симетричні області-біти {a,b}=a⊕b: (0,0)=0⊕0=0, (0,0)=1⊕1=0. Таким чином, результат підсумовування буде загальним для кожної пари взаємодіючих підвекторів, розмірність яких визначається номером даної змінної, від 0 до $2^n - 1$.

Приклад 8. Визначити всі похідні першого порядку по аналітичній формі логічної функції $f(x) = x_1x_2 \vee \bar{x}_1x_3$. Для даної функції виконуються наступні обчислення:

$$\frac{df(x_1,x_2,x_3)}{dx_1} = f(0,x_2,x_3) \oplus f(1,x_2,x_3) =$$
$$= (0 \cdot x_2 \vee \bar{0} \cdot x_3) \oplus (1 \cdot x_2 \vee \bar{1} \cdot x_3) =$$
$$= (0 \vee 1 \cdot x_3) \oplus (x_2 \vee 0 \cdot x_3) =$$
$$= x_3 \oplus x_2 = x_2\bar{x}_3 \vee \bar{x}_2x_3.$$

$$\frac{df(x_1,x_2,x_3)}{dx_2} = f(x_1,0,x_3) \oplus f(x_1,1,x_3) =$$
$$= \bar{x}_1 x_3 \oplus (x_1 \vee x_3) =$$
$$\overline{\bar{x}_1 x_3}(x_1 \vee x_3) \vee \bar{x}_1 x_3 \overline{(x_1 \vee x_3)} =$$
$$= (x_1 \vee \bar{x}_3)(x_1 \vee x_3) \vee \bar{x}_1 x_3 \bar{x}_1 \bar{x}_3 = x_1.$$
$$\frac{df(x_1,x_2,x_3)}{dx_3} = f(x_1,x_2,0) \oplus f(x_1,x_2,1) =$$
$$= x_1 x_2 \oplus (\bar{x}_1 \vee x_2) =$$
$$\overline{x_1 x_2}(\bar{x}_1 \vee x_2) \vee x_1 x_2 \overline{(\bar{x}_1 \vee x_2)} =$$
$$= (\bar{x}_1 \vee \bar{x}_2)(\bar{x}_1 \vee x_2) \vee x_1 x_2 x_1 \bar{x}_2 = \bar{x}_1.$$

Для трьох змінних отримано 4 умови активізації, які відповідають чотирьом логічним шляхам в схемній структурі диз'юнктивної форми даної функції.

Обчислення трьох похідних першого порядку по таблиці істинності дає наступний результат:

| $X_1$ | $X_2$ | $X_3$ | Y | $Y_1^0$ | $Y_1^1$ | $Y_1'$ | $Y_2^0$ | $Y_2^1$ | $Y_2'$ | $Y_3^0$ | $Y_3^1$ | $Y_2'$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |

Якщо виключити з розгляду таблицю істинності, а використовувати кубітне покриття, то на змістовному рівні процес обчислення похідних матиме такий вигляд:

| Y | $Y_1'$ | $Y_2'$ | $Y_3'$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |

Пояснення. Для отримання похідної за першою змінною необхідно xor-скласти, зсунуті назустріч один одному, дві сусідні тетради кубіт-вектора Y, а

результат записати в обидві тетради: {a,b}=a⊕b, (0110,0110)=0101⊕0011. Для отримання похідної за другою змінною слід послідовно xor-скласти сусідні пари кубіт-вектора Y а загальний результат записувати в кожну пару: {a,b}=a⊕b: (00,00)=01⊕01, (11,11)=00⊕11. Для отримання похідної по третій змінній слід послідовно xor-скласти сусідні біти кубіт-вектора Y а загальний результат записати в кожен біт сусідів: {a,b}=a⊕b: (1,1)=0⊕1, (1,1)=0⊕1, (0,0)=0⊕0, (0,0)=0⊕0.

Природно, що кубіт-похідна по будь-якій вхідній змінній, як вектор, має відносну симетрію рівності підвекторів за побудовою: похідна першої змінної має симетричну рівність двох тетрад, похідна другої змінної має симетричну рівність кожних сусідніх пар, похідна третьої змінної має симетричну рівність кожних сусідніх бітів.

Приклад 9. Визначити всі похідні першого порядку за таблицею істинності логічної функції трьох змінних:

$$f(x) = \bar{x}_2\bar{x}_3 \vee x_1x_2x_3.$$

Результат взяття похідних за таблицею істинності [38] наведеної функціональності на основі виконання операцій над стовпцями вхідних змінних представлена у вигляді:

$$\frac{df}{dx_1} =$$

| $x_1$ | $x_2$ | $x_3$ | Y | $Y_2^0$ | $Y_2^1$ | $Y_2^{\oplus}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 |

$= x_2x_3.$

$$\frac{df}{dx_2} =$$

| $x_1$ | $x_2$ | $x_3$ | Y | $Y_2^0$ | $Y_2^1$ | $Y_2^{\oplus}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 |

$= \overline{x}_1\overline{x}_3 \vee x_1\overline{x}_3 \vee x_1x_3 = \overline{x}_3 \vee x_1x_3.$

$$\frac{df}{dx_3} =$$

| $x_1$ | $x_2$ | $x_3$ | Y | $Y_3^0$ | $Y_3^1$ | $Y_3^{\oplus}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 |

$= \overline{x}_1\overline{x}_2 \vee x_1\overline{x}_2 \vee x_1x_2 = \overline{x}_2 \vee x_1x_2.$

Як альтернатива, нижче наведено результати виконання процедур взяття похідних першого порядку для трьох змінних по кубітному покриттю функціональності:

| $X_1$ | $X_2$ | $X_3$ | Y | $Y_1^0$ | $Y_1^1$ | $Y_1'$ | $Y_2^0$ | $Y_2^1$ | $Y_2'$ | $Y_3^0$ | $Y_3^1$ | $Y_3'$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |

Далі представлений процес обчислення похідних на кубітному покритті без розгляду таблиці істинності, що на формальному рівні має такий вигляд:

| Y | $Y_1'$ | $Y_2'$ | $Y_3'$ |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Інтерпретація отриманих кубіт-похідних. Природно, що похідні є функції, задані векторами. Вони можуть бути записані в аналітичній формі (ДНФ) за одиничними значеннями змінних, які формують адреси комірок кубіт-вектора:

$$Y_1' = 011 \vee 111 = \overline{X}_1 X_2 X_3 \vee X_1 X_2 X_3 = X_2 X_3 (\overline{X}_1 \vee X_1) = X_2 X_3.$$
$$Y_2' = 000 \vee 010 \vee 100 \vee 101 \vee 110 \vee 111 = \overline{X}_1 \overline{X}_3 \vee X_1.$$
$$Y_3' = 000 \vee 001 \vee 100 \vee 101 \vee 110 \vee 111 = \overline{X}_1 \overline{X}_2 \vee X_1.$$

Мінімізація булевих функцій, відповідних похідним, приводить до аналітичних виразів, де відсутні змінні, за якими береться похідна. Таким чином, всі результати обчислення похідних від трьох форм (аналітична, таблична, векторна) завдання функції є ідентичними. Найбільш технологічним є метод взяття похідної по кубітному покриттю. Метод має меншу обчислювальну складність в силу компактного представлення функціональності. Використання аналітичної форми передбачає істотне підвищення складності алгоритмів, пов'язаної із застосуванням законів булевої алгебри і мінімізації функцій, що обмежує її застосування для вирішення практичних завдань.

Для порівняння далі коротко описаний метод отримання тесту $T = [T_{ij}], i = \overline{1,k}; j = \overline{1,n}$ комбінаційної функціональності, заданої кубічним покриттям або таблицею істинності, який містить пункти [103]:

1) $f'(x_i) = f(x_1, x_2, ..., x_i = 0, ..., x_n) \oplus f(x_1, x_{2,...,} x_i = 1, ..., x_n);$
2) $T = \bigcup_{i=1}^{n} [f'(x_i) * (x_i = 0) \vee (x_i = 1)];$
3) $T_{ij} = T_{i-1,j} \leftarrow T_{ij} = X;\ T_{1j} = 1 \leftarrow T_{1j} = X;$
4) $T = T \setminus T_i \leftarrow T_i = T_{i-r},\ r = \overline{1, i-1},\ i = \overline{2, n}.$

1) Обчислення похідних за всіма n змінними функціональності шляхом використання кубічного покриття. 2) Об'єднання всіх умов (векторів) активізації в таблицю, де кожному вектору шляхом конкатенації (*) ставиться у відповідність зміна змінної, за якою була взята похідна, що означає подвоєння кількості тестових наборів відносно загальної кількості (k) умов активізації. 3) Мінімізація тестових векторів шляхом видалення повторюваних вхідних послідовностей. Рис. 3.7 ілюструє таблиці ходу отримання тесту відповідно до пунктів 2-3 алгоритму для функціональності $f(x) = \bar{x}_2\bar{x}_3 \vee x_1x_2x_3$, представленої схемної структурою.

| $x_1$ | $x_2$ | $x_3$ | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 → |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

| $x_1$ | $x_2$ | $x_3$ | Y |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |

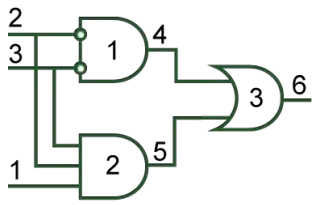



Рис. 3.7 – Отримання тесту для схемної структури булевої функції

Як альтернатива описаному вище, далі пропонується технологічно простий метод синтезу тестів на основі взяття похідних по кубітним покриттям функціональних елементів без розгляду станів вхідних змінних.

1) Початкове завдання логічної функціональності кубітним покриттям.

2) Виконання операцій зустрічного зсуву частин кубіт-вектора і подальшого покоординатного xor-підсумовування для отримання векторів похідних для кожної вхідної змінної.

3) Логічне об'єднання векторів похідних, що формує тест-вектор, рівний за розміром кубітному покриттю.

4) У разі необхідності отримання мінімального тесту вирішується завдання покриття (вже на матриці кубіт-похідних) шляхом знаходження мінімальної кількості пар одиничних координат кубіт-вектора всіх змінних, де пара одиниць повинна перевіряти одиночні константні несправності кожного входу. Процедура вибору пари одиниць в кубіт-похідній визначається наявністю двох представників, по одному від будь-якої парної і будь-якої непарної частин вектора.

Далі в таблиці представлено результати синтезу тесту для логічної функції від трьох змінних, заданої рівнянням: $f(x) = \bar{x}_2\bar{x}_3 \vee x_1x_2x_3$, що має кубітне покриття (10001001),

| $X_1$ | $X_2$ | $X_3$ | Y | $Y_1'$ | $Y_2'$ | $Y_3'$ | T |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | *1* |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | *1* |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | *0* |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | *0* |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | *0* |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | *1* |

| $X_1$ | $X_2$ | $X_3$ | Y |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |

З урахуванням виконання пункту 3 алгоритм синтезу дає мінімальний тест перевірки вхідних змінних, який містить 5 наборів, що представлено стовпцем Т, а також продубльовано в явному вигляді правою таблицею.

Інтерес представляє той факт, що результат виконання процедури взяття похідної по кубіт-вектору вже містить тест активізації кожної змінної. Об'єднаний тест перевіряє всі константні несправності вхідних змінних, а також може бути використаний для діагностування несправностей, оскільки для істотних входів всі похідні-вектори будуть різними. Фактично взяття похідної по змінній на кубіт-покритті формує Q-тест, не більше і не менше.

### 3.4 Дедуктивний аналіз несправностей цифрових структур

Використовується для визначення якості тесту щодо введеного класу несправностей, як правило, одиночних константних. Існує розвинена теорія

дедуктивного аналізу [38], орієнтована на паралельну обробку списків несправностей. Базові поняття теорії моделювання несправностей представлені апаратом транспортування списків дефектів через примітивні функціональні логічні елементи [104-110]. Нижче визначено дедуктивні функції паралельного моделювання несправностей на вичерпному тесті для функціональних елементів and, or, not. Отримання дедуктивного перетворювача 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, який має 3 координати, причому остання з них визначає стан виходу двовходового елемента and (or);L – вихідний список несправностей; X – список несправностей на конкретному вході примітиву; x={0,1} – логічне значення на вході примітиву. У наступному перетворенні $T_t = (T_{t1}, T_{t2}), (t = \overline{1,2})$– тест-вектор, який 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 инвариантность інверсії до вхідного набору для транспортування дефектів. Вона трансформується в повторювач. Тому дана функція не фігурує на виходах дедуктивних елементів. Спільна апаратурна

реалізація дедуктивних функцій для решти двовходових елементів and, or на вичерпному тесті представлена універсальною схемою (рис. 3.8) дедуктивно-паралельного аналізу несправностей.

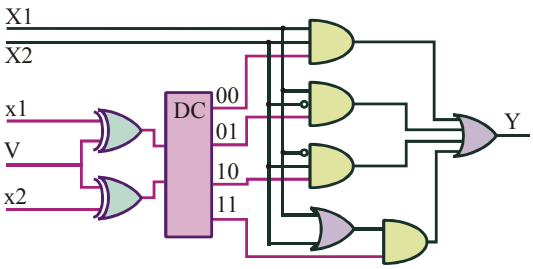



Рис. 3.8 – Симулятор несправностей примітивів

У симуляторі представлено булеві (x1,x2) і регістрові (X1,X2) для кодування несправностей входи, змінна V 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 перевіряються несправностей.

Далі пропонується технологічна реалізація дедуктивного моделювання на кубітной формі завдання функциональностей, яка відрізняється від наведеної вище паралелізмом виконання логічних операцій, а також можливістю застосування методу для будь-яких цифрових структур.

Сукупність кубіт-похідних для всіх вхідних змінних, обчислених по кубітному покриттю, являє собою кубітную матрицю для реалізації дедуктивного методу моделювання несправностей. Рядок матриці формує умови для транспортування списків несправностей від зовнішніх входів до виходу за правилом: одиничні значення створюють об'єднання вхідних списків, а нульові сигнали вказують на входи, списки яких повинні бути відняті з результату об'єднання. Наявність всіх нульових сигналів в рядку створює умови перетину вхідних списків між собою.

Як приклад пропонується побудова дедуктивних формул транспортування списків несправностей від вхідних змінних до виходу

функціональності, заданої вхідними наборами, кубітним покриттям з векторними похідними:

| $x_1$ | $x_2$ | $x_3$ | Y | $X_1$ | $X_2$ | $X_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

У загальному випадку формула дедуктивного моделювання логічних функциональностей, представлених у вигляді кубітних векторів, має такий вигляд:

$$L = \vee_{i=1}^{2^n} (x_{i1} \wedge x_{i2} \wedge \ldots \wedge x_{ij} \wedge \ldots \wedge x_{in}) \wedge (X_{i1} \vee X_{i2} \vee \ldots \vee X_{ij} \vee \ldots \vee X_{in}),$$

$$L = \vee_{i=1}^{2^n} (x_{i1} \wedge \ldots \wedge x_{in}) \wedge (X_{i1} \vee \ldots \vee X_{in}) ,$$

$$L = x \wedge X, x = x_{ij}, X = X_{ij}, i = \overline{1, 2^n}, j = \overline{1, n}.$$

Тут L – список вихідних несправностей; x – матриця вхідних тестових наборів; X – матриця кубітних похідних від кубітного покриття; n – кількість вхідних змінних.

Алгоритм побудови дедуктивної формули для заданої функціональності містить наступні пункти:

1) Завдання кубіт-вектора функціональності.

2) Обчислення кубіт-похідних для вхідних змінних з метою отримання відповідної матриці.

3) Формування аналітичної або матрично-векторної форми обчислення вихідних списків несправностей шляхом логічного множення матриць вхідних тестових впливів і матриці похідних.

Нижче наведено процес обчислення аналітичної і векторної форм для дедуктивного моделювання несправностей логічної функціональності:

$$T = (000,001,010,011,100,101,110,111).$$

$$Q = (011,001,010,100,011,011,011,111)].$$

$$L = (000 \wedge 011) \vee (001 \wedge 001) \vee (010 \wedge 010) \vee (011 \wedge 100) \vee$$
$$\vee (100 \wedge 011) \vee (101 \wedge 011) \vee (110 \wedge 011) \vee (111 \wedge 111).$$

$$L = (\bar{x}_1\bar{x}_2\bar{x}_3 \wedge \bar{X}_1X_2X_3) \vee (\bar{x}_1\bar{x}_2x_3 \wedge \bar{X}_1\bar{X}_2X_3) \vee (\bar{x}_1x_2\bar{x}_3 \wedge \bar{X}_1X_2\bar{X}_3) \vee$$
$$\vee (\bar{x}_1x_2x_3 \wedge X_1\bar{X}_2\bar{X}_3) \vee (x_1\bar{x}_2\bar{x}_3 \wedge \bar{X}_1X_2X_3) \vee (x_1\bar{x}_2x_3 \wedge \bar{X}_1X_2X_3) \vee$$
$$\vee (x_1x_2\bar{x}_3 \wedge \bar{X}_1X_2X_3) \vee (x_1x_2x_3 \wedge X_1X_2X_3).$$

Для порівняння технологічної складності запропонованого методу нижче представлена процедура отримання дедуктивних формул моделювання несправностей [38] на основі аналітичного завдання функціональностей:

$$L = \bar{x}_1\bar{x}_2\bar{x}_3\{[(\bar{X}_2 \oplus 0)(\bar{X}_3 \oplus 0) \vee (X_1 \oplus 0)(X_2 \oplus 0)(X_3 \oplus 0)] \oplus 1\}$$
$$\vee \bar{x}_1\bar{x}_2x_3\{[(\bar{X}_2 \oplus 0)(\bar{X}_3 \oplus 1) \vee (X_1 \oplus 0)(X_2 \oplus 0)(X_3 \oplus 1)] \oplus 0\}$$
$$\vee \bar{x}_1x_2\bar{x}_3\{[(\bar{X}_2 \oplus 1)(\bar{X}_3 \oplus 0) \vee (X_1 \oplus 0)(X_2 \oplus 1)(X_3 \oplus 0)] \oplus 0\}$$
$$\vee \bar{x}_1x_2x_3\{[(\bar{X}_2 \oplus 1)(\bar{X}_3 \oplus 1) \vee (X_1 \oplus 0)(X_2 \oplus 1)(X_3 \oplus 1)] \oplus 0\}$$
$$\vee x_1\bar{x}_2\bar{x}_3\{[(\bar{X}_2 \oplus 0)(\bar{X}_3 \oplus 0) \vee (X_1 \oplus 1)(X_2 \oplus 0)(X_3 \oplus 0)] \oplus 1\}$$
$$\vee x_1\bar{x}_2x_3\{[(\bar{X}_2 \oplus 0)(\bar{X}_3 \oplus 1) \vee (X_1 \oplus 1)(X_2 \oplus 0)(X_3 \oplus 1)] \oplus 0\}$$
$$\vee x_1x_2\bar{x}_3\{[(\bar{X}_2 \oplus 1)(\bar{X}_3 \oplus 0) \vee (X_1 \oplus 1)(X_2 \oplus 1)(X_3 \oplus 0)] \oplus 0\}$$
$$\vee x_1x_2x_3\{[(\bar{X}_2 \oplus 1)(\bar{X}_3 \oplus 1) \vee (X_1 \oplus 1)(X_2 \oplus 1)(X_3 \oplus 1)] \oplus 1\}.$$

$$L = \bar{x}_1\bar{x}_2\bar{x}_3(\overline{\bar{X}_2\bar{X}_3 \vee X_1X_2X_3}) \vee \bar{x}_1\bar{x}_2x_3(\bar{X}_2X_3 \vee X_1X_2\bar{X}_3) \vee \bar{x}_1x_2\bar{x}_3(X_2\bar{X}_3 \vee X_1\bar{X}_2X_3)$$
$$\vee \bar{x}_1x_2x_3(X_2X_3 \vee X_1\bar{X}_2\bar{X}_3) \vee x_1\bar{x}_2\bar{x}_3(\overline{\bar{X}_2\bar{X}_3 \vee \bar{X}_1X_2X_3}) \vee x_1\bar{x}_2x_3(\bar{X}_2X_3$$
$$\vee \bar{X}_1X_2\bar{X}_3) \vee x_1x_2\bar{x}_3(X_2\bar{X}_3 \vee \bar{X}_1\bar{X}_2X_3) \vee x_1x_2x_3(\overline{X_2X_3 \vee \bar{X}_1\bar{X}_2\bar{X}_3}).$$

$$L = \bar{x}_1\bar{x}_2\bar{x}_3(\overline{\bar{X}_2\bar{X}_3} \wedge \overline{\bar{X}_1X_2X_3}) \vee \bar{x}_1\bar{x}_2x_3(\bar{X}_2X_3 \vee X_1X_2\bar{X}_3) \vee \bar{x}_1x_2\bar{x}_3(X_2\bar{X}_3 \vee X_1\bar{X}_2X_3)$$
$$\vee \bar{x}_1x_2x_3(X_2X_3 \vee X_1\bar{X}_2\bar{X}_3) \vee x_1\bar{x}_2\bar{x}_3(\overline{\bar{X}_2\bar{X}_3} \wedge \overline{\bar{X}_1X_2X_3}) \vee x_1\bar{x}_2x_3(\bar{X}_2X_3$$
$$\vee \bar{X}_1X_2\bar{X}_3) \vee x_1x_2\bar{x}_3(X_2\bar{X}_3 \vee \bar{X}_1\bar{X}_2X_3) \vee x_1x_2x_3(\overline{\bar{X}_2\bar{X}_3} \wedge \overline{\bar{X}_1\bar{X}_2\bar{X}_3}).$$

$$L = \bar{x}_1\bar{x}_2\bar{x}_3[(X_2 \vee X_3) \wedge (\bar{X}_1 \vee \bar{X}_2 \vee \bar{X}_3)] \vee \bar{x}_1\bar{x}_2x_3(\bar{X}_2X_3 \vee X_1X_2\bar{X}_3) \vee \bar{x}_1x_2\bar{x}_3(X_2\bar{X}_3$$
$$\vee X_1\bar{X}_2X_3) \vee \bar{x}_1x_2x_3(X_2X_3 \vee X_1\bar{X}_2\bar{X}_3) \vee x_1\bar{x}_2\bar{x}_3[(X_2 \vee X_3) \wedge (X_1 \vee \bar{X}_2$$
$$\vee \bar{X}_3)] \vee x_1\bar{x}_2x_3(\bar{X}_2X_3 \vee \bar{X}_1X_2\bar{X}_3) \vee x_1x_2\bar{x}_3(X_2\bar{X}_3 \vee \bar{X}_1\bar{X}_2X_3)$$
$$\vee x_1x_2x_3[(\bar{X}_2 \vee \bar{X}_3) \wedge (X_1 \vee X_2 \vee X_3)].$$

$$L = \bar{x}_1\bar{x}_2\bar{x}_3(\bar{X}_1X_2 \vee \bar{X}_1X_3 \vee \bar{X}_2X_3 \vee X_2\bar{X}_3) \vee \bar{x}_1\bar{x}_2x_3(\bar{X}_2X_3 \vee X_1X_2\bar{X}_3)$$
$$\vee \bar{x}_1x_2\bar{x}_3(X_2\bar{X}_3 \vee X_1\bar{X}_2X_3) \vee \bar{x}_1x_2x_3(X_2X_3 \vee X_1\bar{X}_2\bar{X}_3)$$
$$\vee x_1\bar{x}_2\bar{x}_3(X_1X_2 \vee X_1X_3 \vee \bar{X}_2X_3 \vee X_2\bar{X}_3) \vee x_1\bar{x}_2x_3(\bar{X}_2X_3 \vee \bar{X}_1X_2\bar{X}_3)$$
$$\vee x_1x_2\bar{x}_3(X_2\bar{X}_3 \vee \bar{X}_1\bar{X}_2X_3) \vee x_1x_2x_3(X_1\bar{X}_2 \vee X_1\bar{X}_3 \vee X_2\bar{X}_3 \vee \bar{X}_2X_3).$$

Обчислення кубітних форм для дедуктивного моделювання несправностей основних логічних примітивів: or, and, xor представлено в таблиці:

| $x_1$ | $x_2$ | $Y^{\vee}$ | $X_1^{\vee}$ | $X_2^{\vee}$ | $Y^{\wedge}$ | $X_1^{\wedge}$ | $X_2^{\wedge}$ | $Y^{\oplus}$ | $X_1^{\oplus}$ | $X_2^{\oplus}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

Тут визначено два вектор-стовпці табличного завдання вхідних змінних $(x_1, x_2)$, кубіт-вектор функції or - $Y^{\vee}$, дві похідні $(X_1^{\vee}, X_2^{\vee})$ для кожної вхідної змінної. Далі показано: кубіт-вектор завдання функції and і дві колонки похідних, а також кубіт-вектор функції xor і два стовпці похідних по вхідним змінним. Формули дедуктивного моделювання тривіально записуються по рядках таблиці:

$$L^{\vee} = \bar{x}_1\bar{x}_2(X_1 \vee X_2) \vee \bar{x}_1x_2(\bar{X}_1 \vee X_2) \vee x_1\bar{x}_2(X_1 \vee \bar{X}_2) \vee x_1x_2(\bar{X}_1 \vee \bar{X}_2)$$

$$L^{\wedge} = \bar{x}_1\bar{x}_2(\bar{X}_1 \vee \bar{X}_2) \vee \bar{x}_1x_2(X_1 \vee \bar{X}_2) \vee x_1\bar{x}_2(\bar{X}_1 \vee X_2) \vee x_1x_2(X_1 \vee X_2)$$

$$L^{\oplus} = \bar{x}_1\bar{x}_2(X_1 \vee X_2) \vee \bar{x}_1x_2(X_1 \vee X_2) \vee x_1\bar{x}_2(X_1 \vee X_2) \vee x_1x_2(X_1 \vee X_2) = (X_1 \vee X_2)$$

Тут вхідні змінні $x_i$ з'єднуються між собою знаком кон'юнкції, а похідні – змінні транспортування списків вхідних несправностей, позначені символом $X_i$, об'єднуються між собою знаком диз'юнкції, відповідно до станів координат стовпців-похідних.

Таким чином, запропонована технологія моделювання несправностей, що базується на використанні векторних кубітних форм завдання функциональностей і похідних, не має аналогів за доступністю розуміння, простотою реалізації і швидкодії. Щоб отримати дедуктивні формули

транспортування дефектів будь-якої функціональності, необхідно і достатньо отримати вектор-стовпці похідних всіх вхідних змінних за допомогою операцій зсуву і xor-підсумовування. Обчислювальна складність даних операцій залежить від апаратних витрат і в межі може бути зведена до лінійної залежності від кількості вхідних змінних.

На рис. 3.9 представлений процесор кубітного моделювання цифрових пристроїв, що містить структури: справного інтерпретативного моделювання, дедуктивного аналізу несправностей, призначеного для оцінки якості тесту і побудови таблиці несправностей, а також модулів тестування і діагностування дефектів на стадіях проектування і експлуатації. Основна відмінність від існуючих рішень полягає у використанні Q-покриття, представленого у формі вектора станів функціональності, що дає можливість істотно підвищити швидкодію моделювання за рахунок виконання паралельних регістрових операцій.

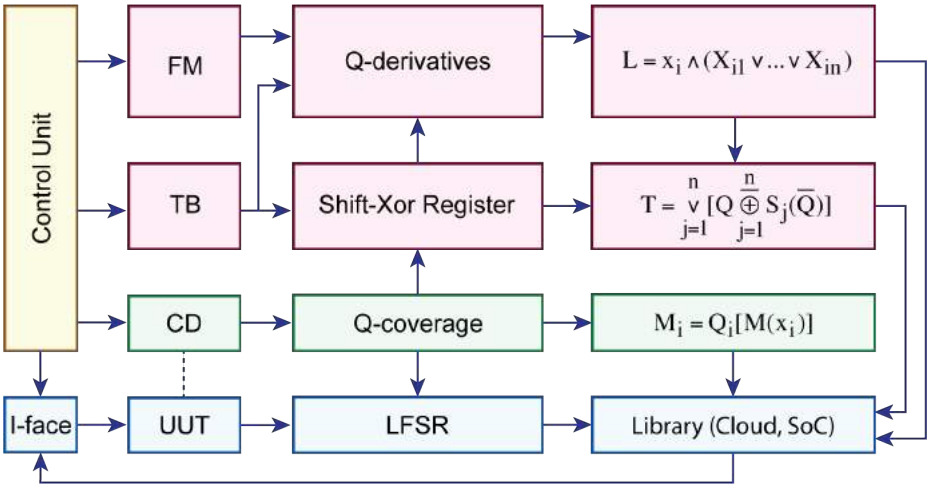



Рис. 3.9 – Процесор кубітного моделювання цифрових пристроїв

Control Unit – пристрій управління симулятором, який синхронізує роботу блоків справного моделювання та структурних компонентів дедуктивного аналізу несправностей. FM – Fault Matrix, матриця вхідних несправностей розглянутої функціональності цифрового пристрою. TB – Test Bench, упорядкована сукупність вхідних перевіряльних послідовностей, де поточний вхідний набір ідентифікується як $x_i$. CD – Circuit Description –

схемний опис цифрового пристрою, де функціональні елементи представлені кубітнимі покриттями Q-coverage. Обробка останніх здійснюється блоком справного моделювання $M_i = Q_i[M(x_i)]$, який реалізує адресні транзакції між кубітним покриттям і вектором моделювання M. Результати справного моделювання вхідних наборів формують матрицю GM (Good simulation Matrix), яка записується в Library. Блок Shift-Xor Register формує матрицю похідних у кубітній формі (Q-derivatives), застосовуючи регістрові операції зсуву і xor. L-блок для створення вихідного списку несправностей використовує формулу аналізу вхідного набору і рядки матриці похідних

$$L = x_i \wedge (X_{i1} \vee \ldots \vee X_{in}).$$

Результати дедуктивного аналізу формують списки несправностей, що відповідають вхідним наборам, які об'єднуються в DM – Fault Detected Matrix і заносяться в Library. Крім того, T-модуль формує Q-тест і оцінює його якість у метриці одиночних константних несправностей зовнішніх входів і виходів функціональних елементів, які заносяться в Library. Тести для функциональностей разом з матрицями несправностей формують бібліотеку Library={Signature, Q-coverage, Q-test, Quality, DM, GM}, яка може багаторазово використовуватися як хмарний або вбудований в SoC сервіс для тестування і/або діагностування функциональностей UUT (Unit Under Test) на основі використання інтерфейсу I-face, що підтримує стандарти IEEE 1500 SECT, IP (Internet Protocol). Пошук тестових сервісів в бібліотеці здійснюється за допомогою кубітного вектора, попередньо згорнутого в 16-розрядний двійковий код – сигнатуру (Sign), на основі регістра зсуву з лінійними зворотними зв'язками (LFSR), що дає можливість структурувати бібліотеку для швидкого отримання інформації та проведення тестування в режимі online. Блок LFSR виконує дві функції стиснення в 16-ковий код-сигнатуру: Q-покриття і реакції UUT на вхідні тестові впливи для подальшого діагностування місця, причини і виду дефекту.

## 3.5 Висновки

Нижче наведено формулювання наукової новизни і практичної значущості описаних досліджень.

1) Удосконалено векторні моделі кубітного представлення структур і компонентів цифрових систем, що базуються на адресному кодуванні вхідних сигналів і відрізняються технологічністю та швидкодією побудованих на їх основі процедур синтезу й аналізу.

2) Удосконалено методи синтезу тестів і моделювання несправностей, які відрізняються паралельним виконанням логічних регістрових операцій над кубітними покриттями схемних компонентів.

3) Вперше розроблено метод і секвенсор безумовного синтезу тестів для функціональних логічних компонентів, який характеризується паралельним виконанням регістрових логічних операцій (shift, or, not, nxor) над кубітним вектором і його похідними, що дає можливість істотно зменшити час генерування вхідних наборів і тестування пристрою в режимі embedded online.

2) Вперше розроблено метод взяття похідних для генерації тестів функціональних компонентів, який характеризується паралельним виконанням регістрових логічних операцій (shift, or, not, nxor) над кубітним вектором, що дає можливість істотно зменшити час генерування вхідних наборів і тестування пристрою за рахунок апаратної надлишковості.

3) Вперше розроблено дедуктивний метод моделювання несправностей для функціональних компонентів, який характеризується паралельним виконанням регістрових логічних операцій (shift, or, not, nxor) над кубітним вектором і його похідними, що дає можливість істотно зменшити час верифікації та тестування цифрового пристрою в режимі embedded online.

4) Запропоновано процесор кубітного моделювання цифрових пристроїв, імплементований в SoC або Cloud Service, для аналізу справної поведінки і несправностей на основі використання кубітних покриттів функціональних елементів, який відрізняється від відомих реалізацій

застосуванням мінімального набору регістрових логічних операцій і високою швидкодією.

5) Практична значущість досліджень полягає в можливості хмарної реалізації швидкодіючого методу синтезу тестів і моделювання несправностей для функціональних логічних компонентів на основі паралельного виконання регістрових логічних операцій (shift, or, not, nxor) над кубітним вектором і його похідними, що дає можливість генерувати вхідні набори і оцінювати їх якість в режимі online. Крім того, хмарний мікросервіс синтезу тестів і моделювання дефектів для функціональних логічних компонентів може бути затребуваний для навчальних і наукових цілей в процесах синтезу та аналізу цифрових архітектур.

6) Запропонований метод синтезу тестів для функциональностей на основі кубітного покриття може бути використаний в якості вбудованого BIST-компонента для сервісного обслуговування SoC на основі стандарту граничного сканування IEEE 1500 SECT або в якості хмарного online сервісу тестування апаратних модулів за допомогою IP-протоколу.

7) Подальші дослідження в цій галузі будуть спрямовані на створення програмно-апаратних генераторів тестів, симуляторів несправностей, справної поведінки, алгоритмів діагностування та бібліотечних рішень, вбудованих в інфраструктуру кристалів і/або хмарні сервіси, що використовують кубітний опис функціональності логічного компонента.

# РОЗДІЛ 4

# СИНТЕЗ І АНАЛІЗ КУБІТНИХ МОДЕЛЕЙ ЦИФРОВИХ СИСТЕМ

Пропонується хмарний сервіс QuaSim для моделювання і верифікації цифрових систем, що базується на транзакціях між адресовними компонентами пам'яті для реалізації будь-якої функціональності. Описано новий підхід до синтезу та аналізу цифрових систем, що використовує векторну форму (квант) завдання комбінаційних і послідовностних структур для їх імплементації в елементи пам'яті, що істотно відрізняється від загальноприйнятої теорії проектування дискретних пристроїв на основі таблиць істинності компонентів. Використовуються квантові або кубітні структури даних [97, 111-114] для реалізації обчислювальних процесів з метою підвищення швидкодії аналізу цифрових систем і зменшення обсягів пам'яті на основі унарного кодування станів вхідних, внутрішніх і вихідних змінних та імплементації кубітних векторів в елементи пам'яті FPGA, що реалізують комбінаційні і послідовностні примітиви. Впровадження квантових memory-based-only моделей опису цифрових компонентів в практику проектування обчислювальних систем безпосередньо впливає на збільшення виходу придатної продукції, підвищення надійності комп'ютерних виробів, зниження вартості проектування і виготовлення, а також забезпечує автономне відновлення працездатності в режимі remote & online без участі людини.

## 4.1 Введення

Мета – істотне поліпшення якості та надійності обчислювальних пристроїв за рахунок адресовних схемних елементів, що дозволяє виконувати online ремонт, а також підвищення швидкодії методів моделювання, тестування і верифікації складних цифрових систем, завдяки зменшенню

розмірності моделей функціональних примітивів та адресної реалізації всіх компонентів структур даних.

Задачі: 1) Розробка автоматної моделі квантового процесора. 2) Синтез кубітних моделей цифрових примітивів: логіка, тригери, регістри і лічильники. 3) Синтез і аналіз квантових моделей цифрових схем. 4) Моделювання цифрових пристроїв на основі використання квантових векторів опису примітивів.

Мотивація і стан питання: 1) Сучасна система на кристалі містить 94% пам'яті і лише 6% логіки, яка доставляє більше 90% проблем, пов'язаних з верифікацією, тестуванням, діагностуванням та відновленням працездатності [100, 115]. Звичайно, швидкодія логічних схем на порядок вище, ніж у пам'яті, проте велика частка обчислювальних процесів припадає на обмін інформацією в структурах пам'яті. Тому переваги комбінаційної логіки в реальних обчислювальних системах обробки великих даних компенсуються великими часовими витратами (близько 90%), пов'язаними з транзакціями в пам'яті. 2) Реалізація процесора тільки на основі використання елементів пам'яті робить його однорідним за структурою і типам функціональних примітивів, що доставляє очевидні технологічні зручності процесам проектування, виробництва і експлуатації, у тому числі верифікації, вбудованому тестуванню і діагностуванню, а головне – ремонту в режимі online за рахунок використання на кристалі універсальних адресовних spare-компонентів пам'яті [100, 115]. 3) Моделювання в процесі верифікації проектованих обчислювачів на основі адресних моделей компонентів робить дану процедуру технологічно простою за рахунок регулярних структур даних і використання єдиної операції транзакції на елементах пам'яті, а також більш швидкодіючою за рахунок можливості паралельної квантоподібної обробки великих масивів однотипної пам'яті [97, 113-114, 116, 118, 119]. 4) Енергоспоживання при заміні логіки на елементи пам'яті зростає на кілька відсотків, що насправді буде платою за перераховані вище істотні переваги, пов'язані, в кінцевому рахунку, зі

збільшенням виходу придатної продукції, підвищенням надійності обчислювальних виробів, зниженням вартості проектування і виготовлення, а також автономним відновленням працездатності в режимі remote & online, без участі людини. Однак енергозберігаючі рішення з реалізації обчислювальних процесів на пам'яті [116-117, 121-122] дають підстави вважати, що такого програшу взагалі не буде.

4.2 Квантові або кубітні структури даних

На ринку електронних технологій існує конкуренція між базами імплементації ідеї [111-114,120]: 1) Гнучка (м'яка) реалізація проекту пов'язана з синтезом інтерпретативної моделі програмної форми функціональності або в апаратному виконанні програмовних логічних пристроїв на основі FPGA, CPLD; переваги полягають у технологічності модифікації проекту, недоліки – у невеликій швидкодії функціонування цифрової системи. 2) Жорстка реалізація орієнтована на використання компілятивних моделей при розробці програмних додатків або на імплементацію проекту в кристали VLSI [97, 100, 115, 121-122]. Переваги та недоліки жорсткої реалізації є інверсними по відношенню до м'якого виконання проектів: висока швидкодія і неможливість модифікації. З урахуванням викладених базових варіантів реалізації ідеї пропонуються квантові структури даних, орієнтовані на підвищення швидкодії гнучких моделей програмного або апаратного виконання проекту, а також на можливість online ремонту в процесі експлуатації.

Квантові структури опису цифрових систем. Кубіт (n-кубіт) є векторною формою унітарного (унарного) кодування універсуму з n примітивів для завдання булеана станів $2^{2^n}$ за допомогою $2^n$ двійкових змінних. Наприклад, якщо n = 2, то 2-кубіт задає 16 станів за допомогою чотирьох змінних. Якщо n=1, то кубіт задає чотири стани на универсумі з двох примітивів (10) і (01) за допомогою двох двійкових змінних (00,01,10,11) [113,120]. При цьому допускається суперпозиція (одночасне існування) у векторі $2^n$ станів,

позначених примітивами. Кубіт (n-кубіт) дає можливість використовувати паралельні логічні операції замість поелементних теоретико-множинних для істотного прискорення процесів аналізу дискретних систем.

Далі кубіт ототожнюється з n-кубітом або двійковим вектором, якщо це не заважає розумінню викладеного матеріалу. Оскільки квантові обчислення пов'язані з аналізом кубітних структур даних, то далі будемо застосовувати визначення «квантовий» для ідентифікації технологій, які використовують три властивості квантової механіки: паралелізм обробки (двійкових векторів), суперпозицію станів і їх переплутування. Синонімами кубіта при завданні двійкового вектора опису логічної функції є: Q-покриття, Q-вектор, квантовий вектор [113-114,120] як уніфікована векторна форма суперпозиційного завдання вихідних станів, відповідних адресним кодам вхідних змінних логічного елемента.

Кубіт в цифровій системі використовується в якості форми завдання структурного примітиву, інваріантної до технологій реалізації функціональності (hardware, software). Більш того, синтез цифрових систем на основі кубітних структур не прив'язаний жорстко до теореми Поста, що визначає п'ять умов (класів) існування функціонально повного базису. На пропонованому рівні абстракції n-кубіт дає більш широкі можливості для векторного завдання будь-якої n-входової функції з булеана потужністю $|B(A)| = 2^{2^n}$, який неодмінно містить всі функціональності, що задовольняють п'яти класам теореми Поста. Формат структурного кубітного компонента цифрової схеми $Q^* = (X, Q, Y)$ містить інтерфейс (вхідні та вихідну змінні), а також кубіт-вектор Q, що задає функцію $Y = Q(X)$, розмірність якого визначається ступеневою функцією від кількості вхідних ліній $k = 2^n$.

## 4.3 Синтез квант-вектора комбінаційної схеми

Кубіт (квант) комбінаційної схеми є вектор станів виходу на впорядкованій множині всіх вхідних слів, які ототожнюються з адресами

комірок пам'яті вектора. Синтез Q-вектора (покриття) схемної структури (без таблиць істинності логічних елементів) на основі примітивів, заданих Q-векторами зводиться до отримання узагальненого кубіт-вектора шляхом декартової процедури виконання логічної операції над розрядами кубітних векторів. Декартова процедура для двох чотирирозрядних кубітів, які суперпозиціонуются логічною операцією or (and, xor), представлена в наступній таблиці:

| $\vee,\wedge,\oplus$ | b(0) | b(1) | b(2) | b(3) |
|---|---|---|---|---|
| a(0) | $c(0) = a(0) \vee b(0)$ | $c(1) = a(0) \vee b(1)$ | $c(2) = a(0) \vee b(2)$ | $c(3) = a(0) \vee b(3)$ |
| a(1) | $c(4) = a(1) \vee b(0)$ | $c(5) = a(1) \vee b(1)$ | $c(6) = a(1) \vee b(2)$ | $c(7) = a(1) \vee b(3)$ |
| a(2) | $c(8) = a(2) \vee b(0)$ | $c(9) = a(2) \vee b(1)$ | $c(10) = a(2) \vee b(2)$ | $c(11) = a(2) \vee b(3)$ |
| a(3) | $c(12) = a(3) \vee b(0)$ | $c(13) = a(3) \vee b(1)$ | $c(14) = a(3) \vee b(2)$ | $c(15) = a(3) \vee b(3)$ |

Приклади, які використовують логічні суперпозиції двох кубітів для отримання Q-векторів схемних структур $c_1 = (a_1 \wedge a_2) \vee (b_1 \vee b_2), c_2 = (a_1 \wedge a_2) \wedge (b_1 \vee b_2), c_3 = (a_1 \wedge a_2) \oplus (b_1 \vee b_2)$, представлені наступною таблицею:

| a(and) =<br>b(or) = | 0001<br>0111 |
|---|---|
| $c_1$ = a(and) $\vee$ b(or) | 0111 0111 0111 1111 |
| $c_2$ = a(and) $\wedge$ b(or) | 0000 0000 0000 0111 |
| $c_3$ = a(and) $\oplus$ b(or) | 0111 0111 0111 1000 |

Тут побудовані Q-покриття трьох схем, що складаються з трьох елементів кожна, де два логічних примітиви суперпозиційно об'єднуються третім елементом (or, and, xor). В результаті виходять три вектори, кожен з яких має розмірність в 16 біт. Обчислювальна складність процедури синтезу Q-покриття комбінаційної схеми дорівнює добутку довжин Q-векторів p примітивів, що входять в неї: $\eta = \prod_{i=1}^{p} \mathrm{card}(Q_i)$.

Складнішою є проблема синтезу Q-покриття схеми, вхідні лінії (розгалуження, що сходяться) якої мають гальванічні або проводові з'єднання (тут за змінною $a_2$): $c = (a_1 \wedge a_2) \vee (a_2 \vee a_3)$. В даному випадку після

синтезу Q-покриття схеми необхідно виконати його верифікацію щодо існування суперечливих адрес на змінних $a_2$ з метою мінімізації Q-вектора шляхом подальшого виключення зазначених адрес з розгляду, що зменшує розмірність Q-покриття до $\mathrm{card}(Q) = 2^q$ координат, де q – загальна кількість вхідних змінних схеми:

| Q = | 0111 0111 0111 1111 |
|---|---|
| $a_1$ = | 0000 0000 1111 1111 |
| $a_2$ = | 0000 1111 0000 1111 |
| $a_2$ = | 0011 0011 0011 0011 |
| $a_3$ = | 0101 0101 0101 0101 |

=

| Q = | 0111 0111 0111 1111 |
|---|---|
| $a_1$ = | 0000 0000 1111 1111 |
| $a_2$ = | 00xx xx11 00xx xx11 |
| $a_2$ = | 00xx xx11 00xx xx11 |
| $a_3$ = | 0101 0101 0101 0101 |

=

| Q = | 0111 0111 |
|---|---|
| $a_1$ = | 0000 1111 |
| $a_2$ = | 0011 0011 |
| $a_2$ = | 0011 0011 |
| $a_3$ = | 0101 0101 |

=

| Q = | 0111 0111 |
|---|---|
| $a_1$ = | 0000 1111 |
| $a_2$ = | 0011 0011 |
| $a_3$ = | 0101 0101 |

Процедура синтезу Q-покриття: 1) будується таблиця відповідності адрес розрядам Q-вектора схеми; 2) далі суперечливі координати у двох рядках $a_2$ відмічаються символом x; 3) потім всі стовпці з даними символами виключаються з таблиці; 4) після чого виходять два ідентичні рядки $a_2$, які об'єднуються в одну; 5) що дає в результаті Q-вектор комбінаційної схеми, але вже значно меншої розмірності. Переваги запропонованого Q-методу синтезу обчислювальних пристроїв полягають в компактності їх опису Q-векторами і в високій швидкодії адресного моделювання логічних елементів, створюють умови для створення ринково привабливої «квантової» теорії проектування цифрових систем на кристалах, що використовує векторно-кубітную форму завдання структурних компонентів.

### 4.4 Мінімізація квант-вектора схеми

Синтез квант-вектора схеми по Q-покриттях компонентів пов'язаний з мінімізацією або зменшенням розмірності Q-вектора шляхом виключення несуттєвих змінних. Як правило істотність залежить від гальванічних з'єднань вхідних та внутрішніх ліній цифрового пристрою, які накладають обмеження, пов'язані з суперечливістю сигналів на лініях зв'язку. Тому правило мінімізації адресного простору полягає в усуненні адресних кодів, які створюють протиріччя за з'єднаними змінними. Нехай є Q-вектор схеми і його адресний

простір, де змінні b,c,d (a,b,c) з'єднані гальванично. Нижче наведено таблиці перетворення або мінімізації адресного простору з метою отримання зменшеного Q-вектора:

| Q = | 0111 0111 0111 1111 |
|---|---|
| a = | 0000 0000 1111 1111 |
| b = | 0000 1111 0000 1111 |
| c = | 0011 0011 0011 0011 |
| d = | 0101 0101 0101 0101 |

| Q = | 0xxx xxx1 0xxx xxx1 |
|---|---|
| a = | 0000 0000 1111 1111 |
| b = | 0xxx xxx1 0xxx xxx1 |
| c = | 0xxx xxx1 0xxx xxx1 |
| d = | 0xxx xxx1 0xxx xxx1 |

=

| Q | 0101 |
|---|---|
| a = | 0011 |
| b = | 0101 |

| Q = | 0111 0111 0111 1111 |
|---|---|
| a = | 00xx xxxx xxxx xx11 |
| b = | 00xx xxxx xxxx xx11 |
| c = | 00xx xxxx xxxx xx11 |
| d = | 0101 0101 0101 0101 |

=

| Q | 0111 |
|---|---|
| a = | 0011 |
| d = | 0101 |

По-перше, тут слід зазначити, що в таблицях спостерігається дзеркальна осьова симетрія з інверсією сигналів на координатах адресного простору, яка створює властивість, що описується наступним виразом: $L \oplus R = 1 \rightarrow L_{ij} \oplus R_{ij} = 1$.

Дану обставину слід використовувати для зменшення розмірності аналізованого простору в два рази і відповідного зниження обчислювальної складності задачі синтезу квантової вектор-функціональності цифрової схеми. По-друге, кількість різних варіантів взаємодії на q вхідних змінних, пов'язаних з гальваничним з'єднанням різних сполучень вхідних ліній, визначається функціональною залежністю, граничні значення якої знаходяться в інтервалі: $\mathrm{card}(Q) = [2^q - 3^q]$. Проте, існує ефективна процедура для мінімізації розмірності Q-вектора шляхом виявлення суперечностей в кодах-стовпцях на координатах (Aij), відповідних гальванично пов'язаним w-змінним за j-параметром. Таку процедуру досить виконати на половині адресного простору $\mathrm{card}(Q) = 2^q / 2$. Інша частина суперечливих стовпців видаляється відповідно до дзеркальних відображень номерів тих стовпців, які були видалені з першої половини таблиці кодів адрес:

$$\{Q_i, Q_{2^q-i}\} = \varnothing \leftrightarrow (\overset{w}{\underset{j=1}{\wedge}} A_{ij}) \oplus (\overset{w}{\underset{j=1}{\vee}} A_{ij}) = 1,\ i \le 2^q / 2$$

Якщо в стовпці Ai на групі з w пов'язаних змінних зафіксовано, що кон'юнкція їх станів дорівнює нулю, а диз'юнкція має значення одиниці, то i-стовпець і його дзеркальне відображення $2^q - i$ видаляються з адресного простору A. Це автоматично призводить до виключення з Q-вектора двох отриманих $\varnothing$-координат (в таблицях позначені символами x), які відповідають даним стовпчикам.

Природно, що також спостерігається симетрія простору векторів-відстаней за Хеммінгом, отриманих шляхом xor-взаємодії між сусідніми рядками таблиці адресного простору, для яких суперпозиція лівої і правої частин дає результат $L \oplus R = 0 \rightarrow L_{ij} \oplus R_{ij} = 0$:

| Q = | 0111 0111 0111 1111 |
|---|---|
| a ⊕ b | 0000 1111 1111 0000 |
| b ⊕ c | 0011 1100 0011 1100 |
| c ⊕ d | 0110 0110 0110 0110 |
| d ⊕ a | 0101 0101 1010 1010 |

$= (L, R);\ (L \oplus R) =$

| Q = | 0111 0111 0111 1111 |
|---|---|
| a ⊕ b | 0000 0000 |
| b ⊕ c | 0000 0000 |
| c ⊕ d | 0000 0000 |
| d ⊕ a | 0000 0000 |

$\leftrightarrow (L = \overline{R})$

Чи доцільно мінімізувати логічну функцію, описану квант-вектором? Відповідь: мінімізація Q-векторів для отримання нормальних або дужкових форм не має практичного значення, істотно тільки зменшення розмірності вектора функціонального опису, що може бути лише наслідком визначення неістотності деяких вхідних (адресних) змінних. Проте, існує проблема розбиття квант-вектора на складові частини меншої розмірності, що пов'язано з імплементацією функціональності в конструктивні компоненти LUT FPGA. В цьому випадку виконується розбиття Q-вектора на два рівних підвектори Q=(L, R), які з'єднуються в структурно-адресну організацію функціональності за допомогою мультиплексора $Q = (\overline{a} \wedge L) \vee (a \wedge R)$. Якщо змінна мультиплексування a=0, то функціональність Q формується за допомогою

комірок лівого L-вектора, в іншому випадку, коли a=1, значення функції Q формується бітами правого R-вектора. Алгоритми розбиття та імплементації складних логічних функцій є в кожній промислової системі синтезу, моделювання і верифікації компонентів SoC.

## 4.5 Імплементація куба функціональності в технологічну структуру PLD

Вирішення задачі зводиться до розбиття куба функціональності (див. Розд. 2) на сегменти, яким можна поставити у відповідність бібліотечні примітиви конкретного кристалу. На кристалі PLD (FPGA) є в наявності певна кількість чотиривходових примітивів – елементів пам'яті для зберігання таблиці істинності LUT (Look Up Table). Примітиви можуть об'єднуватися в логічні сектори (Slice), що складаються з двох примітивів, об'єднаних мультиплексором, що дає можливість збільшити розрядність вхідних змінних до п'яти. Подальше об'єднання двох секторів за допомогою мультиплексора збільшує розрядність вхідних змінних до шести. При цьому на кристалі технологічно передбачено, що кожні 2 пари секторів мультиплексуються в блоки CLB (Configurable Logic Block), що дає можливість збільшити кількість входів синтезованої логічної функції в межах одного CLB до семи змінних. Структура зазначених вище модулів (LUT, Slice, CLB) представлена на рис. 4.1. Тут значення функції від семи змінних формується на виході мультиплексора MUXF7. Подальше збільшення розрядності вхідних змінних пов'язане з примусовою структурною організацією компонентів CLB в ієрархічні структури більш високого рівня: 1 CLB здатний формувати будь-яку функцію від 7 змінних, 2 CLB – 8 змінних, 4 CLB – 9 змінних, 8 CLB – 10 змінних. У загальному випадку функціональна залежність кількості змінних n від кількості CLB-блоків N має наступний вигляд:

$$n = \log_2 N + 7.$$

Для оцінки апаратної складності при синтезі або реалізації обчислювальної структури необхідно мати зворотну залежність кількості логічних блоків CLB від складності логічної функції, приведеної до кількості

змінних n або потужності куба функціональності $N = 2^n$. Наприклад, для функції від n=10 змінних необхідно мати на кристалі 8 CLB. У загальному випадку число логічних блоків CLB, в залежності від кількості змінних синтезованої функції, так само

$$N = 2^{n-7}, \; n = 7,8,9, \ldots.$$

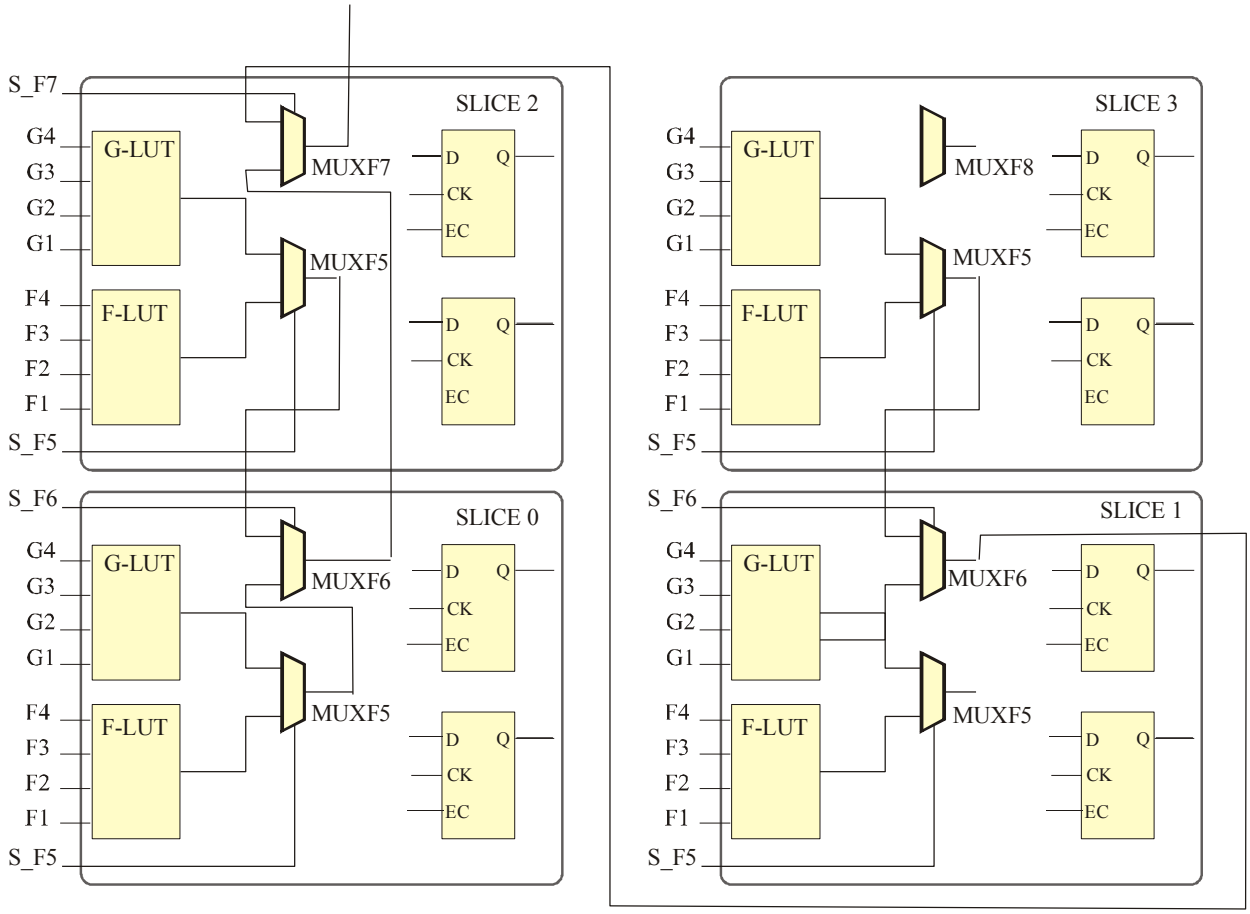



Рис. 4.1 – Структура моделей PLD

Що стосується кількості примітивів LUT, то тут оцінка апаратної складності буде представлена наступним виразом:

$$N^* = 8L \times 2^{n-7} + 4M \times 2^{n-7} = (8L + 4M) \times 2^{n-7}.$$

Тут 8L, 4M – апаратна складність реалізації LUT і мультиплексора, що входять до складу CLB. При цьому важливі характеристики реалізації

функціональності – структурна глибина, як кількість рівнів ієрархії, виражена у кількості LUT і/або мультиплексорах на найдовшому логічному шляху і його затримка D – також залежать від числа змінних (затримки мультиплексора):

$$S = (n - 3);\ D = (n - 3) \times D_M.$$

Приклад: є куб цифрової структури, що містить $p = 4096$, $(n = 12)$ бітів (змінних).

Післясинтезні характеристики такої функціональності матимуть вигляд:

$$N = 2^{12-7} = 32;$$
$$N^* = (8L + 4M) \times 2^{n-7} = (8L + 4M) \times 2^5 = 32 \times (8L + 4M);$$
$$S = (12 - 3) = 9;$$
$$D = 9 \times D_M.$$

При наявності часових характеристик примітивів для конкретного кристалу програмовної логіки, а також оцінок апаратної вартості реалізації структурних компонентів (LUT, Slice, CLB) можна підрахувати точніше параметри функціональності, імплементованої в кристал PLD (FPGA).

Запропоновано кубітні (квантові) структури даних і обчислювальних процесів для істотного підвищення швидкодії при вирішенні задач дискретної оптимізації та відмовостійкого проектування. Представлено суперпозиційний метод синтезу куба функціональності для її імплементації в кристали програмовної логіки.

### 4.6 Модель квантового процесора

Квантовий процесор може бути будь-якої кінцевої розмірності: вектор, матриця, куб. Для структури, що містить два виміри, він представлений матрицею стовпців або Q-векторів, які формують відповідні їм комірки M-вектора моделювання (рис. 4.2, ліва частина). Вектор M, спільно з X-вектором кортежів вхідних змінних примітивів створює структуру взаємних зв'язків між стовпцями-елементами. Адреса комірки Q-покриття, що формує стан невхідного і-розряду M-вектора, визначається вмістом комірок M-вектора,

знайдених за адресами, заданими і-кортежем вектору вхідних змінних. Кожен вектор Qi, так само як і кортеж Xi вектора номерів вхідних ліній, має адресний зв'язок з Mi-коміркою вектора моделювання. Квантовий процесор може входити компонентом до складу більш складної системи. Квантова модель процесора має таку структуру:

$$W = < Q, M, X >,$$
$$Q = (Q_1, Q_2, ..., Q_i, ..., Q_n),$$
$$Q_i = (Q_{i1}, Q_{i2}, ..., Q_{ij}, ..., Q_{ik_i});$$
$$M = (M_1, M_2, ..., M_i, ..., M_n);$$
$$X = (X_1, X_2, ..., X_i, ..., X_n),$$
$$X_i = (X_{i1}, X_{i2}, ..., X_{ij}, ..., X_{im_i});$$
$$M_i = Q_i[M(X_i)];\ k_i = 2^{m_i}.$$

В аналітичній моделі W представлені: 1) Упорядкована адресно-доступна Q-сукупність квантових примітивів, які формують функціональність системи. 2) Вектор моделювання M, що зв'язує всі примітиви в єдину систему на основі ідентифікації еквіпотенційних ліній, які створюють формат з істотних змінних: вхідних, внутрішніх і вихідних. 3) Вектор X кортежів упорядкованих номерів вхідних змінних для кожного квантового примітиву, які формують адреси доступу до комірок Q-векторів примітивів (рис. 4.2, ліва частина). Вектор кількості вхідних змінних примітиву $|X|$ формує адресний простір або довжину кожного Q-покриття. Його можна представити у вигляді таблиці кортежів вхідних змінних, які формують номери ліній вектора моделювання для обчислення адрес доступу до квантових покриттів (рис. 4.2, середня частина). Таблицю кортежів можна також представити у вигляді матриці масок входів, визначених у форматі вектора моделювання, для паралельного формування адрес і одночасного зчитування вихідних станів примітивів з матриці Q-покриттів (рис. 4.2, права частина). Зі структури X-матриці вхідних ліній видно, що кванти, які формують виходи: (8, 9, 10), (11, 12) і (13, 14) можна обробляти паралельно. 4) Характеристичне рівняння, що задає алгоритм функціонування квантового процесора на основі використання

тільки операцій транзакції (зчитування-запис) між Q-векторами примітивів і вектором моделювання.

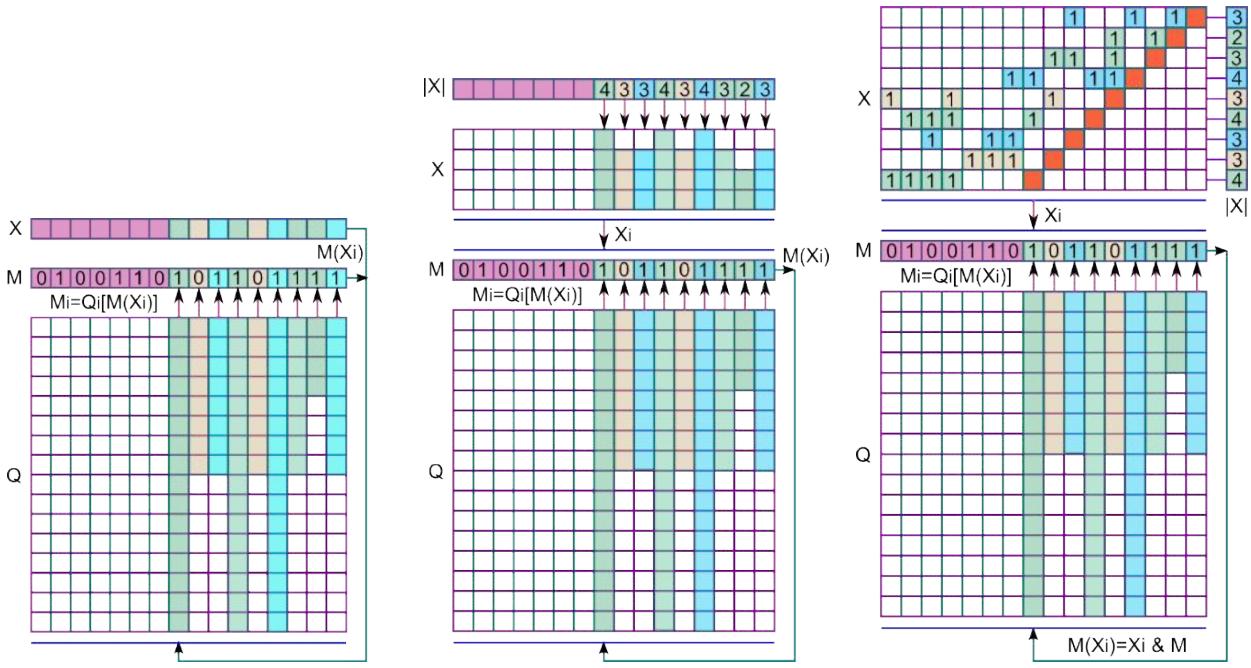



Рис. 4.2 – Кубітні структури даних квантового процесору

Схема цифрового пристрою, яка відповідає наведеному вище опису структур даних: M-вектор моделювання, X-матриця входів і Q-матриця покриттів, представлена на рис. 4.3. Вона містить 9 примітивів, кожен з яких має Q-покриття в формі квант-вектора, що реалізує певну функціональність. Особливість квантових структур даних, що представляють модель цифрової схеми, полягає в повній адресовності всіх компонентів пристрою без гальванічних проводових з'єднань.

Аксіоми квантового (only memory-based) процесора: 1) В квантовому процесорі немає нічого, крім адресовної пам'яті. 2) Обчислювальний процес представлений єдиною універсальною транзакцією між адресовними компонентами пам'яті Mi = Qi [M(Xi)]. 3) Транзакція є універсальною процедурою зчитування-запису даних на непорожній множині адресовних елементів пам'яті. 4) Всі компоненти пам'яті є online-repaired, завдяки їх псевдогальванічній адресній (address-connected) зв'язності. 5) Комбінаційні логічні елементи (reusable logic), так само як і послідовні (sequential

components), виконуються на елементах пам'яті. 6) З'єднання всіх компонентів в обчислювальну систему здійснюється за допомогою (цифрової) ідентифікації псевдо-гальванічних з'єднань вхід-вихідних змінних компонентів схеми, які формують вектор моделювання, який зберігає стани всіх істотних ліній цифрової системи. 7) Всі компоненти квантової моделі цифрової системи: W=<Q,M,X>, у тому числі функціональні модулі, вектор моделювання, вектор адрес вхідних змінних, є online перепрограмовними, тобто – online ремонтопридатними. 8) Примітив цифрової системи має формат W=<Q,Y,X>, оскільки окремий елемент не має зв'язків і вектора M, що створюють систему з окремих компонентів.

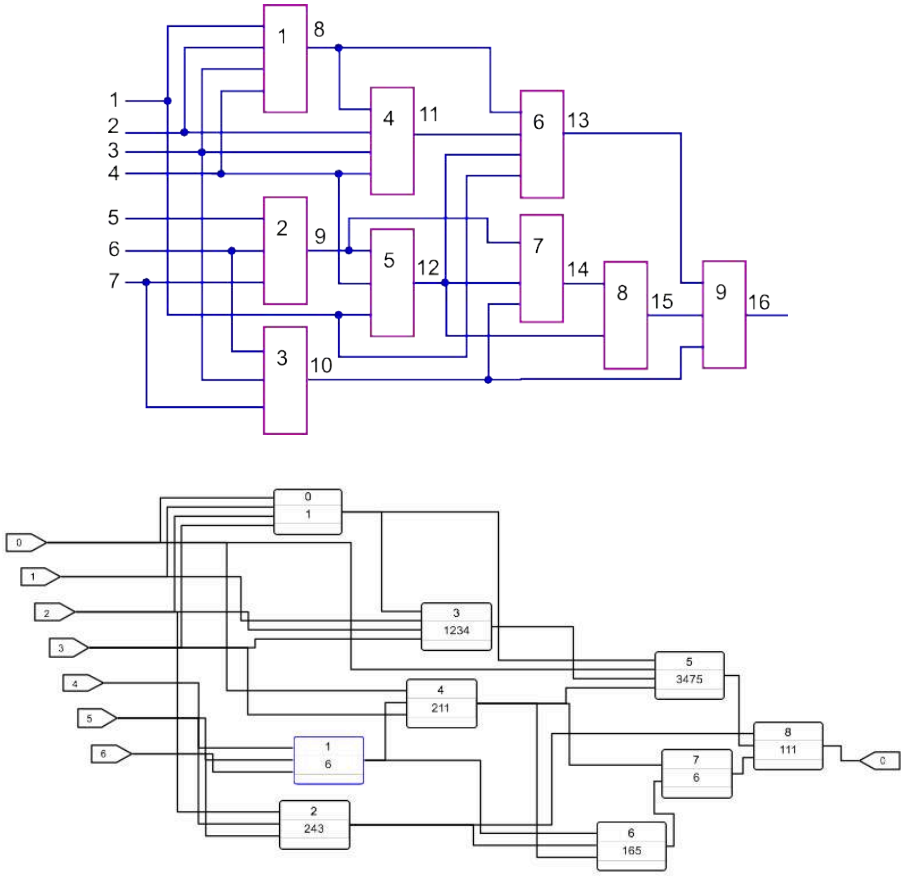
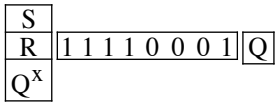


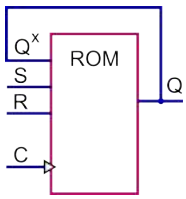

Рис. 4.3 – Схемна модель цифрового пристрою

Згідно введеної квантової моделі, описи послідовних примітивів (тригери, регістри, лічильники) можна представляти Q-покриттями або кубітними векторами, які мають псевдозмінні для завдання внутрішнього

стану. Наприклад, функціональний опис SR-тригера трансформується в квантовий примітив, заданий Q-покриттям, а потім реалізується на адресовному елементі пам'яті FPGA з діаграмами перевірки, які наведено на рис. 4.4.

| S | R | $Q^X$ | Q |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |



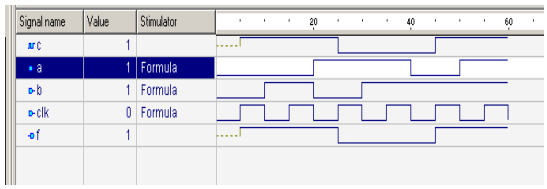
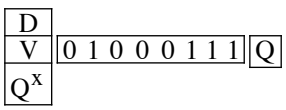



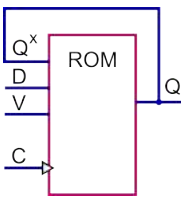

Рис. 4.4 – Реалізація SR-тригера на елементі пам'яті

Таблиця істинності тригера представлена в формі вектора вихідних станів $Q(S,R,Q^X)=(11110001)$, який записується в елемент постійної пам'яті, що підтримує три адресних входи, сигнал синхронізації, а також зворотний зв'язок, який з'єднує вихід елемента пам'яті з одним адресним входом. HDL-реалізація в системі проектування Active HDL 9.1 (Aldec Inc.), а також результати верифікації синтезованого SR-тригера підтверджують коректність схемотехнічного рішення.

Інший приклад пов'язаний з синтезом на елементі постійної пам'яті синхронного DV-тригера. Таблиця істинності тригера трансформована у вектор вихідних станів $Q(D,V,Q^X)=(01000111)$, який записується в елемент пам'яті, що має три адресних входи, сигнал синхронізації, а також зворотний зв'язок, якій з'єднує вихід примітиву пам'яті з одним адресним входом. Зазначені компоненти, у тому числі часові діаграми верифікації HDL-коду моделі DV-тригера представлені на рис. 4.5.

| D | V | $Q^X$ | Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



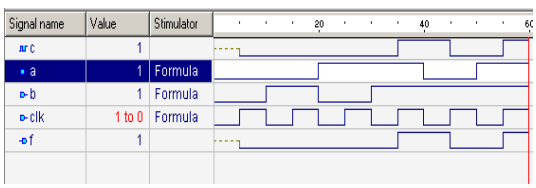
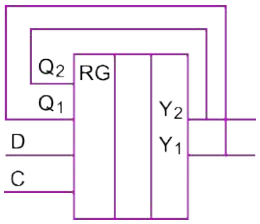
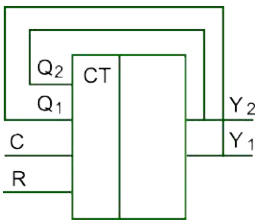




Рис. 4.5 – Реалізація DV-тригера на елементі пам'яті

На рис. 4.6 наведено моделі двох послідовних примітивів: дворозрядного регістра та лічильника. Їх відмінність полягає в завданні двох виходів, стани яких формуються однією і тією ж множиною вхідних змінних.




| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| D | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Q1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Q2 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| Y1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Y2 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |




| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| C | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| Q1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| Q2 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| Y1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| Y2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |

Рис. 4.6 – Memory-based моделі регістра та лічильника

Регістр на змінних (C, D, Q1, Q2, Y1, Y2) виконує функцію зсуву вправо інформації від входу D за розрядами: (D-Y1-Y2), при R=1, і збереження даних при C=0. Лічильник, визначений на змінних (R, C, Q1, Q2, Y1, Y2), реалізує функцію інкремента за розрядами (Y1, Y2), при RC=(11), а також режим зберігання інформації, при (R or C=0 ). Таким чином, для реалізації дворозрядного регістра або лічильника необхідно два 16-бітових елементи пям'яті, що працюють синхронно від одних і тих же входів:

| | | |
|---|---|---|
| C | | |
| D | 0 0 1 1 0 0 1 1 0 0 0 0 1 1 1 1 | Y1 |
| Q1 | 0 1 0 1 0 1 0 1 0 0 1 1 0 0 1 1 | Y2 |
| Q2 | | |

| | | |
|---|---|---|
| R | | |
| C | 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 | Y1 |
| Q1 | 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 | Y2 |
| Q2 | | |

Тут кожна квантова модель представлена двома векторами, де кожен з них формує функцію розряду регістра або лічильника, як стан комірки вектора, отриманої при формуванні адреси A вхідними змінними: {Y1, Y2} = A(C, D, Q1, Q2), {Y1, Y2} = A(R, C, Q1, Q2) відповідно. Моделювання

примітиву зводиться до тривіальної процедури формування адреси, за якою знаходиться стан виходу примітиву, як вміст комірки квантового вектора.

## 4.7 Алгоритм моделювання квантових покриттів цифрових компонентів

Використовує memory-based only моделі для адресного аналізу цифрових систем з метою їх верифікації. Реалізація таких структур пов'язана з комиркамі пам'яті (LUT (Look Up Table) FPGA), які здатні зберігати інформацію у вигляді Q-вектора, де кожен біт або розряд має свою адресу, що ототожнюється з вхідним словом. Програмна реалізація алгоритму моделювання таких структур стає конкурентоспроможною за швидкодією на ринку проектування цифрових систем на кристалах за рахунок адресації функціональних примітивів.

Одновимірній Q-вектор опису функціональності можна прив'язати до вихідної (внутрішньої) лінії пристрою, стан якої формується в процесі моделювання розглянутого Q-покриття. Тоді регістрова реалізація комбінаційного пристрою може бути представлена вектором моделювання M, невхідні лінії якого безпосередньо пов'язані з виходами функціональних елементів. Впорядковані значення вхідних змінних задають адресу біта Q-вектора, що формує стан розглянутої невхідної лінії. Якщо функціональності описуються одновиходовими примітивами, то кожен з них можна ототожнити з номером або координатою невхідної лінії, на яку навантажений даний елемент. Якщо функціональність багатовиходова, то Q-покриття являє собою таблицю з кількістю рядків, що дорівнює кількості виходів. Ефект від такого примітиву полягає в паралелізмі одночасного обчислення станів кількох виходів за одне звернення до матриці за поточною адресою. Дана обставина є істотним аргументом на користь синтезу узагальнених кубітів для фрагментів цифрового пристрою або всієї схеми з метою їх паралельної обробки на одному часовому такті. Модель функціонування цифрової структури спрощується до обчислення двох адрес при формуванні вектора моделювання

$M_i = Q_i[M(X_i)]$ шляхом виключення складної адреси виходу примітиву в процесі запису станів виходів в координати М-вектора.

Алгоритм моделювання квантових примітивів цифрової системи використовує аналітичну структуру (k – кількість вхідних змінних і-примітиву, * – операція конкатенації бітів, А – адреса біта Q-вектора):

$$\boxed{M_i = Q_i(A)} \leftarrow \boxed{A = \overset{k}{\underset{j=1}{*}} M(X_{ij})} \leftarrow \boxed{\begin{matrix} M \\ X_i \\ Q_i \end{matrix}}$$

Даному аналітичному виразу можна поставити у відповідність наступні пункти алгоритму формування двійкових станів М-вектора моделювання цифрової схеми, зображені на рис. 4.7: 0) Ініціювання початкових умов і параметрів. 1) Завдання чергового набору двійкових станів на вхідних координатах вектора моделювання. 2) Визначення і-номера чергового оброблюваного примітиву шляхом виконання операції інкрементування. 3) Виконання процедури конкатенації станів бітів М-вектора, відповідних номерам вектора вхідних змінних Xi. Зчитування відповідного біта з функціонального кубіт-покриття Qi за допомогою бінарної вектор-адреси сконкатенованих бітів М-вектора. Занесення зчитаного з кубіта біта у вектор моделювання М за адресою і. (М-вектор може мати координати з символами X, що дає можливість виконувати потрійне моделювання цифрових пристроїв для вирішення задач тестування і верифікації.) 4) Якщо не всі примітиви оброблені i<n, виконується перехід до пункту 2 алгоритму. 5) Якщо не всі вхідні набори оброблені t<m, виконується перехід до пункту 1. 6) Кінець моделювання.

Виходячи з характеристичного рівняння квантової моделі цифрової системи можна зробити висновок, що сучасний <MQT> (Memory-Quant-Transaction) процесор слід представляти як адресну організацію структури функціональних примітивів пам'яті без гальванічних або проводових зв'язків, на яких визначені адресні транзакції даних в часі і просторі для досягнення поставленої мети.

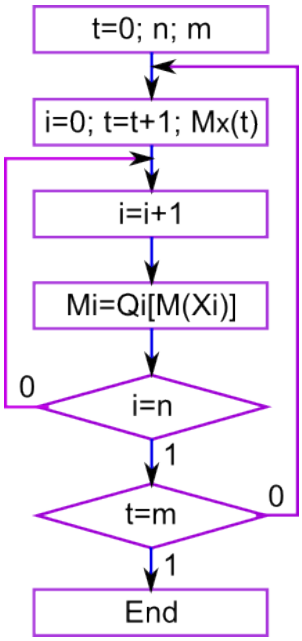
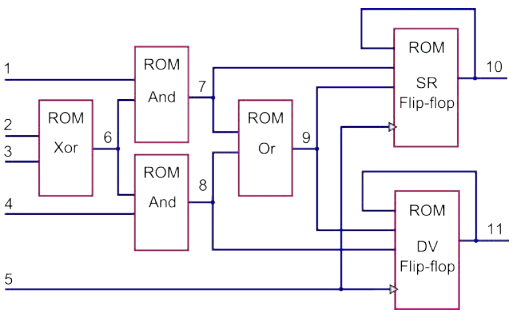



Рис. 4.7 – Алгоритм моделювання квантових покриттів цифрової системи

На рис. 4.8 представлена схема з тригерами і комбінаційної логікою, яка також описана у вигляді елементів пам'яті, куди занесено вихідні стани таблиці істинності кожного логічного елемента. Структури даних, необхідні для моделювання цифрового пристрою зведено в таблицю, де основними компонентами є: M – вектор моделювання або стану занумерованих ліній, який в даному випадку має 5 вхідних, 6 внутрішніх і вихідних ліній, стани яких підлягають визначенню; X – вектор кортежів номерів вхідних ліній примітивів, які необхідні для формування адреси з метою отримання по ній стану виходу елемента Qi, функціональність якого задається Q-вектором.




| L | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| M | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| X | . | . | . | . | . | 2 | 1 | 6 | 7 | 10 | 11 |
| | . | . | . | . | . | 3 | 6 | 4 | 8 | 7 | 9 |
| | . | . | . | . | . | . | . | . | . | 9 | 8 |
| Q | . | . | . | . | . | 0 | 0 | 0 | 0 | 1 | 0 |
| | . | . | . | . | . | 1 | 0 | 0 | 1 | 1 | 1 |
| | . | . | . | . | . | 1 | 0 | 0 | 1 | 1 | 0 |
| | . | . | . | . | . | 0 | 1 | 1 | 1 | 1 | 0 |
| | . | . | . | . | . | . | . | . | . | 0 | 0 |
| | . | . | . | . | . | . | . | . | . | 0 | 1 |
| | . | . | . | . | . | . | . | . | . | 0 | 1 |
| | . | . | . | . | . | . | . | . | . | 1 | 1 |

Рис. 4.8 – Memory-based комбінаційна схема з тригерами

Приклад виконання алгоритму моделювання схемної квантової структури. Всі примітиви повинні бути впорядковані за принципом: черговий елемент аналізується, якщо все попередники для нього були оброблені. В процесі моделювання адресно витягнутий стан комірки поточного Q-покриття заноситься в розряд Мі вектора моделювання. Результати послідовної обробки всіх Q-векторів схемної структури формують стани ліній М-вектора, для наведеного вище прикладу комірки (6 - 11). Початкові стани невизначеностей на псевдовходах функціональних примітивів довизначають сигналами нуля або одиниці в залежності від внутрішньої технологічної культури компанії, що надає промислові засоби моделювання і верифікації. Кількість вхідних змінних примітиву q пов'язана з довжиною Q-вектора співвідношенням: $\mathrm{card}(Q) = 2^q$. Правильність роботи алгоритму моделювання була верифікована на тестових і реальних схемах із залученням засобів Active HDL 9.1 (Aldec Inc.). Особливість структурно-функціонального завдання цифрової системи полягає в поданні всіх примітивів елементами пам'яті, куди записуються Q-вектори вихідних станів.

Висновки: 1) Будь-які структурні компоненти обчислювальних пристроїв, комбінаційні і/або послідовностні, а також системи в цілому можна описувати кубітними Q-векторами і реалізовувати в елементах пам'яті FPGA, CPLD або VLSI. Це надає ринку електронних технологій можливість не використовувати комбінаційну reusable логіку при синтезі обчислювальних пристроїв, яка доставляє розробникам серйозні проблеми, пов'язані з тестуванням, верифікацією і ремонтом жорсткої проводової реалізації цифрових виробів. 2) Memory-based інтерпретативне адресно-орієнтоване моделювання комбінаційних і послідовностних примітивів цифрових пристроїв стає порівнянним за швидкодією з компілятивним аналізом дискретних об'єктів. Крім того, стає можливим реалізовувати на програмовних логічних пристроях апаратне моделювання цифрових систем, де комбінаційні

і послідовносні функціональні примітиви будуть представлені стандартними елементами пам'яті, в які зашиваються Q-вектори.

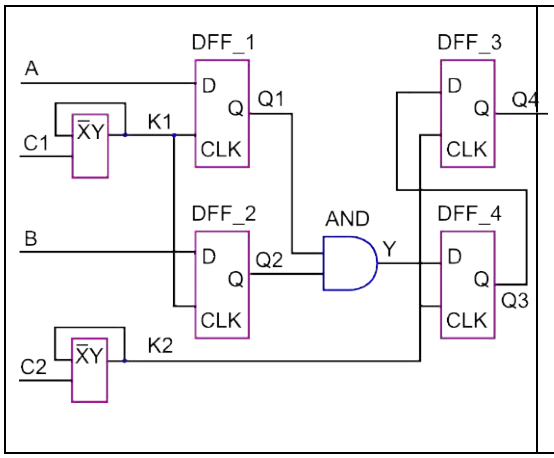
4.8 Моделювання синхронних цифрових схем

Сигнали синхронізації доставляють певні незручності для опису моделей послідовностних компонентів (тригери, регістри, лічильники) і реалізації алгоритмів аналізу. Це пов'язано зі схемотехнічним виконанням управління за переднім або заднім фронтом, які дозволяють виконання транзакцій між master-slave компонентами. Іншими словами, синхронні примітиви мають два послідовно з'єднаних елементі, орієнтовані на низький і високий рівні сигналів запису даних в перший і другий момент часу відповідно. Однак для логічного моделювання урахування подробиць, пов'язаних зі схемотехнічними рішеннями, може істотно уповільнити час аналізу цифрових схем. Тому тут необхідні логічно адекватні моделі реальних процесів, що приводять до підвищення швидкодії алгоритмів обробки компонентів. При цьому накладається обмеження, пов'язане тільки з адресним характером аналізу всіх компонентів схеми. Для забезпечення можливості розгляду синхровходу, як логічної змінної, що формує адресу квантового вектора, пропонується модель розбиття послідовностного примітиву на два елементи: 1) Логічний квант видачі дозвільного сигналу при формуванні переднього (заднього) фронту в двох часових модельних тактах. 2) Квант реалізації штатної функціональності (тригера, регістра, лічильника) послідовностного компонента. Таким чином, модель синхронного D-тригера може бути описана у формі двох Q-покриттів, адресно обчислюючих стани виходів:

| | | |
|---|---|---|
| CLK(t – 1)<br>CLK(t) | 0 1 0 0 | C(t) |

| | | |
|---|---|---|
| Q(t – 1)<br>D(t)<br>C(t) | 0 0 0 1 0 0 1 1 | Q(t) |

С урахуванням викладеного вище, модель цифрової схеми з двома сигналами синхронізації, представлена на рис. 4.9, буде мати структури даних, що складаються з сукупності Q-покриттів, які формують поточний вектор моделювання M, але з урахуванням значень координат даного вектора в попередній момент часу M(t-1). Збільшення кількості змінних за рахунок введення двох елементів синхронізації зменшує сукупну розмірність таблиці квантових векторів, яка при 7 змінних матиме 56 координат.




| L | A | $C_1$ | B | $C_2$ | $K_1$ | $Q_1$ | $Q_2$ | Y | $K_2$ | $Q_3$ | $Q_4$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $M^{t-1}$ | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $M^t$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X | . | . | . | . | $C_1^{t-1}$ | $Q_1^{t-1}$ | $Q_2^{t-1}$ | $Q_1$ | $C_2^{t-1}$ | $Q_3^{t-1}$ | $Q_4^{t-1}$ |
| | . | . | . | . | $C_1^t$ | A | B | $Q_1$ | $C_2^t$ | Y | $Q_3$ |
| | . | . | . | . | . | $K_1$ | $K_1$ | . | . | $K_2$ | $K_2$ |
| Q | . | . | . | . | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | . | . | . | . | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| | . | . | . | . | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | . | . | . | . | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| | . | . | . | . | . | 0 | 0 | . | . | 0 | 0 |
| | . | . | . | . | . | 0 | 0 | . | . | 0 | 0 |
| | . | . | . | . | . | 1 | 1 | . | . | 1 | 1 |
| | . | . | . | . | . | 1 | 1 | . | . | 1 | 1 |

Рис. 4.9 – Схема з D-тригерами і елементами синхронізації

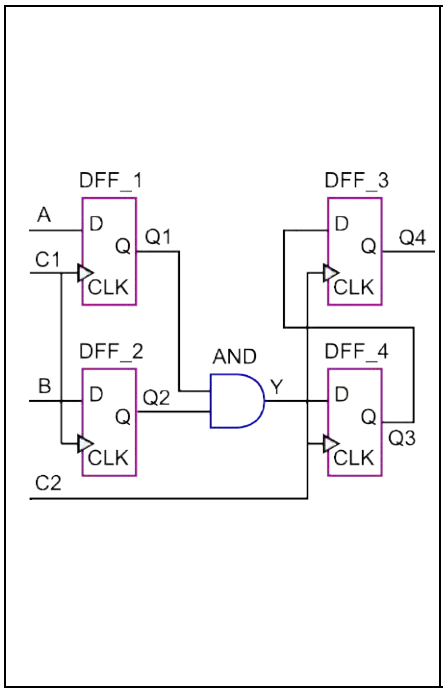
Якщо не вводити дві додаткові змінні (елементи синхронізації), то обсяг пам'яті для Q-покриттів буде дорівнювати 80 коміркам (рис. 4.10).

Наведена схемна реалізація максимально орієнтована на структури даних промислових засобів моделювання і верифікації. Однак квантові вектори для завдання функціональностей тригерів створюють необхідні умови для підвищення швидкодії інтерпретативного аналізу, тестування і діагностування схемних компонентів.

Проблема зменшення сукупного обсягу Q-покриттів схеми пов'язана з кількістю змінних, які формують адреси координат Q-вектора. Природно, що будь-яке розбиття кількості змінних на дві рівних упорядкованих підмножини дає можливість істотно зменшити розмірність пам'яті для запису вже двох Q-векторів. У загальному випадку функціональна залежність зменшення

розмірності вихідного Q-вектора, визначеного на n-змінних, при розподілі на 2 підсхеми з рівним числом результуючих змінних (n/2), має наступний вигляд:

$$Q = \frac{2^n}{2^{n/2} + 2^{n/2}} = \frac{2^n}{2 \times 2^{n/2}} = \frac{2^n}{2^{n/2+1}} = 2^{n-(n/2+1)} = 2^{n/2-1}.$$




| L | A | $C_1$ | B | $C_2$ | $Q_1$ | $Q_2$ | Y | $Q_3$ | $Q_4$ |
|---|---|---|---|---|---|---|---|---|---|
| $M^{t-1}$ | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $M^t$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X | . | . | . | . | $Q_1^{t-1}$ | $Q_2^{t-1}$ | $Q_1$ | $Q_3^{t-1}$ | $Q_4^{t-1}$ |
| | . | . | . | . | A | B | $Q_2$ | Y | $Q_3$ |
| | . | . | . | . | $C_1^{t-1}$ | $C_1^{t-1}$ | . | $C_2^{t-1}$ | $C_2^{t-1}$ |
| | . | . | . | . | $C_1^t$ | $C_1^t$ | . | $C_2^t$ | $C_2^t$ |
| Q | . | . | . | . | 0 | 0 | 0 | 0 | 0 |
| | . | . | . | . | 0 | 0 | 0 | 0 | 0 |
| | . | . | . | . | 0 | 0 | 0 | 0 | 0 |
| | . | . | . | . | 0 | 0 | 1 | 0 | 0 |
| | . | . | . | . | 0 | 0 | . | 0 | 0 |
| | . | . | . | . | 1 | 1 | . | 1 | 1 |
| | . | . | . | . | 0 | 0 | . | 0 | 0 |
| | . | . | . | . | 0 | 0 | . | 0 | 0 |
| | . | . | . | . | 1 | 1 | . | 1 | 1 |
| | . | . | . | . | 0 | 0 | . | 0 | 0 |
| | . | . | . | . | 1 | 1 | . | 1 | 1 |
| | . | . | . | . | 1 | 1 | . | 1 | 1 |
| | . | . | . | . | 1 | 1 | . | 1 | 1 |
| | . | . | . | . | 1 | 1 | . | 1 | 1 |
| | . | . | . | . | 1 | 1 | . | 1 | 1 |
| | . | . | . | . | 1 | 1 | . | 1 | 1 |

Рис. 4.10 – Схема з D-тригерами на основі внутрішньої синхронізації

Наприклад, розбиття вектора з восьми змінних на два Q-покриття з 4 змінними дає зменшення обсягу пам'яті у 8 разів. Однак слід мати на увазі, що кожне розбиття функціонального модуля на k підсхем відносно зовнішніх входів потребує k додаткових d-циклів для обчислення стану виходу всієї схеми $T = d(k+1)$. Зниження швидкодії розбиття функціональності є платою за істотне зменшення обсягу пам'яті для зберігання структур даних цифрової системи. У загальному випадку розбиття функціональності на k однакових частин призводить до отримання такої залежності виграшу обсягу пам'яті від числа розбиттів на підмножини вектора вхідних змінних:

$$Q = \frac{2^n}{k \times 2^{n/k}} = \frac{1}{k} \times 2^{n-n/k}.$$

Тут параметр розбиття k приймає значення, кратні ступеню двійки: 2, 4, 8, 16. Однак значення k не повинно бути більше, ніж n/2. Однак слід зауважити, що необов'язково кількість розбиття має приймати значення,

кратні ступеню двійки. У загальному випадку, на векторі вхідних змінних може існувати m розбиттів, кожне з яких має більше однієї змінної. При цьому виконується умова розбиття, що сума всіх змінних, які беруть участь у розбитті, не може бути більше n:

$$Q = \frac{2^n}{2^{k_1} + 2^{k_2} + \ldots + 2^{k_i} + \ldots + 2^{k_m}},\ k_1 + k_2 + \ldots + k_i + \ldots + k_m = n.$$

Формула показує виграш від розбиття функціональності у вигляді відношення розмірності вихідного квант-вектора до сукупного обсягу Q-векторів, отриманих після розбиття. Щоб оцінити ефективність розбиття функціональності на схемні фрагменти, необхідно враховувати не тільки зменшення обсягу пам'яті для зберігання структур даних, але й негативні наслідки, пов'язані з вартістю аналізу збільшеної кількості схемних компонентів, яка регламентується в кожному конкретному випадку коефіцієнтом d:

$$Q = \frac{2^n}{d \times m \times (2^{k_1} + 2^{k_2} + \ldots + 2^{k_i} + \ldots + 2^{k_m})}.$$

Підводячи підсумок в частині модифікації теорії справного моделювання (fault-free) цифрових систем, можна відзначити наступні факти. Автомат моделювання синхронних цифрових пристроїв, як правило, представлений моделлю Мура:

$$S(t) = f[X(t), S(t-1)];$$
$$Y(t) = f[X(t), S(t)].$$

Тут фігурують вхідні (X) і внутрішні стани автомата в двох сусідніх часових фреймах S(t), S(t-1), а також правила визначення вихідних значень Y(t) для ініціювання обчислювальних процедур. Пропонується модифікація зазначеної моделі автомата Мура для аналізу цифрових систем, сутність якої полягає в заміні функціональних відносин адресними (A) операціями:

$$S(t) = A[X(t), S(t-1)];$$
$$Y(t) = A[X(t), S(t)].$$

Адресний або квантовий автомат, визначений вище дозволяє: 1) Уникнути жорстких гальванічних міжз'єднань між елементами комбінаційних і послідовностних схем при їх апаратній імплементації тільки в елементи пам'яті. 2) Отримати властивість гнучкої замінюваності компонентів цифрової системи в режимі online, завдяки їх адресовності. 3) Істотно спростити всі процеси моделювання, верифікації та тестування шляхом використання тільки процедур обчислення адреси компонента схеми або комірки його пам'яті. 4) Уніфікувати процеси проектування цифрових виробів шляхом їх зведення до формування функціональностей на основі обчислення адрес або до транзакцій на елементах пам'яті. 5) Підвищити ефективність процедур моделювання цифрових схем за рахунок зменшення обсягу інтерпретативних моделей і спрощення способу їх обробки, коли замість вичерпного аналізу таблиць пропонується обчислення адреси комірки квантового вектора. 6) Виконувати всі обчислювальні процедури на основі використання квантових покриттів і вектора моделювання, заданого в двох сусідніх автоматних тактах, згідно з визначенням квантового автомата. 7) Технологічно простіше стає використовувати інфраструктуру [97,115] стандартів тестопридатного проектування (IEEE 1500, 1149) для покомпонентного тестування, діагностування та відновлення працездатності адресно доступних функціональних блоків в режимі online.

## 4.9 Структура хмарного сервісу QuaSim для моделювання цифрових пристроїв

QuaSim є засобом для аналізу, тестування і верифікації цифрових проектів невеликої розмірності і призначений для використання в навчальному процесі в якості хмарного сервісу, доступного для студентів з будь-якого мобільного пристрою або комп'ютера.

Мета – істотне підвищення якості навчального процесу за рахунок надання технологічних мікро-сервісів аналізу цифрових пристроїв з

одночасною візуалізацією схем, тестів, результатів моделювання і таблиць істинності функціональних елементів.

Задачі: 1) Створення структури хмарного моделювання цифрових пристроїв на платформі компьютингових сервісів Google. 2) Розробка модуля (мікросервіса) Q-element, що реалізує створення моделі та візуалізацію логічного або функціонального примітиву схеми. 3) Проектування модуля генерування кубітних моделей примітивів і більш складних цифрових пристроїв, а також засобів їх оперативної візуалізації. 4) Розробка модуля або панелі управління, що інтегрує симулятор, генератор логічних елементів, схемних конструкцій і вхід-вихідних портів. 5) Розробка власне модуля для аналізу цифрових схем на основі рекурсивної обробки логічних елементів. 6) Створення службових бібліотек для зберігання: готових функціональних елементів, складних цифрових схем, тестів і результатів їх аналізу, а також службової інформації. 7) Тестування і верифікація хмарного сервісу Q-simulator, призначеного для інтерпретативного моделювання цифрових пристроїв.

Сутність квантового методу аналізу полягає в адресній реалізації всіх функціональних компонентів цифрових систем і структур даних, що дає можливість істотно підвищити швидкодію інтерпретативного моделювання та якість обслуговування проектів за рахунок швидкої заміни некондиційних логічних елементів шляхом їх переадресації.

Структура хмарного сервісу містить наступні основні мікросервіси: 1) Q-element генерує квантові описи логічних елементів в структурі цифрової функціональності. 2) Модуль View виконує візуалізацію схемних елементів, портів входів і виходів на екрані монітора. 3) Модуль Collapse керує вікнами монітора та їх розмірами за допомогою відповідних іконок. 4) Контролер Split візуалізує роботу всіх контролерів при складанні схеми на екрані, а також здійснює масштабування деталей проекту. 5) Функція Evaluate формує стан виходу поточного елемента шляхом вибору вмісту з комірки кубіта за її

адресою. 6) Модуль Q-sim реалізує власне алгоритм моделювання всіх ліній схеми шляхом побудови на першому кроці рекурсивної моделі для послідовно-паралельної обробки елементів. На другому кроці обчислюються стани всіх виходів логічних елементів. Моделювання закінчується після обробки всіх тестових вхідних послідовностей. Якщо схема має глобальні або локальні зворотні зв'язки, то моделювання здійснюється до фіксації однакових значень сигналів на всіх лініях схеми. Якщо схема не встановлюється в стійкий стан на вхідних наборах, то фіксується генераторний режим після виконання n (= 20) ітерацій. В цьому випадку всім постійно змінюваним лініям присвоюється значення двійкової невизначеності X={0,1}. 7) Модуль керування бібліотекою елементів, схем і проектів здійснює зчитування, запис і підключення фрагментів. 8) Модуль часових діаграм здійснює візуалізацію тесту з вихідними сигналами на моніторі в формі безперервних сигналів, розділених на такти в абсолютному або модельному часі. 9) Всі модулі хмарного сервісу запрограмовані мовою Swift, з використанням операційної системи OSX 10.9, компілятора XCode 7. Кількість вихідних файлів 36, загальна кількість рядків коду - 1450.

На рис. 4.11 наведено візуалізацію результатів графічного проектування схеми з тригерами на моніторі комп'ютера. Схема повністю відповідає функціональності, представленої на рис. 4.3. Вона містить чотири вхідних порти для подачі робочих або тестових впливів, а також два вихідних порти. Структура містить чотири логічних елементи і два тригери. Мнемонічний опис компонентів схеми приведено до універсальної формі прямокутника, він різниться тільки номером примітиву в складі пристрою, а також типом функціональності, яка задається кубіт-вектором, представленим десятковим числом. У схемі, представленій на рис. 4.11, елементи мають порядкові номери (у верхній частині) і цілі числа для ідентифікації функциональностей: 0/6 - 0110, 1/1 - 0001 2/1 - 0001 3/7 - 0111, 4/143 - 11110001, 5 / 226 - 01000111. Тут двійковий вектор відповідає десятковому еквіваленту числа для завдання

функціональності. Оскільки кубіт-вектор не має в явному вигляді задання вхідних наборів, то його можна розглядати як неявну або компактну форму теоретико-множинної по суті таблиці істинності. Навіщо явно вказувати вхідні значення, якщо вони складають строго послідовну адресацію вихідних значень. Таким чином, таблиця істинності, як сукупність вхідних сигналів і відповідних їм вихідних значень завжди програє перед кубіт-векторною формою подання функциональностей в плані обсягу і швидкодії аналізу даних. Не існує принципових відмінностей між описами комбінаційного елемента, схеми або послідовностного примітиву, оскільки всі вони формально представлені кубіт-векторами, які розміщуються в адресовній пам'яти. Більш того, всі примітиви схеми також є адресовними, а структура схеми також може бути описана у вигляді кубіт-вектора. Таким чином, можна прийти до такої реалізації обчислювального пристрою, де немає нічого крім адресної пам'яті або кубітних векторів різної довжини, в яких функціональності визначаються впорядкованими наборами нульових і одиничних сигналів. Переваги даного сервісу QuaSim кубітного опису і моделювання цифрових пристроїв полягають в наступному: 1) усі функціональні елементи і схеми задаються Q-векторами, що уніфікує процедури синтезу та аналізу цифрових пристроїв; 2) технологічно просто міняти або корегувати функціональність схеми або будь-якого примітиву шляхом заміни окремих бітів Q-вектора; 3) уніфікація кубітної форми опису примітивів схеми дає можливість застосувати до них єдину процедуру аналізу функциональностей, яка зводиться до обчислення адреси $M_i = Q_i[M(X_i)]$, що робить процес програмування хмарного сервісу QuaSim технологічно простим у виконанні і не залежним від функціональної та структурної складності цифрових структур; 4) простий і зрозумілий початківцю графічний інтерфейс робить хмарний сервіс конкурентоспроможним на ринку освітніх послуг, де складні та важкі засоби моделювання від провідних компаній планети є недоступними для університетів у зв'язку з їх високою вартістю, а для

студентів – часовитратними за складністю підготовки HDL-специфікацій при розгляді невеликих навчальних проектів; 5) уніфікація форми опису примітивів створює умови для технологічного вирішення задач синтезу, моделювання несправностей, тестування, верифікації та діагностування, на основі операцій з кубіт-векторами; 6) недоліком кубітної або квантової технології опису та аналізу цифрових структур можна вважати деяке зменшення швидкодії моделювання в порівнянні з існуючими промисловими компіляторами, для ASIC і VLSI проектів, де обсяг reusable logic є домінуючим для досягнення високої швидкодії.

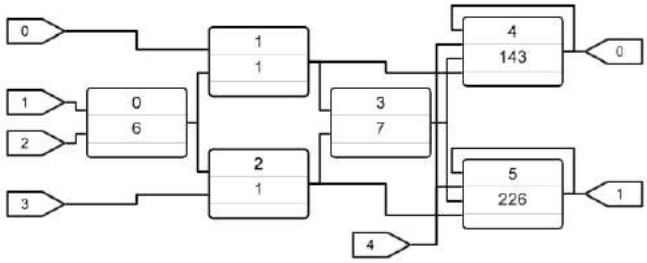



Рис. 4.11 – Скріншот структури з тригерами

Структура взаємодіючих компонентів хмарного сервісу QuaSim представлена на рис. 4.12. Квантове або кубітне представлення моделі цифрового пристрою разом з інтерпретативним симулятором складають ядро системи, інтегрованої у великі дані кіберпростору або Інтернету. Це дає можливість використовувати в якості вихідних даних відкриті специфікації і тестбенчі, описані мовами VHDL, Verilog. Такі дані і/або тестові приклади є практично у всіх провідних компаніях, університетах і тематичних конференціях IEEE, TTTC, ISCAS. Крім того, занурення QuaSim сервісу в інтернет-простір передбачає також вивантаження результатів його роботи, пов'язаних з аналізом і синтезом навчальних або ринково орієнтованих проектів в сервіси зберігання даних на платформах Google, Amazon, Microsoft, IBM, Facebook. Природно, що інтеграція хмарного сервісу з кіберпространством передбачає наявність парсер-мікросервісів для перетворення специфікацій з мов опису апаратури у внутрішню мову QuaSim, а також має існувати і зворотнє перетворення даних з кубітного представлення

в стандарти HDL-мов. Парсерізація забезпечує можливість використання відкритих в інтернеті проектів для їх вивчення і порівняння в системі моделювання QuaSim, а також робить доступними внутрішні проектні рішення QuaSim для всіх бажаючих на ринку освітніх сервісів.

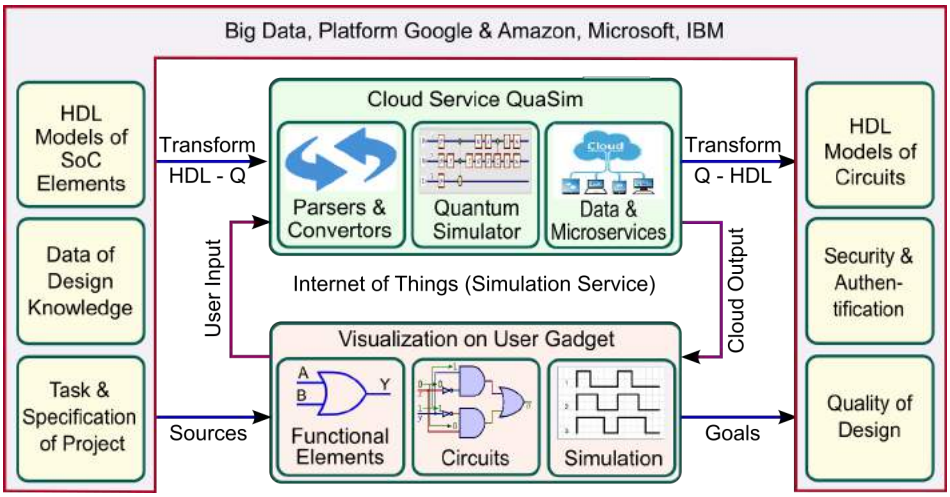



Рис. 4.12 – Хмарний сервіс моделювання цифрових пристроїв

Блок Security контролює доступ користувачів з метою їх статистичного обліку і передбачає аутентифікацію кожного на основі пароля, прізвища, імені, доповнене будь-яким дійсним (корпоративним) атрибутом зі списку: {електронний цифровий підпис, e-mail, цифровий ключ, номер телефону}.

Тестування і верифікація хмарного сервісу моделювання цифрових систем здійснювалася окремо для кожного мікросервісу, а потім у взаємодії всіх модулів: 1) перевірка правильності генерування логічних і більш складних функціональних елементів; 2) перевірка структурного синтезу цифрової схеми і засобів візуалізації; 3) верифікація і тестування алгоритмів двійкового і трійкового синхронного інтерпретативного справного моделювання вхідних впливів на 40 схемах, комбінаційного та послідовностного типів. При цьому використовувалися тестові набори, алгоритмічно генеровані і складені користувачем; 4) перевірка сервісних модулів, що забезпечують працездатність основних мікросервісів: бібліотеки елементів і схем, аутентифікація користувача, модуль формування статистичних даних за проектами і користувачами; 5) верифікація

інтерфейсних мікросервісів, що забезпечують інтеракції між хмарними back-end і призначеними для користувача front-end модулями.

### 4.10 Хмарний сервіс тестування

Представлена доменно і IP-ідентифікована структура взаємодії кіберфізичних компонентів, що реалізують хмарний комп'ютинг-сервіс моделювання і тестування цифрових пристроїв в контейнерно-орієнтованому середовищі компанії Docker, що працює в середовищі Google Computing Engine.

Мета – зменшення часу проектування і верифікації мікросервісів за рахунок контейнерної технології, а також популяризація нових методів кубітного синтезу та аналізу цифрових пристроїв в середовищі фахівців, завдяки своїй хмарної доступності.

Задачі: 1) Створення хмарного середовища проектування та верифікації на основі контейнерів. 2) Розробка мікросервісів моделювання і тестування цифрових пристроїв. 3) Верифікація структури і функціональностей хмарного сервісу.

Дослідження ринку проектування сучасних програмних продуктів показали, що в останні три роки набирають популярність гнучкі і масштабовані технології проектування програмних продуктів, що використовують контейнеризацію, орієнтовані на мінімізацію циклу time-to-maret. Основна ідея контейнера полягає у створенні захищеного і доступного віртуального середовища проектування і експлуатації в кіберпросторі, на основі використання локальної або глобальної апаратури Infrastructure, рис. 4.13.

На інфраструктуру встановлюється практично будь-яка операційна система Host operating system, зручна для розробника, наприклад, Ubuntu, Debian. У локально виконаній hardware інфраструктурі контейнерів можна інсталювати також OSX і Windows.

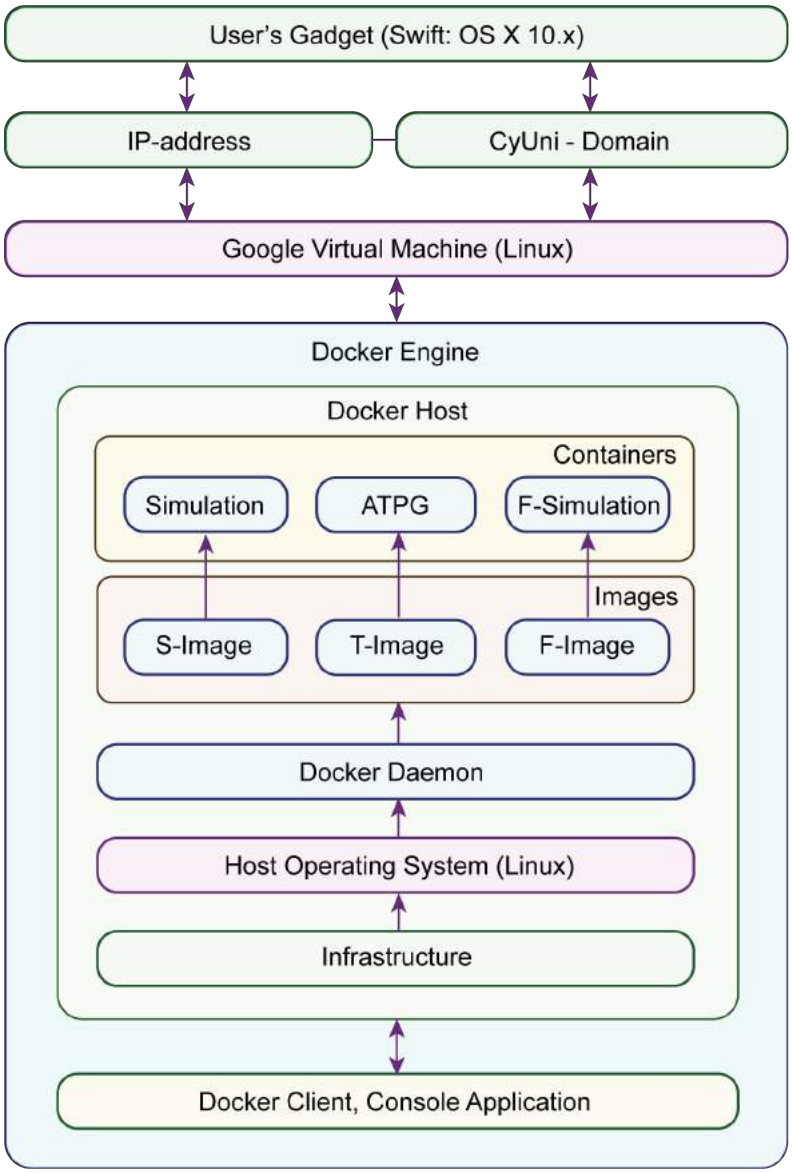



Рис. 4.13 – Хмарний сервіс тестування і верифікації цифрових пристроїв

Далі розробник інсталює Docker Engine, якщо він не встановлений спочатку, який за допомогою Docker Daemon (реалізація на Golang, C), керує контейнерами, запущеними на основі попередньо побудованих образів (Images). Контейнери Containers являють собою легку віртуальну машину, на якій працюють функціональні блоки: Simulation, ATPG, F-simulation, реалізовані на Swift (або Jawa), і відповідні бібліотеки. Docker Engine представляє собою компьютингову систему контейнеризації, яка складається з двох взаємодіючих між собою компонентів: Docker Host, Docker Client (Console Application), які можна інтерпретувати як виконавчий Host і керуючий (Client) механізми хмарної системи тестування і моделювання. Далі

Docker Engine є компонентом, який входить до складу Google Virtual Machine, якій ставиться у відповідність IP-address and CyUni domain, що робить розробку, а далі хмарний сервіс, доступним для фахівців і студентів.

Останньою ланкою у вертикальній зв'язці (Back-End, Front-End) є User's Gadget (Swift OS X.10x). На кінцевому пристрої користувача установлюються додатки, здатні привести результати роботи хмарних сервісів моделювання і тестування до мультивіконного формату візуалізації даних, зручного для сприйняття людиною. Слід зазначити, що набагато більш технологічним є створення Front-End додатку на JavaScript, HTML, CSS, який буде працювати через Internet browser. Це гарантує відсутність блокування клієнта з боку постачальника конкретної ОС (vendor lock). Для мобільних пристроїв доцільніше використовувати додаток індивідуального клієнта під кожну ОС.

Переваги структурної організації хмарного сервісу тестування і моделювання цифрових пристроїв: 1) захищеність і доступність контейнерів для командної розробки і верифікації моделей і методів синтезу та аналізу обчислювальних виробів; 2) глобальна доступність сервісів в кіберпросторі планети для вивчення і застосування інноваційних технологій проектування, що базуються на кубітно-векторних моделях опису цифрових компонентів; 3) контейнерна розширюваність хмарного комп'ютингу при реалізації нових мікросервісів, відповідних додаткових методів і алгоритмів тестування і моделювання; 4) просторово-часова інваріантність проектування і експлуатації кібер-фізичної системи, спрямована на доступність хмарних сервісів з будь-якої точки планети в будь-який час доби; 5) інфрструктурна незалежність хмарного віртуального комп'ютингу забезпечує безвідмовну роботу невидимої апаратури, її своєчасну замінюваність на стороні Google (Amazon, IBM) без додаткових фінансових витрат для власника сервісів.

Тестування і верифікація хмарних сервісів кубітного синтезу тестів, аналізу несправностей і справної поведінки було виконано на стандартних прикладах Benchmark circuits (c5, c17, c432, c499, c880, c1355, c1908, c2670,

c3540, c5315) з бібліотек конференції ISCAS. Приклад опису схемної структури c17 на мові Verilog представлений у вигляді наступного лістингу:

```
// Verilog
// c17
// Ninputs 5
// Noutputs 2
// NtotalGates 6
// NAND2 6
module c17 (N1, N2, N3, N6, N7, N22, N23);
input N1, N2, N3, N6, N7;
output N22, N23;
wire N10, N11, N16, N19;
nand NAND2_1 (N10, N1, N3);
nand NAND2_2 (N11, N3, N6);
nand NAND2_3 (N16, N2, N11);
nand NAND2_4 (N19, N11, N7);
nand NAND2_5 (N22, N10, N16);
nand NAND2_6 (N23, N16, N19);
endmodule
```

Результати порівняльного тестування програмної реалізації методів за основними параметрами: час моделювання справної поведінки і несправностей, представлено діаграмами на рис. 4.14 і 4.15.

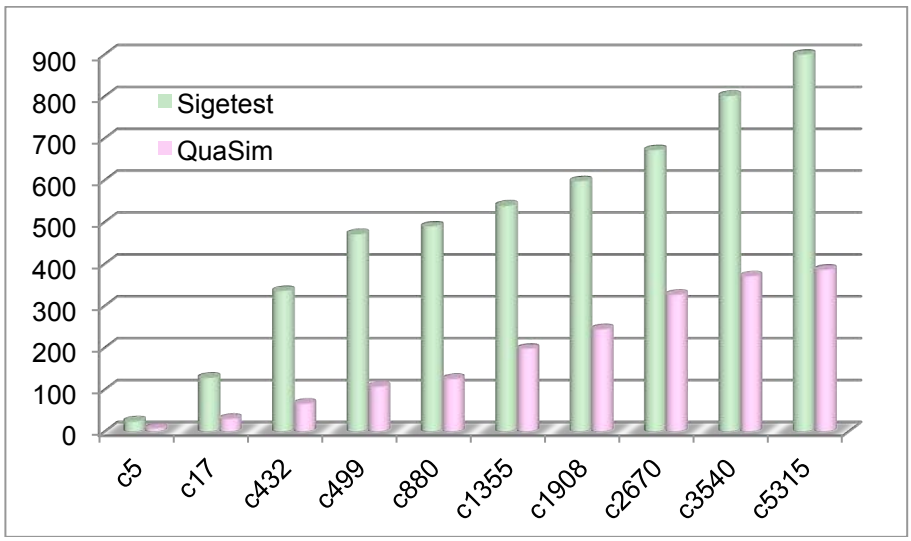



Рис. 4.14 – Порівняльний аналіз двох методів справного моделювання

Моделювання здійснювалося для схем (c5, c17, c432, c499, c880, c1355, c1908, c2670, c3540, c5315) на 10 000 вхідних наборів: 1) базовий метод – програмний продукт SiGeTest [Хаханов В.І., Обрізан В.І., Мірошниченко Я.В., Мельникова О.В. SIGETEST – система моделювання тестів перевірки несправностей цифрових пристроїв // Каталог анотацій на розробки за

матеріалами першого українсько-китайського форуму «Наука-виробництво».– Харків, ХНУРЕ.– 2007.– С. 25-26]; 2) кубітний метод моделювання, реалізація QuaSim. Одиниця виміру – мілісекунда. Порівняльний аналіз синтезу тестів не проводився, оскільки новий метод синтезу тестів реалізований тільки для black box функциональностей, заданих кубітним покриттям.

Виграш у швидкодії справного моделювання на 10 прикладах цифрових схем – не менше, ніж в два рази. Обидва методи використовують інтерпретативний алгоритм, орієнтований на імплементацію в програмовну логіку. Перевага програмної реалізації зазначеного алгоритму полягає в інтерактивному характері модифікації схемної структури в процесі виконання програми моделювання.

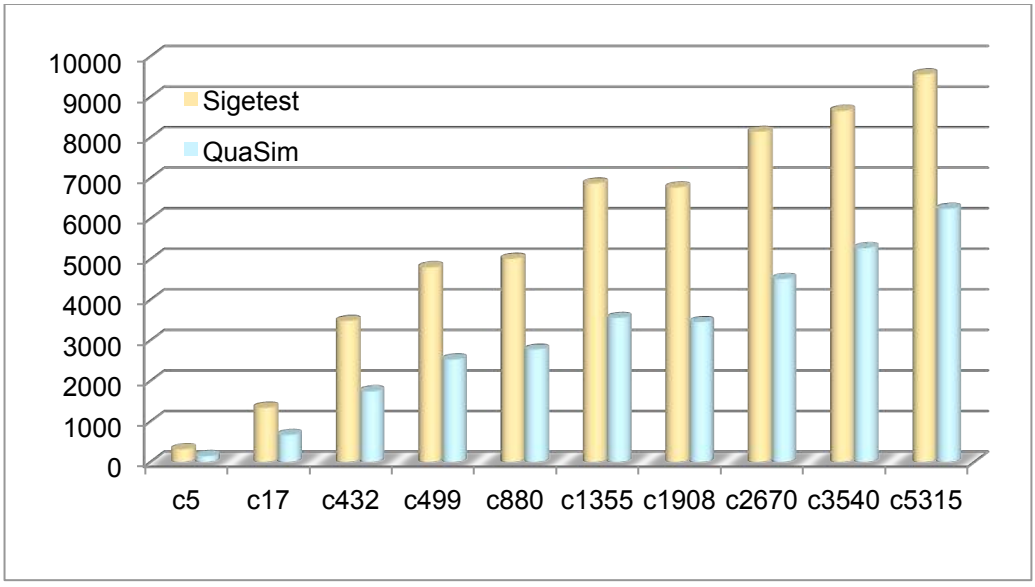



Рис. 4.15 – Порівняльний аналіз двох методів моделювання несправностей

Дедуктивні методи моделювання несправностей, при їх програмній реалізації, на порядок складніші, ніж алгоритми справного моделювання. Тому кількість мілісекунд тут на порядок більше, ніж на попередній діаграмі. Діаграма показує також, що за швидкодією майже в два рази виграє нова реалізація дедуктивного методу QuaSim, яка використовує паралельні операції над кубітами функціональних елементів.

Таким чином, порівняльний аналіз нових методів моделювання, в порівнянні з базовим, показав суттєві переваги у швидкодії, які, поряд з високою технологічністю паралельних операцій, ставлять їх у ряд перспективних напрямів memory-driven проектування цифрових систем на основі використання хмарного комп'ютингу. Інтеграція моделей і методів кубітного аналізу із середовищем моделювання Riviera фірми Aldec Inc представлена на рис. 4.16. Нові структури даних і методи синтезу тестів, справного моделювання та аналізу несправностей, додані в систему, дозволяють удосконалити існуючий процес верифікації та на 12% скоротити час розробки цифрового виробу (time-to-market).

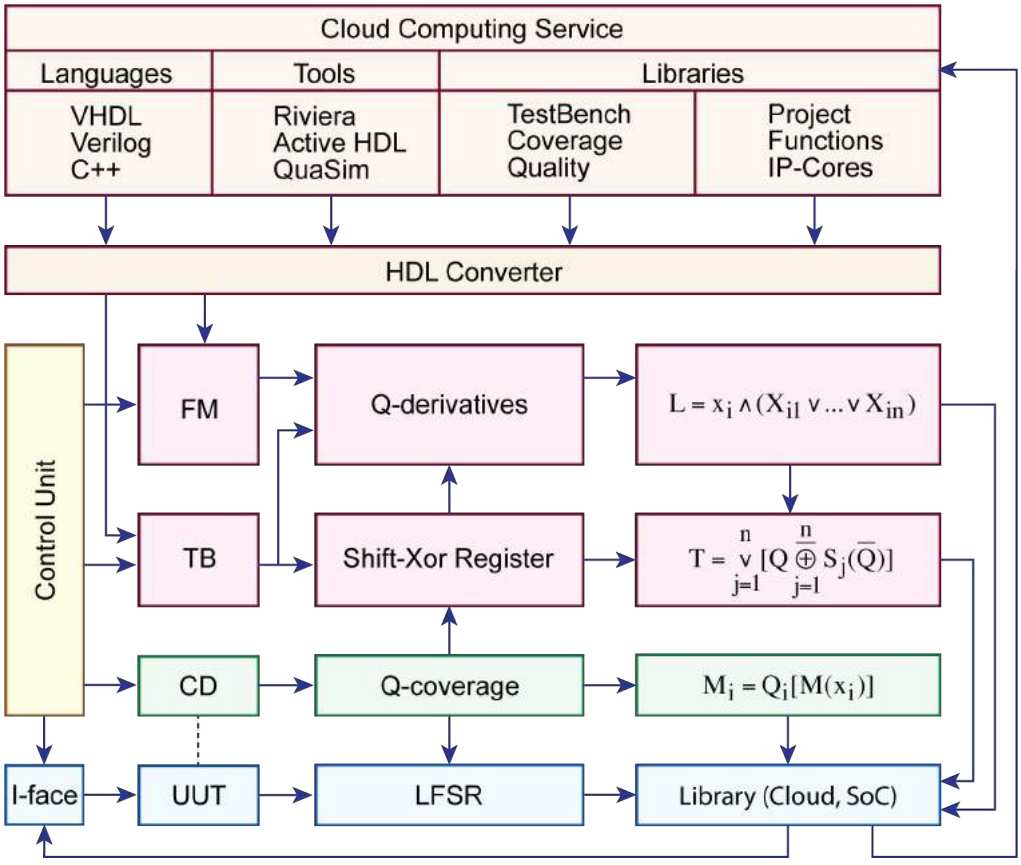



Рис. 4.16 – Інтеграція методів з продуктом Riviera, Aldec

Таким чином, кіберфізична організація взаємодії користувачів з хмарним компьютинговим сервісом є найбільш перспективною формою проектування, тестування цифрових виробів, яка орієнтована на вільний доступ до віртуальних кіберпродуктів з будь-якої точки фізичного простору.

## 4.11 Висновки

Запропоновано новий підхід до проектування цифрових пристроїв, який характеризується: 1) удосконаленим методом інтерпретативного паралельного моделювання компонентів цифрових систем на кристалах, який відрізняється застосуванням автоматної МАТ-моделі, що використовує тільки адресовні структури пам'яті і операції транзакції; 2) методами синтезу та аналізу, що базуються на суперпозиції кубіт-векторних примітивів завдання всіх типів функціональностей, імплементованих в елементи пам'яті, що дає можливість істотно підвищити швидкодію засобів моделювання, тестування і верифікації, а також значно спростити процедури створення реальних і віртуальних комп'ютерних систем; 3) оригінальними структурами даних для моделювання і верифікації цифрових систем, які дають можливість суттєво спростити алгоритми реалізації та підвищити їх швидкодію за рахунок адресовних функціональних квантів і паралельності обробки компонентів; 4) реалізацією всіх обчислювальних структур і процесів на основі використання введеного квантового адресного автомата, який дає можливість безпосередньо використовувати інфраструктуру стандартів тестопридатного проектування для підвищення виходу придатної продукції за рахунок online ремонту функціональних примітивів.

Практична значущість нового підходу синтезу та аналізу цифрових систем полягає в наступному: 1) реалізація процесора тільки на основі використання елементів пам'яті робить його однорідним за структурою і типам функціональних примітивів, що доставляє очевидні технологічні зручності процесам проектування, виробництва і експлуатації, у тому числі верифікації, вбудованого тестування і діагностування, а головне – ремонту в режимі online за рахунок використання на кристалі універсальних адресовних spare-компонентів пам'яті; 2) моделювання в процесі верифікації проектованих обчислювачів на основі адресних моделей компонентів робить дану процедуру технологічно простою за рахунок регулярних структур даних і використання

єдиної операції транзакції на елементах пам'яті, а також більш швидкодіючою за рахунок можливості паралельної квантоподібної обробки великих масивів однотипної пам'яті; 3) імплементація квантових only memory-based моделей опису цифрових компонентів і систем безпосередньо пов'язана зі збільшенням виходу придатної продукції, підвищенням надійності обчислювальних виробів, зниженням вартості проектування і виготовлення, а також автономним відновленням працездатності в режимі remote & online, без участі людини.

Практичні висновки за елементами кубітної теорії комп'ютингу. Згідно квантової теорії, електрон може перебувати в двох місцях одночасно. Дане твердження не суперечить відомому рівнянню Шредінгера, де координата кіберфізічного простору здатна мати в один і той же час два електрони або два стійких стани. Кубіт $Q = (Q_1, Q_2, \dots, Q_i, \dots, Q_n)$ – дискретний стан точки у кіберфізичному просторі, який визначається суперпозицією скінченної кількості примітивів (носіїв) універсуму, але представлених вектором. Квант функціональності (XQY) – універсальна за формою елементарна структура (частка), призначена для створення як завгодно складних компьютингових систем. Квант функціональності – кубіт з ідентифікаторами вхідних і вихідних змінних, які формують адреси примітивів (комірок пам'яті) для цілеспрямованого виконання транзакцій (зчитування-запис). Квант функціональності призначений для реалізації як завгодно складних memory-driven обчислювальних архітектур, вільних від логіки. Структура пам'яті, що використовує манхеттенську або трикутну топологію, створює квазіоптимальні умови для підвищення швидкодії адресних транзакцій між кубіт-векторами функциональностей. Послідовність обробки квантів функциональностей обчислювальної архітектури задається кубітом алгоритму, заданого орієнтованим графом, який також реалізується на пам'яті. Створюються два memory-driven механізми керування і виконання компьютингової системи, що використовує тільки кубітну векторну форму

опису функцій і структур. Адреса доступу до комірки кубіта формується на основі конкатенації станів вхідних змінних (X) кванта функциональностей. Вміст розряду кубіта Q[M(X)] за сконкатенованою адресою M(X) визначає стан кванта функціональності, який записується в комірку (Y) вектора стану (M) обчислювальної системи. Таким чином, computing (рис. 4.17), глобально і локально, віртуально і фізично, програмно і апаратно, являє собою реалізацію в пам'яті тривіальної транзакції: M(Y) = Q[M(X)] – рівняння Memory-Address-Transaction комп'ютингу. І нічого більше.

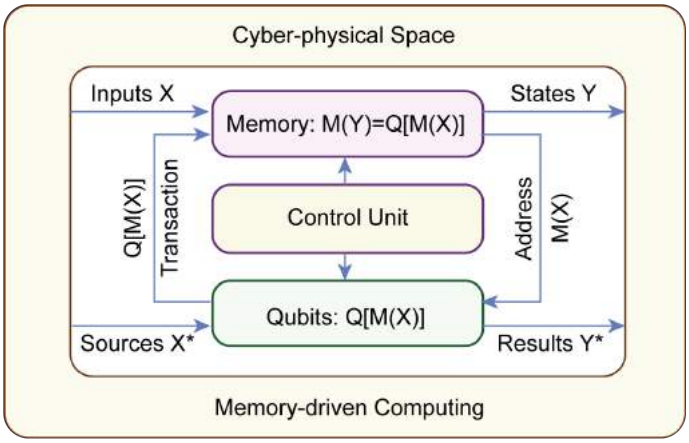



Рис. 4.17 – MAT-комп'ютинг

Іншими словами, будь-який обчислювальний процес зводиться до адресних операцій зчитування-запису на пам'яті. Практична значущість даної парадигми полягає у виключенні апаратної логіки з комп'ютингу, що дає можливість: 1) зробити регулярною архітектуру комп'ютера на основі використання тільки матриць пам'яті; 2) виконувати всі арифметичні, логічні та спеціальні операції на матричній структурі пам'яті, без звернення до зовнішнього модулю АЛУ, якій значно зменшує швидкодію; 3) технологічно вирішувати всі проблеми надійності комп'ютерних систем на основі вбудованого online тестування і ремонту відмовних модулів пам'яті шляхом їх переадресації на справні з ремонтного запасу; 4) мати практично необмежені можливості в межах одного кристала для паралельного виконання логічних

операцій будь-якої регістрової розмірності; 5) істотно зменшувати час проектування спеціалізованих процесорів за рахунок виключення складних і нерегулярних блоків АЛУ з шинами обміну інформацією; 6) перепрограмувати в режимі online логіку елементів пам'яті для виконання інших операцій.

В даний час спостерігається стійке домінування інноваційної кіберкультури, що містить Нано-Адитивні, Біо-Інформаційні, Кібер-Фізичні, Соціально-Когнітивні технології, на ринку науки, освіти, індустрії, транспорту і забезпечення якості життя.

Основні атрактори досліджень пов'язані з ключовими словами: Qubit Function, Qubit Structure, Qubit Coverage, Qubit Vector, Quantum of Functionality, Memory-Address-Transaction, Quantum Simulation, Qubit Design and Testing, Memory-Driven Computing, Cloud Computing, IoT Computing, Big Data Computing, Cyber-Physical Computing and Cosmological Computing.

В роботі вирішено проблему усунення протиріччя між двовимірним класичним описом функцій і структур і одновимірним поданням регістрових змінних в апаратному компьютингу за рахунок приведення функцій і структур до одновимірної кубітно-векторної метрики в цілях технологічного вирішення задач синтезу і аналізу цифрових систем на основі паралельних логічних операцій, що створюють memory-driven computing.

Сутність представленого науково-технологічного дослідження полягає у створенні векторних структур даних і кубітних методів синтезу, тестування і моделювання, інтегрованих в хмарну інфраструктуру сервісного обслуговування компонентів цифрових систем на кристалах з метою підвищення якості комп'ютингу і виходу придатної продукції за рахунок адресовності всіх обчислювальних процесів і явищ.

Основна інноваційна ідея запропонованої МАТ-моделі комп'ютингу полягає в синтезі і аналізі векторних цифрових структур на основі адресовних елементів пам'яті, що виключають використання reusable or new logic.

Мета дослідження була спрямована на суттєве підвищення виходу придатної програмно-апаратної продукції і якості комп'ютингу за рахунок створення інфраструктури хмарних сервісів моделювання, тестування і відновлення працездатності на основі кубітно-векторних структур даних опису функціональних елементів і підвищення швидкодії кубітних методів синтезу та аналізу.

Напрями майбутніх досліджень: 1) Створення теорії кубітного проектування і тестування цифрових систем на кристалах. 2) Розробка хмарних сервісів кубітного синтезу та аналізу цифрових систем на кристалах. 3) Створення хмарного сервісу моніторингу та управління науково-освітніми процесами "Smart Cyber University". 4) Розробка хмарного сервісу моніторингу та streetlight-free управління транспортом. 5) Створення e-infrastructure і хмарних сервісів моніторингу і human-free управління соціальними групами. 6) Розробка трикутних 2D-3D топологій для створення оптимальних інфраструктур кіберфізичного простору (triangle-driven міста, комп'ютингові архітектури, кіберекосистема планети).

# ВИСНОВОК

Проведені науково-технологічні дослідження в рамках дисертаційної роботи характеризуються успішним вирішенням актуальної науково-практичної задачі приведення опису функцій і структур до єдиної одновимірної кубітно-векторної метрики в цілях технологічного вирішення всіх задач синтезу й аналізу обчислювальних пристроїв шляхом виконання паралельних логічних операцій в рамках стійкої тенденції memory-driven computing.

Автором одержано такі наукові та практичні результати:

1. Запропоновано нову модель сталого розвитку кіберфізичного комп'ютингу та напрями її використання для прогнозування аттракторів в області IT-індустрії на основі просторово-часового аналізу технологічних процесів, які характеризуються трьома фазами розвитку і дають можливість прогнозувати майбутні тренди розвитку кіберкультури планети.

2. Удосконалено кубітно-векторні моделі для опису структур і компонентів цифрових систем на базі адресного кодування вхідних сигналів, які відрізняються від аналогів технологічністю і швидкодією побудованих на їх основі процедур тестування, логічного синтезу та аналізу.

3. Удосконалено метод паралельного регістрового моделювання компонентів цифрових систем на кристалах, який відрізняється від аналогів застосуванням автоматної МАТ-моделі, що використовує тільки адресовні структури пам'яті і операції транзакції. Запропоновано новий підхід до проектування цифрових пристроїв, який характеризується застосуванням методів синтезу та аналізу, які базуються на суперпозиції кубіт-векторних примітивів візначення всіх типів функциональностей, імплементованих в елементи пам'яті, що дає можливість істотно підвищити швидкодію засобів моделювання, тестування і верифікації, а також значно спростити процедури створення реальних і віртуальних комп'ютерних систем.

4. Удосконалено методи синтезу тестів на основі булевих похідних і дедуктивного моделювання несправностей, які відрізняються паралельним виконанням логічних регістрових операцій над кубітними покриттями схемних компонентів, що дає можливість істотно зменшити час верифікації та тестування цифрового пристрою.

5. Створено хмарну інфраструктуру сервісного обслуговування для online проектування, тестування і верифікації цифрових проектів, яка відрізняється візуалізацією цифрових схем і доступністю мікросервісів моделювання для зменшення періоду налагодження HDL-коду.

6. Здійснено тестову верифікацію компонентів хмарної інфраструктури сервісного обслуговування, а також методів тестування, моделювання, синтезу та аналізу на реальних прикладах цифрових схем і елементів. Використано мови програмування: SWIFT, C++, Verilog, Python 2.7 і платформи: Microsoft Windows, X Window і Macintosh OS X. Наукові результати дозволили підвищити швидкодію процедур аналізу структур даних, тестування та інтерпретативного адресного моделювання цифрових схем, на 15% зменшити час налагодження HDL-проектів в процесі проектування SoC.

Наукові висновки і рішення є достовірними, що підтверджується достатньою кількістю проведених експериментів, тестуванням і моделюванням реальних функціональних модулів з відкритих бібліотек компаній. Значущість наукових досліджень підтверджується інтеграцією методів аналізу і верифікації з сервісами проектування компанії Aldec. Результати дисертації в складі моделей, методів та інфраструктури сервісів також впроваджено в навчальний процес Харківського національного університету радіоелектроніки.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронний ресурс: [Gartner: 21 Billion IoT Devices To Invade By 2020. http://www.informationweek.com/mobile/mobile-devices/gartner-21-billion-iot-devices-to-invade-by-2020/d/d-id/1323081 ]

2. In-Memory Computing technology - The holy grail of analytics? Deloitte. https://www2.deloitte.com/content/dam/Deloitte/de/Documents/technology-media-telecommunications/TMT_Studie_In_Memory_Computing.pdf

3. Chen C. GFlink: An In-Memory Computing Architecture on Heterogeneous CPU-GPU Clusters for Big Data / C. Chen, K. Li, A. Ouyang, Z. Tang and K. Li // 2016 45th International Conference on Parallel Processing (ICPP) .- Philadelphia, PA.- 2016.- P. 542-551. doi: 10.1109 / ICPP.2016.69

4. Changing the Way Businesses Compute ... and Compete. In-Memory Computing and Real-Time Business Intelligence.- White Paper.- Intel® Xeon® Processor E7 v2 Family.- In-Memory Computing.- 2013.

5. Kang M. In-Memory Computing Architectures for Sparse Distributed Memory / M. Kang, N.R. Shanbhag // IEEE Transactions on Biomedical Circuits and Systems.- Aug. 2016.- vol. 10, no. 4. - P. 855-863.

6. Chi P. Processing-in-Memory in ReRAM-based Main Memory / P. Chi, Shuangchen Li, et al // SEAL-lab Technical Report - 2016. - no. 2015-001 (April 29, 2016).

7. Schabel J. Processor-in-memory support for artificial neural networks / J. Schabel, L. Baker, S. Dey, W. Li, P.D. Franzon // 2016 IEEE International Conference on Rebooting Computing (ICRC) .- San Diego , CA.- 2016.- P. 1-8. doi: 10.1109 / ICRC.2016.7738697

8. DeBenedictis E.P. Optimal adiabatic scaling and the processor-in-memory-and-storage architecture (OAS + PIMS) / E.P. DeBenedictis, J.E. Cook, M.F. Hoemmen, T.S. Metodi // Proceedings of the 2015 IEEE / ACM International

Symposium on Nanoscale Architectures (NANOARCH-15) .- Boston, MA.- 2015.- P. 69-74. doi: 10.1109 / NANOARCH.2015.7180589

9. Sterling T.L. Gilgamesh: A Multithreaded Processor-In-Memory Architecture for Petaflops Computing / T.L. Sterling, H.P. Zima // Supercomputing, ACM / IEEE 2002 Conference.- 2002.- P. 48-48.

10. Scrbak M. Processing-in-Memory: Exploring the Design Space. Chapter: Architecture of Computing Systems / M. Scrbak, M. Islam, K.M. Kavi, M. Ignatowski, N. Jayasena. - ARCS 2015. Vol. 9017 of the series Lecture Notes in Computer Science.- Springer.- 2015.- P. 43-54.

11. Яковлев Ю.С. Реконфигурируемые PIM–системы: методология построения, примеры моделей / Ю.С. Яковлев // Математичні машини і системи.– 2007.– № 2.– С. 27-42.

12. Палагин А.В. Особенности подхода к выбору ПЛИС для проектирования PIM-систем / А.В. Палагин, Ю.С. Яковлев, Е.В. Елисеева // Математичні машини і системи.– 2012.– № 3.– С. 19-28.

13. Яковлев Ю.С. Разработка и моделирование процессорного элемента операционной среды PIM системы / Ю.С. Яковлев, Б.М. Тихонов // Электронное моделирование.– 2009.– Т. 31.– №2.– С. 65-80.

14. Elliott D.G. Computational RAM: implementing processors in memory / D.G. Elliott, M. Stumm, W.M. Snelgrove, C. Cojocaru, R. Mckenzie // IEEE Design & Test of Computers.- Jan-Mar, 1999.- vol. 16, no. 1.- P. 32-41. doi: 10.1109 / 54.748803

15. Chiueh H. The address translation unit of the data-intensive architecture (DIVA) system. / H. Chiueh, J. Draper, S. Mediratta, J. Sondeen // Proceedings of the 28th European Solid-State Circuits Conference.- Florence, Italy.- 2002.- P. 767-770.

16. Azarkhish E. Design and Evaluation of a Processing-in-Memory Architecture for the Smart Memory Cube / E. Azarkhish, D. Rossi, I. Loi, L. Benini // F.Hannig et al. (Eds.): ARCS 2016.- 2016.- LNCS 9637.- P. 19-31.

17. Akin B. Data reorganization in memory using 3D-stacked DRAM / B. Akin, F. Franchetti, J.C. Hoe // 2015 ACM / IEEE 42nd Annual International

Symposium on Computer Architecture (ISCA) .- Portland, OR.- 2015.- P . 131-143. doi: 10.1145 / 2749469.2750397

18. Елисеева Е.В. Структура и функции управляющего пакета интеллектуальной системы памяти / Е.В. Елисеева // Комп'ютерні засоби, мережі та системи. – 2009.– № 8.– С. 130-137.

19. Oskin M. Active Pages: a computation model for intelligent memory / M. Oskin, F.T. Chong, T. Sherwood // Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235) .- Barcelona.- 1998.- P. 192-203.

20. Seung-Moon Yoo. FlexRAM Architecture Design Parameters / Seung-Moon Yoo, J. Renau, M. Huang, J. Torrellas // Електронний ресурс:

https://masc.soe.ucsc.edu/docs/flexramTR.pdf

21. Torrellas J. FlexRAM: Toward an advanced Intelligent Memory system: A retrospective paper / J. Torrellas // 2012 IEEE 30th International Conference on Computer Design (ICCD) .- Montreal, QC.- -2012 P. 3-4.

22. Fraguela B.B. Programming the FlexRAM Parallel Intelligent Memory System / B.B. Fraguela, J. Renauy, P. Feautrierz, D. Paduay, J. Torrellasy // PPoPP'03.- June 11-13, 2003.- San Diego, California, USA. - 12 p.

23. Gaeke B.R. Memory-intensive benchmarks: IRAM vs. cache-based machines / B.R. Gaeke, P. Husbands, X.S. Li, L. Oliker, K.A. Yelick, R. Biswas // Proceedings of International Parallel and Distributed Processing Symposium.- 2002. 7 p. doi: 10.1109 / IPDPS.2002.1015506

24. DIVA (Data Intensive Architecture). AFRL-IF-RS-TR-2004-144. Final Technical Report. June 2004. Електронний ресурс:
https://www.researchgate.net/publication/235045186_DIVA_Data_Intensive_Architecture

25. Draper J. The Architecture of the DIVA ProcessingInMemory Chip / J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C.W. Kang, I. Kim, G. Daglikoca // ICS'02.- June 22-26, 2002.- New York, USA.

26. Hamdioui S. Memristor based computation-in-memory architecture for data-intensive applications / S. Hamdioui et al // 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE) .- Grenoble.- 2015.- P. 1718-1725.

27. Tessier R. Reconfigurable Computing Architectures / R. Tessier, K. Pocek, A. DeHon // Proceedings of the IEEE.- March 2015.- vol. 103, no. 3.- P. 332-354. doi: 10.1109 / JPROC.2014.2386883

28. Todman T.J. Reconfigurable computing: architectures and design methods / T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk, P. Cheung // IEEE Proceedings - Computers and Digital Techniques.- Mar 2005.- vol. 152, no. 2.- P. 193-207. doi: 10.1049 / ip-cdt: 20045086

29. Compton K. An Introduction to Reconfigurable Computing / K. Compton, S. Hauck // Електронний ресурс: http://kmorrow.ece.wisc.edu/Publications/Compton_ReconfigIntro.pdf

30. Compton K. Reconfigurable Computing: A Survey of Systems and Software / K. Compton, S. Hauck // ACM Computing Surveys.- June 2002.- Vol. 34, No. 2.- P. 171-210.

31. Ogawa E. Reconfigurable IBM PC Compatible SoC for Computer Architecture Education and Research / E. Ogawa, Y. Matsuda, T. Misono, R. Kobayashi, K. Kise // 2015 IEEE 9th International Symposium on Embedded Multicore / Many-core Systems-on-Chip .- 2015.- P. 65-72.

32. Wijtvliet M. Coarse grained reconfigurable architectures in the past 25 years: Overview and classification / M. Wijtvliet, L. Waeijen, H. Corporaal // 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS) .- Agios Konstantinos .- 2016.- P. 235-244. doi: 10.1109 / SAMOS.2016.7818353

33. Hou Z. System level power consumption modeling and optimization for coarse-grained reconfigurable architectures / Z. Hou, Z. Zhao, W. Sheng, W. He // 2016 International Conference on Integrated Circuits and Microsystems (ICICM) .- Chengdu.- 2016.- P. 1-6. doi: 10.1109 / ICAM.2016.7813532

34. Coarse Grained Reconfigurable Architectures // Електронний ресурс: [www.fpl.uni-kl.de]

35. Hartenstein R. Coarse grain reconfigurable architectures / R. Hartenstein // Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001 (Cat. No.01EX455) .- Yokohama.- 2001.- P. 564 -569. doi: 10.1109 / ASPDAC.2001.913368

36. Wijtvliet M. Coarse grained reconfigurable architectures in the past 25 years: Overview and classification / M. Wijtvliet, L. Waeijen, H. Corporaal // 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS) .- Agios Konstantinos.- 2016.- P. 235-244.

37. Liang S. A Coarse-Grained Reconfigurable Architecture for Compute-Intensive MapReduce Acceleration / S. Liang, S. Yin, L. Liu, Y. Guo, S. Wei // IEEE Computer Architecture Letters.- July-Dec. 1, 2016.- vol. 15, no. 2.- P. 69-72, doi: 10.1109 / LCA.2015.2458318.

38. Проектирование и тестирование цифровых систем на кристаллах / В. И. Хаханов, Е. И. Литвинова, И. В. Хаханова, О. А. Гузь. – Харьков : ХНУРЭ, 2009. – 484 с.

39. Chin S.Y.I. Power Implications of Implementing Logic Using FPGA Embedded Memory Arrays / S. Chin, C.S. p. Lee, S.J.E. Wilton // 2006 International Conference on Field Programmable Logic and Application.- Madrid, 2006.- P. 1-8.

40. Taylor C.N. FPGA Implementation Details / C.N. Taylor // Електронний ресурс: http://ece320web.groups.et.byu.net/CourseNotes/FPGA.pdf

41. FPGA Architecture. White Paper.- Altera.- July 2006.- ver. 1.0.- 14p.

42. Тюрин С.Ф. Логический элемент ПЛИС-FPGA для реализации ДНФ / С.Ф. Тюрин, А.М. Морозов, И.С. Понуровский // Вестник ИжГТУ.– 2013.– № 2(58).– С.95-98.

43. Wolf W. FPGA-Based System Design / W. Wolf. - Pearson Education.- 2004.- 576 p.

44. Borowik G. On Memory Capacity to Implement Logic Functions / G. Borowik, T. Łuba, P. Tomaszewicz // EUROCAST 2011. Part II. Lecture Notes in Computer Science.- Volume 6928.- 2012. - Springer Berlin Heidelberg.- P. 343-350.

45. Intel® FPGAs. Електронний ресурс: https://www.altera.com/products/soc/overview.html

46. Cong J. Boolean Matching for LUT-Based Logic Blocks With Applications to Architecture Evaluation and Technology Mapping / J. (Jingsheng) Cong, Y.-Y. Hwang // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.- Sept . 2001. - vol. 20, no. 9.- P. 1077-1090.

47. Battezzati N. Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications / N. Battezzati et al. // Springer Science + Business Media.- LLC 2011.- DOI 10.1007 / 978-1-4419-7595-9_2.

48. Kuon I. FPGA Architecture: Survey and Challenges / I. Kuon, R. Tessier, J. Rose // Foundations and Trends in Electronic Design Automation. -2007.- vol. 2, no. 2.- p. 135-253.

49. Introduction to FPGA Design with Vivado High-Level Synthesis. UG998 (v1.0) .- July 2, 2013.- Xilinx. //Електронний ресурс: http://www.xilinx.com

50. Зотов В. Особенности архитектуры нового поколения ПЛИС с архитектурой FPGA фирмы Xilinx / В. Зотов // Компоненты и технологии.– № 12.– 2010.– С. 17-24.

51. 7 Series FPGAs. Configurable Logic Block. User Guide. UG474 (v1.8) .- Xilinx.- Sep. 27, 2016 // Електронний ресурс:http://www.xilinx.com

52. Synthesis and Simulation Design Guide. Xilinx. UG626 (v 12.3) .- September 21 2010 // Електронний ресурс: www.xilinx.com

53. Chong N. FPGA design and system optimizations with new technologies / N. Chong, J. Jing, H. Liu, G. Refai-Ahmed, S. Wu, X. Wu // 2016 IEEE International Electron Devices Meeting (IEDM. - San Francisco, CA, USA.- 2016.- P. 2.1.1-2.1.4. doi: 10.1109 / IEDM.2016.7838028

54. A Fault Tolerant FPGA Architecture. - ProQuest.- 2008.- 127 p.

55. Wegrzyn M. Tracing Fault Effects in FPGA Systems / M.Wegrzyn and J. Sosnowski // Intl Journal of Electronics and Telecommunications.- 2014.- vol. 60, no. 1.- P. 103-108.

56. Jamuna S. Detection and Diagnosis of Faults in the Routing Resources of a SRAM based FPGAs / S. Jamuna, V.K. Agrawal // Intl Journal of Computer Applications (0975 - 8887) .- Sep. -2012 vol. 53, no.13.- P. 18-22.

57. Parreira A. A novel approach to FPGA based hardware fault modeling and simulation / A. Parreira, JP Teixeira, M. Santos // Proc. 6th IEEE Int. Workshop Des. Diagnostics Electron. Circuits Syst.- P. 17-24, DOI: 10.1.1.1.3581.

58. Jamuna S. Detection and Diagnosis of Faults in the Routing Resources of a SRAM based FPGAs / S. Jamuna, V.K. Agrawal // International Journal of Computer Applications (0975 - 8887) .- Sept. -2012 vol. 53, no.13.- P. 18-22.

59. Sterpone L. Scalable FPGA graph model to detect routing faults / L. Sterpone, G. Cabodi, SF Finocchiaro, C. Loiacono, F. Savarese, B. Du // 2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS) .- Sant Feliu de Guixols.- 2016.- P. 155-160. doi: 10.1109 / IOLTS.2016.7604690

60. Stott E. Timing Fault Detection in FPGA-Based Circuits / E. Stott, J.M. Levine, P.Y.K. Cheung, N. Kapre // 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines.- Boston, MA.- 2014 .- P. 96-99. doi: 10.1109 / FCCM.2014.32

61. Jamuna S. Implementation of bistcontroller for fault detection in CLB of FPGA / S. Jamuna, V.K. Agrawal // 2012 International Conference on Devices, Circuits and Systems (ICDCS) .- Coimbatore.- -2012 P. 99-104. doi: 10.1109 / ICDCSyst.2012.6188682

62. Rehman S.U. Test and diagnosis of FPGA cluster using partial reconfiguration / S.U. Rehman, M. Benabdenbi, L. Anghel // 2014 10th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME) .- Grenoble.- 2014.- P. 1-4. doi: 10.1109 / PRIME.2014.6872671

63. Rehman S.U. BIST for logic and local interconnect resources in a novel mesh of cluster FPGA / S.U. Rehman, M. Benabdenbi, L. Anghel // 2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS) .- New York City, NY.- 2013.- P. 296-301. doi: 10.1109 / DFT.2013.6653622

64. Raj G.G. Fault scanning and repairing in processor based system using dynamic reconfiguration / G.G. Raj, B. Kannan, T. Aravind // 2013 IEEE International Conference on Smart Structures and Systems (ICSSS) .- Chennai.- 2013.- P. 120-124. doi: 10.1109 / ICSSS.2013.6623013

65. Das N. Build-in-Self-Test of FPGA for diagnosis of delay fault / N. Das, P. Roy, H. Rahaman, P. Dasgupta // 2011 3rd Asia Symposium on Quality Electronic Design (ASQED) .- Kuala Lumpur.- 2011.- P. 54-61. doi: 10.1109 / ASQED.2011.6111702

66. Stott E. Fault tolerant methods for reliability in FPGAs / E. Stott, P. Sedcole, P.Y.K. Cheung // 2008 International Conference on Field Programmable Logic and Applications.- Heidelberg.- 2008.- P. 415-420. doi: 10.1109 / FPL.2008.4629973

67. Stroud C. BIST-based diagnosis of FPGA interconnect / C. Stroud, J. Nall, M. Lashinsky, M. Abramovici // Proceedings of International Test Conference.– 2002.– P. 618-627.

68. Abramovici M. BIST-based test and diagnosis of FPGA logic blocks / M. Abramovici, C.E. Stroud // IEEE Transactions on Very Large Scale Integration (VLSI) Systems.– 2001.– vol. 9, no. 1.– P. 159-172.

69. Abraham J. The future of fault tolerant computing / J. Abraham, R. Iyer, D. Gizopoulos, D. Alexandrescu, Y. Zorian // 2015 IEEE 21st International On-Line Testing Symposium (IOLTS).– Halkidiki.– 2015. – P. 108-109.

70. Zorian Y. Design, test & repair methodology for FinFET-based memories / Y. Zorian // 2014 International Test Conference.– Seattle, WA, USA.– 2014.– P. 1.

71. Sargsyan V. An efficient approach for memory repair by reducing the number of spares / V. Sargsyan, Valery A. Vardanian, Samvel K. Shoukourian,

Yervant Zorian, Avetik Yessayan // 2015 IEEE East-West Design & Test Symposium. – 2015. – Batumi, Georgia. – P. 1-4.

72. Marinissen E.J. IoT: Source of test challenges / E.J. Marinissen et al // 21th IEEE European Test Symposium (ETS).– Amsterdam.– 2016.– P. 1-10.

73. Koal T. Dependability and life time enhancements for nano-electronic systems / T. Koal, M. Schölzel, H.T. Vierhaus // Signal Processing Algorithms, Architectures, Arrangements, and Applications. 2011.– Poznan.– P. 1-6.

74. Habermann S. Built-in self repair by reconfiguration of FPGAs / S. Habermann, R. Kothe, H.T. Vierhaus, // 12th IEEE International On-Line Testing Symposium (IOLTS'06). – Lake Como. – 2006. – P. 2.

75. Kharchenko V. Concepts of Green IT Engineering: Taxonomy, Principles and Implementation / V. Kharchenko, O. Illiashenko. – Springer-Verlag Berlin Heidelberg. – 2016.

76. Бондаренко М.Ф. Структура логического ассоциативного мультипроцессора / М.Ф. Бондаренко, В.И. Хаханов, Е.И. Литвинова // Автоматика и телемеханика. – 2012. – С. 71-92.

77. Хаханов В.И. Кубитные структуры данных вычислительных устройств / В.И. Хаханов, В. Гариби, Е.И. Литвинова, А.С. Шкиль // Электронное моделирование. – 2015. – № 1. – С. 76-99.

78. Хаханов В.И. Кубитные технологии анализа и диагностирования цифровых устройств / В.И. Хаханов, Т. Бани Амер, С.В. Чумаченко, Е.И. Литвинова // Электронное моделирование. – Том 37, № 3. – С. 17-40.

79. Hahanov V. Big Data Driven Cyber Analytic System / V. Hahanov, E. Litvinova, W. Gharibi, S. Chumachenko // 2015 IEEE Int. Congress on Big Data.– New York, USA. – P. 615-622.

80. Dehorter N. Tuning of fast-spiking interneuron properties by an activity-dependent transcriptional switch / N. Dehorter, G. Ciceri, G. Bartolini, L. Lim, I. del Pino, O. Marín. – Science.– 11 Sept. 2015: 349 (6253). – P. 1216-1220.

81. P.A. Merolla A million spiking-neuron integrated circuit with a scalable communication network and interface / Paul A. Merolla et all // Science, 8 August 2014.– Vol. 345, no. 6197. – P. 668-673.

82. ГОСТ 34.003-90. Автоматизированные системы. Термины и определения.

83. ISO / IEC 38500: 2008. Corporate governance of information technology.

84. Marr B. Forbes. 9 Attractors in the Computer Market. 17 'Internet Of Things' Facts Everyone Should Read. Електронний ресурс: http://www.forbes.com/sites/bernardmarr/2015/10/27/17-mind-blowing-internet-of-things-facts-everyone-should-read/

85. Galer S. Forbes. IDC Releases Top Ten 2016 IT Market Predictions. Електронний ресурс: http://www.forbes.com/sites/sap/2015/11/05/idc-releases-top-ten-2016-it-market-predictions/

86. Бондаренко М.Ф. Квантовые технологии реализации мозгоподобных вычислительных структур академика В.М. Глушкова / М.Ф. Бондаренко, В.И. Хаханов // Матеріали XIX Міжнародної конференції з автоматичного управління. – Київ. – 26-28 вересня, 2012. – С. 32-33.

87. Палагин А.В. Архитектурно-структурная организация компьютерных средств класса “Процессор-в-памяти” / А.В. Палагин, Ю.С Яковлев., Б.М. Тихонов, И.М. Першко // Математичні машини і системи. – 2005. – № 3. – 14 с.

88. Палагин А.В. Системная интеграция средств компьютерной техники. / А.В. Палагин, Ю.С. Яковлев.– Винница: “Универсум”.– 2005.

89. Большой взрыв. Рождение и развитие Вселенной. Електронний ресурс: http://ru-universe.livejournal.com/712762.html?page=1

90. Kelly K.F. The booms and busts of molecular electronics / K.F. Kelly, C.C.M. Mody // IEEE Spectrum. – 2015.– Vol. 52, Iss. 10. –P. 52-60.

91. Сет Л. Программируя Вселенную: Квантовый компьютер и будущее науки / Л. Сет. – М: Альпина нон-фикшн. – 2014.– 256 с.

92. Nielsen M.A. Quantum Computation and Quantum Information / M.A. Nielsen, I. L. Chuang.– Cambridge University Press.– 2010. – 676 p.

93. Whitney M.G. Practical Fault Tolerance for Quantum Circuits / M.G. Whitney. – PhD dissertation. – University of California, Berkeley. – 2009. – 229 p.

94. Nakahara M. Quantum Computing. An Overview / M. Nakahara. – Higashi-Osaka: Kinki University.– 2010. – 53 p.

95. Курош А.Г. Курс высшей алгебры / А.Г. Курош.– Изд-во. Москва: Наука. – 1968. – 426 с.

96. Горбатов В.А. Основы дискретной математики / В.А. Горбатов. – М.: Высшая школа.– 1986. – 311 с.

97. Hahanov V.I. Qubit Model for solving the coverage problem / V.I. Hahanov, E.I. Litvinova, S.V. Chumachenko et al // Proc. of IEEE East-West Design and Test Symposium. – Ukraine. – Sept., 2012. – P. 142 - 144.

98. Hahanov V. «Quantum» Processor for Digital Systems Analysis / V. Hahanov, W. Gharibi, I. Iemelianov, D. Shcherbin // Proceedings of IEEE East-West Design & Test Symposium (EWDTS-2015). – 2015. – Batumi, Georgia. – P. 104-110.

99. Рябцев В.Г. Метод и средства визуализации алгоритмов тестов диагностирования запоминающих устройств / В.Г. Рябцев, Д.Н. Муамар // Электронное моделирование.– 2010. – Том 32, № 3. – С. 43-52.

100. Zorian Y. Test solutions for nanoscale Systems-on-Chip: Algorithms, methods and test infrastructure / Y. Zorian, S. Shoukourian // Nineth International Conference on Computer Science and Information Technologies Revised Selected Papers.– Yerevan.– 2013.– P. 1-3.

101. Tshagharyan G. Overview study on fault modeling and test methodology development for FinFET-based memories / G. Tshagharyan, G. Harutyunyan, S.

Shoukourian, Y. Zorian // 2015 IEEE East-West Design & Test Symposium (EWDTS).– Batumi.– 2015.– P. 1-4.

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

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

104. Molnar L. Fault simulation methodes / L. Molnar, A. Gontean // 2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC) .– Timisoara.– 2016.– P. 194-197.

105. Hadjitheophanous S. Scalable parallel fault simulation for shared-memory multiprocessor systems / S. Hadjitheophanous, S.N. Neophytou, M.K. Michael // 2016 IEEE 34th VLSI Test Symposium (VTS) .– Las Vegas, NV.– 2016.– P. 1 -6.

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

107. Ubar R. Combinational fault simulation in sequential circuits / R. Ubar, J. Kõusaar, M. Gorev, S. Devadze // 2015 IEEE International Symposium on Circuits and Systems (ISCAS).– Lisbon.– 2015.– P. 2876-2879.

108. Gorev M. Fault simulation with parallel exact critical path tracing in multiple core environment / M. Gorev, R. Ubar, S. Devadze // 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE).– Grenoble.– 2015.– P. 1180-1185.

109. Pomeranz I. Fault simulation with test switching for static test compaction / I. Pomeranz // 2014 IEEE 32nd VLSI Test Symposium (VTS).– Napa, CA.– 2014.– P. 1-6.

110. Mirkhani S. EAGLE: A regression model for fault coverage estimation using a simulation based metric / S. Mirkhani, J.A. Abraham // 2014 International Test Conference.– Seattle, WA.– 2014.– P. 1-10.

111. Metodi T. Quantum Computing for Computer Architects / T. Metodi, F. Chong.– Synthesis Lectures on Computer Architecture.– Morgan & Claypool.– 2006.– 154 p.

112. Stig S. Quantum approach to informatics / S. Stig, Kalle-Antti Suominen. – John Wiley & Sons, Inc.– 2005.– 249 p.

113. Hahanov V.I. Qubit data structure of computing devices / V.I. Hahanov, W. Gharibi, E..I Litvinova, A.S. Shkil // Electronic modeling.– № 1.– 2015.– P.76-99.

114. Hahanov V. MQT-model for Virtual Computer Design / V. Hahanov, T. Bani Amer, I. Hahanov // Proc. of Microtechnology and Thermal Problems in Electronics (Microtherm).– 2015.– P. 182-185.

115. Zorian Y. Embedded-memory test and repair: infrastructure IP for SoC yield / Y. Zorian, S. Shoukourian // IEEE Design & Test of Computers.– vol. 20, iss. 3.– P. 58 - 66.

116. Dugganapally I.P. Multi-level, Memory-Based Logic Using CMOS Technology / I.P. Dugganapally, S.E. Watkins, B. Cooper // 2014 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) .– Tampa, FL.– P. 583-588 .

117. Yueh W. A Memory-Based Logic Block With Optimized-for-Read SRAM for Energy-Efficient Reconfigurable Computing Fabric / W. Yueh, S. Chatterjee, M. Zia, S. Bhunia, S. Mukhopadhyay // IEEE Transactions on Circuits and Systems II: Express Briefs.– vol. 62, iss. 6.– P. 593-597.

118. Matsunaga S. MTJ-based nonvolatile logic-in-memory circuit, future prospects and issues / S. Matsunaga, J. Hayakawa, S. Ikeda, K. Miura, T. Endoh, H. Ohno, T. Hanyu // Design, Automation & Test in Europe Conference & Exhibition, DATE '09.– 2009. – P. 433 - 435.

119. Harada S. Design of a Logic-in-Memory Multiple-Valued Reconfigurable VLSI Based on a Bit-Serial Packet Data Transfer Scheme / S. Harada, Xu Bai, M. Kameyama, Y. Fujioka // 2014 IEEE 44th International Symposium on Multiple-Valued Logic (ISMVL).– 2014.– P. 214 - 219.

120. Hahanov V.I. Qubit technology analysis and diagnosis of digital devices / V.I. Hahanov, T. Bani Amer, S.V. Chumachenko, E.I. Litvinova // Electronic modeling. – 2015.– vol. 37, no 3.– P. 17-40.

121. Melikyan V.Sh. A method of eliminating false paths during statistical static analysis of timing delays of digital circuits / V.Sh. Melikyan / Elektronica i svyaz.– 2009.– vol. 2-3, no. 1.– P. 93-96.

122. Melikyan V.Sh. Interconnections model delays for the logic analysis of ECL circuits / V.Sh. Melikyan, AO Vatyan // SUAB.– vol. 2. Computer Engineering.– Moscow.– 1997.– P. 187-194.

123. Хаханов В.И. Метрика для анализа Big Data / В.И. Хаханов, А.С. Мищенко, В.И. Обризан, Tami Bani Amer // Радиоэлектроника и информатика.– 2014.– № 2.– С. 26-29.

124. Хаханов В.И. Кубитные технологии анализа и диагностирования цифровых устройств / В.И. Хаханов, Т. Бани Амер, С.В. Чумаченко, Е.И. Литвинова // Электронное моделирование. – Том 37, № 3.– 2015.– С. 17-40. (входить до міжнародних наукометричних баз Cambridge Scientific Abstracts, Computer and Information Systems Abstracts, INIS Collection, Inspec, ВИНИТИ РАН).

125. Bani Amer T. Синтез Q-тестов по кубитному покрытию функциональностей / Tamer Bani Amer, I. Iemelianov, M. Liubarskyi, V. Hahanov // Радиоэлектроника и информатика.– 2014.– № 2. – С. 26-29.

126. Хаханов В.И. Процессорные структуры для анализа Big Data / В.И. Хаханов, Е.И. Литвинова, С.В. Чумаченко, И. Емельянов, Т. Bani Amer // Радіоелектронні і комп'ютерні системи. – 2016.– № 6 (80).– С. 163-175.

127. Bani Amer T. Кубитная форма описания вычислительных структур

/ T. Bani Amer, С.В. Чумаченко, И.В. Емельянов // Радиоэлектроника и информатика.– 2016.– № 1.– С.47-52.

128. Хаханов В. Облачное управление физическими и кадровыми ресурсами / В. Хаханов, С. Чумаченко, Е. Литвинова, А. Мищенко, И. Емельянов, Т. Бани Амер // Australian Journal of Scientific Reseach.– № 1(5).– 2014.– С. 202-212.

129. Bani Amer T. Компьютинговые модели облачных сервисов / T. Bani Amer, В.И. Хаханов, И.В. Емельянов, М. Любарский // АСУ и приборы автоматики.– 2015.– Вип. 173.– С.48-57.

130. Bani Amer T. Кубитные модели описания цифровых устройств / T. Bani Amer, И.В. Хаханов, Е.И. Литвинова, И.В. Емельянов // АСУ и приборы автоматики.– 2016. – Вып. 174.– С. 24-41.

131. Emelyanov I. Qubit Modeling Digital Systems / I. Emelyanov, I. Hahanova, T. Bani Amer // Proc. of IEEE East-West Design and Test Symposium.– Kiev, 26-29 September.– 2014.– P. 246-248. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

132. Hahanov I. Automaton MQT-model for virtual computer design / I. Hahanov, T. Bani Amer, I. Hahanova, S. Dementiev, A. Arefiev // Proceedings of 13th International Conference: The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2015.– Lvov, Ukraine.– P. 161-165. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

133. Hahanov V. MQT-model for Virtual Computer Design / V. Hahanov, T. Bani Amer, I. Hahanov // Proc. of Microtechnology and Thermal Problems in Electronics (Microtherm), 23-25 June 2015.– Poland.– P. 182-185.

134. Gerasimenko K. Method for Functional Testing Critical Control Systems / K. Gerasimenko, V. Hahanov, T. Bani Amer, A. Pryimak // Proceedings of IEEE East-West Design & Test Symposium 2015.– Batumi, Georgia.– P. 149-153. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

135. Hahanov V. Cloud-Driven Traffic Control: Feasibiliti and Advanteges /

V. Hahanov, S. Chumachenko, T. Bani Amer, I. Hahanov // Proc. of the 4th Mediterranean Conference on Embedded Computing (MECO).– 2015.– Budva, Montenegro.– P.17-20. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

136. Hahanov V. Smart Resources Control / V. Hahanov, T. Bani Amer, S. Chumachenko, E. Litvinova // The 4th International Academic Congress “Science and Education in the Modern World”.– Vol. II. – New Zealand, Auckland.– 2015.– P. 1021-1034.

137. Hahanov I. QuaSim – Cloud Service for Digital Circuits Simulation / I. Hahanov, W. Gharibi, I. Iemelianov, T. Bani Amer // Proceedings of IEEE East-West Design & Test Symposium.– 2016.– Yerevan, Armenia.– P. 141-158. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

138. Soklakova T. Technological culture of Big Data / T. Soklakova, I. Iemelianov, T. Bani Amer, I. Hahanov // Матеріали XIII Міжнародної конференції TCSET-2016. – 23-26 лютого, 2016.– Львів–Славське.– С.549-554. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

139. Bani Amer T. Кибер-компьютинг – новый бренд IoT-рынка / T. Bani Amer, И. Емельянов // Материалы XX Международного молодежного форума «Радиоэлектроника и молодежь в XXI веке».– 2016.– Часть 5.– С.36-37.

140. Hahanov V. Qubit Test Synthesis of the Functionality / V. Hahanov, T. Bani Amer, E. Litvinova, T. Soklakova, M. Liubarskyi, N. Shavlak, K. Dziuba // Proceedings of 14th International Conference: The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2017.– Lvov, Ukraine.– P. 161-165. (Входить до міжнародних наукометричних баз Scopus, IEEE Xplore).

# ДОДАТОК А

# ДОКУМЕНТИ, ЩО ПІДТВЕРДЖУЮТЬ ВПРОВАДЖЕННЯ

ЗАТВЕРДЖУЮ»
Перший проректор ХНУРЕ
_____ Ключник І.І.
"06" 03 2016 р.

**СПРАВКА**

про впровадження в навчальний процес ХНУРЕ результатів дисертаційної роботи Тамер Бані-Амер (Tamer Bani-Amer) «Хмарний сервіс-комп'ютинг для тестування і моделювання SoC-компонентів», представленої на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.05 – комп'ютерні системи та компоненти

Голова комісії: Литвинова Є.І.

Члени комісії: Чумаченко С.В, Шкіль О.С.

В період з 05.10.2016 по 10.10.2016 комісія провела роботу з визначення впровадження в навчальний процес хмарних компьютингових сервісів тестування і верифікації цифрових пристроїв, розроблених на кафедрі АПОТ ХНУРЕ при 30% участі здобувача Тамера Бані Амера.

Комісією встановлено, що розроблені моделі і методи кубітного тестування, моделювання та оцінки якості вхідних наборів реалізовані у вигляді хмарних мікросервісов, придатних для їх використання в навчальних цілях для курсів: «Проектування та тестування цифрових систем на ПЛІС» для бакалаврів напрямку «комп'ютерна інженерія».

Основними результатами, впровадженими в навчальний процес слід вважати: векторні структури даних і кубітні методи синтезу, тестування і моделювання, інтегровані в хмарну інфраструктуру сервісного обслуговування компонентів цифрових систем на кристалах з метою підвищення якості виробів і виходу придатної продукції за рахунок адресованих обчислювальних процесів і явищ. Основна інноваційна ідея запропонованої МАТ-моделі обчислень полягає в синтезі і аналізі векторних цифрових структур на основі адресованих елементів пам'яті, що виключають використання reusable or new logic. Приведення опису функції і структури до єдиного одномерного формату, означає технологічно вирішувати всі завдання синтезу та аналізу для функціональностей і графів у кубітно-векторній метриці, що створює memory-driven computing на основі виконання паралельних логічних операцій.

Хмарні компьютингові сервіси тестування, моделювання несправностей і справної поведінки цифрових пристроїв є працездатними, перевіреними на представницькій вибірці стандартних тестових схем з бібліотеки конференції ISCAS.

Комісія рекомендує використання нової кубітно-векторної технології і хмарних сервісів проектування, синтезу тестів і моделювання цифрових пристроїв в навчальному процесі для студентів факультету КІУ.

Голова комісії проф. каф. АПОТ Литвинова Є.І.

Члени комісії зав. каф. АПОТ, проф. Чумаченко С.В.,

доц. каф. АПОТ Шкиль О.С.

ALDEC
The Design Verification Company

ООО «Алдек-КТС»
ул. Космическая 23а
61145, г.Харьков, Украина

ЕГРПОУ 36374067
Р/с 26008154763 в ВАТ ХОД «Райффайзен Банк Аваль»,
МФО 380805
Тел.: (+38 057) 760-47-25

**СПРАВКА**

о внедрении облачных компьютинговых сервисов тестирования и верификации цифровых устройств в методологическое и технологическое обеспечение компании ООО «АЛДЕК-КТС» в целях повышения качества проектирования цифровых изделий.

Внедрение в научно-исследовательскую и производственную деятельность моделей и методов кубитного синтеза и анализа цифровых устройств, облачных компьютинговых сервисов тестирования и верификации, разработанных на кафедре АПВТ, ХНУРЭ при 30 процентном участии соискателя Тамера Бани Амера позволило на 12 процентов уменьшить общее время разработки, необходимое для создания и выхода годной продукции на рынок.

Установлено, что разработанные модели и методы кубитного тестирования, моделирования и оценки качества входных наборов, реализованные в виде облачных микросервисов, потенциально позволяют на 5-6 процентов повысить выход годной продукции за счет online восстановления работоспособности memory-driven логических схем. Основными результатами, внедренными в производственный процесс следует считать: векторные структуры данных и кубитные методы синтеза, тестирования и моделирования, интегрированные в облачную инфраструктуру сервисного обслуживания компонентов цифровых систем на кристаллах в целях повышения качества изделий и выхода годной продукции за счет адресуемости всех вычислительных процессов и явлений.

Основная инновационная идея предложенной «Memory-Address-Transaction» модели вычислений заключается в синтезе и анализе векторных цифровых структур на основе адресуемых элементов памяти, исключающих использование reusable or new logic.

Приведение описания функции и структуры к единому одномерному формату, означает – технологично решать все задачи синтеза и анализа для функциональностей и графов в кубитно-векторной метрике, создающей memory-driven computing на основе выполнения параллельных логических операций.

Облачные компьютинговые сервисы тестирования, моделирования неисправностей и исправного поведения цифровых устройств являются работоспособными, проверенными на представительной выборке стандартных тестовых схем из бибилиотек конференции ISCAS.

Разработанные модели и методы кубитно-векторной технологии и облачных сервисов проектирования, синтеза тестов и моделирования цифровых устройств интегрированы в состав методологического обеспечения среды моделирования Riviera компании Aldec Inc. и используются в процессе проектирования цифровых систем на кристаллах, имплементируемых в блоки FPGA, CPLD.

Директор ООО «Алдек-КТС», к.т.н. _____ / Зайченко С.О. /

25.11.2016г.

# ДОДАТОК Б

## Swift codes of Applications

```
//
// AppDelegate.swift
// QuantumModeling
//

import Cocoa

@NSApplicationMain
class AppDelegate: NSObject, NSApplicationDelegate {

    var fileURL: URL?
    let openPanel = NSOpenPanel ()

    // MARK: methods
    func applicationDidFinishLaunching (_ aNotification: Notification)
{
        let newDocButton = NSButton ()
        newDocButton.bezelStyle = .texturedRounded
        newDocButton.title = "New Document"
        newDocButton.sizeToFit ()
        newDocButton.target = self
        newDocButton.action = #selector (AppDelegate.createNewDoc)
        openPanel.accessoryView = newDocButton
// NSDocumentController.sharedDocumentController (). BeginOpenPanel
(openPanel, forTypes: [ "txt"]) {(result) in
// if result == NSCancelButton {
// let newDoc = CircuitViewDocument (circuitView: SimpleCircuit ())
// do {
// try newDoc.writeToURL (self.openPanel.URLs [0], ofType: "txt")
//} catch {
// print (error)
//}
//}
//}

    }

    func createNewDoc () {
        let newDoc = CircuitViewDocument (circuitView: SimpleCircuit
())
        newDoc.makeWindowControllers ()
    }

    func applicationWillTerminate (_ aNotification: Notification) {
        // Insert code here to tear down your application
    }

    // MARK: Methods
```

```
    @IBAction func saveButtonTappped (_ sender: NSMenuItem) {
        if fileURL == nil {
            saveAsButtonTapped (sender)
            return
        }

    }

    @IBAction func saveAsButtonTapped (_ sender: NSMenuItem) {
        let savePanel = NSSavePanel ()
        let window = NSApplication.shared (). keyWindow!
        savePanel.beginSheetModal (for: Window, completionHandler: {
            result in
            if result == NSModalResponseOK {
                let newDoc = NSDocument ()
            }
        })
    }
}

//
// QVectorTableController.swift
// QuantumModeling
//

import Cocoa

class QVectorTableController: NSViewController {

    @IBOutlet weak var truthTableView: NSTableView!
    // MARK: Properties
    var truthTable = [([Bool], [Bool])] ()
    var elements = [Element] ()
    var circuit: Circuit?

    override func viewDidLoad () {
        super.viewDidLoad ()
        assembleTruthTable ()
      // let width = 42 + 3 + max (60, circuit! .inputPorts.count *
14)
      // let height = 20 * Int (pow (2, Double (circuit!
.inputPorts.count)))
    // view.frame = CGRectMake (view.frame.minX, view.frame.minY,
CGFloat (width), CGFloat (height))

        truthTableView.tableColumns [0] .sizeToFit ()
        for outElem in elements {
            let tableColumn = NSTableColumn (identifier: "Output
\(outElem.index) ")// Додати індекс портів
            truthTableView.addTableColumn (tableColumn)
            tableColumn.headerCell = NSTableHeaderCell (textCell:
"Output \(outElem.index) ")
            tableColumn.sizeToFit ()
        }
        var width = CGFloat (0)
```

```
        for column in truthTableView.tableColumns {
            width + = column.width + 3
        }

        let rows = truthTable.count +1
        let height = CGFloat (min (rows * 20, 500))
        view.setFrameSize (CGSize (width: width +10, Height: height))
    }

    fileprivate func assembleTruthTable () {
// guard circuit! = Nil else {
// print ( "circuit not found")
// return
//}
// let inputsNumber = circuit! .inElems.count
// let inputStatesNum = Int (pow (2, Double (inputsNumber)))
// for i in 0 .. <inputStatesNum {
// var inputValues = [Bool] (count: inputsNumber, repeatedValue:
false)
// for var j = inputsNumber-1, z = 0; j> = 0; j - = 1, z + = 1 {
// inputValues [z] = Bool ((i & (1 << j)) >> j)
//}
// let outputValues = circuit! .simulateForInput (inputValues)
// truthTable.append ((inputValues, outputValues))
//}
        let inputsNumber = circuit! .inPorts.count
        let inputStatesNum = Int (pow (2, Double (inputsNumber)))
        for i in 0.. <inputStatesNum {
            var inputValues = [Bool] (repeating: false, Count:
inputsNumber)
            var j = inputsNumber - 1
            var z = 0
            while j> = 0 {
                inputValues [z] = Bool (((i & (1<< j)) >> j) as
NSNumber)
                z + = 1
                j - = 1
            }
            var outputValues = [Bool] ()
            for elem in elements {
                outputValues.append (circuit! .modelElement (elem,
forInputValues: inputValues))
            }
            truthTable.append ((inputValues, outputValues))
        }
    }

}

extension QVectorTableController: NSTableViewDataSource {
    func numberOfRows (in tableView: NSTableView) -> Int {
        return truthTable.count
    }
}
```

```
extension QVectorTableController: NSTableViewDelegate {

    func tableView (_ tableView: NSTableView, ViewFor tableColumn:
NSTableColumn?, Row: Int) -> NSView? {
        let cell = tableView.make (withIdentifier: "QTableCell",
Owner: self) as! NSTableCellView
        if tableColumn! .identifier == "InputSetsColumn" {
            var str = ""
            for i in truthTable [row].0 {
                str + = String (Int (i as NSNumber))
                str + = ""
            }
            cell.textField! .stringValue = str
        } else {
            let index = tableView.column (withIdentifier: tableColumn!
.identifier) -1
            cell.textField! .stringValue = String (Int (truthTable
[row].1[Index] as NSNumber))
        }
        return cell
    }

}

//
// CircuitDelegate.swift
// QuantumModeling
//
// Created by Ivan Hahanov on 12.03.16.
// Copyright © 2016 Ivan Hahanov. All rights reserved.
//

import Foundation

protocol CircuitDelegate {
    // func modelingVectorChanged ()
    func onAddingElement (_ element: Element)
    func reloadData ()
}
//
// CircuitDesignerDataSource.swift
// QuantumModeling
//
// Created by Ivan Hahanov on 13.03.16.
// Copyright © 2016 Ivan Hahanov. All rights reserved.
//

import Cocoa

protocol CircuitDesignerDataSource {
    var circuit: CircuitView? {get set }
    var updatedFrame: NSRect { get }
    var circuitDesignerView: CircuitDesignerView! {get set }
    var tracker: NSBezierPath? {get set }
    var dragInterType: DragInteractionType { get set }
```

```
                                }
//
// CircuitDesignerView.swift
// QuantumModeling
//

import Cocoa

class CircuitDesignerView: NSView {

    var dataSource: CircuitDesignerDataSource?

    override func updateTrackingAreas () {
        super.updateTrackingAreas ()
        removeTrackingArea (trackingAreas [0])
        let trackingArea = NSTrackingArea (rect: bounds, options:
[.activeInKeyWindow, .mouseMoved], owner: self, UserInfo: nil)
        addTrackingArea (trackingArea)
    }

    override var isFlipped: Bool {
        return true
    }


    override func draw (_ dirtyRect: NSRect) {
        super.draw (dirtyRect)

        NSColor.black.setStroke ()

        for (_, Link) in dataSource! .circuit! .links {
            link.stroke ()
        }
        if dataSource? .dragInterType == .linking {
            NSColor.blue.setStroke ()
        } else if dataSource? .dragInterType == .selection {
            NSColor.black.setStroke ()
            NSColor.blue.withAlphaComponent (0.1) .setFill ()
            dataSource? .tracker? .fill ()
        }

        dataSource! .tracker? .stroke ()
    }}
//
// CircuitEditorWindowController.swift
// QuantumModeling
//
// Created by Ivan Hahanov on 13.06.16.
// Copyright © 2016 Ivan Hahanov. All rights reserved.
//

import Cocoa

class CircuitEditorWindowController: NSWindowController {
```

```
    // MARK: Properties
    @IBOutlet weak var toolbar: NSToolbar!
    @IBOutlet weak var simulateButton: NSToolbarItem!

    // MARK: Methods
    override func windowDidLoad () {
        super.windowDidLoad ()
        let splitViewController = contentViewController as!
SplitViewController
        let editorViewController =
splitViewController.childViewControllers [1] as! EditorViewController
   // NSNotificationCenter.defaultCenter (). AddObserver
(editorViewController, selector: Selector (
"resizeDesignerViewToFit"), name: NSWindowDidResizeNotification,
object: window)

    }

    @IBAction func collapsingSideMenu (_ sender: AnyObject) {
        let splitVC = contentViewController as! SplitViewController

        let segControl = sender as! NSSegmentedControl
        if segControl.selectedSegment == 0 {
            splitVC.splitViewItems [0] .isCollapsed =!
SplitVC.splitViewItems [0] .isCollapsed
        } else {
            splitVC.splitViewItems [2] .isCollapsed =!
SplitVC.splitViewItems [2] .isCollapsed
        }
    }

    override func prepare (for segue: NSStoryboardSegue, sender: Any?)
{
        if segue.identifier == "SimulationTableSegue" {
            let editorViewController = contentViewController as!
EditorViewController
            let tableViewController = segue.destinationController as!
ViewController
            tableViewController.circuit = editorViewController.circuit
        } else if segue.identifier == "QTableSegue" {
            let splitViewController = contentViewController as!
SplitViewController
            let editorViewController =
splitViewController.childViewControllers [1] as! EditorViewController
            let targerViewController = segue.destinationController as!
QVectorTableController
            targerViewController.circuit =
editorViewController.circuit
            for outPort in editorViewController.circuit.outPorts {
                if outPort.element! = nil {
                    targerViewController.elements.append
(outPort.element!)
                }
            }
        }
```

```
    }


}

extension CircuitEditorWindowController: NSToolbarDelegate {

}

//
// CircuitTableDataSource.swift
// QuantumModeling
//
//

import Cocoa

protocol CircuitTableDataSource {
    var circuit: CircuitView? {get set }
   // var cellSize: CGSize {get}
 // var circuitTableView: CircuitTableView! {Get set}
                                }
////
//// CircuitTableView.swift
//// QuantumModeling
////
////
//
import Cocoa
//
class CircuitTableView: NSView {
//
// var dataSource: CircuitTableDataSource?
//
// override func drawRect (dirtyRect: NSRect) {
// super.drawRect (dirtyRect)
//
// let cellSize = dataSource! .cellSize
//
// // 1
// var startPoint = NSPoint (x: 0, y: bounds.height - cellSize.height-
3)
// var endPoint = NSPoint (x: bounds.width, y: startPoint.y)
// var rectangle = NSRect (x: 0, y: bounds.height - cellSize.height,
width: cellSize.width, height: cellSize.height)
// let paragraphStyle = NSMutableParagraphStyle ()
// paragraphStyle.alignment = .Center
// let attrs = [NSFontAttributeName: NSFont (name: "Helvetica", size:
cellSize.height * 0.75) !, NSParagraphStyleAttributeName:
paragraphStyle]
// NSColor.blackColor (). SetStroke ()
//
// let L: NSString = "L"
// L.drawInRect (rectangle, withAttributes: attrs)
// drawLine (startPoint, endPoint)
```

```
//
// var i = 1
// let sortedElements = dataSource! .circuit! .elements.sort ({
// if $ 0.1 is InputPortElementView && $ 1.1 is QElementView {
// return true
//} else if $ 1.1 is InputPortElementView && $ 0.1 is QElementView {
// return false
//} else {
// return $ 0.0 <$ 1.0
//}})
// for (index, _) in sortedElements {
// let shift = CGFloat (i) * cellSize.width
// let x = shift
// let y = bounds.height - cellSize.height
// let rect = NSRect (x: x, y: y, width: cellSize.width, height:
cellSize.height)
//
// let strValue: NSString = "\ (index)"
// strValue.drawInRect (rect, withAttributes: attrs)
// i + = 1
//}
//
// // 2
// let M: NSString = "M"
// startPoint.y - = cellSize.height
// endPoint.y - = cellSize.height
// rectangle.origin.y - = cellSize.height
// M.drawInRect (rectangle, withAttributes: attrs)
// drawLine (startPoint, endPoint)
//
// var vectorShift = dataSource! .circuit! .modelingVectorLength-1
// let modelingVector = dataSource! .circuit! .modelingVector
// for i in 1 .. <dataSource! .circuit! .modelingVectorLength + 1 {
// let shift = CGFloat (i) * cellSize.width
// let x = shift
// let y = bounds.height - 2 * cellSize.height
// let rect = NSRect (x: x, y: y, width: cellSize.width, height:
cellSize.height)
//
// let value = (modelingVector & (UInt64 (1) << UInt64 (vectorShift)))
>> UInt64 (vectorShift)
// vectorShift - = 1
//
// let strValue: NSString = "\ (value)"
// strValue.drawInRect (rect, withAttributes: attrs)
//
//}
//
//
// // 3
// var lowestValue = 0
// let XTableXShift = dataSource! .circuit! .inputPorts.count
// let XTableYShift = 2
//
// i = XTableXShift
```

```
// for (_, element) in sortedElements {
// if element is QElementView {
// let qElementView = element as! QElementView
// let height = qElementView.links.count
// if lowestValue <height {
// lowestValue = height
//}
// let horizontalShift = CGFloat (i + 1) * cellSize.width
// var j = XTableYShift
// let sorted = qElementView.links.sort ({$ 0.0 <$ 1.0})
// for (key, _) in sorted {
// let verticalShift = bounds.height - cellSize.height - CGFloat (j) *
cellSize.height
//
// let x = horizontalShift
// let y = verticalShift
//
// let rect = NSRect (x: x, y: y, width: cellSize.width, height:
cellSize.height)
// let value = key
// let strValue: NSString = "\ (value)"
// strValue.drawInRect (rect, withAttributes: attrs)
// j + = 1
//
//}
// i + = 1
//}
//}
//
// let X: NSString = "X"
// var vertShift = bounds.height - 2 * cellSize.height - (CGFloat
(lowestValue) * cellSize.height / 2) -cellSize.height / 2
// startPoint.y - = cellSize.height * CGFloat (lowestValue)
// endPoint.y - = cellSize.height * CGFloat (lowestValue)
// rectangle.origin.y = vertShift
// X.drawInRect (rectangle, withAttributes: attrs)
// drawLine (startPoint, endPoint)
//
// // 4
// let QTableXShift = XTableXShift
// let QTableYShift = lowestValue * Int (cellSize.height) + 2 * Int
(cellSize.height)
// lowestValue = 0
// i = QTableXShift
// for (_, element) in sortedElements {
// if element is QElementView {
// let qElementView = element as! QElementView
// let height = Int (pow (2.0, Double (qElementView.links.count)))
// if lowestValue <height {
// lowestValue = height
//}
// let x = CGFloat (i + 1) * cellSize.width
// var currentShift = height - 1
// for j in QTableYShift .. <height + QTableYShift {
//
```

```
// let y = bounds.height - CGFloat (QTableYShift) - cellSize.height -
cellSize.height * CGFloat (j-QTableYShift)
// let rect = NSRect (x: x, y: y, width: cellSize.width, height:
cellSize.height)
// let qVector = qElementView.qVector
// let value = ((qVector & (1 << UInt (currentShift))) >> UInt
(currentShift))
// currentShift - = 1
// let strValue: NSString = "\ (value)"
// strValue.drawInRect (rect, withAttributes: attrs)
//
//}
// i + = 1
//}
//
//}
//
// let Q: NSString = "Q"
// vertShift = bounds.height - CGFloat (QTableYShift) - (CGFloat
(lowestValue) * cellSize.height / 2) -cellSize.height / 2
// rectangle.origin.y = vertShift
// Q.drawInRect (rectangle, withAttributes: attrs)
//
// startPoint = NSPoint (x: cellSize.width, y: bounds.height)
// endPoint = NSPoint (x: cellSize.width, y: 0)
// drawLine (startPoint, endPoint)
//
// startPoint.x + = CGFloat (XTableXShift) * cellSize.width
// endPoint.x + = CGFloat (XTableXShift) * cellSize.width
// drawLine (startPoint, endPoint)
//}
//
//
//}
//
// public func drawLine (startPoint: CGPoint, _ endPoint: CGPoint) {
// let path = NSBezierPath ()
// path.lineWidth = 2
// path.moveToPoint (startPoint)
// path.lineToPoint (endPoint)
// path.stroke ()
}
//
// CircuitView.swift
// QuantumModeling
//
//

import Cocoa

typealias HashCode = Double

protocol CircuitView: Circuit, NSCoding {

    var elemViews: [HashCode: ElementView] { get set }
```

```
    var links: [HashCode: LinkView] { get set }

    func linkElementAtHash (_ from: HashCode, ToElemAtHash to:
HashCode)
    func deleteLinkAtHash (_ hash: HashCode)
    func deleteQElemAtHash (_ hash: HashCode)
    func deleteInputPortAtHash (_ hash: HashCode)
    func deleteOutputPortAtHash (_ hash: HashCode)
    func addInPortElementView ()
    func addQElementView ()
    func addOutPortElementView ()
    func isReady () ->Bool
    //
// CircuitViewDocument.swift
// QuantumModeling
//
//

import Cocoa

class CircuitViewDocument: NSDocument {

    var circuit: CircuitView

    / *
    override var windowNibName: String? {
        // Override returning the nib file name of the document
        // If you need to use a subclass of NSWindowController or if
your document supports multiple NSWindowControllers, you should remove
this method and override -makeWindowControllers instead.
        return "CircuitDocument"
    }
    * /

    override func makeWindowControllers () {
        let editorWC = NSStoryboard (name: "Main", Bundle:
Bundle.main) .instantiateController (withIdentifier:
"CircuitEditorWindowController") as! CircuitEditorWindowController
        let editorVC = ((editorWC.contentViewController as!
SplitViewController) .childViewControllers [1] as!
EditorViewController)
        editorVC.circuit = circuit
        addWindowController (editorWC)
    }

    override func windowControllerDidLoadNib (_ aController:
NSWindowController) {
        super.windowControllerDidLoadNib (aController)
        // Add any code here that needs to be executed once the
windowController has loaded the document's window.
    }

    override func data (ofType typeName: String) Throws -> Data {
        var data = Data ()
        let circuitData = NSArchiver.archivedData (withRootObject:
```

```
circuit)
        let str = circuitData.base64EncodedString (options:
NSData.Base64EncodingOptions.lineLength64Characters)
        data = str.data (using: String.Encoding (rawValue: 0))!
        return data
        // Insert code here to write your document to data of the
specified type. If outError! = NULL, ensure that you create and set an
appropriate error when returning nil.
        // You can also choose to override -fileWrapperOfType: error
:, -writeToURL: ofType: error :, or -writeToURL: ofType:
forSaveOperation: originalContentsURL: error: instead.
        // throw NSError (domain: NSOSStatusErrorDomain, code:
unimpErr, userInfo: nil)
    }

    override func read (from data: Data, OfType typeName: String)
Throws {
        circuit = NSUnarchiver.unarchiveObject (with: data) as!
CircuitView
        // Insert code here to read your document from the given data
of the specified type. If outError! = NULL, ensure that you create and
set an appropriate error when returning false.
        // You can also choose to override -readFromFileWrapper:
ofType: error: or -readFromURL: ofType: error: instead.
        // If you override either of these, you should also override -
isEntireFileLoaded to return false if the contents are lazily loaded.
    // throw NSError (domain: NSOSStatusErrorDomain, code: unimpErr,
userInfo: nil)
    }

    override class func autosavesInPlace () -> Bool {
        return true
    }

    init(CircuitView: CircuitView) {
        self.circuit = circuitView
    }

}
//
// CircuitViewDocument.swift
// QuantumModeling
//
//

import Cocoa

class CircuitViewDocument: NSDocument {

    var circuit: CircuitView

    / *
    override var windowNibName: String? {
    // Override returning the nib file name of the document
    // If you need to use a subclass of NSWindowController or if your
```

```
document supports multiple NSWindowControllers, you should remove this
method and override -makeWindowControllers instead.
    return "CircuitDocument"
    }
    * /

    override func makeWindowControllers () {
        let editorWC = NSStoryboard (name: "Main", Bundle:
Bundle.main) .instantiateController (withIdentifier:
"CircuitEditorWindowController") as! CircuitEditorWindowController
        let editorVC = ((editorWC.contentViewController as!
SplitViewController) .childViewControllers [1] as!
EditorViewController)
        editorVC.circuit = circuit
        addWindowController (editorWC)
    }

    override func windowControllerDidLoadNib (_ aController:
NSWindowController) {
        super.windowControllerDidLoadNib (aController)
        // Add any code here that needs to be executed once the
windowController has loaded the document's window.
    }

    override func data (ofType typeName: String) Throws -> Data {
    var data = Data ()
    let circuitData = NSArchiver.archivedData (withRootObject:
circuit)
    let str = circuitData.base64EncodedString (options:
NSData.Base64EncodingOptions.lineLength64Characters)
    data = str.data (using: String.Encoding (rawValue: 0))!
    return data
    // Insert code here to write your document to data of the
specified type. If outError! = NULL, ensure that you create and set an
appropriate error when returning nil.
    // You can also choose to override -fileWrapperOfType: error :, -
writeToURL: ofType: error :, or -writeToURL: ofType: forSaveOperation:
originalContentsURL: error: instead.
    // throw NSError (domain: NSOSStatusErrorDomain, code: unimpErr,
userInfo: nil)
    }

    override func read (from data: Data, OfType typeName: String)
Throws {
    circuit = NSUnarchiver.unarchiveObject (with: data) as!
CircuitView
    // Insert code here to read your document from the given data of
the specified type. If outError! = NULL, ensure that you create and
set an appropriate error when returning false.
    // You can also choose to override -readFromFileWrapper: ofType:
error: or -readFromURL: ofType: error: instead.
    // If you override either of these, you should also override -
isEntireFileLoaded to return false if the contents are lazily loaded.
    // throw NSError (domain: NSOSStatusErrorDomain, code: unimpErr,
userInfo: nil)
```

```
    }

    override class func autosavesInPlace () -> Bool {
        return true
    }

    init(CircuitView: CircuitView) {
        self.circuit = circuitView
    }

}
//
// DictionaryExtensions.swift
// QuantumModeling
//
//

import Cocoa



enum Errors: Error {
    case unconvertable (String)
}
//
// EditorViewController.swift
// QuantumModeling
//
//

import Cocoa

enum DragInteractionType {
    case linking
    case selection
    case drag
}

class EditorViewController: NSViewController, NSTextFieldDelegate {

    // MARK: Outlets
    @IBOutlet weak var scrollView: NSScrollView!
    @IBOutlet weak var circuitDesignerView: CircuitDesignerView!
    @IBOutlet weak var heightConstraint: NSLayoutConstraint!
    @IBOutlet weak var widthConstraint: NSLayoutConstraint!

    // MARK: Properties
    var circuit: CircuitView = SimpleCircuit ()
    var circuitDesignerModel: CircuitDesignerDataSource =
CircuitViewSupply ()

    var selectedObjects = [HashCode: Selectable] ()
    var linkHash: HashCode?
    var hitHash: HashCode?
    var dragInterType = DragInteractionType.drag
```

```
    var prevDragLoc = CGPoint.zero

    // MARK: Methods
    override func viewDidLoad () {
        super.viewDidLoad ()
        view.wantsLayer = true
        view.acceptsTouchEvents = true
        circuitDesignerView.dataSource = circuitDesignerModel
        circuitDesignerModel.circuitDesignerView = circuitDesignerView
        circuit.delegate = self
        scrollView.hasVerticalScroller = true
        scrollView.hasHorizontalScroller = true
      // scrollView.borderType = .NoBorder
        circuitDesignerModel.circuit = circuit
        scrollView.translatesAutoresizingMaskIntoConstraints = false
        circuitDesignerView.translatesAutoresizingMaskIntoConstraints
= false
        let trackingArea = NSTrackingArea (rect:
circuitDesignerView.bounds, options: [.mouseMoved,
.activeInKeyWindow], owner: self, UserInfo: nil)
        circuitDesignerView.addTrackingArea (trackingArea)
        scrollView.allowsMagnification = true
        scrollView.maxMagnification = 1
        scrollView.minMagnification = 0.5
    }

    override func viewDidLayout () {
        reloadData ()
    }

// private func resizeDesignerViewBounds () {
// guard! Circuit.elemViews.isEmpty else {return}
//// var occupiedArea = (circuit.elements.first! .1 as! ElementView)
.view.frame
//// for (_, element) in circuit.elements {
//// let elemView = element as! ElementView
//// occupiedArea = CGRectUnion (occupiedArea, elemView.view.frame)
////}
//// if occupiedArea.width> circuitDesignerView.bounds.width {
//// circuitDesignerView.bounds = CGRectMake (0, 0,
occupiedArea.width, circuitDesignerView.bounds.height -
(circuitDesignerView.bounds.height-occupiedArea.width))
////}
////
//// let frame = CGRectMake (circuitDesignerView.frame.minX,
circuitDesignerView.frame.minY, max (circuitDesignerView.bounds.width,
occupiedArea.width + 20), max (circuitDesignerView.bounds.height,
occupiedArea.height + 20) )
//// circuitDesignerView.frame = frame
// var occupiedArea = (circuit.elements.first! .1 as! ElementView)
.view.frame
// for (_, element) in circuit.elements {
// let elemView = element as! ElementView
// occupiedArea = CGRectUnion (occupiedArea, elemView.view.frame)
```

```
//}
// circuitDesignerView.frame = CGRectUnion (occupiedArea,
circuitDesignerView.frame)
//}

    override func prepare (for segue: NSStoryboardSegue, sender: Any?)
{
        if segue.identifier == "QTableSegue" {
            let targerViewController = segue.destinationController as!
QVectorTableController
            targerViewController.circuit = circuit
        } else if segue.identifier == "PopoverSegue" {
            let targerViewController = segue.destinationController as!
QVectorTableController
            targerViewController.circuit = circuit
        }
    }

    // MARK: Help Functions
    fileprivate func positionElement (_ elemView: ElementView,
AtCoords coordinates: NSPoint) {
        let roundedCoords = CGPoint (x: coordinates.x-
(coordinates.x.truncatingRemainder (dividingBy: 10)), Y:
coordinates.x- (coordinates.x.truncatingRemainder (dividingBy: 10)))
        elemView.frame.origin = roundedCoords
    }

    fileprivate func moveObjectsToLoc (_ location: NSPoint) {
        let roundedLocation = NSPoint (x: location.x-
(location.x.truncatingRemainder (dividingBy: 10)), Y: location.y-
(location.y.truncatingRemainder (dividingBy: 10)))
        let dx = roundedLocation.x - prevDragLoc.x
        let dy = roundedLocation.y - prevDragLoc.y
        moveObjctsByDelta (dx: dx, dy: dy)
// for (_, obj) in selectedObjects {
// if let elementView = obj as? ElementView {
// elementView.frame.origin.x + = roundedLocation.x - prevDragLoc.x
// elementView.frame.origin.y + = roundedLocation.y - prevDragLoc.y
//
//} else {
//
//}
//}
        prevDragLoc = roundedLocation
    }

    fileprivate func moveObjctsByDelta (dx: CGFloat, Dy: CGFloat) {
        for (_, Obj) in selectedObjects {
            if let elementView = obj as? ElementView {
                elementView.frame.origin.x + = dx
                elementView.frame.origin.y + = dy

            } else {

            }
```

```
        }

    }

    fileprivate func moveElementsByDelta (dx: CGFloat, Dy: CGFloat) {
        for (_, ElemView) in circuit.elemViews {
            elemView.frame.origin.x + = dx
            elemView.frame.origin.y + = dy
        }
    }

    fileprivate func resizeDesignerViewToFit () {
        guard! circuit.elemViews.isEmpty else {return}
        var occupiedArea = circuit.elemViews.first !.1.frame
        for (_, ElemView) in circuit.elemViews {
            let elemView = elemView
            occupiedArea = occupiedArea.union (elemView.frame)
        }
        let fullArea = occupiedArea.union (circuitDesignerView.frame)

        var point = scrollView.documentVisibleRect.origin
        if occupiedArea.minX < 0 {
            widthConstraint.constant = fullArea.width + 100
            moveElementsByDelta (dx: -occupiedArea.minX + 100, Dy: 0)
            point.x + = -occupiedArea.minX + 100
            scrollView.contentView.scroll (to: point)
        }
        if occupiedArea.minY < 0 {
            heightConstraint.constant = fullArea.height + 100
            moveElementsByDelta (dx: 0, Dy: -occupiedArea.minY + 100)
            point.y + = -occupiedArea.minY + 100
            scrollView.contentView.scroll (to: point)
        }
    }

    fileprivate func hitObject (_ point: NSPoint) -> (HashCode,
Selectable)? {
// if let elemView = NSApplication.sharedApplication (). KeyWindow?
.contentView? .hitTest (point) as? ElementView {
// return (elemView.index, elemView)
//}
        // let convertedLocation = NSApplication.sharedApplication ().
KeyWindow! .contentView! .convertPoint (point, toView:
circuitDesignerView)
        let splitViewController = parent as! SplitViewController
        let convertedLocation = splitViewController.view.convert
(point, to: circuitDesignerView)
        for (Hash, elemView) in circuit.elemViews {
            if elemView.frame.contains (convertedLocation) {
                return (Hash, elemView)
            }
        }

        for (Hash, link) in circuit.links {
            if link.containsPoint (convertedLocation) {
```

```
                return (Hash, link)
            }
        }
        return nil
    }

    fileprivate func selectObject (_ object :(HashCode,Selectable)) {
        selectedObjects [object.0] = Object.1
        object.1.selected = true
    }

    fileprivate func deselectObjectAtHash (_ hash: HashCode) {
        selectedObjects [hash] ?. selected = false
        selectedObjects [hash] = nil
    }

    func deselectObjects () {
        for (_, Obj) in selectedObjects {
            obj.selected = false
        }
        selectedObjects = [:]
    }

    // MARK: NSTextFieldDelegate
    override func controlTextDidEndEditing (_ obj: Notification) {
        NSApplication.shared (). KeyWindow? .makeFirstResponder (view)

        deselectObjects ()
        circuitDesignerView! .setNeedsDisplay (circuitDesignerView!
.bounds)
    }

    // MARK: Handling Mouse
    override func mouseMoved (with theEvent: NSEvent) {
        if hitObject (theEvent.locationInWindow)! = nil {
            NSCursor.pointingHand (). Set ()
            return
        }
        NSCursor.arrow (). Set ()
    }

    override func mouseDown (with theEvent: NSEvent) {
        let splitViewController = parent as! SplitViewController
        let convertedLocation = splitViewController.view.convert
(theEvent.locationInWindow, to: circuitDesignerView)

        prevDragLoc = CGPoint (x: convertedLocation.x-
(convertedLocation.x.truncatingRemainder (dividingBy: 10)), Y:
convertedLocation.y- (convertedLocation.y.truncatingRemainder
(dividingBy: 10)))
        NSApplication.shared (). KeyWindow? .makeFirstResponder (view)
// presentedViewControllers? [0] .dismissController (self)
        let object = hitObject (theEvent.locationInWindow)
        if object! = nil {
            if (TheEvent.modifierFlags.rawValue &
```

```
NSEventModifierFlags.control.rawValue) == 0 {
                if selectedObjects [object !.0] == nil {
                    deselectObjects ()
                }
                dragInterType = .drag
            } else {
                deselectObjects ()
                dragInterType = .linking
            }
            selectObject (object!)
            if theEvent.clickCount == 2 {
                if let link = object !.1 as? LinkView {
                    if link.from is InputPortElementView {return}
                    let qTableController = storyboard!
.instantiateController (withIdentifier: "QVectorTableController") as!
QVectorTableController
                    qTableController.circuit = circuit
                    let rect = CGRect (x: convertedLocation.x, y:
convertedLocation.y, width: 1, Height: 1)
                    qTableController.elements = [link.from]
                    presentViewController (qTableController,
asPopoverRelativeTo: rect, of: circuitDesignerView, preferredEdge:
.maxX, behavior: .semitransient)
                } else if let qElemView = object !.1 as? QElementView
{
                    let qTableController = storyboard!
.instantiateController (withIdentifier: "QVectorTableController") as!
QVectorTableController
                    let testCircuit = SimpleCircuit ()
                    testCircuit.addQElementView ()
                    testCircuit.qElems [0] .qVector =
qElemView.qElem.qVector
                    for _ in qElemView.qElem.elements {
                        testCircuit.addInPortElementView ()
                        let inPort = testCircuit.inPorts.last
                        testCircuit.qElems [0] .elements.append
(InPortElement (inputPort: inPort!))
                    }
                    qTableController.circuit = testCircuit
                    qTableController.elements = [testCircuit.qElems
[0]]
                    presentViewController (qTableController,
asPopoverRelativeTo: qElemView.frame, of: circuitDesignerView,
preferredEdge: .maxX, behavior: .semitransient)
                }
            }
            return
        } else {
            dragInterType = .selection
        }
        deselectObjects ()
   // circuitDesignerView.setNeedsDisplayInRect
(circuitDesignerView.bounds)
    }
```

```
    override func mouseDragged (with theEvent: NSEvent) {
  // elementInputField? .removeFromSuperview ()
      // let newLocation = view.convertPoint
(theEvent.locationInWindow, toView: circuitDesignerView!)
        // let newLocation = NSApplication.sharedApplication ().
KeyWindow! .contentView! .convertPoint (theEvent.locationInWindow,
toView: circuitDesignerView)
        let splitViewController = parent as! SplitViewController
        let newLocation = splitViewController.view.convert
(theEvent.locationInWindow, to: circuitDesignerView)
        if dragInterType == .drag {
            moveObjectsToLoc (newLocation)
        } else if dragInterType == .linking {
            circuitDesignerModel.dragInterType = .linking
            if ! (SelectedObjects.first ?.1 is OutputPortElementView)
{

                let mouseTracker = NSBezierPath ()
                mouseTracker.lineWidth = 2
                mouseTracker.move (to: prevDragLoc)
                mouseTracker.line (to: newLocation)
                circuitDesignerModel.tracker = mouseTracker

                let hit = hitObject (theEvent.locationInWindow)
                if hit! = nil &&! (Hit !.1 is InputPortElementView)
&&! (Hit !.1 is LinkView) {
                    hitHash = hit !.0
                    circuit.elemViews [hitHash!] !. highlighted = true
// circuit.linkElementAtHash (selectedObjects.first! .0, toElemAtHash:
hit! .0)
// linkHash = hit! .0 / selectedObjects.first! .0
// circuitDesignerModel.tracker = nil
                } else {
                    if hitHash! = nil {
                        circuit.elemViews [hitHash!] !. highlighted =
false
                        hitHash = nil
                    }
                 // circuitDesignerModel.tracker = Tracker
(fromElement: selectedObjects.first! .1 as! ElementView, cursor:
newLocation)
                }
            }
        } else {
            circuitDesignerModel.dragInterType = .selection
            let roundedPrev = CGPoint (x: Double (Int (prevDragLoc.x))
+0.5, Y: Double (Int (prevDragLoc.y)) +0.5)
            let roundedNew = CGPoint (x: Double (Int (newLocation.x))
+0.5, Y: Double (Int (newLocation.y)) +0.5)
            let selectionArea = CGRect (x: roundedPrev.x, y:
roundedPrev.y, width: roundedNew.x-roundedPrev.x, height:
roundedNew.y-roundedPrev.y)
            let selectionAreaPath = NSBezierPath (rect: selectionArea)
            selectionAreaPath.lineWidth = 1
            circuitDesignerModel.tracker = selectionAreaPath
```

```
            for (Hash, elemView) in circuit.elemViews {
                if elemView.frame.intersects (selectionArea) {
                    selectObject ((hash, elemView))
                } else {
                    deselectObjectAtHash (hash)
                }
            }

        }
        reloadData ()
    }

    override func mouseUp (with theEvent: NSEvent) {
        linkHash = nil
        circuitDesignerModel.tracker = nil
        resizeDesignerViewToFit ()
        if hitHash! = nil {
            circuit.elemViews [hitHash!] !. highlighted = false
            circuit.linkElementAtHash (selectedObjects.first !.0,
ToElemAtHash: hitHash!)
            hitHash = nil
        }
// let splitViewController = parentViewController as!
SplitViewController
// let newLocation = splitViewController.view.convertPoint
(theEvent.locationInWindow, toView: circuitDesignerView)
// for (_, obj) in selectedObjects {
// if let elemView = obj as? ElementView {
// if elemView.frame.minX <0 {
// moveObjectsToLoc (CGPointMake (100, newLocation.y))
//}
// if elemView.frame.minY <0 {
// moveObjectsToLoc (CGPointMake (newLocation.x, 100))
//}
//}
//}

        reloadData ()
    }

    // MARK: Handling Keyboard
    override func keyDown (with theEvent: NSEvent) {

        if Int (theEvent.characters! .unicodeScalars.first! .value) ==
NSDeleteCharacter {
            for (Hash, obj) in selectedObjects {
                if obj is QElementView {
                    circuit.deleteQElemAtHash (hash)
                } else if obj is InputPortElementView {
                    circuit.deleteInputPortAtHash (hash)
                } else if obj is OutputPortElementView {
                    circuit.deleteOutputPortAtHash (hash)
                } else if obj is LinkView {
                    circuit.deleteLinkAtHash (hash)
                }
```

```
            }
            deselectObjects ()
            reloadData ()

        }
    }


}

// MARK: CircuitDelegate
extension EditorViewController: CircuitDelegate {

    func onAddingElement (_ element: Element) {
        let elemView = element as! ElementView
        let roundedCoords = CGPoint (x: prevDragLoc.x-
(prevDragLoc.x.truncatingRemainder (dividingBy: 10)), Y:
prevDragLoc.y- (prevDragLoc.y.truncatingRemainder (dividingBy: 10)))
        elemView.frame.origin = roundedCoords

        circuitDesignerView! .addSubview (elemView)
        reloadData ()
    }

    func reloadData () {
        circuitDesignerView! .setNeedsDisplay (circuitDesignerView!
.bounds)
  // circuitDesignerView.frame = circuitDesignerModel.updatedFrame
    }

}

extension EditorViewController: NSSplitViewDelegate {

}
```