

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кваліфікаційна наукова праця
на правах рукопису

ДЕМЧЕНКО ОЛЕГ ІВАНОВИЧ

УДК 658:512.011: 681.326: 519.713

ДИСЕРТАЦІЯ
ВЕКТОРНО-ЛОГІЧНІ МОДЕЛІ ЦИФРОВИХ СХЕМ ДЛЯ СИМУЛЯЦІЇ ТА
ВІЗУАЛІЗАЦІЇ

123 – Комп'ютерна інженерія

Технічні науки

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень.

Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело _____ О. І. Демченко

Науковий керівник: Чумаченко Світлана Вікторівна, доктор технічних наук, професор



Харків – 2026

ПЕРЕДМОВА

Автор виражає глибоку вдячність засновнику теорії векторно-логічного комп'ютингу, на підставі якої виконано дисертаційне дослідження, – доктору технічних наук, професору Хаханову Володимиру Івановичу, котрого вважає своїм наставником та мотиватором з часів студентства протягом більш 20 років, що керував аспірантською підготовкою перші два роки й продовжував по сей час надавати консультації з провадження цієї роботи.

Демченко Олег

АНОТАЦІЯ

Демченко О.І. Векторно-логічні моделі цифрових схем для симуляції та візуалізації. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 123 «Комп'ютерна інженерія». – Харківський національний університет радіоелектроніки, Міністерство освіти і науки України, Харків, 2026.

Дисертація присвячена розв'язку науково-практичної задачі щодо економічних векторно-логічних обчислень, орієнтованих на безпроцесорну обробку у пам'яті (in-memory), що засновані на транзакціях читання-запису над розумними структурами даних, якими виступають логічні вектори, таблиці істинності та матриці.

Мета дослідження – зниження часових та енергетичних витрат fault/good-value as address simulation цифрових схем за рахунок надлишковості векторно-логічних моделей їх елементів.

Задачі дослідження:

1. Аналіз стану проблеми та актуальних технологій прогресивного корпусування інтегральних мікросхем, моделювання несправностей цифрових систем та їх компонентів.

2. Вдосконалення розумних структур даних векторної логіки, що зменшують обчислювальну складність алгоритмів fault/good-value as address simulation за рахунок експоненціальної надмірності моделей елементів логічних схем.

3. Розробка архітектури для моделювання схеми та рендерингу процесів синхронізації векторно-логічного modeling for fault/good-value as address simulation, що містить побудову моделі об'єкта у пам'яті комп'ютера та дозволяє візуально супроводжувати процес modeling for simulation цифрової схеми.

4. Розробка векторно-логічних механізмів modeling for fault/good-value as address simulation з лінійною алгоритмічною складністю.

5. Програмна реалізація розумних структур даних, алгоритмів та механізмів fault/good-value as address simulation, синхронний рендеринг всіх компонентів процесу симуляції на вікно монітору з метою візуального супроводжування процесу modeling for simulation цифрової схеми, їх верифікація на бібліотеці стендових схем ISCAS та імплементація до хмарного сервісу MOSI-HDL.

Наукова новизна дослідження представлена векторно-логічними моделями елементів цифрової схеми, механізмами good-value simulation test set as address та fault as address simulation цифрової схеми і fault as address simulation вхідного набору, використовуючи квадратичну матрицю симуляції.

Практичне значення полягає у швидкому розвитку економічно ефективних механізмів та технологій для моделювання та симуляції в пам'яті завдяки інтелектуальному рендерингу та ієрархічній синхронізації всіх процесів моделювання та симуляції проєкту. Вихід запропонованих механізмів моделювання на EDA-ринок передбачає використання стандарту IEEE 1500 SECT для перевірки функціональності та структури SoC на рівнях IP-ядра, гейту, RTL та системи цифрового опису проєкту.

Об'єкт дослідження – векторно-логічний in-memory комп'ютинг для вирішення задач моделювання цифрових систем.

Предмет дослідження – векторно-логічне моделювання несправностей, як адрес, для верифікації цифрових схем.

Наукова новизна результатів досліджень:

1. Удосконалено розумні структури даних, які орієнтовані на безпроцесорний векторно-логічний modeling for fault/good-value as address simulation за допомогою read-write транзакцій, що дозволило зменшити обчислювальну складність алгоритмів симуляції до лінійної.

2. Вперше запропоновано векторно-логічну модель елементів цифрової схеми, що дозволяє здійснити good-value simulation вхідних тестових наборів та

fault as address simulation як адрес бітів логічного та дедуктивного вектора на основі read–write транзакцій та знизити обчислювальну складність алгоритмів тестування логічних функціональностей до лінійної.

3. Вперше запропоновано архітектуру для моделювання схеми та рендерингу процесів синхронізації векторно-логічного modeling for fault/good-value as address simulation, що містить побудову моделі об'єкта у пам'яті комп'ютера та дозволяє проєктувальнику візуально супроводжувати процес modeling for simulation цифрової схеми (вручну будувати тести для покриття несправностей логічних схем).

4. Вперше запропоновано механізми векторно-логічного modeling for fault/good-value as address simulation, що відзначаються лінійною алгоритмічною складністю.

Практична значущість результатів дослідження визначається верифікацією розумних структур даних та алгоритмів моделювання на десятках схем та функціональних елементів з бібліотеки ISCAS. Розроблено хмарний сервіс MOSI-HDL (<http://mosihdl.work>) для економічного вирішення задач modelling for simulation (MOSI) на основі векторно-логічного in-memory комп'ютингу, що орієнтований на використання read-write транзакцій без інструкцій процесора, який містить: 1) векторно-логічні моделі елементів цифрової схеми для здійснення good-value simulation вхідних тестових наборів та симуляцію несправностей як адрес бітів логічного та дедуктивного вектора на основі read-write транзакцій; 2) механізми good-value simulation of test set as address цифрової схеми без процесора; механізми fault as address simulation цифрової схеми без участі процесора; 3) механізм fault as address simulation вхідного набору, використовуючи квадратичну матрицю симуляції; 4) механізми рендерингу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора.

Інтегральною перевагою або новизною даних сервісів є синхронізація процесів моделювання по всіх вікнах, що представляють всі суттєві процедури для розуміння отриманого результату.

Запропоновані моделі, методи та механізми апробовані на міжнародних конференціях з напрямку EDA. Моделі, методи, механізми та алгоритми імплементовано у хмарний сервіс MOSI-HDL (<http://mosihdl.work>), реалізований мовою C++. Розробку представлено на двох виставках Міжнародного молодіжного форуму у 2025 та 2026 роках, де вона стала переможцем.

Обмеження MOSI-HDL обумовлені тим, що він є діючим прототипом та постійно розвивається та доповнюється новими функціоналами. Наприклад, поточна версія підтримує лише комбінаційні схеми без ієрархії, а також інтеграцію з відомими зовнішніми сервісами автоматичного генератора тестів. Перспективи розвитку полягають у створенні інструментарію для генерації тестів, що покривають усі несправності логічних схем будь-якої конфігурації.

Ключові слова: візуалізація, модель, структури даних, моделювання, симуляція, енергоефективність, хмарні технології, комп'ютерні системи (компоненти), двійкові послідовності (вектори), логічні вирази, алгоритм, матриця, VHDL-модель, несправність, векторна логіка.

СПИСОК ОПУБЛІКОВАНИХ РОБІТ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Список публікацій здобувача, в яких опубліковані основні наукові результати дисертації:

1. Демченко О.І., Чумаченко С.В. Технології прогресивного корпусування інтегральних мікросхем / О.І. Демченко, С.В. Чумаченко // Системи управління, навігації та зв'язку. – 2025. – Т. 3, № 81. – С. 199–202. – DOI: 10.26906/SUNZ.2025.3.199. (Фахове видання. Категорія Б)

2. Хаханов В.І., Обрізан В.І., Рахліс Д., Хаханова Г.В., Хаханов І.В., Демченко О.І., Воронов А.О. Механізми схемотехнічного моделювання на основі векторної логіки / В.І. Хаханов, В.І. Обрізан, Д.Ю. Рахліс, Г.В. Хаханова, І.В. Хаханов, О.І. Демченко, А.О. Воронов // Радіоелектроніка, інформатика, управління. – 2025. – № 4. – С. 219–232. – DOI: 10.15588/1607-3274-2025-4-20. (Фахове видання. Web of Science. Категорія А.)

3. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector-logic Models of Digital Circuits for Simulation and Rendering / V. Hahanov, S. Chumachenko, E. Litvinova, A. Voronov, O. Demchenko, N. Maksymova // IAES International Journal of Robotics and Automation (IJRA). — 2026. — Vol. 15, no. 2. — P. 319–330. — ISSN 2089-4856. — DOI: 10.11591/ijra.v15i2.pp319-330. (Закордонне видання. Scopus, Q3).

4. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector logic for robotic system on chip design and test / V. Hahanov, S. Chumachenko, E. Litvinova, A. Voronov, O. Demchenko, N. Maksymova // IAES International Journal of Robotics and Automation (IJRA). – 2026. – Vol. 15, no. 2. – P. 415–426. – ISSN 2089-4856. – DOI: 10.11591/ijra.v15i2.pp415-426. (Закордонне видання. Scopus, Q3)

Результати, які засвідчують апробацію матеріалів дисертації:

5. Hahanov V., Litvinova E., Chumachenko S., Hahanov I., Demchenko O., Voronov A. Vector Logic Modeling Self-Tested Circuits / V. Hahanov, E. Litvinova, S. Chumachenko, I. Hahanov, O. Demchenko, A. Voronov // 2025 IEEE East-West Design & Test Symposium (EWDTS), Tbilisi, Georgia, 12–15 Dec. 2025 : proceedings. – 2025. – 4 p. – DOI: 10.1109/EWDTS67441.2025.11303706. (Scopus)

6. Чумаченко С.В., Демченко О.І. Технології прогресивного корпусування ІМС: основні виклики / С.В. Чумаченко, О.І. Демченко // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матеріали 15-ї міжнар. наук.-техн. конф., Баку – Харків – Жиліна, 24–25 квіт. 2025 р. – Т. 2. – С. 26–27. – Режим доступу: <https://repository.kpi.kharkov.ua/handle/KhPI-Press/93979> (доступ 09.06.2026).

7. Демченко О.І. Програмне середовище MOSI HDL для моделювання векторно-логічних структур / О.І. Демченко // Радіoeлектроніка та молодь у XXI столітті : матеріали XXX міжнар. молодіж. форуму, 22–24 квіт. 2026 р., Харків. – Харків: ХНУРЕ, 2026. – Т. 5. – С. 42–43. – Режим доступу: <https://drive.google.com/file/d/1nd5a4ajsX2Uet6uXvPlQY9dSZu7HTUmq/view> (доступ 09.06.2026).

8. Хаханов І.В., Кулак Г.К., Воронов А.О., Демченко О.І., Максимова Н.Г., Пономарьова В.І. (науковий керівник – д-р техн. наук, проф. Хаханов В.І.). Розробка «Хмарний сервіс MObeling for Simulation (MOSI)» / І.В. Хаханов, Г.К. Кулак, А.О. Воронов, О.І. Демченко, Н.Г. Максимова, В.І. Пономарьова // XXIX Міжнародний молодіжний форум «Радіoeлектроніка та молодь у XXI столітті». Каталог виставки технічної творчості, 16–19 квіт. 2025 р. – 2025. – С. 23–24. – Режим доступу: https://drive.google.com/file/d/1NbFKwy4X_G-kZqzVfQUi0_dnQGItKQ7R/view (доступ 09.06.2026).

9. Демченко О.І., Воронов А.О., Максимова Н.Г. (науковий керівник – д-р техн. наук, проф. Хаханов В.І.). Розробка «HDL Modeling for Simulation Cloud Service (MOSI-HDL)» / О.І. Демченко, А.О. Воронов, Н.Г. Максимова // XXX Міжнародний молодіжний форум «Радіoeлектроніка та молодь у XXI столітті». Каталог виставки технічної творчості молоді, 22–24 квіт. 2026 р. – 2026. – С. 32. – (англійською мовою). – Режим доступу: https://drive.google.com/file/d/1c4ZysLsElmI5fFhCqrZKjuKSkTj_UyBG/view (доступ 09.06.2026).

ABSTRACT

Demchenko O.I. Vector-Logic Models of Digital Circuits for Simulation and Rendering. – Qualifying scientific work on the rights of the manuscript.

The dissertation for the competition PhD scientific degree on a specialty 123 "Computer engineering". – Kharkiv National University of Radio Electronics, Ministry of Education and Science of Ukraine, Kharkiv, 2026.

The dissertation is devoted to the solution of a scientific and practical problem regarding economical vector-logic calculations, oriented to processor-free processing in memory (in-memory), which are based on read-write transactions over intelligent data structures, which are logical vectors, truth tables and matrices.

The purpose of the research is to reduce the time and energy costs for fault/good-value as address simulation of digital circuits due to the redundancy of vector-logic models of their elements.

Research objectives:

1. Analysis of the state of the problem and current technologies of progressive packaging of integrated circuits, modeling of faults of digital systems and their components.
2. Improvement of intelligent vector logic data structures that reduce the computational complexity of fault/good-value as address simulation algorithms due to the exponential redundancy of models of elements of logical circuits.
3. Development of an architecture for circuit modeling and rendering of synchronization processes of vector-logical modeling for fault/good-value as address simulation, which includes building an object model in computer memory and allows for visual monitoring of the modeling for simulation process of a digital circuit.
4. Development of vector-logical modeling for fault/good-value as address simulation mechanisms with linear algorithmic complexity.

5. Software implementation of intelligent data structures, algorithms and fault/good-value as address simulation mechanisms, synchronous rendering of all components of the simulation process on the monitor window for the purpose of visual monitoring of the modeling for simulation process of a digital circuit, their verification on the ISCAS bench circuit library and implementation in the MOSI-HDL cloud service.

The scientific novelty of the research is represented by vector-logic models of digital circuit elements, good-value simulation test set as address and fault as address simulation mechanisms of the digital circuit and fault as address simulation of the input set, using a quadratic simulation matrix.

The practical significance lies in the rapid development of cost-effective mechanisms and technologies for modeling and simulation in memory due to intelligent rendering and hierarchical synchronization of all processes of modeling and simulation of the project. The release of the proposed modeling mechanisms to the EDA market involves the use of the IEEE 1500 SECT standard to verify the functionality and structure of the SoC at the levels of the IP core, gate, RTL and digital description system of the project.

The object of the research is vector-logic in-memory computing for solving problems of modeling digital systems.

The subject of the research is vector-logic modeling of faults, as addresses, for verification of digital circuits.

Scientific novelty of the research results:

1. Intelligent data structures focused on processor-free vector-logical modeling for fault/good-value as address simulation using read-write transactions have been improved, which has reduced the computational complexity of simulation algorithms to linear.

2. For the first time, a vector-logical model of digital circuit elements has been proposed, which allows for good-value simulation of input test sets and fault as address simulation as address bits of a logical and deductive vector based on read-write

transactions and reduces the computational complexity of logical functionality testing algorithms to linear.

3. For the first time, an architecture for circuit modeling and rendering of vector-logical modeling for fault/good-value as address simulation synchronization processes has been proposed, which includes building an object model in computer memory and allows the designer to visually accompany the modeling for simulation process of a digital circuit (manually build tests to cover faults in logical circuits).

4. For the first time, vector-logic modeling mechanisms for fault/good-value as address simulation, characterized by linear algorithmic complexity, are proposed.

Practical Significance of the Research Results is determined by the verification of smart data structures and simulation algorithms on dozens of circuits and functional elements from the ISCAS benchmark library.

The MOSI-HDL cloud service (<http://mosihdl.work>) has been developed for the cost-effective solution of modeling-for-simulation (MOSI) problems based on vector-logic in-memory computing. It is tailored for the use of read-write transactions without processor instructions and includes:

1. Vector-logic models of digital circuit elements to perform good-value simulation of input test sets and the simulation of faults as bit addresses of a logical and deductive vector based on read-write transactions.

2. Processorless mechanisms for the good-value simulation of a digital circuit's test set as an address.

3. Processorless mechanisms for the "fault as address" simulation of a digital circuit.

4. A mechanism for the "fault as address" simulation of an input set using a square simulation matrix.

5. Mechanisms for rendering and synchronizing simulation results across four scalable windows displayed on a monitor screen.

The integral advantage or novelty of these services lies in the synchronization of simulation processes across all windows, which represent all essential procedures necessary for understanding the obtained result.

The proposed models, methods, and mechanisms have been approved at international conferences in the field of EDA. The models, methods, and algorithms are implemented in the MOSI-HDL cloud service, developed in C++. The development was presented at two exhibitions of the International Youth Forum in 2025 and 2026, where it became the winner.

Keywords: Visualization, Model, Method, Data Structures, Modeling, Binary Sequences (Vectors), Simulation, Energy Efficiency, Cloud Technologies, Computer Systems (Components), Logic (Boolean) Expressions, Algorithm, Matrices, VHDL-Model, Fault, Vector Logic.

LIST OF PUBLICATIONS OF THE APPLICANT

List of publications of the applicant in which the main scientific results of the dissertation are published:

1. Демченко О.І., Чумаченко С.В. Технології прогресивного корпусування інтегральних мікросхем / О.І. Демченко, С.В. Чумаченко // Системи управління, навігації та зв'язку. – 2025. – Т. 3, № 81. – С. 199–202. – DOI: 10.26906/SUNZ.2025.3.199. (Фахове видання. Категорія Б)

2. Хаханов В.І., Обрізан В.І., Рахліс Д., Хаханова Г.В., Хаханов І.В., Демченко О.І., Воронов А.О. Механізми схемотехнічного моделювання на основі векторної логіки / В.І. Хаханов, В.І. Обрізан, Д.Ю. Рахліс, Г.В. Хаханова, І.В. Хаханов, О.І. Демченко, А.О. Воронов // Радіоелектроніка, інформатика, управління. – 2025. – № 4. – С. 219–232. – DOI: 10.15588/1607-3274-2025-4-20. (Фахове видання. Web of Science. Категорія А.)

3. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector-logic Models of Digital Circuits for Simulation and Rendering / V. Hahanov, S. Chumachenko, E. Litvinova, A. Voronov, O. Demchenko, N. Maksymova // IAES International Journal of Robotics and Automation (IJRA). —

2026. – Vol. 15, no. 2. – P. 319–330. – ISSN 2089-4856. – DOI: 10.11591/ijra.v15i2.pp319-330. (Закордонне видання. Scopus, Q3).

4. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector logic for robotic system on chip design and test / V. Hahanov, S. Chumachenko, E. Litvinova, A. Voronov, O. Demchenko, N. Maksymova // IAES International Journal of Robotics and Automation (IJRA). – 2026. – Vol. 15, no. 2. – P. 415–426. – ISSN 2089-4856. – DOI: 10.11591/ijra.v15i2.pp415-426. (Закордонне видання. Scopus, Q3)

Results that confirm the validation of the dissertation materials:

5. Hahanov V., Litvinova E., Chumachenko S., Hahanov I., Demchenko O., Voronov A. Vector Logic Modeling Self-Tested Circuits / V. Hahanov, E. Litvinova, S. Chumachenko, I. Hahanov, O. Demchenko, A. Voronov // 2025 IEEE East-West Design & Test Symposium (EWDTS), Tbilisi, Georgia, 12–15 Dec. 2025 : proceedings. – 2025. – 4 p. – DOI: 10.1109/EWDTS67441.2025.11303706. (Scopus)

6. Чумаченко С.В., Демченко О.І. Технології прогресивного корпусування ІМС: основні виклики / С.В. Чумаченко, О.І. Демченко // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матеріали 15-ї міжнар. наук.-техн. конф., Баку – Харків – Жиліна, 24–25 квіт. 2025 р. – Т. 2. – С. 26–27. – Режим доступу: <https://repository.kpi.kharkov.ua/handle/KhPI-Press/93979> (доступ 09.06.2026).

7. Демченко О.І. Програмне середовище MOSI HDL для моделювання векторно-логічних структур / О.І. Демченко // Радіoeлектроніка та молодь у XXI столітті : матеріали XXX міжнар. молодіж. форуму, 22–24 квіт. 2026 р., Харків. – Харків: ХНУРЕ, 2026. – Т. 5. – С. 42–43. – Режим доступу: <https://drive.google.com/file/d/1nd5a4ajsX2Uet6uXvPlQY9dSZu7HTUmq/view> (доступ 09.06.2026).

8. Хаханов І.В., Кулак Г.К., Воронов А.О., Демченко О.І., Максимова Н.Г., Пономарьова В.І. (науковий керівник – д-р техн. наук, проф. Хаханов В.І.).

Розробка «Хмарний сервіс MOdeling for Simulation (MOSI)» / І.В. Хаханов, Г.К. Кулак, А.О. Воронов, О.І. Демченко, Н.Г. Максимова, В.І. Пономарьова // XXIX Міжнародний молодіжний форум «Радіoeлектроніка та молодь у XXI столітті». Каталог виставки технічної творчості, 16–19 квіт. 2025 р. – 2025. – С. 23–24. – Режим доступу: https://drive.google.com/file/d/1NbFKwy4X_G-kZqzVfQUl0_dnQGItKQ7R/view (доступ 09.06.2026).

9. Демченко О.І., Воронов А.О., Максимова Н.Г. (науковий керівник – д-р техн. наук, проф. Хаханов В.І.). Розробка «HDL Modeling for Simulation Cloud Service (MOSI-HDL)» / О.І. Демченко, А.О. Воронов, Н.Г. Максимова // XXX Міжнародний молодіжний форум «Радіoeлектроніка та молодь у XXI столітті». Каталог виставки технічної творчості молоді, 22–24 квіт. 2026 р. – 2026. – С. 32. – (англійською мовою). – Режим доступу: https://drive.google.com/file/d/1c4ZysLsElmI5fFhCqrZKjuKSkTj_UyBG/view (доступ 09.06.2026).

ЗМІСТ

Перелік скорочень, умовних позначень, одиниць і термінів.....	18
Вступ	21
1 Аналіз стану проблеми та актуальних технологій моделювання несправностей цифрових систем.....	33
1.1 Передмова	33
1.2 Технології прогресивного корпусування інтегральних мікросхем. Стан питання	35
1.3 Техніки моделювання несправностей	44
1.4 Комп'ютинг автоматизованого проєктування (Design Automation Computing).....	47
1.5 Архітектури та тестування цифрових проєктів.....	48
1.6 Висновки до розділу 1	53
1.7 Перелік джерел посилання до розділу 1	55
2 Розумні структури даних та векторна логіка їх обробки.....	67
2.1 Передмова	67
2.2 Синтез карти тестування для найпростішої логіки.....	68
2.3 Інжиніринг верифікації логічних функціональностей	70
2.4 Побудова карти тестування функціональності	73
2.5 Розумні структури даних та векторна логіка їх обробки	78
2.6 Висновки до розділу 2	85
2.7 Перелік джерел посилання до розділу 2	88
3 Математичний апарат декартової та векторної логіки для механізмів моделювання	91
3.1 Передмова	91
3.2 Стан питання.....	92
3.3 Векторна та декартова логіка.....	94
3.4 Моделювання логічного вектора для цифрової структури на основі декартової логіки.....	101
3.5 Моделювання логічного вектора схеми шляхом good-value simulation ..	105
3.6 Метод декартової суперпозиції логічних векторів	108
3.7 Обговорення результатів та висновки.....	109

3.8 Перелік джерел посилання до розділу 3	111
4 Верифікація структур даних, механізмів, алгоритмів. Хмарний сервіс MOSI–HDL для моделювання та візуалізації.....	115
4.1 Передмова	115
4.2 Стан питання.....	116
4.3 Рендеринг векторно-логічного моделювання цифрової схеми	119
4.4 Матриця моделювання несправностей	121
4.5 Моделювання дедуктивного вектора та тестової карти	125
4.6 Моделювання таблиці істинності за допомогою дедуктивного вектора ..	129
4.7 Good-value та fault simulation цифрових схем	133
4.8 Обчислювальна складність алгоритмів моделювання.....	135
4.9 Зниження обчислювальної складності алгоритмів симуляції	139
4.10 Обговорення результатів	144
4.11 Архітектура програми MOSI HDL та приклади роботи.....	147
4.12 Обробка даних в MOSI HDL	150
4.13 Висновки до розділу 4	164
4.14 Перелік джерел посилання до розділу 4	165
Висновки.....	171
Додаток А Список публікацій здобувача, в яких опубліковані основні наукові результати дисертації	173
Додаток Б Результати, які засвідчують апробацію матеріалів дисертації	174
Додаток В Документи, що підтверджують впровадження результатів	175
Додаток Г Дипломи переможців виставок науково-технічної творчості	176

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ОДИНИЦЬ І ТЕРМІНІВ

ДНФ – диз’юнктивна нормальна форма представлення логічної функції;

ІМС – інтегральна мікросхема;

КНФ – кон’юнктивна нормальна форма представлення логічної функції;

ASIC – Application-Specific Integrated Circuit (спеціалізовані інтегральні мікросхеми, розроблені для вирішення конкретних завдань);

ATPG – Automatic Test Pattern Generation (автоматична генерація тестових наборів для перевірки працездатності цифрових схем);

CoWoS – Chip-on-Wafer-on-Substrate (технологічна платформа інтеграції кристалів на рівні кремнієвого інтерпозера);

CS – Computational Storage (обчислювальне сховище, яке переносить обробку даних безпосередньо на пристрій, розвантажуючи хост-процесор);

DUT – Device Under Test (реалізація проєкту, яка на даний момент проходить верифікацію або тестування);

EDA – Electronic Design Automation (ринок і системи автоматизованого проєктування електроніки);

EMIB – Embedded Multi-die Interconnect Bridge (технологія вбудованого кремнієвого міжкристального містка для з’єднання чиплетів);

FC – Functional Coverage (метрика плану тестування, що визначає повноту покриття функціональних можливостей проєкту);

Fast Lookup Table-based Estimation (швидка оцінка на основі довідкових таблиць);

FPGA – Field-Programmable Gate Array (програмована логічна інтегральна схема, варіант апаратної реалізації цифрових систем);

HBM – High Bandwidth Memory (високошвидкісна пам’ять із великою пропускнуою здатністю);

HDL – Hardware Description Language (мова опису апаратури, наприклад, Verilog, VHDL, що використовується для моделювання цифрових систем);

IEEE – Institute of Electrical and Electronics Engineers (Інститут інженерів з електротехніки та електроніки, міжнародна організація, що розробляє галузеві стандарти);

ISCAS – International Symposium on Circuits and Systems (Міжнародний симпозиум по схемам та системам);

IP-core – Intellectual Property core (інтелектуальна власність, готовий функціональний блок або ядро проєкту);

LSI – Local Silicon Interconnect (локальні кремнієві з'єднання для забезпечення високої щільності трасування між кристалами);

ML – Machine Learning (машинне навчання, алгоритми пошуку закономірностей у даних без прямого програмування);

MOSI – MOdelling for SIMulation (хмарний сервіс для економічного моделювання та візуалізації процесів симуляції цифрових схем);

MVC – Model-View-Controller (патерн проєктування програмного забезпечення «модель – вигляд – контролер»);

PoP – Package-on-Package (метод вертикального компонування пакетів кристалів один на одному в одному корпусі);

RDL – Redistribution Layer (перерозподільний шар для розведення контактних виводів усередині пакета);

RISC-V – Reduced Instruction Set Computer - Five (відкрита архітектура процесора з обмеженим набором команд п'ятої версії);

RSMT – Rectilinear Steiner Minimum Tree (прямолінійне мінімальне дерево Штейнера);

RTL – Register-Transfer Level (рівень регістрових передач при проєктуванні цифрових логічних систем);

SiP – System-in-Package (система в корпусі, що інтегрує кілька різних кристалів у єдиний високоінтегрований модуль);

SoC – System on Chip (система на кристалі, яка об'єднує процесор, пам'ять та інші компоненти в одному виробі);

STA – Static Timing Analysis (статичний часовий аналіз параметрів затримок сигналів на елементах схеми);

TSH – Through Silicon Hole (вертикальні отвори через кремній без повної металізації, альтернатива TSV);

TSV – Through Silicon Via (металізовані міжперехідні з'єднання через кремній, що проходять крізь кристал);

UHDM – Universal Hardware Data Model (універсальна апаратна модель даних, що використовується при парсингу проєктів);

UUT – Unit Under Test (виріб або блок, що проходить тестування);

VLSI – Very Large Scale Integration (надвелика інтегральна схема –HBIC);

VPI – Verilog Procedural Interface (програмний інтерфейс Verilog для взаємодії із симуляційними моделями);

WLP – Wafer-Level Package (технологія пакування на рівні цілої пластини).

ВСТУП

Modelling for simulation (моделювання для симуляції) цифрових схем забезпечує перевірку та якість цифрового продукту шляхом тестування та діагностики під час проєктування, виробництва та експлуатації. Найпоширенішими є механізми good-value моделювання (моделі та алгоритми) для вхідних даних або тестових наборів, що використовуються для перевірки цифрових проєктів. Для цього потрібен testbench як друга модель цифрового проєкту, який також може містити помилки проєктування.

Більш складними і тому менш поширеними є механізми моделювання несправностей, які використовуються для синтезу тестів, оцінки їх якості та діагностики несправностей і дефектів. Крім того, механізми моделювання несправностей (fault modeling) повинні мати компонент good-value моделювання. Як правило, те, що знаходиться “під капотом” промислових систем моделювання та симуляції, є конфіденційною інформацією, яка не підлягає публікації у відкритих джерелах. Лише небагато фахівців з моделювання несправностей цілком володіють теорією, методами та механізмами, що вбудовані в нові промислові сервіси та академічні продукти. Тому для навчання студентів науковці використовують академічні статті, що розкривають суть методів.

За 80 років розвитку ринку EDA з'явилося не більше десятку методів моделювання та симуляції несправностей або дефектів. Причому їх суть однакова – несправність виявляється, якщо вона змінює вихідний стан елемента або схеми на вхідному або тестовому наборах. Водночас, моделювання несправностей для будь-якої системи є ключовим технологічним фактором, від якого залежить ринок EDA та академічне середовище. Тут слід розглянути два типи існуючих моделей цифрових проєктів. Компіляційні або жорстко-програмні моделі вимагають надлишкового програмного коду для моделювання набору фактичних несправностей та спеціальних алгоритмів для їх

моделювання. Інтерпретаційні або гнучкі явні експоненціально надлишкові моделі цифрового проєктування містять усі несправності за адресами моделі справної поведінки. Це є конструктивною великою перевагою надлишкових векторно-логічних моделей. Тому вони мають прості або нульові алгоритми для свого моделювання. Компіляційні моделі є швидкими, оскільки вони використовують інструкції процесора. Інтерпретативні моделі розроблені для енергоефективних безпроцесорних масових обчислень майбутнього в пам'яті та використовують транзакції читання-запису, які наразі виконуються за 0,6 нс.

Усі сервіси моделювання ринку створюються з використанням технології компіляції. Це означає, що вони використовують інструкції HDL, розроблені для потужних процесорів. Економічними перевагами механізмів компіляції є високе енергоспоживання та низька затримка. Натомість можна запропонувати гнучкі механізми інтерпретаційної векторної логіки, good value, and fault-in-memory simulation на основі транзакцій читання-запису без інструкцій процесора.

Векторно-логічне адресне моделювання цифрових схем для симуляції має наступну архітектуру (рис. 0.1).



Рисунок 01 – Архітектура векторно-логічного моделювання [рисунок сформовано автором та оброблено за допомогою ШІ-сервісу Gemini]

Розробляються прості, розумні, надлишкові та явні структури даних, передбачувані пам'яттю, які суттєво зменшують алгоритмічну складність моделювання *good-value* та *fault-as-address simulation*.

Ці алгоритми використовують транзакції читання-запису без інструкцій процесора, коли вони реалізовані в архітектурі обчислень в пам'яті, що значно зменшує споживання енергії та затримку для механізмів моделювання. Код HDL для логічних схем використовується виключно для побудови явних векторно-логічних структур даних, які передбачувані за розміром пам'яті та використовуються для моделювання *fault-as-address simulation*, *good-value test set as address simulation*, моделювання векторно-логічних схем, синтезу мінімальних тестів та діагностики схем. Мови HDL були створені як стандарти для комунікації між розробниками обладнання та для розробки потужних компіляторів, що дозволяють моделювати великі цифрові проекти під час верифікації. Сьогодні також створені бібліотеки (ISCAS) стендових схем, які дослідники використовують для верифікації та порівняння механізмів моделювання. Тому будь-яка академічна або промислова система моделювання повинна мати інтерфейс HDL для введення проєктів, щоб перетворити опис HDL у внутрішні структури даних, у цьому конкретному випадку, векторно-логічне моделювання для механізмів симуляції (рис. 0.2).



Рисунок 02 – Архітектура та механізми розроблювальної системи моделювання [рисунок сформовано автором та оброблено за допомогою ШІ-сервісу *Gemini*]

Імена дослідників, які зробили значний внесок у теорію та практику design, test and fault simulation: Y. Zorian, R. Ubar, Vazgen Melikyan, Samvel Shoukourian, J. Bergeron, Z. Navabi, A. Jerraya, D. Armstrong, J. Abraham, H Fujiwara, T. Nishida, X. Wang, Kang Li, Hans-Joachim Wunderlich, Hussam Amrouch, Patrick Groeneveld, Adit D. Singh, Giorgio Di Natale, Paolo Prinetto, H. Fatih Ugurdag, Fadi Maamari, P. Mueller, A. Ivanov, A., Irith Pomeranz, Sudhakar M. Reddy, J. Hayes. Aleksandr Palagin, Volodymyr Opanasenko, Oleksandr Drozd, Artur Jutman, Maksim Jenihhin, N. Takahashi, Hana Kubatova, E.J. Marinissen, V.D. Agrawal, J.P. Roth, A. Matrosova, P. Parkhomenko, A. Birger, Roy Kaushik.

Дисертація присвячена вирішенню науково-практичної задачі щодо розробки економічно ефективного векторно-логічного моделювання в пам'яті для симуляції несправностей як адрес цифрових схем, заснованих на транзакціях читання-запису без інструкцій процесора. Запропоновано експоненціально надлишкову векторно-логічну модель компонентів цифрових схем, які значно знижують обчислювальну складність алгоритмів для good-value та fault simulation, використовуючи адреси бітів логічного вектора. Векторно-логічну матрицю несправностей введено як основний механізм для моделювання несправностей та правильної поведінки, підтримуючи фіксований обсяг пам'яті для кожного тестового вхідного набору. Представлено векторно-логічні алгоритми, здатні ефективно моделювати good-value та несправності на матриці моделювання, використовуючи бітові адреси логічних та дедуктивних векторів для симуляції вхідних наборів та несправностей. Описано прості векторно-логічні алгоритми для виведення дедуктивного вектора та генерації таблиці істинності виявлених комбінацій несправностей на вхідному наборі елемента, визначеному його логічним вектором. Запропоновано алгоритми моделювання good-value та несправностей як адрес з майже однаковою обчислювальною складністю через надлишковість у векторно-логічних структурах даних та в матрицях дедуктивних векторів для елементів схеми.

Розроблено хмарний сервіс MOSI-HDL (MOdeling for SIMulation), перевагою або новизною якого є синхронізація процесів моделювання у всіх

масштабованих вікнах, при цьому дані представляють важливі компоненти та процедури для конструктивного розуміння результатів. Відповідні рядки таблиць моделювання good-value та несправностей глобально синхронізовані на моніторі з матрицею моделювання та good-value та несправними станами всіх ліній схеми. Рядки матриці моделювання локально синхронізовані з рядками коду Verilog, а виділений логічний елемент схеми – з таблицею істинності всіх виявлених комбінацій несправностей для вхідного набору. Кодування на C++ та верифікація векторного логічного моделювання для механізмів моделювання виконуються з використанням репрезентативного набору схемних тестів з бібліотеки ISCAS. Векторно-логічне моделювання для симуляції є прикладом нового масового обчислення в пам'яті, пропонуючи економічно ефективні рішення з точки зору енергії та часу для проєктування, верифікації, тестування та діагностики цифрових продуктів. Результати цього дослідження можуть бути корисними для вчених, студентів та фахівців, що працюють у галузі проєктування та тестування обчислень в пам'яті на основі транзакцій читання-запису.

Об'єкт дослідження – in-memory комп'ютинг та векторно-логічні моделі й механізми для розв'язання задач моделювання та симуляції цифрових схем.

Предмет дослідження – in-memory моделювання несправностей, як адрес, для верифікації цифрової логіки та схем.

Математичний апарат: дискретна математика, теорія проєктування, теорія автоматів, технічна діагностика, мова C++.

Мета дослідження – зниження часових та енергетичних витрат fault/good-value as address simulation цифрових схем за рахунок надлишковості векторно-логічних моделей їх елементів.

Задачі дослідження:

1. Аналіз стану проблеми та актуальних технологій прогресивного корпусування інтегральних мікросхем, моделювання несправностей цифрових систем та їх компонентів.

2. Вдосконалення розумних структур даних векторної логіки, що зменшують обчислювальну складність алгоритмів fault/good-value as address simulation за рахунок експоненціальної надмірності моделей елементів логічних схем.

3. Розробка архітектури для моделювання схеми та рендерингу процесів синхронізації векторно-логічного modeling for fault/good-value as address simulation, що містить побудову моделі об'єкта у пам'яті комп'ютера та дозволяє візуально супроводжувати процес modeling for simulation цифрової схеми.

4. Розробка векторно-логічних механізмів modeling for fault/good-value as address simulation з лінійною алгоритмічною складністю.

5. Програмна реалізація розумних структур даних, алгоритмів та механізмів fault/good-value as address simulation, синхронний рендеринг всіх компонентів процесу симуляції на вікно монітору з метою візуального супроводжування процесу modeling for simulation цифрової схеми, їх верифікація на бібліотеці стендових схем ISCAS та імплементація до хмарного сервісу MOSI-HDL.

Наукова новизна результатів досліджень:

1. Удосконалено розумні структури даних, які орієнтовані на безпроцесорний векторно-логічний modeling for fault/good-value as address simulation за допомогою read-write транзакцій, що дозволило зменшити обчислювальну складність алгоритмів симуляції до лінійної.

2. Вперше запропоновано векторно-логічну модель елементів цифрової схеми, що дозволяє здійснити good-value simulation вхідних тестових наборів та fault as address simulation як адрес бітів логічного та дедуктивного вектора на основі read-write транзакцій та знизити обчислювальну складність алгоритмів тестування логічних функціональностей до лінійної.

3. Вперше запропоновано архітектуру для моделювання схеми та рендерингу процесів синхронізації векторно-логічного modeling for fault/good-value as address simulation, що містить побудову моделі об'єкта у пам'яті комп'ютера та дозволяє проєктувальнику візуально супроводжувати процес

modeling for simulation цифрової схеми (вручну будувати тести для покриття несправностей логічних схем).

4. Вперше запропоновано механізми векторно-логічного modeling for fault/good-value as address simulation, що відзначаються лінійною алгоритмічною складністю.

Практична значущість результатів дослідження визначається верифікацією розумних структур даних та алгоритмів моделювання на десятках схем та функціональних елементів з бібліотеки ISCAS. Розроблено хмарний сервіс MOSI-HDL (<http://mosihdl.work>) для економічного вирішення задач modelling for simulation (MOSI) на основі векторно-логічного in-memory комп'ютингу, що орієнтований на використання read-write транзакцій без інструкцій процесора, який містить: 1) здійснення good-value simulation вхідних тестових наборів та симуляцію несправностей як адрес бітів логічного та дедуктивного вектора на основі read-write транзакцій; 2) механізми good-value simulation of test set as address цифрової схеми без процесора, механізми fault as address simulation цифрової схеми без участі процесора; 3) механізм fault as address simulation вхідного набору, використовуючи квадратичну матрицю симуляції; 4) механізми рендерингу результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора.

Інтегральною перевагою або новизною даних сервісів є синхронізація процесів моделювання по всіх вікнах, що представляють всі суттєві процедури для розуміння отриманого результату.

Запропоновані моделі, методи та механізми апробовані на міжнародних конференціях з напрямку EDA. Моделі, методи та алгоритми імплементовано у хмарний сервіс MOSI-HDL, реалізований мовою C++. Для верифікації механізмів моделювання оброблені схеми ISCAS85. Програма MOSI HDL містить близько 13 тисяч рядків коду на C++ версії C++17, у тому числі невелика частина – модуль web, написаний на JavaScript + HTML. Сам модуль моделювання (model) становить понад 2 тисячі рядків коду, найбільший обсяг займає графічний модуль (view) – понад 8 тисяч рядків. Підтримуються

Windows та Ubuntu. Розробку представлено на двох виставках Міжнародного молодіжного форуму у 2025 та 2026 роках, де вона стала переможцем. Обмеження MOSI-HDL обумовлені тим, що він є діючим прототипом та постійно розвивається та доповнюється новими функціоналами. Наприклад, поточна версія підтримує лише комбінаційні схеми без ієрахії, а також інтеграцію з відомими зовнішніми сервісами автоматичного генератору тестів. Планується також створити інструментарій для генерації тестів, що покривають усі несправності логічних схем будь-якої конфігурації.

Структура новизни дисертації представлена на рис. 0.3: формула, розумна структура даних, векторно-логічні моделі, механізми, архітектура та завдання з верифікації цифрових проєктів.

Формула	Використання векторної логіки при modeling експоненційно надмірних структур даних елементів логічної схеми для fault/good-value as address simulation та синхронної візуалізації всіх компонентів процесу modeling for simulation на вікна монітора.			
Розумні структури даних [2-5]	Логічні вектори та матриці завдання комбінацій несправностей всіх логічних функцій від n-змінних [4].	Побудова тестової карти функціональності або структури, представленої логічним вектором [5]	Квадратична матриця симуляції [3] – для реалізації механізму fault as address simulation вхідного набору.	Матриця перекодування для побудови дедуктивних векторів [2] .
Векторно-логічні моделі та механізми [2,3,7,9].	Векторно–логічна модель елементів цифрової схеми для good–value simulation на основі read–write транзакцій [3].	Механізм good-value simulation цифрової схеми для синтезу логічного вектора [2].	Механізми fault as address simulation цифрової схеми без участі процесора [3].	Механізми рендерингу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора. [3,7,8,9].
Задачі верифікації цифрових проєктів [2-5, 7-9]	Верифікація розумних структур даних та алгоритмів моделювання на десятках схем та функціональних елементів з бібліотеки ISCAS [8]	Опис архітектури та програмна реалізація механізмів моделювання та рендерингу [9].	Розробка та апробація сервісів MOSI-HDL [3, 7-9].	Інтеграція зовнішніх бібліотек, синхронізація механізмів [9].

Рисунок 0.3 – Карта новизни дисертаційного дослідження

Обґрунтованість наукових положень. Отримані в процесі виконання досліджень наукові висновки і практичні результати з моделювання несправностей як адрес та рендерінгу схем є достовірними, що підтверджується достатньою кількістю проведених експериментів, точністю розрахунків, апробацією результатів на міжнародних науково-практичних конференціях, впровадженням результатів в освітній процес.

Впровадження результатів дисертації. Результати дисертації у складі векторно-логічних моделей та механізмів впроваджені у навчальний процес Харківського національного університету радіоелектроніки (акт про впровадження від 15.06.2026).

Особистий внесок здобувача. Всі наукові і практичні результати отримані автором особисто. У роботах, опублікованих зі співавторами, здобувачеві належать (Додатки А, Б):

[1] – аналітичний огляд технологій прогресивного корпусування та технік моделювання цифрових систем на основі огляду літературних джерел, висновки;

[2] – механізм good-value simulation цифрової схеми для синтезу логічного вектора з подальшим його використанням для побудови карти тестування несправностей на повному тесті за три матричні операції, що використовує read-write транзакції над бітами логічних векторів і не потребує інструкцій процесора; вдосконалені структури даних, механізми моделювання карти тестування за логічним вектором схеми;

[3] – архітектура для моделювання та рендерінгу схеми; векторно-логічна модель елементів цифрової схеми, що дозволяє здійснювати good-value simulation вхідних тестових наборів та симуляцію несправностей як адрес бітів логічного та дедуктивного вектора на основі read-write транзакцій; механізми векторно-логічного modeling for fault/good-value as address simulation, механізми рендерінгу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора;

[4] – розумні структури даних у вигляді логічних векторів та матриць завдання комбінацій несправностей всіх логічних функцій від n -змінних;

[5] – побудова тестової карти функціональності або структури, представленої логічним вектором;

[6] – аналіз стану проблеми на основі огляду літературних джерел, висновки;

[7] – опис архітектури та функціоналу програмного застосунку. верифікація механізмів рендерингу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора;

[8] – опис структури програмного застосунку, верифікація розумних структур даних та алгоритмів моделювання на схемах та функціональних елементах з бібліотеки ISCAS;

[9] – опис архітектури та програмна реалізація механізмів моделювання та рендерингу, інтеграція зовнішніх бібліотек, синхронізація механізмів.

Внесок співавторів робіт:

[1] – постановка завдання, керівництво – Чумаченко С.В.;

[2] – концептуалізація дослідження, формулювання мети та постановка задач, методологія, валідація, формальний аналіз, дослідження, керівництво, аналіз результатів – Хаханов В.І.; вдосконалені механізми моделювання карти тестування за логічним вектором схеми – Демченко О.І., Обрізан В.І.; валідація результатів, підготовка та редагування рукопису – Рахліс Д.Ю.; дослідження, порівняння з аналогами – Хаханова Г.В.; процедури (механізми) моделювання логічного вектора схеми методом декартової суперпозиції векторів, програмна реалізація – Хаханов І.В.; дослідження, написання огляду та висновків, програмна реалізація, механізм good-value simulation цифрової схеми для синтезу логічного вектора з подальшим його використанням для побудови карти тестування несправностей на повному тесті за три матричні операції, що використовує read-write транзакції над бітами логічних векторів і не потребує інструкцій процесора; вдосконалені структури даних – Демченко О.І.;

дослідження, написання огляду та висновків, програмна реалізація – Воронов А.О.;

[3] – концептуалізація дослідження, формулювання мети та постановка задач, методологія, валідація, формальний аналіз, дослідження, керівництво, аналіз результатів – Хаханов В.І.; формальний аналіз, дослідження, написання драфту, адміністрування проєкту – Чумаченко С.В.; дослідження, переклад англійською мовою, підготовка та редагування рукопису – Литвинова Є.І.; – програмна реалізація, дослідження, написання огляду та висновків – Воронов А.О.; дослідження, написання огляду та висновків, узагальнення архітектури для моделювання та рендерингу схеми; практична апробація моделей й механізмів fault/good-value as address simulation для програмної реалізації й візуалізації, механізми рендерингу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора – Демченко О.І.; дослідження, валідація, написання огляду та висновків – Максимова Н.Г.;

[4] – концептуалізація дослідження, формулювання мети та постановка задач, методологія, валідація, формальний аналіз, дослідження, керівництво, аналіз результатів – Хаханов В.І.; формальний аналіз, дослідження, написання драфту, адміністрування проєкту – Чумаченко С.В.; дослідження, переклад англійською мовою, підготовка та редагування рукопису – Литвинова Є.І.; програмна реалізація, дослідження, написання огляду та висновків, практична апробація механізмів візуалізації графових структур – Воронов А.О.; дослідження, написання огляду та висновків, узагальнення розумних структур даних, опис алгоритмів та їх програмна реалізація – Демченко О.І.; дослідження, валідація, написання огляду та висновків – Максимова Н.Г.

Апробація результатів дисертації. Результати роботи були представлені та обговорені на наступних конференціях: IEEE East-West Design and Test Symposium, 2026; 15 Міжнародна науково-технічна конференція «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління (Current directions of development of information and communication technologies and control tools)», Баку – Харків – Жиліна, 2025; XXX

Міжнародний молодіжний форум «Радіoeлектроніка та молодь у ХХІ столітті», Харків, 2026; а також на двох виставках технічної творчості молоді в рамках Міжнародного молодіжного форуму «Радіoeлектроніка та молодь у ХХІ столітті» з отриманням призових місць – у 2025 (нагорода – Гран-прі у номінації «Розробка програмного забезпечення для апаратної платформи») та 2026 (нагорода – I місце у номінації «Розробка програмного забезпечення для апаратної платформи») роках.

Публікації. Результати дисертаційної роботи відображені у 9 друкованих працях, серед яких 3 статті, а саме: 1 стаття – в міжнародному науковому журналі за кордоном, що індексується Scopus, 2 статті – у наукових журналах, включених до «Переліку наукових фахових видань України», з них 1 – у журналі категорії А, що входить до наукометричної бази Web of Science, та 1 – категорії Б, а також 3 тез доповідей у матеріалах міжнародних наукових конференцій, з них 1 входить до науково-метричної бази Scopus. Результати роботи також апробовано на виставках технічної творчості молоді при Міжнародному молодіжному форумі «Радіoeлектроніка та молодь у ХХІ столітті» та висвітлено у каталозі виставок.

Структура дисертації: 177 сторінок (з них 131 с. представляють основний текст) і містить: 4 розділи, 71 рисуноків, список джерел з 151 назви на 24 с., 3 додатки на 5 с., анотації на 13 с.

1 АНАЛІЗ СТАНУ ПРОБЛЕМИ ТА АКТУАЛЬНИХ ТЕХНОЛОГІЙ МОДЕЛЮВАННЯ НЕСПРАВНОСЕЙ

1.1 Передмова

Розвиток мікроелектроніки та постійне зростання потреби у високопродуктивних, компактних і енергоефективних інтегральних мікросхем (ІМС) зумовлює необхідність удосконалення відповідних технологій. Одним із ключових напрямків розвитку напівпровідникової індустрії в останні роки є прогресивне корпусування та пакування (Advanced packaging), оскільки це дозволяє подолати обмеження традиційних підходів. Типові застосування рішень на основі прогресивних компонок включають: ІМС для високопродуктивних обчислень (High Performance Computing), ІМС для штучного інтелекту та машинного навчання, високопродуктивні графічні процесори, мережеві ASIC та напівпровідникові компоненти для мобільних пристроїв (смартфони 5G, ноутбуки, смарт-годинники). Сучасні технології, зокрема 2,5 та 3-вимірні інтеграції, забезпечують підвищення характеристик у багатьох аспектах, де традиційні технології мають обмеження. Використання чиплетів дозволяє збільшити рівень інтеграції, тоді як у випадку монокристалівних рішень це стає дедалі складнішим як у технічному, так і в економічному плані. Поєднання пам'яті та процесора в рамках одного корпусу дозволяє суттєво підвищити продуктивність обчислень. У випадку компонок гетерогенних кристалів технологія прогресивного корпусування є практично єдиним ефективним рішенням, оскільки враховує електричні, механічні та технологічні особливості кожного кристала. Однак впровадження цих технологій супроводжується значними викликами, серед яких надвисока щільність між'єднань між кристалами (сотні тисяч і більше), вразливість до термічного та механічного стресу, гірший тепловий режим роботи ІМС. Це робить задачу моделювання, верифікації та тестування між'єднань чиплетів

критично важливою. Вибір оптимальної моделі тестування між'єднань є ключовим для подальшого розвитку гетерогенних мікросистем. Складність і вартість верифікації кристалів зростає через мікрометрові розміри контактних майданчиків, неможливість впроваджувати надмірну кількість контрольних точок (оскільки додатковий метал додає ємнісне навантаження і погіршує цілісність сигналів), а також прихованість значної частини розводки в пакеті. У цьому контексті розроблюваний стандарт IEEE P3405 відіграє важливу роль, оскільки регламентує методології оцінки між'єднань і сприяє підвищенню їхньої надійності. З огляду на глобальне зростання попиту на передові напівпровідникові рішення, вдосконалення технологій Advanced packaging є стратегічно важливим для електронної промисловості. Високий рівень інвестицій провідних компаній свідчить про серйозність намірів. Так, TSMC у 2025 році планує інвестувати 100 мільярдів доларів США у розширення виробництва на основі Advanced Packaging, що доповнює вже заплановані вкладення у розмірі 65 мільярдів доларів. Intel також має намір інвестувати 100 мільярдів доларів США, а Samsung планує вкласти понад 40 мільярдів доларів США.

Метою розділу є аналіз сучасних тенденцій і основних викликів, пов'язаних із прогресивними технологіями корпусування ІМС, моделювання несправностей для подальшого вибору оптимальної моделі тестування між'єднань чиплетів. Об'єктом дослідження виступають технології компонування та корпусування ІМС, моделювання несправностей. Предметом дослідження є технологічні рішення та конструктивні особливості прогресивного корпусування й компонування ІМС, а також моделювання несправностей при їх проектуванні. До них належать 2.5-вимірна інтеграція, що забезпечує надвисоку щільність між'єднань, такі як інтеграція на рівні кремнієвого інтерпозера (CoWoS) і кремнієвого міжкристального містка (EMIB). Розглядаються також матеріали та пов'язані технологічні процеси, які сприяють створенню ефективних між'єднань.

1.2 Технології прогресивного корпусування інтегральних мікросхем. Стан питання

Основними чинниками розвитку прогресивного корпусування (пакування – Advanced Packaging) є необхідність вирішення низки викликів, що стоять перед виробниками спеціалізованих ІМС (ASIC). Серед них: продуктивність, енергоефективність, масо-габаритні характеристики, тривалість виробничого циклу та вартість.

Наприклад, динамічна пам'ять, змонтована за традиційною схемою та з'єднана з процесором на друкованій платі, має обмежену пропускну здатність через відносно високий імпеданс таких з'єднань. Цей приклад також демонструє проблему інтеграції гетерогенних кристалів, тобто виконаних за різними технологіями. Пам'ять та процесор можуть суттєво відрізнитись як з точки зору і техпроцесу так і електричного інтерфейсу що додатково ускладнює інтеграцію. Вибір оптимальної моделі тестування міжз'єднань є ключовим для подальшого розвитку гетерогенних мікросистем. У цьому контексті розроблюваний стандарт IEEE P3405 відіграє важливу роль, оскільки регламентує методології оцінки міжз'єднань і сприяє підвищенню їхньої надійності [1, 2].

Типові застосування рішень на основі прогресивних компоновок включають: ІМС для високопродуктивних обчислень (High Performance Computing), ІМС для Artificial Intelligence та Machine Learning, високопродуктивні графічні процесори, мережеві ASIC та напівпровідникові компоненти для мобільних пристроїв [3]. Вибір оптимальної моделі тестування міжз'єднань є суттєвою складовою для розвитку гетерогенних мікросистем [4-6]. Складність і вартість верифікації кристалів зростає через мікрометрові розміри, надмірну кількість контрольних точок [7], а також прихованість значної частини розводки в пакеті. У цьому контексті важливу роль відіграє стандарт IEEE P3405, оскільки регламентує методології оцінки міжз'єднань і сприяє підвищенню їхньої надійності.

Крім того, класичні технології дротового монтажу (Wirebonding) або монтажу методом перевернутого кристалу (Flip Chip) на стандартних кульках припою (С4-бампи) забезпечують відносно невелику кількість з'єднань: сотні контактів при розміщенні по периметру кристала і тисячі при розміщенні по всій його площі [8]. Однак із зростанням рівня інтеграції ІМС цього стає недостатньо.

Вагомим викликом є виробництво великих монолітних кристалів. Більшість літографічних процесів обмежують зону експонування кристала одним reticle (26×33 мм), а в деяких випадках – навіть до $26 \times 16,5$ мм [9]. Навіть якщо вдається досягти більшої зони експонування, зворотно-пропорційна залежність між площею кристала та виходом справних чипів різко підвищує собівартість готового виробу. Це додатково посилюється значним зростанням вартості кремнієвих пластин для техпроцесів 7 нм і менше. Наприклад, ціна 300-мм кремнієвої пластини для 2 нм техпроцесу TSMC може сягати 30 тисяч доларів США. Щоб задовольнити ці вимоги, було розроблено цілу низку двовимірних (2D), дво- з половиною-вимірних (2,5D) і навіть тривимірних (3D) інтеграцій.

Інтеграція ІМС з внутрішнім розведенням виводів Fan-In Wafer-Level Package (WLP) – це двовимірна технологія пакетування ІМС, при якій розведення виводів здійснюється ще на етапі цілої пластини, на відміну від традиційного процесу, коли збірка окремих кристалів у пакети відбувається після розрізання пластини. Технологія забезпечує корпус із габаритами, наближеними до розміру кристала, з перерозподільним шаром (RDL) у межах його периметра, однак має обмеження щодо кількості виводів.

Коли існує необхідність отримати більшу кількість виводів то розповсюджують зону контактів за периметр кристалу. Таку технологію відносять до інтеграції з зовнішнім розведенням виводів (Fan-Out-WLP). Існує реалізація двома способами Chip First та Chip Last. У Chip First кристал спочатку встановлюється та обволочується компаундом, після чого прикріплюється до

підложки і наостанок формується RDL. У Chip Last пакет спочатку створюється починаючи з RDL на підложці, а потім додається кристал і компаунд.

Система на кристалі (SoC) інтегрує ІМС з різними функціями, такими як центральний процесор, графічний процесор, пам'ять тощо, в один кристал для системи або підсистеми. Коли одного кристала недостатньо то можлива інтеграція Package-on-Package (PoP) цей метод компонування дозволяє вертикально поєднувати пакети кристалів в одному корпусі. Два або більше (через проблеми тепловідводу цього уникають [10]) пакетів встановлюються один на одному зі інтерфейсом для маршрутизації сигналів між ними. PoP забезпечує вищу щільність компонентів у пристроях, таких як мобільні телефони, цифрові камери, хоча і при цьому збільшується загальна висота компонента.

Особливу увагу заслуговують 2,5-вимірні інтеграції ІМС, до яких відносять різні технології на основі інтерпозера або кремнієвого моста. Інтерпозер – це тонка пластина (~100–200 мкм), виготовлена переважно з кремнію, хоча можливе використання й інших матеріалів, таких як скло або органічні сполуки. У цій пластині створені так звані міжперехідні з'єднання (Through Silicon Via – TSV для кремнію або Through Glass Via – TGV у випадку скла) [11]. Типовий діаметр таких з'єднань знаходиться в діапазоні 10–25 мкм. Після травлення або лазерної обробки отвори заповнюються міддю шляхом металізації. Інтерпозери підтримують високу щільність контактів – до 10^6 см^{-2} , що суттєво перевищує можливості керамічних або органічних пакетів, які зазвичай мають щільність близько 10^3 см^{-2} . Крім того, крок з'єднань (pitch) виводів на інтерпозерах може становити всього 5 мкм, тоді як у керамічних пакетах він на порядок більший. За функціональністю інтерпозери поділяються на: пасивні, які лише забезпечують електричні міжз'єднання та активні, що містять додаткові функції, наприклад, ретрансляцію сигналів.

Прикладом активних інтерпозерів є широкий клас ІМС пам'яті та логіки. Інтеграція логіки безпосередньо в інтерпозер покращує його функціональність, але водночас збільшує вартість та ускладнює проектування систем на його

основі. Натомість пасивні інтерпозери зустрічаються частіше і зазвичай містять кілька шарів металізації для передачі сигналів і живлення, деякі мають вбудовані пасивні компоненти, такі як конденсатори.

Технологія інтерпозерів була розроблена як проміжний етап до 3D-інтегрованих систем, оскільки широке впровадження останніх потребує подолання численних технічних і логістичних викликів. Зокрема, відсутність єдиної моделі та налагодженого ланцюга постачання між різними учасниками виробничого процесу ускладнює їх впровадження. Через це інтерпозери, які еволюціонували з технологій багатокристалівних модулів, мають менше виробничих обмежень і нижчу собівартість порівняно з 3D-інтеграцією. Це робить їх життєздатним рішенням для забезпечення все більш складної комунікації між кристалами. Завдяки своїм перевагам технологія інтерпозерів швидко розвивається, а вдосконалення процесів виробництва та матеріалів сприяє її перетворенню на окрему перспективну технологію, а не просто проміжне рішення на шляху до 3D-систем. Серед недоліків які виникають при експлуатації інтерпозерів з металізованими міжперехідними з'єднання є ризик деформації структури пакету внаслідок суттєвої різниці між коефіцієнт теплового розширення міді та кремнію ($16,5 \times 10^{-6} / ^\circ\text{C}$ проти $2,6 \times 10^{-6} / ^\circ\text{C}$).

Альтернативою до TSV є використання вертикального з'єднання кристалів на двосторонньому інтерпозері за допомогою отворів через кремній (Through Silicon Hole – TSH) на підложці та зростанні мідних штифтів (10-20 мкм в діаметрі) на кристалах. Іншими словами, металізація отворів в підложці не виконується, натомість через ці отвори відбувається з'єднання мідних штифтів.

Однією з найвідоміших реалізацій технології інтерпозера є інтеграція CoWoS (Chip-on-Wafer-on-Substrate) від компанії TSMC. Першим виробом на платформі CoWoS був продукт для Xilinx в 2011 році з розміром інтерпозера відповідаючому 1×reticle (26x33 мм). Технологія існує в варіантах CoWoS-S, CoWoS-R та CoWoS-L [12].

CoWoS-S платформа інтеграції з кремнієвим інтерпозером і високощільними міжз'єднаннями, на великій площі (до $3,3 \times \text{reticle} \sim 2700 \text{ мм}^2$) інтерпозера для розміщення різних функціональних верхніх кристалів, включно з логічними чиплетами та високошвидкісною пам'яттю (HBM).

CoWoS-R платформа інтеграції з органічним інтерпозером та RDL зовнішнім розведенням виводів) – використовує інтерпозер з RDL для з'єднання між системою на кристалі (SoC) та високошвидкісною пам'яттю (HBM), забезпечуючи гетерогенну інтеграцію. Інтерпозер RDL складається з полімерного матеріалу та мідних дорожок.

CoWoS-L – це один із варіантів технології Chip Last на платформі CoWoS. Використовується інтерпозер із Local Silicon Interconnect (LSI) для міжкристальних з'єднань та шари перерозподілу (RDL) для передачі живлення та сигналів. Основні характеристики CoWoS: кристали LSI забезпечують високу щільність трасування міжкристальних з'єднань через декілька шарів субмікронних мідних провідників. Інтерпозер містить шари RDL із широким кроком для передачі сигналів і живлення. Це знижує втрати високочастотних сигналів під час швидкісної передачі даних. Серед інших недоліків варто відмітити: нижчий вихід при збільшенні розміру інтерпозера, складна верифікація 2,5-вимірних і 3-вимірних структур.

На рис. 1.1 представлений опис пакету інтерпозера з чиплетами на мові Verilog. Характерною властивістю є наявність одного інтерпозера на одразу кілька чиплетів. Порти пакету проходять через інтерпозер починаючи з нижчого рівня (порти з суфіксом bot) і закінчуючи рівнем вищим рівнем (порти з суфіксом top). За потреби деякі з портів розгалужуються на кілька різних кристалів, тоді як деякі біти портів призначені конкретним кристалом. Інакше – можлива підтримка різної топології розведення (1 до 1, 1 до багатьох).

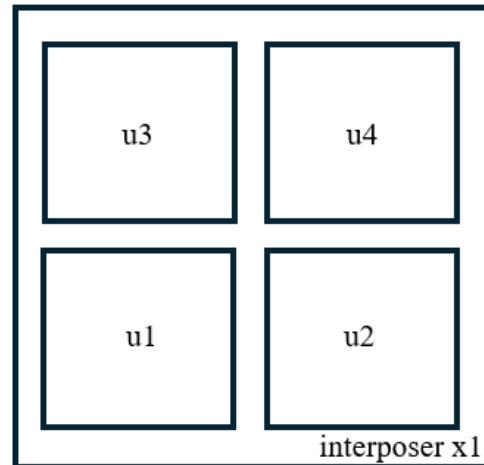
Інший варіант 2,5-вимірної інтеграції є інтеграція на основі невеликого кремнієвого міжкристального містка – EMIB (Embedded Multi-die Interconnect Bridge). Дана технологія пакетування напівпровідників розроблена Intel, де невеликий вбудований кремнієвий міст відповідає за з'єднання кількох

кристалів або чиплетів у межах одного корпусу. На відміну від одного великого кремнієвого інтерпозера, міст ЕМІВ охоплює лише ту область, яка необхідна для з'єднання конкретних кристалів, що робить сам міст більш компактним і економічно вигідним рішенням [13].

```
module interposer(
    output wire    clk_top,
    input wire    clk_bot,
    output wire    rst_n_top,
    input wire    rst_n_bot,
    inout wire [63:0] data_top,
    inout wire [63:0] data_bot);
```

```
    assign clk_top = clk_bot;
    assign rst_n_top = rst_n_bot;
    assign data_top = data_bot;
```

```
endmodule
```



```
module package_interposer(
    input wire clk_pkg,
    input wire rst_n_pkg,
    input wire ctl_pkg,
    inout wire [63:0] data_pkg);
```

```
    wire clk_chp;
    wire rst_n_chp;
    wire [63:0] data_chp;
```

```
    chiplet_die u1( .clk(clk_chp), .rst_n(rst_n_chp), .data(data_chp[15:0]) );
    chiplet_die u2( .clk(clk_chp), .rst_n(rst_n_chp), .data(data_chp[31:16]) );
    chiplet_die u3( .clk(clk_chp), .rst_n(rst_n_chp), .data(data_chp[47:32]) );
    chiplet_die u4( .clk(clk_chp), .rst_n(rst_n_chp), .data(data_chp[63:48]) );
    interposer x1( .clk_bot(clk_pkg), .clk_top(clk_chp),
                  .rst_n_bot(rst_n_pkg), .rst_n_top(rst_n_chp),
                  .data_bot(data_pkg), .data_top(data_chp) );
```

```
endmodule
```

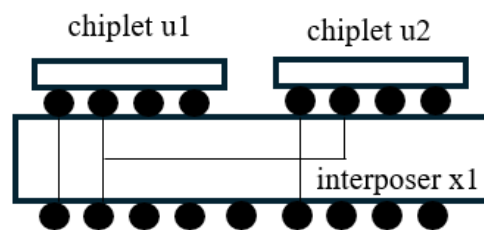


Рисунок 1.1 – Опис CoWoS пакету мовою Verilog

ЕМІВ дозволяє інтегрувати кілька кристалів у високо інтегровану систему SiP з можливістю використання кількох мостів ЕМІВ у межах одного корпусу. Цей підхід вимагає тісної спільної розробки кристалів, ЕМІВ і загальної архітектури корпусу для оптимізації продуктивності та вартості, на відміну від більш орієнтованого на інтерфейси кремнієвого інтерпозера. Ключова перевага технології полягає в можливості інтегрувати велику кількість кристалів в одному корпусі використовуючи при цьому невеликі кремнієві містки (1-5 мм ширини, до 10 мм довжини і 100 мкм товщини). Типовий розмір мікробампів складає 55 мкм, при цьому розмір бампів на містку може відрізнатись для інтеграції кристалів з іншим кроком з'єднань. Це може бути перевагою в тому сенсі, що відбувається локалізація мікробампів (використовується там де потрібно, а не по всій площі у випадку кремнієвого інтерпозера). До недоліків слід віднести меншу щільність з'єднань порівнянно до кремнієвого інтерпозера, меншу швидкодію сигналів і гірший тепловий режим через використання полімерних матеріалів які оточують кремнієвий міст.

На рис. 1.2 представлений можливий опис ЕМІВ пакет з чиплетами мовою Verilog. Характерною властивістю є наявність кількох мостів, кожен з яких поєднує пару чиплетів. Чиплети також можуть мати з'єднання поза мостом, наприклад зазвичай живлення реалізовується таким чином.

Висновки. Через об'єктивні виклики як на фізичному, так і на економічному рівні, традиційні підходи до корпусування та пакетування ІМС поступово замінюються технологіями Advanced Packaging. У зв'язку з цим все більше рішень використовуватимуть 2,5D/3D інтеграцію на основі з'єднувального кристала (міжкристального містка чи інтерпозера) з надвисокою щільністю міжз'єднань (сотні тисяч – мільйони з'єднань між кристалами). Як зазначено в дорожній карті TSMC, компанія активно працює над збільшенням площі інтерпозера (переробляється літографічний процес, змінюються фотошаблони тощо), що дозволить збільшити кількість міжз'єднань у майбутньому. З освоєнням 2,5D інтеграцій, очікується активний рух у бік повноцінних 3D інтеграцій на основі активних інтерпозерів та споріднених

технологій, які включають поєднання активних елементів і міжкристальних з'єднань. При цьому існуючі труднощі у досягненні надійних міжз'єднань не тільки не зникнуть, а й будуть ще більш множитись і загострюватись.

```

module emib(
  inout wire      clk_left,
  inout wire      clk_right,
  inout wire      rst_n_left,
  inout wire      rst_n_right,
  inout wire [31:0] data_left,
  inout wire [31:0] data_right);

```

```

assign clk_right = clk_left;
assign rst_n_right = rst_n_left;
assign data_right = data_left;

```

```

endmodule

```

```

module package_emib(
  input wire clk_pkg,
  input wire rst_n_pkg,
  input wire ctl_pkg,
  inout wire [63:0] data_pkg);

```

```

wire clk_chp_12, clk_chp_34;
wire rst_n_chp_12, rst_n_chp_34;
wire [31:0] data_chp_12, data_chp_34;

```

```

chiplet_die u1( .clk(clk_pkg), .rst_n(rst_n_pkg), .data(data_chp_12[15:0]) );
emib      x12( .clk_left(clk_pkg), .clk_right(clk_chp_12),
               .rst_n_left(rst_n_pkg), .rst_n_right(rst_n_chp_12),
               .data_left(data_pkg[31:0]), .data_right(data_chp_12) );
chiplet_die u2( .clk(clk_chp_12), .rst_n(rst_n_chp_12), .data(data_chp_12[31:16]) );

```

```

chiplet_die u3( .clk(clk_pkg), .rst_n(rst_n_pkg), .data(data_chp_34[15:0]) );
emib      x34( .clk_left(clk_pkg), .clk_right(clk_chp_34),
               .rst_n_left(rst_n_pkg), .rst_n_right(rst_n_chp_34),
               .data_left(data_pkg[63:32]), .data_right(data_chp_34) );
chiplet_die u4( .clk(clk_chp_34), .rst_n(rst_n_chp_34), .data(data_chp_34[31:16]) );

```

```

endmodule

```

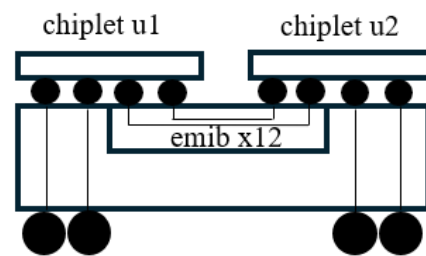
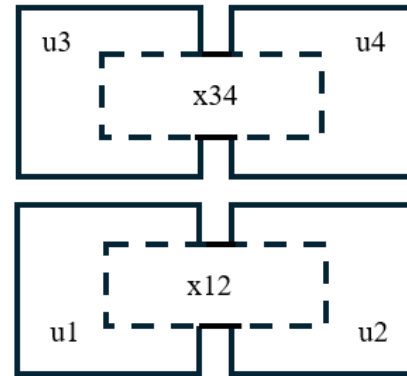


Рисунок 1.2 – Опис EMIB пакету мовою Verilog

Шляхами подолання цих викликів є побудова оптимальної моделі тестування багатокристальних ІМС на різних етапах їх виробничого циклу – від проєктування до тестування на етапах виробництва (Pre-Bond, Mid-Bond, Post-Bond).

Перспективним є подальший розвиток описів багатокристальних структур за допомогою Verilog/SystemVerilog. Мови опису апаратури мають розвинений функціонал, здатний точно описати структурну компоновку таких виробів і провести симуляцію в рамках функціонального тестування. Більш того, це дозволить здійснювати симуляції з урахуванням параметрів затримок сигналів (STA – статичний часовий аналіз) на різних елементах міжз'єднань.

Така модель повинна враховувати існуючі технологічні обмеження на розмір контактних майданчиків тестових точок (у випадку мікробампів – це десятки нанометрів), їх обмежену кількість (збільшення розміру тестових майданчиків погіршує електричні характеристики високочастотних сигналів і не несе функціональної користі після верифікації кристала), їх недоступність для безпосередніх операцій після певних етапів пакування, а також ризик фізичного пошкодження мікробампів через вплив зондувального обладнання.

Основна проблема: яку модель слід заадресувати – це можливість швидкої і надійної верифікації багатотисячних міжкристальних з'єднань. Можливим підходом, який варто дослідити, це ранжирування провідників у wire-clusters (IEEE P3405) по певній геометричній характеристиці, щоб відокремити менші по кількості підкластери які мають більший вплив в рамках свого підкластера чим поза ним.

Необхідно також використовувати прості векторно-логічні моделі для тестування чіплет-з'єднань, заздалегідь зробивши сегментацію сукупності з'єднань, оскільки їх у пакеті кристалів може бути біля 100 тис. [6, 14].

1.3 Техніки моделювання несправностей

Моделювання несправностей є важливим для АТРГ, діагностики та класифікації несправностей (рис. 1.3). Метриками моделювання несправностей є швидкість, пам'ять, моделювання функціональних блоків та здатність до затримки, послідовні схеми та багатозначне моделювання несправностей для обробки невідомого X та високоімпедансного Z . Симулятор несправностей зазвичай, окрім моделі схеми, потребує стимулів та очікуваних реакцій (які необхідні для моделювання істинних значень): модель несправності, список несправностей. Одночасне моделювання несправностей – це моделювання, кероване подіями, яке включає разом хороші/погані події. Моделювання частин несправних схем, які відрізняються від відповідних схем. Складність управління пам'яттю. Практично всі промислові системи моделювання несправностей мають списки несправностей непередбачуваного розміру та структури даних непередбачуваного розміру. Усі шість базових механізмів моделювання використовують процесор з високим рівнем енергоспоживання [15-20].

Fault simulation technique	Complexity	Memory	Data structure	Level	Delay	Speed	Fault model	Multy-valued
Serial fault simulation (1963)	$n \times n^3$	Predictable	Add fault model, fault list	Gate, system	No problem	Slowest	Any	Easy
Parallel fault simulation (1965)	$\frac{1}{w} \times n^3$	Predictable	Register Memory	Gate	Not capable	Middle	Logic	Difficult
Deductive fault simulation (1972)	n^2	Unpredictable	Deductive formulas	Gate	Not capable	Middle	Any	Difficult
Concurrent fault simulation (1974)	$\frac{1}{3} \times n^2$	Unpredictable	Add fault model, fault list	Gate, RTL	Capable	Faster	Logic	Easy
PPSFP – Parallel pattern single fault propagation (1985)	$\frac{1}{w} \times n^3$	Unpredictable	Add fault model, fault list	Gate	Capable	Middle	Logic	Easy
Differential fault simulation (1989)	$\frac{1}{2} \times n^2$	Unpredictable	Add fault model, fault list	Gate, RTL	Not capable	Middle	Any	Difficult
Vector fault simulation (2024)	$\frac{1}{2} \times \frac{1}{3} \times n^2$	Predictable	No, As true-value simulation	Gate, RTL, System	Not capable	Faster	Logic	Capable

Рисунок 1.3 – Систематизація порівняння технік моделювання несправностей

Векторно-логічний комп'ютинг [81] пропонує моделювання несправностей як адрес цифрової схеми за допомогою алгоритму моделювання справної поведінки тестових двійкових наборів як адрес. Моделювання несправностей на вхідному наборі логічної схеми означає обробку матриці

несправностей розміром n^2 , n – кількість вхідних внутрішніх та вихідних ліній. По головній діагоналі матриці апіорі розміщуються одиниці, що ідентифікують несправність ліній схеми. Після обробки матриці ці одиниці перетворюються на виявлені константні несправності, знаки яких завжди інверсні справним значенням ліній схеми. Несправністю є будь-яке логічне значення $\{0,1\}$, що кодується 1, яке змінює вихід елемента на вхідному тестовому наборі. Матриця моделювання несправностей перетворює процес аналізу несправностей алгоритм справного моделювання ліній схеми на дедуктивних векторах. Проблеми моделювання комбінацій несправностей розгалужень, що сходяться, не існує. Оскільки виявлені на лініях несправності не передаються елементами схеми, а позначаються одиницею в координаті стовпця матриці аналізованої лінії. Матриця моделювання також є розумною структурою даних для обробки схем зі зворотними зв'язками.

Новизна полягатиме у використанні розумних структур даних на основі логічного вектора для створення економічних та простих алгоритмів in-memory аналізу великих даних, як адрес таблиці істинності.

Практична значимість полягатиме у програмній реалізації лінійних за обчислювальною складністю механізмів тестування логіки та схем, які одночасно виконують синтез тестів та моделювання несправностей, як адрес.

Ринкова привабливість такого підходу визначається:

1) економікою обробки великих даних як адрес in-memory комп'ютингу [21-28], вільним від процесора, що дозволяє економити електроенергію (36%) і час обробки великих даних (22%);

2) істотним зниженням обчислювальної складності запропонованих алгоритмів логічних схем за рахунок експоненційної надмірності розумних структур даних [29-32];

3) інваріантністю запропонованих векторно-логічних структур даних для моделювання несправностей як адрес вентильного, регістрового та системного рівня подання проектів;

4) передбачуваністю розмірів пам'яті для зберігання розумних структур даних та обчислювальної складності алгоритмів для моделювання несправностей логіки та цифрових схем [33-36];

5) мінімальною кількістю даних про проект для синтезу тестів на основі моделювання логічних несправностей, як адрес [37, 38];

6) використанням двох векторно-матричних паралельних операцій для синтезу карти тестування логічних функціональностей [39-42];

7) впровадженням тестової константи – універсальної матриці логічних несправностей, як відстаней між тестовими наборами, та несправностями, як адресами таблиці істинності, що у кілька разів зменшує складність алгоритмів моделювання несправностей [43-46];

8) векторно-логічна модель структури дозволяє тестувати несправності переходів будь-якого графа розробленими механізмами синтезу карти тестування [47-49]. За допомогою кодованих унітарних даних на універсумі примітивів можна побудувати логічний вектор будь-якої схеми, а потім його верифікувати розробленими механізмами [50].

На ринку EDA існує проблема проектування, а саме: можна спроектувати виріб, але не завжди можна верифікувати за прийнятний час (time-to-market) з необхідною якістю (yield). Усі технології верифікації явно або неявно використовують рівняння конволюції відношень [51]

$$L \oplus T \oplus F = 0 \quad (1.1)$$

між специфікацією L , реалізацією T та помилками F (несправностями). Існують наступні механізми вирішення проблеми верифікації:

1) *debugging* – здійснює порівняння стану змінних після виконання кожного оператора;

2) механізм асерцій або темпоральної логіки здійснює порівняння стану змінних у певних точках програми та часу;

3) стандарти граничного сканування IEEE 11.49, 1500, 1687 – передбачають розбивку проекту на частини (IP-core), що дозволяє спростити задачу тестування виробу в процесі експлуатації, дозволяють зменшити довжину тесту IP-компонентів проекту та покращити його якість за рахунок реєстрів граничного сканування [52, 53].

1.4 Комп'ютинг автоматизованого проектування (Design Automation Computing)

Останні 30 років спостерігається тренд зменшення алгоритмічної складності під час вирішення задач Design Automation рахунок експоненційної складності розумних структур даних. Prompt Computing за цим трендом зводять всі завдання автоматичного проектування до формування запитів до розумної моделі [9, 11, 13-15, 26, 56, 57-62]. При цьому Design Automation комп'ютинг передбачає гармонійне відношення між метою і ресурсами для її досягнення, між Latency і Redundancy алгоритму та моделі (рис. 1.4).

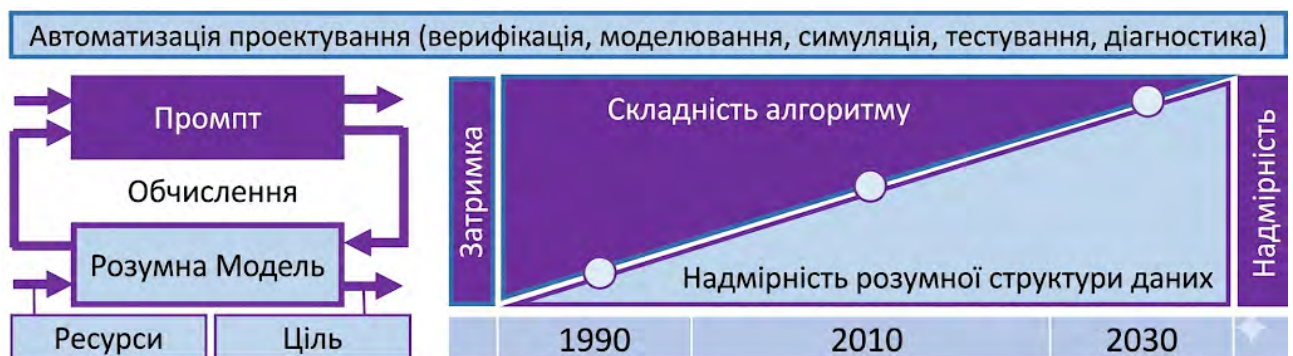


Рисунок 1.4 – Схема конвертації комп'ютингу автоматизованого проектування до промт-комп'ютингу [рисунок складено автором та оброблено за допомогою сервісу Gemini]

Можна припустити, що XXI століття це час енергозберігаючого комп'ютингу, який мінімізує час вирішення практичних завдань за рахунок

надмірності розумних структур даних [63-80]. При цьому енергозбереження буде забезпечене масовою імплементацією in-memory комп'ютингу на основі read-write транзакцій, вільних від команд процесора. Одне з таких рішень – це in memory prompt vector-logic computing для побудови карти тестування за логічним вектором, що пропонується в даному дослідженні. Алгоритмічна складність отримання результату моделювання всіх комбінацій несправностей на вичерпному тесті є матричною константою за рахунок експоненціальної надмірності структур даних. Загальна тенденція вирішення задач у сфері Design Automation полягає у масовому використанні штучного інтелекту на всіх етапах життєвого циклу проєкту [51, 52, 53-59, 79, 80].

1.5 Архітектури та тестування цифрових проєктів

Технічна діагностика як область знань, що охоплює теорію, методи та засоби визначення технічного стану об'єктів, своїми завданнями визначає технічний стан, пошук місця та причини відмови чи несправності (рис. 1.5).

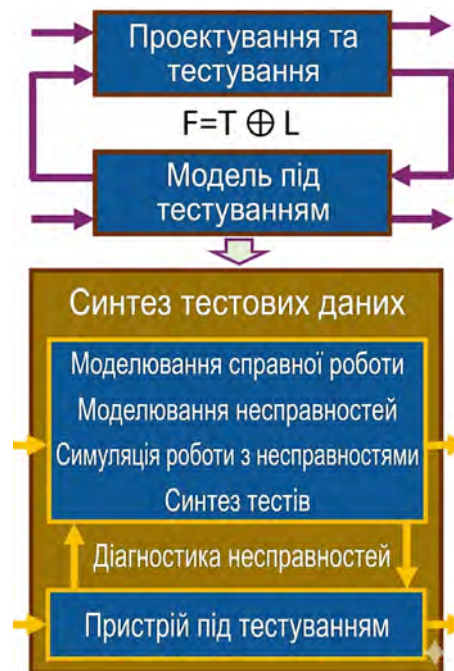


Рисунок 1.5 – Схема задач технічної діагностики [рисунок складено автором та оброблено за допомогою сервісу Gemini]

Використуємо у подальшому наступні поняття та визначення.

Як об'єкт тестування [51] розглядається виріб та його складові, що підлягають перевірці та діагностуванню.

Модель об'єкта тестування – формалізований опис об'єкту, необхідний для розв'язку задач діагностування. Як моделі об'єкта тестування можуть розглядатися диференціальні рівняння, логічні співвідношення, діаграми сигналів, програмні коди, явні структури даних, графові структури.

Модель дефектів – логічні, тимчасові чи фізичні порушення у компонентах цифрового виробу чи моделі, що призводять до його неправильного функціонування.

Достовірність технічного діагностування (yield) – ступінь об'єктивної відповідності результатів діагностування дійсному технічному стану об'єкта.

Тест перевірки – тест перевірки справності або працездатності виробу з точністю до справного стану або підмножини несправних.

Тест діагностування (fault test) – один або кілька тестових впливів та послідовність їх виконання, що забезпечують діагностування чи пошук дефекту.

Моделювання справної поведінки – це процес визначення стану ліній цифрового виробу за відсутності несправностей на тестових наборах або вхідних робочих впливах.

Моделювання несправностей – процес визначення несправностей заданого класу, які перевіряються на тестових наборах чи вхідних робочих впливах.

Повнота покриття тестом (вхідними наборами) – заданого класу несправностей цифрового виробу чи його станів, виражене у відсотках.

Тестування чи верифікація – процес визначення відповідності технічного стану виробу заданої специфікації шляхом порівняння реакції виходів під час подання тестових вхідних наборів.

Синтез тесту – процес визначення мінімальної кількості тестових вхідних наборів, які покривають певний клас несправностей чи справних станів цифрового виробу.

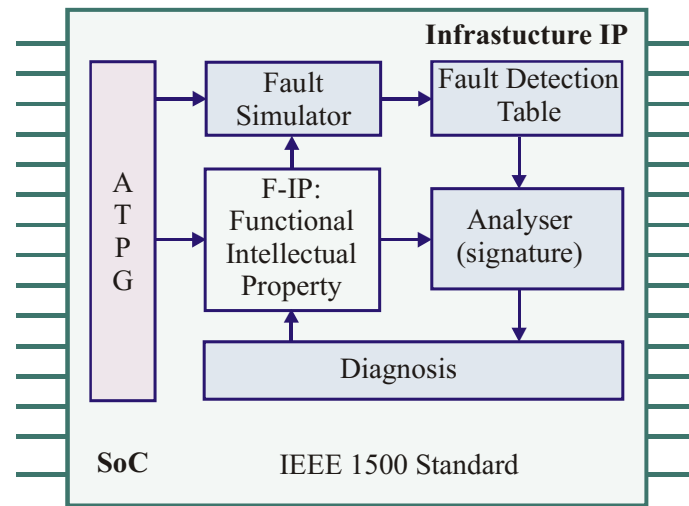
Модель діагностування (testing map) – структура даних, які містять інформацію про справний та всі несправні стани цифрового виробу.

Діагностичний експеримент – процес визначення технічного стану цифрового виробу з точністю до справного стану чи підмножини несправностей.

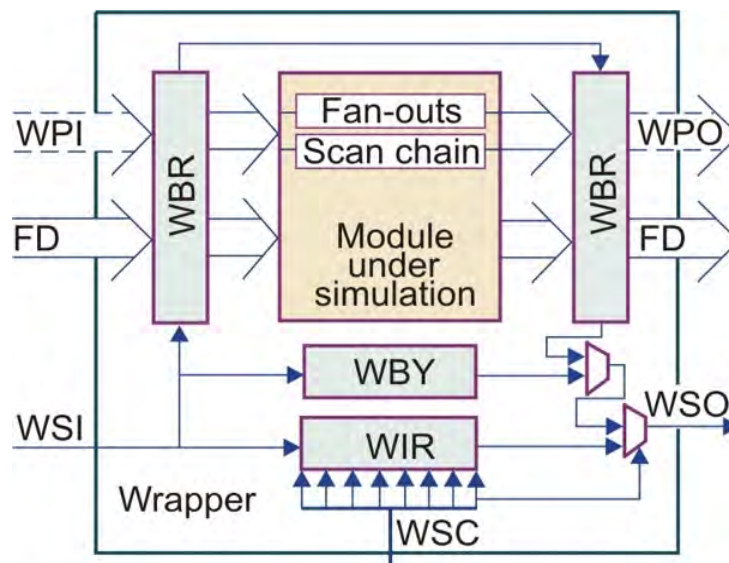
Пошук дефектів – процес визначення несправностей цифрового виробу із заданою точністю чи глибиною діагностування.

Testing map містить структурні відносини між тестами та несправностями, включаючи критичні. Для таких несправностей визначається мінімальний тест їх перевірки, який є testbench для критичних станів функціональності. Далі працює схема верифікації на основі Checker, яка визначає вектор помилок F для коригування поточної реалізації проекту DUT. Процедура верифікації повторюється, поки існують помилки. Testbench для functional coverage набагато економічніше за алгоритмічною складністю справного моделювання, ніж testbench для fault coverage на основі моделювання несправностей. Недолік першого методу – відсутність критеріїв створення мінімального тесту. Testbench – це архітектура тестування пристрою або його частини (SoC) при застосуванні регістру граничного сканування стандарту IEEE 1500 (рис. 1.6) [53]. Перша модель є специфікацією.

Архітектура орієнтована вирішення задач верифікації цифрової логіки з урахуванням синтезу тестів для функціонального покриття чи перевірки комбінацій логічних несправностей. Синтез тесту складає основі використання дедуктивної матриці, отриманої з логічного вектора. Автоматичний синтез тестів на основі побудови карти тестування функціональності визначає практичну мету дослідження.



a



б

Рисунок 1.6 – Схема стандарту для верифікації функціональності цифрового виробу

Дослідження укладається у комбінацію трьох технологій (рис. 1.7): інтегру комп'ютинг обробки великих даних, як адрес, read-write транзакції, вільні від команд процесора, явні розумні структури даних, що мають значну надмірність. Тут також можуть застосовуватися ML-механізми для зв'язування структур даних (Logic Y-vector, Test Truth table, Fault Truth table, Deductive matrix), які дозволяють практично без програмування вирішувати завдання синтезу та аналізу: Testing map, Test minimal, Test Quality, Fault Diagnosis.

(Machine learning – алгоритми пошуку закономірностей у вхідних даних без програмування на основі розумних механізмів їх структуризації з метою розпізнавання шаблонів та прийняття рішення.) Все це дозволило створити унікальну технологію моделювання цифрових пристроїв для синтезу та оцінки якості тестів та вирішення задач діагностування цифрового виробу. При цьому вирішується задача підвищення адекватності та швидкодії моделювання несправностей, як адрес, за рахунок використання надмірності явних структур даних у вигляді логічної векторів, таблиць істинності та дедуктивних матриць.



Рисунок 1.7 – Схема комбінації технологій [рисунок сформовано автором та перераховано за допомогою сервісу Gemini]

Імена дослідників, які зробили значний внесок у теорію та практику design, test and fault simulation: Y. Zorian, R. Ubar, Vazgen Melikyan, Samvel Shoukourian, J. Bergeron, Z. Navabi, A. Jerraya, D. Armstrong, J. Abraham, H. Fujiwara, T. Nishida, X. Wang, Kang Li, Hans-Joachim Wunderlich, Hussam Amrouch, Patrick Groeneveld, Adit D. Singh, Giorgio Di Natale, Paolo Prinetto, H. Fatih Ugurdag, Fadi Maamari, P. Mueller, A. Ivanov, A., Irith Pomeranz; Sudhakar M. Reddy, J. Hayes; Aleksandr Palagin, Volodymyr Opanasenko, Oleksandr Drozd,

Artur Jutman, Maksim Jenihhin, N. Takahashi, Hana Kubatova, E.J. Marinissen, V.D. Agrawal, J.P. Roth, A. Matrosova, P. Parkhomenko, A. Birger, Roy Kaushik.

Як зазначено у аналітиці компанії Gartner [54], обчислювальне сховище (CS) переносить обробку хоста з основної пам'яті центрального процесора (ЦП) на пристрій. Це робить моделювання складних цифрових проєктів та аналіз великих даних економічними з погляду витрат часу та енергії. Це є загальним трендом комп'ютингу протягом найближчого десятиліття (рис. 1.8).

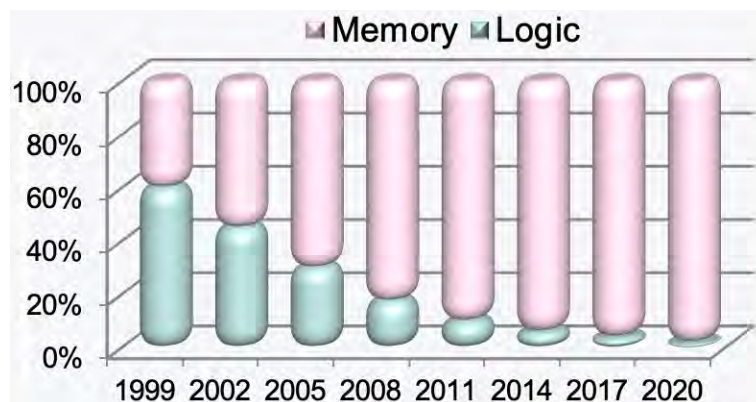


Рисунок 1.8 – Діаграма тенденції збільшення пам'яті на кристалі

Судячи з рис. 1.8, пам'ять збільшує свій вплив і стає все доступнішою у великих обсягах кожному споживачеві. Логіка, реалізована у залізі, прагне нулю. Їй на зміну приходить векторна логіка, що не вимагає синтезу, яка сьогодні має швидкодію [4-6, 14, 45-50] порівняно з логікою процесора або замовленої схеми.

1.6 Висновки до розділу 1

З проведеного аналізу можна зробити два висновки:

1) доступність пам'яті за ціною та обсягом для реалізації in-memory комп'ютингу не має обмежень;

2) використовуючи векторну логіку [81], яка не потребує системи команд центрального процесора, а використовує лише read-write транзакції на розумних структурах даних, можна отримати економію електроенергії до 40% і скоротити час обчислювальних операцій до 30%.

Векторна логіка також не потребує енерго- та часовитратних процедур синтезу в технологічно дозволену систему елементів на кристалі ASIC, VLSI, FPGA, RISC-V.

Синтез моделі для вирішення практичної задачі полягає у зручній для алгоритму суперпозиції (розміщенні) логічних векторів у пам'яті. Виявляється, це наукове та практичне завдання, яке потребує уваги дослідників. Це лише кілька позитивних моментів використання векторної логіки для ІТ-ринку, які чекають нових економічних моделей організації обчислювальних процесів.

Мета дослідження – зниження часових та енергетичних витрат fault/good-value as address simulation цифрових схем за рахунок надлишковості векторно-логічних моделей їх елементів.

Дисертація присвячена розв'язку науково-практичної задачі щодо економічних векторно-логічних обчислень, орієнтованих на безпроцесорну обробку у пам'яті (in-memory), що засновані на транзакціях читання-запису (read-write) над розумними структурами даних, якими виступають логічні вектори, таблиці істинності та матриці.

Задачі дослідження:

1. Аналіз стану проблеми та актуальних технологій прогресивного корпусування інтегральних мікросхем, моделювання несправностей цифрових систем та їх компонентів.

2. Вдосконалення розумних структур даних векторної логіки, що зменшують обчислювальну складність алгоритмів fault/good-value as address simulation за рахунок експоненціальної надмірності моделей елементів логічних схем.

3. Розробка архітектури для моделювання схеми та рендерингу процесів синхронізації векторно-логічного modeling for fault/good-value as address

simulation, що містить побудову моделі об'єкта у пам'яті комп'ютера та дозволяє візуально супроводжувати процес modeling for simulation цифрової схеми.

4. Розробка векторно-логічних механізмів modeling for fault/good-value as address simulation з лінійною алгоритмічною складністю.

5. Програмна реалізація розумних структур даних, алгоритмів та механізмів fault/good-value as address simulation, синхронний рендеринг всіх компонентів процесу симуляції на вікно монітору з метою візуального супроводжування процесу modeling for simulation цифрової схеми, їх верифікація на бібліотеці стендових схем ISCAS та імплементація до хмарного сервісу MOSI-HDL.

Результати розділу опубліковані у роботах [82, 83].

1.7 Перелік джерел посилання до розділу 1

1. Marinissen E. J., Pancholi V., Chuang P.-Y., Keim M. IEEE Std P3405: New Standard-under-Development for Chiplet Interconnect Test and Repair / E. J. Marinissen, V. Pancholi, P.-Y. Chuang, M. Keim // 2024 IEEE 42nd VLSI Test Symposium (VTS), Tempe, AZ, USA. – 2024. – P. 1–11. – DOI: 10.1109/VTS60656.2024.10538776.

2. Chiplet Interconnect Test and Repair in conjunction with IEEE European Test Symposium 2024, 23–24 May 2024. – [Електронний ресурс]. – Режим доступу: <https://ets24.nl/index.php/workshops/citar/> (дата звернення: 23.05.2024).

3. Lau J. H. Semiconductor Advanced Packaging / John H. Lau. – Singapore: Springer, 2021. – 498 p. DOI:10.1007/978-981-16-1376-0

4. Hahanov V., Gharibi W., Chumachenko S., Litvinova E. Vector Synthesis of Fault Testing Map For Logic / V. Hahanov, W. Gharibi, S. Chumachenko, E. Litvinova // IAES International Journal of Robotics and Automation (IJRA). – 2024. – Vol. 13, no. 3. – P. 293–306. – DOI: 10.11591/ijra.v13i3.pp293-306.

5. Hahanov V., Litvinova E., Hahanova H., Chumachenko S., Davitadze Z., Hahanova I., Kulak H., Ponomarova V., Abdullayev V. H. Vector-Logical In-Memory Simulation of Faults as Truth Table Addresses / V. Hahanov et al. // 2024 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, Armenia. – 2024. – P. 1–6. – DOI: 10.1109/EWDTS63723.2024.10873615.

6. Hahanov V., Devadze D., Hahanov I., Chumachenko S., Litvinova E., Obrizan V., Pashkov D., Mishchenko A., Maksymova N. Prompt-Testing of Logic / V. Hahanov et al. // 2024 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, Armenia. – 2024. – P. 1–5. – DOI: 10.1109/EWDTS63723.2024.10873774.

7. Franzon P. D., Marinissen E. J., Bakir M. S. Handbook of 3D Integration / Paul D. Franzon, Erik Jan Marinissen, Muhannad S. Bakir. – Weinheim: Wiley-VCH, 2019. – Vol. 4. – 488 p.

8. Chen A., Lo R. H.-Y. Semiconductor Packaging. Materials Interaction and Reliability / Andrea Chen, Randy Hsiao-Yu Lo. – Boca Raton: CRC Press, 2012. – 187 p.

9. IEEE International Roadmap for Devices and Systems. Lithography and Patterning. – Institute of Electrical and Electronics Engineers, 2023. – DOI: 10.60627/khgq-xy87.

10. Pavlidis V. F., Savidis I. S., Friedman E. G. Three-Dimensional Integrated Circuit Design / Vasilis F. Pavlidis, Ioannis Savidis, Eby G. Friedman. – 2nd ed. – Newnes, 2017. – 768 p.

11. Moore T. M., McKenna R. G. Characterization of Integrated Circuit Packaging Materials / Thomas M. Moore, Robert G. McKenna. – London: Butterworth-Heinemann, 2013. – 274 p.

12. TSMC. Chip on Wafer on Substrate with silicon interposer (CoWoS) [Электронный ресурс] / TSMC. – Режим доступа: <https://3dfabric.tsmc.com/english/dedicatedFoundry/technology/cowos.htm> (дата звернення: 08.04.2025).

13. Mahajan R. et al. Embedded Multidie Interconnect Bridge – A Localized, High-Density Multichip Packaging Interconnect / R. Mahajan et al. // IEEE

Transactions on Components, Packaging and Manufacturing Technology. – Oct. 2019. – Vol. 9, no. 10. – P. 1952–1962. – DOI: 10.1109/TCPMT.2019.2942708.

14. Hahanov V., Chumachenko S., Litvinova E., Hahanov I., Hahanova I. Vector-logic computing for faults-as-address deductive simulation / V. Hahanov et al. // IAES International Journal of Robotics and Automation. – 2023. – Vol. 12, no. 3. – P. 274–288. – DOI: 10.11591/ijra.v12i3.pp274-288.

15. Ulrich E.G. Exclusive Simulation of Activity in Digital Networks / E.G. Ulrich // Communications of the ACM. – Vol. 13, 1969, Feb., P. 102–110. – DOI відсутній.

16. Armstrong D.B. A Deductive Method for Simulating Faults in Logic Circuits / D.B. Armstrong // IEEE Transactions on Computers. – Vol. C-21, May 1972, P. 464–471. – DOI відсутній.

17. Ulrich E.G., Baker T. The Concurrent Simulation of Nearly Identical Digital Networks / E.G. Ulrich, T. Baker // Proceedings 10th Design Automation Workshop, 1973, P. 145–150. – DOI відсутній.

18. Abramovici M., Breuer M.A., Friedman A.D. Digital Systems Testing and Testable Design / M. Abramovici, M.A. Breuer, A.D. Friedman. – New York : IEEE Press, 1990. – 657 p. – DOI відсутній.

19. Chul Young Lee, Walker D.M.H. PROBE: a PPSFP simulator for resistive bridging faults / C.Y. Lee, D.M.H. Walker // 18th IEEE VLSI Test Symposium, Montreal, Quebec, Canada, 2000, P. 105–110. – doi: 10.1109/VTEST.2000.843833.

20. Riahi P.A., Navabi, Z. Lombardi F. A VPI-based combinational IP core module-based mixed level serial fault simulation and test generation methodology / P.A. Riahi, Z. Navabi, F. Lombardi // 2003 Test Symposium, 2003, P. 274–277. – DOI:10.1109/ATS.2003.1250822

21. Wu B., Zhu H., Chen K., Yan C., Liu W. MLiM: High-Performance Magnetic Logic in-Memory Scheme With Unipolar Switching SOT-MRAM / B. Wu et al. // IEEE Transactions on Circuits and Systems I: Regular Papers. – Vol. 70, No. 6, Jun. 2023, P. 2412–2424. – DOI: 10.1109/TCSI.2023.3254607.

22. Xiao C. et al. Resistance-Sum Architecture for Voltage-Controlled SOT-MRAM based Computing-in-Memory With Hybrid References / C. Xiao et al. // 2023 IEEE International Magnetic Conference (INTERMAG Short Papers), Sendai, Japan, 2023, P. 1–2. – DOI: 10.1109/INTERMAGShortPapers58606.2023.10228265.

23. Ahn B., Jang J., Na H., Seo M., Son H., Song Y.H. AI Accelerator Embedded Computational Storage for Large-Scale DNN Models / B. Ahn et al. // 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Incheon, Korea, 2022, P. 483–486. – DOI: 10.1109/AICAS54282.2022.9869991.

24. Yang Z., Zhang C., Hu M., Lin F. OPC: A Distributed Computing and Memory Computing-Based Effective Solution of Big Data / Z. Yang et al. // 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), Chengdu, China, 2015, P. 50–53. – DOI: 10.1109/SmartCity.2015.46.

25. Moreau M. et al. Reliable ReRAM-based Logic Operations for Computing in Memory / M. Moreau et al. // 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), Verona, Italy, 2018, P. 192–195. – DOI: 10.1109/VLSI-SoC.2018.8644780.

26. Kang W., Zhang H., Zhao W. Spintronic Memories: From Memory to Computing-in-Memory / W. Kang, H. Zhang, W. Zhao // 2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Qingdao, China, 2019, P. 1–2. – DOI: 10.1109/NANOARCH47378.2019.181298.

27. Gauchi R. et al. Memory Sizing of a Scalable SRAM In-Memory Computing Tile Based Architecture / R. Gauchi et al. // 2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC), Cuzco, Peru, 2019, P. 166–171. – DOI: 10.1109/VLSI-SoC.2019.8920373.

28. Vasudevan V., Abdullatif A., Kabir S., Campean F. Certifiability Analysis of Machine Learning Systems for Low-Risk Automotive Applications / V. Vasudevan et al. // Computer. – Vol. 57, No. 9, 2024, P. 45–56. – DOI: 10.1109/MC.2024.3401402.

29. Gharibi W., Hahanov V., Litvinova E., Hahanov I. Quantum Structures for Digital Systems Synthesis / W. Gharibi et al. // Proceedings of IEEE East-West Design & Test Symposium (EWDTS-2015). – Batumi, 2015, P. 115–122. – doi: 10.1109/EWDTS.2015.7493180.

30. Hahanov V., Yemelyanov I., Obrizan V., Hahanov I. Quantum Diagnosis and Simulation of SoC / V. Hahanov et al. // Proceedings of XIth International Conference MEMSTECH-2015, 2015, P. 58–60.

31. Hahanov I., Gharibi W., Iemelianov I., Bani Amer T. QuaSim – Cloud Service for Digital Circuits Simulation / I. Hahanov et al. // 2016 Proceedings of IEEE East-West Design & Test Symposium, Yerevan, Armenia, 2016, P. 141–158. – doi: 10.1109/EWDTS.2016.7807667

32. Hahanov I., Chumachenko S., Iemelianov I., Hahanov V., Larchenko L. and T Daniyil. Deductive Qubit Fault Simulation / I. Hahanov et al. // *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, 2017, Lviv, Ukraine, pp. 256-259, doi: 10.1109/CADSM.2017.7916129.

33. Hahanov V., Iemelianov I., Chumachenko S., Hahanov I., Hahanova I. Quantum sequencer for the minimal test synthesis of black-box functionality / V. Hahanov et al. // 2017 IEEE East-West Design & Test Symposium (EWDTS), 2017, Novi Sad, Serbia, pp. 1-6, doi: 10.1109/EWDTS.2017.8110148.

34. Hahanov I., Iemelianov I., Liubarskyi M., Hahanov V. Qubit Description of Functions and Structures for Service Computing Synthesis / I. Hahanov et al. // *Cyber Physical Computing for IoT-driven Services*. – Springer, 2018, P. 71–93. – DOI:10.1007/978-3-319-54825-8_4

35. Hahanov I., Amer T., Iemelianov I., Liubarskyi M., Hahanov V. QuaSim Cloud Service for Quantum Circuit Simulation / I. Hahanov et al. // *Cyber Physical Computing for IoT-driven Services*. – Springer, 2018, P. 135–147. – DOI:10.1007/978-3-319-54825-8_6

36. Hahanov V., Chumachenko S., Litvinova E., Hahanov I., Hahanova A. Methods for Quantum Analysis of Digital Circuits / V. Hahanov et al. // *Proc. IEEE*

International Conference TCSET 2018, Lviv-Slavsk, 2018, P. 441–447. – DOI:10.1109/TCSET.2018.8336317

37. Hahanov V., Liubarskyi M., Gharibi W., Chumachenko S., Litvinova E., Hahanov I. Test Synthesis for Logical X-functions / V. Hahanov et al. // 2018 IEEE East-West Design & Test Symposium (EWDTS), 2018, P. 1–9. – DOI: 10.1109/EWDTS.2018.8524863.

38. Hahanov V., Hahanov I. et al. Quantum Mem-Computing for Design and Test / V. Hahanov et al. // 2018 IEEE Globecom Workshops (GC Wkshps), Abu Dhabi, UAE, 2018, P. 1–7. – DOI: 10.1109/GLOCOMW.2018.8644256.

39. Hahanov V., Litvinova E., Khakhanova H., Hahanova I. Qubit Fault Detection in SoC Logic / V. Hahanov et al. // 2019 IEEE East-West Design & Test Symposium (EWDTS), Batumi, Georgia, Sep. 13–16, 2019, P. 1–7 (108–114). – doi: 10.1109/EWDTS.2019.8884475.

40. Abdullayev V.H., Hahanov I. et al. Structure and Metrics of Emerging Computing / V. H. Abdullayev et al. // 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), Lviv-Slavske, Ukraine, 2020, P. 920–925. – DOI: 10.1109/TCSET49122.2020.235571.

41. Gharibi W., Hahanov V., Man K.L., Chumachenko S., Litvinova E., Hahanov I. Quantum Deterministic Computing / W. Gharibi et al. // 2020 IEEE East-West Design & Test Symposium (EWDTS), Varna, Bulgaria, 2020, P. 1–6. – DOI: 10.1109/EWDTS50664.2020.9224950.

42. Hahanov V., Gharibi W., Chumachenko S., Litvinova E., Abdullayev V.H., Tariq Hama Salih T.H., Hahanov I. Vector-qubit models for SoC logic-structure testing and fault simulation / V. Hahanov et al. // Experience of Designing and Application of CAD Systems in Microelectronics, 2021, P. 24–28. – doi: 10.1109/CADSM52681.2021.9385266.

43. Hahanov V., Gharibi W., Karavay M., Man K.L., Chumachenko S., Litvinova E., Tariq Hama Salih, Devadze D., Hahanov I. Quantum Digital-Analogue

Computing / V. Hahanov et al. // 2021 IEEE East-West Design and Test Symposium (EWDTS 2021) – Proceedings, 2021. – doi: 10.1109/EWDTS52692.2021.9581044.

44. Hahanov V., Litvinova E., Shevchenko O., Chumachenko S., Khakhanova H., Hahanov I. Vector Models for Modeling Logic Based on XOR-Relations / V. Hahanov et al. // 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), Lviv-Slavske, Ukraine, 24–27 Feb. 2022, P. 823–828. – DOI: 10.1109/TCSET55632.2022.9766894.

45. Gharibi W., Hahanov V., Devadze D., Litvinova E., Hahanov I. Deductive Matrix Synthesis for Fault Simulation / W. Gharibi et al. // 2023 IEEE 17th International Conference on the Experience of Designing and Application of CAD Systems, CADSM 2023 – Proceedings, 2023, P. 6–10. – doi: 10.1109/CADSM58174.2023.10076506.

46. Gharibi W., Hahanova A., Hahanov V., Chumachenko S., Litvinova E., Hahanov I. Vector-deductive memory-based transactions for fault-as-address simulation / W. Gharibi et al. // Electronic Modeling. – Vol. 45, No. 1, 2023, P. 03–26. – DOI: 10.15407/emodel.45.01.

47. Gharibi W., Hahanova A., Hahanov V., Chumachenko S., Litvinova E., Hahanov I. Vector–Logic Synthesis of Deductive Matrices for Fault Simulation / W. Gharibi et al. // Electronic Modeling. – Vol. 45, No. 2, 2023, P. 16–33. – DOI: 10.15407/emodel.45.02.016.

48. Hahanov V., Chumachenko S., Litvinova E., Hahanov I., Ponomarova V., Khakhanova H., Kulak G. Faults-as-Address Simulation / V. Hahanov et al. // IAES International Journal of Robotics and Automation (IJRA). – Vol. 13, No. 4, 01 Dec. 2024, P. 452–468. – DOI: 10.11591/ijra.v13i4.pp452-468.

49. Hahanov V., Hahanov I., Miroshnyk A., Shkil A., Rakhlis D., Hahanova I. Modeling Faults as Addresses / V. Hahanov et al. // 2023 IEEE East-West Design & Test Symposium (EWDTS), Batumi, Georgia, 2023, P. 1–7. – DOI: 10.1109/EWDTS59469.2023.10297037.

50. Devadze D., Hahanov I. et al. In-Memory Fault-Free Vector Simulation / D. Devadze et al. // 2023 IEEE East-West Design & Test Symposium (EWDTS), Batumi, Georgia, 2023, P. 1–6. – DOI: 10.1109/EWDTS59469.2023.10297076.
51. Hahanov V. Cyber-Physical Computing for IoT-driven Services / V. Hahanov. – New York : Springer, 2018. – DOI:10.1007/978-3-319-54825-8
52. Pretz K. The Maestro Behind Design-Software Behemoth Synopsys / K. Pretz // IEEE Spectrum, Sept. 2023, P. 52–53. – URL: <https://spectrum.ieee.org/aart-de-geus-profile>.
53. IEEE Standard Testability Method for Embedded Core-based Integrated Circuits / IEEE Std 1500-2022. – 12 Oct. 2022. – DOI: 10.1109/IEEESTD.2022.9916221.
54. What's New in the 2022 Gartner Hype Cycle for Emerging Technologies. – URL: <https://www.gartner.com/en/articles/what-s-new-in-the-2022-gartner-hype-cycle-for-emerging-technologies>.
55. Coluccio A. et al. Hybrid-SIMD: A Modular and Reconfigurable Approach to Beyond von Neumann Computing / A. Coluccio et al. // IEEE Transactions on Computers. – Vol. 71, No. 9, Sept. 2022, P. 2287–2299. – DOI: 10.1109/TC.2021.3127354.
56. Wang P. et al. RC-NVM: Enabling Symmetric Row and Column Memory Accesses for In-memory Databases / P. Wang et al. // 2018 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2018, P. 518–530. – DOI: 10.1109/HPCA.2018.00051.
57. Liu T., Yu T., Wang S., Cai S. An Efficient Degraded Deductive Fault Simulator for Small-Delay Defects / T. Liu et al. // IEEE Access. – Vol. 8, 2020, P. 204855–204862. – DOI: 10.1109/ACCESS.2020.3037292.
58. Burhan M. et al. A Comprehensive Survey on the Cooperation of Fog Computing Paradigm-Based IoT Applications / M. Burhan et al. // IEEE Access. – Vol. 11, 2023, P. 73303–73329. – DOI: 10.1109/ACCESS.2023.3294479.
59. Kaya Z., Garrido M., Takala J. Memory-Based FFT Architecture with Optimized Number of Multiplexers and Memory Usage / Z. Kaya et al. // IEEE

Transactions on Circuits and Systems II: Express Briefs. – Vol. 70, No. 8, Aug. 2023, P. 3084–3088. – DOI: 10.1109/TCSII.2023.3245823.

60. Hahanov V. et al. Vector-Qubit models for SoC Logic-Structure Testing and Fault Simulation / V. Hahanov et al. // 2021 IEEE 16th International Conference CADSM, 2021, P. 24–28. – DOI: 10.1109/CADSM52681.2021.9385266.

61. Hahanov V.I., Hyduke S.M., Gharibi W., Litvinova E.I., Chumachenko S.V., Hahanova I.V. Quantum Models and Method for Analysis and Testing Computing Systems / V.I. Hahanov et al. // 2014 11th International Conference on Information Technology: New Generations (ITNG), Las Vegas, NV, 2014, P. 430–434. – DOI: 10.1109/ITNG.2014.125.

62. Hahanov V., Gharibi W., Litvinova E., Chumachenko S. Qubit-driven Fault Simulation / V. Hahanov et al. // 2019 IEEE Latin American Test Symposium (LATS), Chile, 2019, P. 1–7. – DOI: 10.1109/LATW.2019.8704583.

63. Ubar R., Raik J., Jenihhin M., Jutman A. Structural Decision Diagrams in Digital Test. Theory and Applications / R. Ubar et al. – Springer, Birkhäuser Cham, 2024. – DOI: 10.1007/978-3-031-44734-1.

64. Marinissen E.J., Zorian Y. IEEE Std 1500 Enable Modular SoC Testing / E.J. Marinissen, Y. Zorian // IEEE Design & Test of Computers. – Vol. 26, No. 1, Jan.–Feb. 2009, P. 8–17. – DOI: 10.1109/MDT.2009.12.

65. Kaczmarek B. et al. LBIST for Automotive ICs With Enhanced Test Generation / B. Kaczmarek et al. // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – Vol. 41, No. 7, Jul. 2022, P. 2290–2300. – DOI: 10.1109/TCAD.2021.3100741.

66. Lee S., Cho K., Choi S., Kang S. A New Logic Topology-Based Scan Chain Stitching for Test-Power Reduction / S. Lee et al. // IEEE Transactions on Circuits and Systems II: Express Briefs. – Vol. 67, No. 12, Dec. 2020, P. 3432–3436. – DOI: 10.1109/TCSII.2020.3004371.

67. Papavramidou P., Nicolaidis M. Iterative Diagnosis Approach for ECC-Based Memory Repair / P. Papavramidou, M. Nicolaidis // IEEE Transactions

on Computer-Aided Design of Integrated Circuits and Systems. – Vol. 39, No. 2, Feb. 2020, P. 464–477. – DOI: 10.1109/TCAD.2018.2887052.

68. Zhang H., Liu K., Zhao M., Shen Z., Cai X., Jia Z. Pearl: Performance-Aware Wear Leveling for Nonvolatile FPGAs / H. Zhang et al. // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – Vol. 40, No. 2, Feb. 2021, P. 274–286. – DOI: 10.1109/TCAD.2020.2998779.

69. Wagle A., Vrudhula S. Heterogeneous FPGA Architecture Using Threshold Logic Gates for Improved Area, Power, and Performance / A. Wagle, S. Vrudhula // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – Vol. 41, No. 6, Jun. 2022, P. 1855–1867. – DOI: 10.1109/TCAD.2021.3099780.

70. Lin L., Huang Y., Xu L., Hsieh S.-Y. A Complete Fault-Tolerant Method for Extra Fault Diagnosability of Alternating Group Graphs / L. Lin et al. // IEEE Transactions on Reliability. – Vol. 70, No. 3, Sept. 2021, P. 957–969. – DOI: 10.1109/TR.2020.3021233.

71. Pomeranz I. Bit-Complemented Test Data to Replace the Tail of a Fault Coverage Curve / I. Pomeranz // IEEE Transactions on Very Large-Scale Integration (VLSI) Systems. – Vol. 32, No. 4, Apr. 2024, P. 609–618. – DOI: 10.1109/TVLSI.2024.3365355.

72. Jinling D., Aiqiang X. A fault simulation method based on mutated truth table of logic gates / D. Jinling, X. Aiqiang // 2016 International Conference on Integrated Circuits and Microsystems (ICICM), China, 2016, P. 28–32. – DOI: 10.1109/ICAM.2016.7813557.

73. Wang Q., Jin T., Mohamed M.A. A Fast and Robust Fault Section Location Method for Power Distribution Systems Considering Multisource Information / Q. Wang et al. // IEEE Systems Journal. – Vol. 16, No. 2, Jun. 2022, P. 1954–1964. – DOI: 10.1109/JSYST.2021.3057663.

74. Hu J. et al. Adaptive Multidimensional Parallel Fault Simulation Framework on Heterogeneous System / J. Hu et al. // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – Vol. 42, No. 6, Jun. 2023, P. 1951–1964. – DOI: 10.1109/TCAD.2022.3213617.

75. Ehteram A., Sabaghian-Bidgoli H., Ghasvari H., Hessabi S. A Simple and Fast Solution for Fault Simulation Using Approximate Parallel Critical Path Tracing / A. Ehteram et al. // Canadian Journal of Electrical and Computer Engineering. – Vol. 43, No. 2, Spring 2020, P. 100–110. – DOI: 10.1109/CJECE.2019.2950280.

76. Chang S.-H., Liu C.-N.J., Küster A. Behavioral Level Simulation Framework to Support Error-Aware CNN Training with In-Memory Computing / S.-H. Chang et al. // 2022 SMACD, Italy, 2022, P. 1–4. – DOI: 10.1109/SMACD55068.2022.9816307.

77. Zarei A., Safaei F. LIMITA: Logic-in-Memory Primitives for Imprecise Tolerant Applications / A. Zarei, F. Safaei // IEEE Transactions on Circuits and Systems I: Regular Papers. – Vol. 68, No. 11, Nov. 2021, P. 4686–4699. – DOI: 10.1109/TCSI.2021.3106017.

78. Yao Z., Liu Y., Chen J., Ji J., Zhang M., Gong Y. Active High-Impedance Fault Detection Method for Resonant Grounding Distribution Networks / Z. Yao et al. // IEEE Access. – Vol. 12, 2024, P. 10932–10945. – DOI: 10.1109/ACCESS.2024.3352258.

79. Shan R. Certifying Generative AI: Retrieval-Augmented Generation Chatbots in High-Stakes Environments / R. Shan // Computer. – Vol. 57, No. 9, 2024, P. 35–44. – DOI: 10.1109/MC.2024.3401085.

80. Hessami A., Kriebitz A., Weger G., Watson E., Shaw P. Artificial Intelligence for the Benefit of Everyone / A. Hessami et al. // Computer. – Vol. 57, No. 9, 2024, P. 68–79. – DOI: 10.1109/MC.2024.341179.

81. Hahanov V. Vector Logic Computing. / V. Hahanov. – New York : Springer, 2026. – ISBN 978-3-032-07001-2. – <https://doi.org/10.1007/978-3-032-07000-2> (Springer eBook, in Press). <https://link.springer.com/book/9783032070005>

82. Демченко О.І., Чумаченко С.В. Технології прогресивного корпусування інтегральних мікросхем / О.І. Демченко, С.В. Чумаченко // Системи управління, навігації та зв'язку. – 2025. – Т. 3, № 81. – С. 199–202. – DOI: <https://doi.org/10.26906/SUNZ.2025.3.199>.

83. Чумаченко С.В., Демченко О.І. Технології прогресивного корпусування ІМС: основні виклики / С.В. Чумаченко, О.І. Демченко // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : матеріали 15-ї міжнар. наук.-техн. конф., Баку – Харків – Жиліна, 24–25 квітня 2025 р. – Т. 2. – С. 26–27. – Режим доступу: <https://repository.kpi.kharkov.ua/handle/KhPI-Press/93979> (доступ 09.06.2026).

2 РОЗУМНІ СТРУКТУРИ ДАНИХ ТА ВЕКТОРНА ЛОГІКА ЇХ ОБРОБКИ

2.1 Передмова

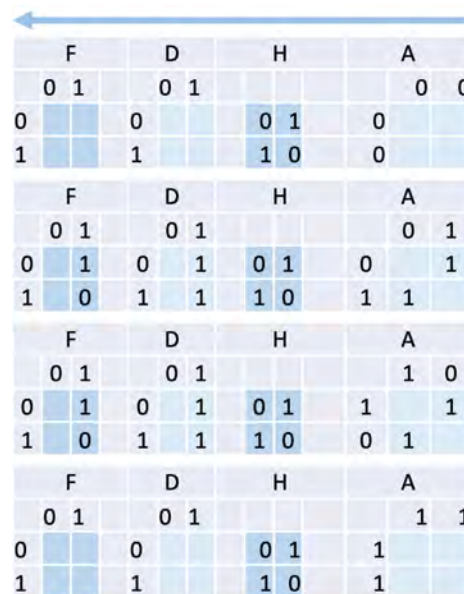
Векторно-логічний комп'ютинг [1, 13] пропонує моделювання несправностей як адрес цифрової схеми за допомогою алгоритму моделювання справної поведінки тестових двійкових наборів як адрес. Моделювання несправностей на вхідному наборі логічної схеми означає обробку матриці несправностей розміром n^2 , n – кількість вхідних внутрішніх та вихідних ліній. По головній діагоналі матриці апріорі розміщуються одиниці, що ідентифікують несправність ліній схеми. Після обробки матриці ці одиниці перетворюються на виявлені константні несправності, знаки яких завжди інверсні справним значенням ліній схеми. Несправністю є будь-яке логічне значення $\{0,1\}$, що кодується 1, яке змінює вихід елемента на вхідному тестовому наборі. Матриця моделювання несправностей перетворює процес аналізу несправностей у алгоритм справного моделювання ліній схеми на дедуктивних векторах. Проблеми моделювання комбінацій несправностей з розгалуженнями, що збігаються, не існує. Оскільки виявлені на лініях несправності не передаються елементами схеми, а позначаються одиницею в координаті стовпця матриці аналізованої лінії. Матриця моделювання також є розумною структурою даних для обробки схем зі зворотними зв'язками.

Розглядаються такі питання: 1) розробка розумних структур даних векторної логіки для зниження обчислювальної складності алгоритмів тестування логічних функціональностей; 2) створення механізмів тестування векторної логіки на основі використання таблиць істинності та матриць, побудованих на логічному векторі; 3) векторне тестування логічних схем шляхом моделювання несправностей як адрес бітів логічного вектора.

2.2 Синтез карти тестування для найпростішої логіки

Математична основа дедуктивного моделювання несправностей полягає у транспортуванні на вихід двійкових комбінацій вхідних дефектів на заданому наборі вхідному за формулою $L=T \oplus F$. Сутність дедуктивного моделювання полягає у зміні логіки елемента F залежно від вхідних умов T . Дедуктивне моделювання, запропоноване 50 років тому Армстронгом, досі є найефективнішим засобом аналізу якості тестів та синтезу таблиць для пошуку дефектів, простежування шляху поширення несправності. Далі пропонується його реалізація на основі векторної форми опису логіки, що виключає логічні аналітичні форми та дає можливість суттєво спростити алгоритми синтезу дедуктивних моделей та їх застосування для інтерпретативного моделювання цифрових елементів та схем великої розмірності.

Найпростіша реалізація векторно-логічного моделювання несправностей як адрес починається з функціональностей, що мають одну змінну. На основі на основі логічного вектора (01, 10) будується дедуктивна матриця та карта тестування несправностей (рис. 2.1).



	F		D		H		A	
	0	1	0	1			0	0
0			0		0	1	0	
1			1		1	0	0	

	F		D		H		A	
	0	1	0	1			0	1
0		1	0	1	0	1	0	1
1		0	1	1	1	0	1	1

	F		D		H		A	
	0	1	0	1			1	0
0		1	0	1	0	1	1	1
1		0	1	1	1	0	0	1

	F		D		H		A	
	0	1	0	1			1	1
0			0		0	1	1	
1			1		1	0	1	

Рисунок 2.1 – Схема синтезу карт тестування для векторної логіки 00, 01, 10, 11

Якщо логічний вектор представлений константним значенням, тобто всі його біти дорівнюють 1 або 0, то карта тестування F для такої функції не існує. Для логічних векторів 01 і 10 карта тестування показує ті константні логічні несправності, відзначені нулем або одиницею, що перевіряються відповідними тестами: тест T0 перевіряє несправність F1, і тест T1 перевіряє несправність F0.

Для двовходової векторної логіки є лише 16 найпростіших функціональностей, кількість яких визначається формулою $Q=2^n$, n – число змінних логічної функції.

На рис. 2.2 представлений синтез карт тестування окремих функціональностей (0111, 0001, 1011, 1101, 0110, 1001).

Test-map	D-matrix	H-matrix	L-matrix
TT 00 01 10 11	TT 00 01 10 11		L 0 1 1 1
00 .1 1. 11	00 1 1 1	0 1 2 3	0 1 1 1
01 .0 .	01 1	1 0 3 2	1 1
10 . 0.	10 1	2 3 0 1	1 1
11 . . 00	11 1	3 2 1 0	1 1

Test-map	D-matrix	H-matrix	L-matrix
TT 00 01 10 11	TT 00 01 10 11		L 0 0 0 1
00 . 11	00 1	0 1 2 3	0 1
01 . 1.	01 1	1 0 3 2	0 1
10 .1 .	10 1	2 3 0 1	0 1
11 .0 0. 00	11 1 1 1	3 2 1 0	1 1 1 1

Test-map	D-matrix	H-matrix	L-matrix
TT 00 01 10 11	TT 00 01 10 11		L 1 0 1 1
00 .1 .	00 1	0 1 2 3	1 1
01 .0 1. 10	01 1 1 1	1 0 3 2	0 1 1 1
10 . . 01	10 1	2 3 0 1	1 1
11 . 0.	11 1	3 2 1 0	1 1

Test-map	D-matrix	H-matrix	L-matrix
TT 00 01 10 11	TT 00 01 10 11		L 1 1 0 1
00 . 1.	00 1	0 1 2 3	1 1
01 . . 10	01 1	1 0 3 2	1 1
10 .1 0. 01	10 1 1 1	2 3 0 1	0 1 1 1
11 .0 .	11 1	3 2 1 0	1 1

Test-map	D-matrix	H-matrix	L-matrix
TT 00 01 10 11	TT 00 01 10 11		L 0 1 1 0
00 .1 1.	00 1 1	0 1 2 3	0 1 1
01 .0 1.	01 1 1	1 0 3 2	1 1 1
10 .1 0.	10 1 1	2 3 0 1	1 1 1
11 .0 0.	11 1 1	3 2 1 0	0 1 1

Test-map	D-matrix	H-matrix	L-matrix
TT 00 01 10 11	TT 00 01 10 11		L 1 0 0 1
00 .1 1.	00 1 1	0 1 2 3	1 1 1
01 .0 1.	01 1 1	1 0 3 2	0 1 1
10 .1 0.	10 1 1	2 3 0 1	0 1 1
11 .0 0.	11 1 1	3 2 1 0	1 1 1

Рисунок 2.2 – Схема синтезу карт тестування для векторної логіки

0111, 0001, 1011, 1101, 0110, 1001

2.3 Інжиніринг верифікації логічних функціональностей

Верифікація – процес визначення відповідності між специфікацією та синтезованим пристроєм за рахунок введення логіко-часової надмірності для пошуку та усунення помилок проектування.

Тестова верифікація – процес визначення помилок проектування на основі тесту, що синтезується (надмірність проекту) для заданого класу несправностей.

Формальна верифікація – процес досягнення якості проекту на основі використання логіко-часової надмірності (темпоральної логіки) та бібліотеки валідних схемотехнічних рішень. Тут використовується темпоральна логіка SystemVerilog, Assertions для формулювання обмежень (constraints) та розпізнавання комбінацій сигналів (functional coverage groups та продукти Cadence Jasper Gold и та Synopsys VC Formal) [2].

Functional coverage (FC) – це метрика плану тестування функціональних можливостей та режимів проекту. FC вручну додається до testbench, підтримується, наприклад, Vivado-симулятором для визначення якості тестів, неперевірених функцій, надлишкових тестів.

Testbench – це частина програмного коду, яка використовується для запуску моделювання, що подає вхідні набори на UUT, отримує вихідні реакції та порівнює їх з очікуваними для виявлення та виправлення помилок у проекті. Тут проблемним питанням виступає – автоматично синтезувати функціональний coverage для цифрового проекту.

Пропонується синтез карти тестування (рис. 2.3) для логічної функціональності, яка вирішує задачу автоматичного синтезу функціонального coverage для проектів з невеликим числом ($n \leq 20$) вхідних змінних під стандарти граничного сканування.

Адресна Н-матриця десяткових адрес розмірністю $2^n \times 2^n$ – це хог-відношення між вхідними тестовими наборами та вектор-адресами таблиці істинності від n -змінних, які формують вектори відстаней або всі комбінації

логічних несправностей, що перевіряються, на вичерпному тесті. Н-матриця – це матриця-константа логічних несправностей для будь-якої логіки від n змінних.

F-matrix									D-matrix									H-matrix								A-matrix										
TT	000	100	010	110	001	101	011	111	TT	000	100	010	110	001	101	011	111											L	0	0	0	0	0	0	0	1
000								111	000								1	0	1	2	3	4	5	6	7		0							1		
100							11.		001							1		1	0	3	2	5	4	7	6		0						1			
010						1.1			010						1			2	3	0	1	6	7	4	5		0						1			
110					1..				011					1				3	2	1	0	7	6	5	4		0					1				
001					.11				100				1					4	5	6	7	0	1	2	3		0					1				
101			.1.						101			1						5	4	7	6	1	0	3	2		0					1				
011		..1							110		1							6	7	4	5	2	3	0	1		0					1				
111	..0	.0.	.00	0..	0.0	00.	000		111		1	1	1	1	1	1	1	7	6	5	4	3	2	1	0		1	1	1	1	1	1	1			

F-matrix									D-matrix									H-matrix								A-matrix										
TT	000	001	010	011	100	101	110	111	TT	000	001	010	011	100	101	110	111											L	0	0	0	1	1	0	1	1
000			.11	1..		11.	111		000				1	1		1	1	0	1	2	3	4	5	6	7		0			1	1		1	1		
001			.1.			1.0	11.	110	001			1				1	1	1	1	0	3	2	5	4	7	6		0			1	1		1	1	
010		..1			1..	1.1	10.		010		1				1	1	1		2	3	0	1	6	7	4	5		0			1	1		1	1	
011		..0	.0.	.00			10.		011		1	1	1				1		3	2	1	0	7	6	5	4		1	1	1	1		1			
100		..1			0..	0.1	01.		100		1				1	1	1		4	5	6	7	0	1	2	3		1	1	1	1		1			
101		..0	.1.	.10			01.		101		1	1	1				1		5	4	7	6	1	0	3	2		0			1	1		1	1	
110				.01	0..		00.	001	110				1	1			1	1	6	7	4	5	2	3	0	1		1	1	1	1		1			
111			.0.			0.0	00.	000	111			1				1	1	1	7	6	5	4	3	2	1	0		1	1	1	1		1			

Рисунок 2.3 – Синтез карти тестування (F-matrix)

Активна матриця – це симетрична щодо головної діагоналі двійкова структура, представлена набором логічних векторів чи його інверсій на вирішення завдань тестування логічних функціональностей. Матриця активності A – це хог-відношення між вхідними наборами зі значеннями виходів та стовпцями таблиці істинності

$$(T_i, L_i) \oplus F_i, \quad (2.1)$$

які формують невпорядковані за адресами активні координати. Рядки матриці активності L формуються за формулою

$$L \ni L_i = l_i \oplus 1, i = \overline{1, 2^n}, \quad (2.2)$$

де l – логічний вектор.

Дедуктивна D-матриця функціональності виходить шляхом приведення координат матриці A-активності, синтезованої на основі логічного вектора L, до стандартного порядку розташування адрес за правилами координатної операції перекодування:

$$D=A_H, A \ni A_i = L \oplus L_i. \quad (2.3)$$

D-матриця містить усі активні 1-координати логічної функціональності та вирішує питання:

- 1) моделювання несправностей;
- 2) визначення критичних станів для синтезу покриття функціональності;
- 3) синтез мінімального тесту;
- 4) синтез таблиці несправностей для пошуку дефектів у цифровому виробі.

Оцінка за витратами пам'яті розмірністю $2^n \times 2^n$.

Активна 1-координата D-матриці здійснює суперпозицію тестового набору з адресою-вектором таблиці покриття або несправностей, де довизначені інверсією вхідних сигналів одиниці є несправності, що перевіряються, які змінюють стан виходу функціональності.

Таку процедуру також можна назвати дедуктивно-векторним методом моделювання. Однак дедуктивне моделювання, запропоноване Армстронгом, було орієнтоване на обробку аналітичних моделей вентильного рівня, що задаються булевими функціями

$$Y=f(x_1, x_2, x_i, x_n) \quad (2.4)$$

на основі застосування рівняння дедукції:

$$F_i=T_i \oplus f(x_1, x_2, x_j, x_n) = f(x_1 \oplus T_{i1}, x_2 \oplus T_{i2}, x_3 \oplus T_{ij}, x_n \oplus T_{in}) \oplus f(T_i). \quad (2.5)$$

Складність розв'язання такого рівняння для генерації дедуктивної формули транспортування несправностей на тестовому вхідному наборі визначається кубічними витратами від числа змінних n^3 . Цей спосіб генерації формули моделювання несправностей також є похідною від рівняння технічної діагностики [3]:

$$F \oplus T \oplus L = 0, \quad (2.6)$$

де $L = f(x_1, x_2, x_j, x_n)$ – аналітична модель логічної функціональності, F_i – формула транспортування несправностей на тестовому наборі T_i .

Цілком виправдано, що провідні компанії EDA відмовились від моделювання несправностей, включаючи дедуктивний метод, внаслідок їх алгоритмічної складності.

Векторно-логічний метод моделювання несправностей вільний від операцій над логічними формульними виразами і має простий алгоритм аналізу векторів, що містить три матричні процедури з квадратичною обчислювальною складністю. Він орієнтований на механізми моделювання несправностей на ринку верифікації цифрових виробів. Він відзначається простотою реалізації, здатний скласти конкуренцію існуючим технологіям верифікації на EDA-ринку завдяки механізмам моделювання несправностей як адрес на розумних структурах даних.

2.4 Побудова карти тестування функціональності

Продемонструємо, як будувати карту тестування (рис. 2.4), використовуючи лише вектор логічної функціональності.

Логічний вектор – це також явна форма визначення логічної функціональності з використанням упорядкованої послідовності з бітів, що

адресуються. Логічний вектор разом із впорядкованим набором явних двійкових адрес утворюють таблицю істинності.

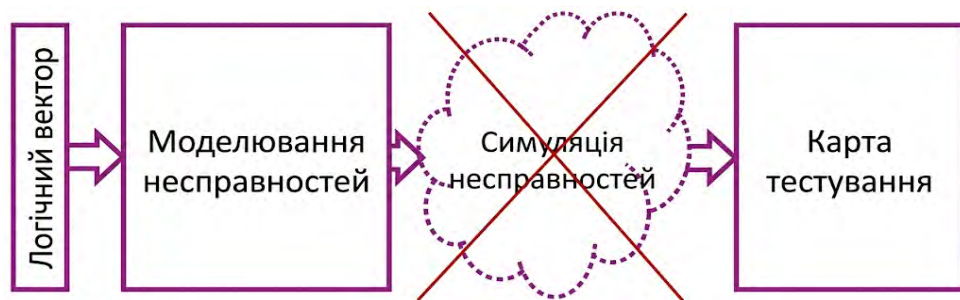


Рисунок 2.4 – Інженерія побудови карти тестування

Розумна структура даних розглядається як набір компонентів, пов'язаних єдиною просторовою метрикою для вирішення задач без програмування. Розумні структури даних для тестування логіки уявляють собою логічний вектор Y , таблиці істинності тесту t і несправностей f , а також матриці відповідно: L – активності, F – несправностей, T – тестів.

Тестування – це процес паралельного й автоматичного синтезу тестів та моделювання несправностей, як адрес, у розумних структурах даних за допомогою рівняння конволюції (2.6). Тестова матриця T є повним набором тестів і несправностей, що перевіряються в кожному двійковому наборі. Побудова T -матриці визначається рівнянням $T=L \oplus F$.

Матриця несправностей F є логічною константою, інші матриці L і T є змінними. Матричною операцією $T=L_F$, одночасно вирішуються наступні завдання: синтез тестів, моделювання та діагностування несправностей будь-якої функціональності, заданої логічним вектором. Рядки матриці активності L формуються за формулою

$$L \ni L_i = Y_i \oplus Y, i = \overline{1, 2^n}, \quad (2.7)$$

де Y – логічний вектор.

Таблиця істинності складається з суперпозиції двох частин (A, Y):

1) упорядкованих за зростанням двійкових адрес A, які формують стандартну константу будь-якої логічної функції від n-змінних;

2) логічного вектора Y, як змінної, що формує цифрову функціональність.

Повний набір адрес таблиці істинності представляє вичерпний тест T для логічної функції. Комбінації одиниць в адресах таблиці істинності – це повний стандартний набір логічних несправностей F для будь-якої цифрової функції від n-змінних.

Декартова хог-операція всіх тест-адрес і fault-адрес $F=T \oplus L$ створює стандартну fault-матрицю, як константу для будь-якої логічної функції (рис. 2.5).

L-matrix: $L_i = Y_i \oplus Y$									F-matrix								T-matrix Standard							
Y	0	0	0	1	1	0	1	1	$F=A \oplus A$								$T=L_F$							
0				1	1		1	1	0	1	2	3	4	5	6	7				1	1		1	1
0				1	1		1	1	1	0	3	2	5	4	7	6			1			1	1	1
0				1	1		1	1	2	3	0	1	6	7	4	5			1			1	1	1
1	1	1	1				1		3	2	1	0	7	6	5	4			1	1	1			1
1	1	1	1				1		4	5	6	7	0	1	2	3			1			1	1	1
0				1	1		1	1	5	4	7	6	1	0	3	2			1	1	1			1
1	1	1	1				1		6	7	4	5	2	3	0	1					1	1		1
1	1	1	1				1		7	6	5	4	3	2	1	0			1			1	1	1

T-matrix Standard $T_{ij} = A_j^F \leftarrow T_{ij} = 1$										$V_{ijt} = \bar{A}_{it}^T \leftarrow T_{ijt} = 1$									
A^F	000	001	010	011	100	101	110	111		A^F	000	001	010	011	100	101	110	111	A^T
				.11	1..		11.	111					.11	1..		11.	111	000	
			.1.			1.1	1.1	111					.1.			1.0	11.	110	001
		..1			1..	1.1	1.1					..1			1..	1.1	10.		010
		..1	.1.	.11			1.1				..0	.0.	.00			10.			011
		..1			1..	1.1	1.1				..1			0..	0.1	01.			100
		1	.1.	.11			1.1				..0	.1.	.10			01.			101
				.11	1..		1.1	111					.01	0..		00.	001	110	
			.1.			1.1	1.1	111					.0.			0.0	00.	000	111

Рисунок 2.5 – Схема синтезу стандартної карти тестування

Оскільки адреси для функції n-змінних і тестів за формою рівні між собою $L=T$, то можна казати про декартовий квадрат хог-відстаней між адресами

таблиці істинності, який (квадрат) генерує константну і стандартну матрицю несправностей $F=A^2=A\oplus A$ будь-якої логічної функції. Ця F є константою, що формує хог-відношення між матрицями з метою синтезу тесту для логічних несправностей: $T=F\oplus L$. Але оскільки матриця F – це сукупність відстаней, то останнє рівняння перетворюється на переадресацію бітів логічної матриці активності $T= L_F$. Після цього кожна координата T -матриці стає вектором вхідних несправностей, що перевіряються, по 1-координатам адрес таблиці істинності несправностей, які інверсні до біт вхідного тестового набору. Тут використовується перетворення 1-координат T -матриці на двійкові коди перевірюваних на тестових наборах несправностей за формулою $T_{ij}=A_j^F \leftarrow T_{ij}=1$. Тут нульові біти T -матриці та неперевірені несправності в карті тестування позначаються точкою. Формула для визначення n -розрядних векторів несправностей у координатах карти тестування має такий вигляд

$$V_{ijt} = \bar{A}_{it}^T \leftarrow T_{ijt}=1, t=\overline{1, n}; i, j=\overline{1, 2^n}. \quad (2.8)$$

Фактично F -матриця є ключовим компонентом для вирішення всіх завдань тестування логічних функціональностей і не тільки [4-8].

Побудова карти тестування полягає у виконанні чотирьох матричних операцій:

- 1) синтез L -матриці на основі виконання хог-декартового твору над бітами логічного вектора цифрової функціональності;
- 2) генерація F -матриці шляхом взяття хог-добутку над явними двійковими адресами таблиці істинності логічного вектора;
- 3) виконання операції перекодування координат L -матриці за допомогою F -матриці, що дозволяє отримати дедуктивну T -матрицю;
- 4) отримання шаблону картки тестування для функціональності, представленої логічним Y -вектором на основі занесення по одиничним координатам T -матриці векторів одиничних значень адрес таблиці істинності;

5) до визначення одиничних координат у векторах таблиці істинності Т-матриці інверсними значеннями тестових вхідних наборів. Отримана карта тестування – це повний комбінаторний набір всіх можливих вхідних несправностей логічної функціональності, які перевіряються на вичерпному тесті.

Інший механізм синтезу карти тестування використовують F-матрицю для кодування одиничних координат L-матриці десятковими кодами несправностей (рис. 2.6) з метою отримання Т-матриці простим множенням $T=L \times F$. Потім одиниці вже двійкового коду десяткових чисел Т-матриці перетворюються на двійкові вектори несправностей карти тестування, знаки яких інверсні до біт тестового набору (правий стовпець карти тестування) або повторення бітів адрес таблиці несправностей. У цьому випадку несправності, що перевіряються, мають два жорсткі зв'язки в карті тестування. Вони прив'язані інверсією до біт тестового набору і до повторення біт адрес таблиці несправностей (верхній рядок матриці карти тестування). Позиції несправностей, що перевіряються в n-розрядному векторі визначаються одиничними координатами двійкового коду десяткового числа Т-матриці. Тут використовуються десяткові коди координат F-матриці

$$T_{ij}=(T_{ij})_2 \leftarrow T_{ij} \neq 0, \quad (2.9)$$

які перетворюються на двійкові коди несправностей, що перевіряються на тестових наборах. Формула для визначення n-розрядних векторів несправностей у координатах карти тестування має такий вигляд:

$$W_{ijt}=A_{jt}^F \text{ or } \bar{A}_{it}^T \leftarrow T_{ijt} = 1, t=\overline{1, n}; i, j=\overline{1, 2^n}. \quad (2.10)$$

Нульові біти в Т-матриці та неперевірені несправності в карті тестування позначаються точкою.

	L-matrix: $L_i=Y_i \oplus Y$								F-matrix								T-matrix Decimal							
Y	0	0	0	1	1	0	1	1	F= $A \oplus A$								T= $L \times F$							
0				1	1		1	1	0	1	2	3	4	5	6	7			3	4		6	7	
0				1	1		1	1	1	0	3	2	5	4	7	6			2	5		7	6	
0				1	1		1	1	2	3	0	1	6	7	4	5			1	6		4	5	
1	1	1	1			1			3	2	1	0	7	6	5	4	3	2	1		6			
1	1	1	1			1			4	5	6	7	0	1	2	3	4	5	6		1			
0				1	1		1	1	5	4	7	6	1	0	3	2			6	1		3	2	
1	1	1	1			1			6	7	4	5	2	3	0	1	6	7	4		3			
1	1	1	1			1			7	6	5	4	3	2	1	0	7	6	5		2			

T-matrix Binary								$W_{ijt}=(A_{jt}^F \text{ or } \overline{A}_{it}^T) \leftarrow T_{ijt} = 1$											
$T_{ij}=(T_{ij})_2 \leftarrow T_{ij} \neq 0$								A^F	000	001	010	011	100	101	110	111	A^T		
			.11	1..		11.	111					.11	1..		11.	111	000		
			.1.	1.1		111	11					.1.	1.0		110	11.	001		
			..1	11.		1..	1.1					..1	10.		1..	1.1	010		
.11	.1.	..1			11.				.00	.0.	..0			10.			011		
1..	1.1	1..			..1				0..	0.1	01.			..1			100		
			11.	..1		.11	.1.					01.	..0		.10	.1.	101		
1..	111	1..			.11				00.	001	0..			.01			110		
111	11.	1.1			.1.				000	00.	0.0			.0.			111		

Рисунок 2.6 – Схеми синтезу десяткової карти тестування

2.5 Розумні структури даних та векторна логіка їх обробки

Розумні структури даних [5, 9]: логічний вектор Y (Y_i – біт), таблиця істинності A (A_i – адреса, X_i – вхідний двійковий набір), матриця відстаней H , матриця активності L , дедуктивна матриця D , матриця несправностей (карта тестування) C . За логічним вектором або таблицею істинності можна моделювати вхідні (тестові) впливи для визначення виходу елемента (рис. 2.7).

Векторна логіка обробки розумних структур даних: $Y \oplus Y$ – синтез матриці активності, $A \oplus A$ – синтез матриці відстаней, L_H – синтез дедуктивної матриці, D_A – синтез матриці несправностей або карти тестування C .

Розв'язання задач design and test містить: вхідні дані – це логічний вектор від n -змінних; вихідні дані – це карта тестування (матриця несправностей) всіх

комбінацій логічних несправностей на вичерпному тесті. Розв’язок – це чотири логічні операції.

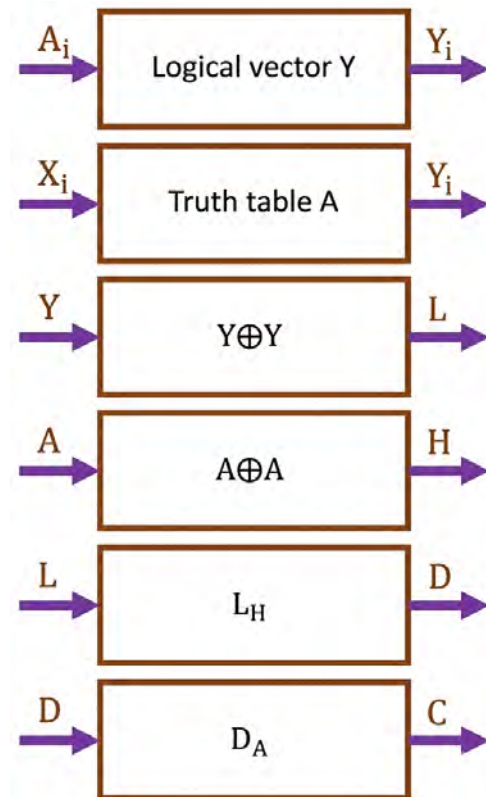


Рисунок 2.7 – Інжиніринг завдань тестування цифрових проєктів [рисунок сформовано автором та стилізовано за допомогою сервісу Gemini]

Використання цих елементарних процедур дає можливість здійснювати моделювання справної поведінки та несправностей. Використання логічного вектора дає можливість уникнути алгоритмів моделювання, отримуючи карту тестування шляхом суперпозиції розумних структур даних: логічний вектор, таблиця істинності, матриці дедукції (рис. 2.8).

L-matrix								H-matrix								D-matrix								C-matrix									
Y	1	1	1	0	0	1	1	1									000	001	010	011	100	101	110	111									
1				1	1				0	1	2	3	4	5	6	7	000				1	1				000				.11	1..		
1				1	1				1	0	3	2	5	4	7	6	001		1				1			001			.1.			1.0	
1				1	1				2	3	0	1	6	7	4	5	010	1						1		010		.1					10.
0	1	1	1			1	1	1	3	2	1	0	7	6	5	4	011	1	1	1	1	1	1	1		011		.0	.0.	.00	1..	1.0	10.
0	1	1	1			1	1	1	4	5	6	7	0	1	2	3	100	1	1	1	1	1	1	1		100		.1	.1.	.11	0..	0.1	01.
1				1	1				5	4	7	6	1	0	3	2	101	1						1		101		.0					01.
1				1	1				6	7	4	5	2	3	0	1	110		1				1			110			.0.			0.1	
1				1	1				7	6	5	4	3	2	1	0	111				1	1				111				.00	0..		

Рисунок 2.8 – Карти тестування логічної функції

Функціональні процедури можна розмістити в послідовності дій для побудови карти тестування, яка містить результат моделювання всіх можливих комбінацій несправностей на вичерпному тесті без моделювання. Використовується чотири матричні операції для побудови карти тестування (рис. 2.9).

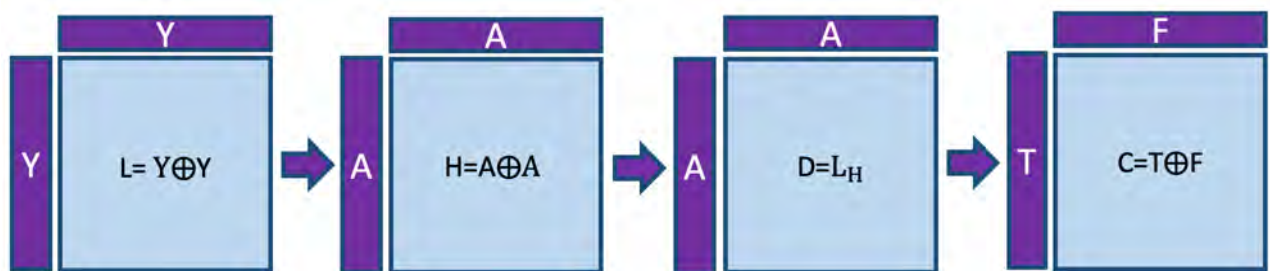


Рисунок 2.9 – Схеми чотирьох матричних операцій для побудови карти тестування

Тут розумні структури даних представлені: логічним вектором Y , адресами таблиці істинності для вектора Y , матрицею активності L , матрицею перекодування за допомогою стандартних адрес матриці активності, комбінацій логічних несправностей, що є картою тестування функціональності. Застосування чотирьох процедур у вигляді матричних операцій до логічного вектора Y , який є вхідною інформацією (запитами), дає можливість отримати карту тестування, синтез якої представлений на рис. 2.10.

L-matrix						H-matrix						D-matrix						C-matrix				
Y	0	1	1	1		A	00	01	10	11		A	00	01	10	11		A	00	01	10	11
0		1	1	1		00	0	1	2	3		00		1	1	1		00		.1	1.	11
1	1					01	1	0	3	2		01		1				01		.0		
1	1				→	10	2	3	0	1	→	10			1		→	10			0.	
1	1					11	3	2	1	0		11				1		11				00

Рисунок 2.10 – Синтез карти тестування для вектора 0111

Висновки. Дослідження полягає у паралельному моделюванні несправностей логіки на вичерпному тесті з нульовою складністю алгоритму. Іншими словами пропонується адресне моделювання несправностей логіки на інтелектуальних структурах даних без алгоритму моделювання. Побудова моделі розумних структур даних використовує чотири послідовні процедури синтезу наступних матриць, які формують рішення:

1) синтез логічної L-матриці шляхом взяття декартова $\oplus$ -квадрата на бітах логічного вектора від n-змінних за такою формулою:

$$L = Y \oplus Y = Y_{\oplus}^2; \quad (2.11)$$

2) побудова матриці перекодування шляхом взяття декартова $\oplus$ -квадрата на адресах таблиці істинності від n-змінних за такою формулою:

$$H = A \oplus A = A_{\oplus}^2. \quad (2.12)$$

Адреси виконують роль тестових наборів T та комбінацій логічних несправностей F. Отримана матриця є константою для всіх логічних функцій від n-змінних;

3) створення дедуктивної матриці шляхом переадресації координат логічної L-матриці на H-матриці перекодування за такою формулою: $D=L_H$. Або приведення адресації координат логічної матриці стандарту IEEE стандарту;

4) отримання карти тестування або матриці несправностей, що перевіряються на вичерпному тесті, шляхом виконання координатних операцій на 1-бітах вектор-адрес таблиці істинності в 1-координатах дедуктивної D-матриці за формулою:

$$C=A^1\bar{T} \text{ or } C=A^1\oplus T. \quad (2.13)$$

Знаки F несправностей, що перевіряються, входних змінних визначаються інверсією бітів тестових входних наборів T. Інакше ця операція по 1-координатам D-матриці виглядає компактно так

$$C=T\oplus F, \quad (2.14)$$

де з боку F в операції беруть участь тільки поодинокі координати адрес таблиці істинності несправностей.

Розглянуто нетрадиційне тестування логіки на основі використання логічної константи, представлені у вигляді матриці несправностей. Остання взаємодіє з логічною матрицею з метою отримання карти тестування у вигляді відношення вичерпного тесту та всіх комбінацій перевірюваних несправностей. Вирішуються задачі моделювання несправностей логічних функціональностей на вичерпному тесті без алгоритмів моделювання.

2.6 H-матриця як розумна структура даних

H-матриця [10, 11] – це розумні структури даних, що уявляє собою матрицю десяткових кодів двійкових відстаней між адресами однієї й тієї самої

таблиці істинності. Це явне завдання всіх комбінацій двійкових логічних несправностей у десяткових кодах, які перевіряються на вичерпному тесті. Вона є матрицею перекодування координат логічної матриці за IEEE-стандартом проходження адрес для отримання матриці дедукції. Вона також є способом завдання комбінацій несправностей всіх логічних функцій від n -змінних (бо немає перевірених несправностей для функцій тотожно рівних нулю чи одиниці). Така структура даних має чотири осі симетрії: по вертикалі, по горизонталі, по головній та побічній діагоналі. Чотири квадранти цієї матриці використовуються для організації попарної симетрії та рівності щодо її діагоналей. Структуру зв'язків рядків матриці подано на рис. 2.11, де наявні всі можливі переходи та переплутування на 2^n станах для обробки великих даних.

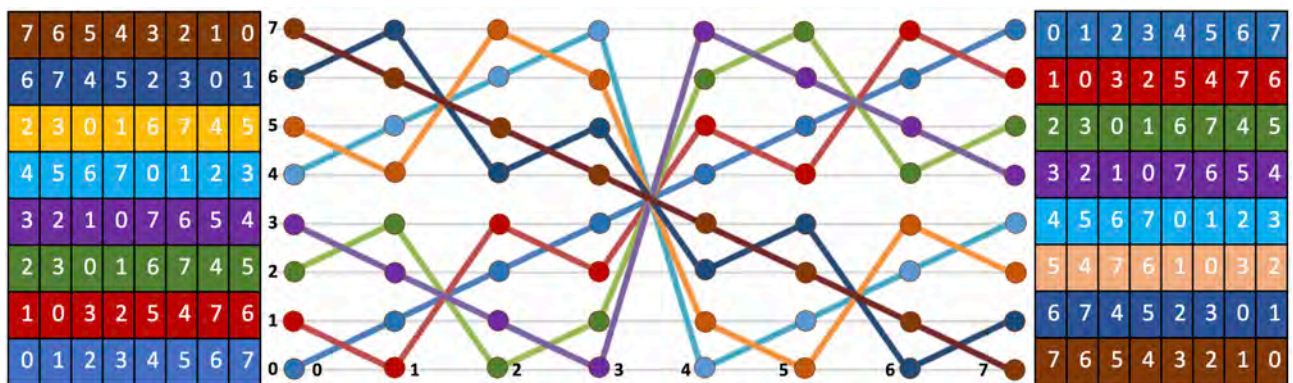


Рисунок 2.11 – Н-матриця відстаней

Н-матриця генерує всі можливі детерміновані похідні між сусідніми станами, які організують плавний та/або швидкий перехід між кодами. Такий механізм відношень між вичерпним тестом і несправностями, який дозволяє без моделювання записати в явному вигляді як карту тестування будь-якої складної логічної функціональності

$$F=L \times H \vee D \times H. \quad (2.15)$$

Це швидкий механізм переадресації координат логічних матриць для отримання дедуктивних матриць елементів $D=L_H$ з метою моделювання несправностей цифрових схем. Її можна інтерпретувати як матрицю відносин між тестами та несправностями будь-яких детермінованих оцифрованих об'єктів чи процесів. Логічна константа H , яка бере участь у всіх процесах тестування цифрових систем:

$$F=H\oplus T, T=H\oplus F. \quad (2.16)$$

Константа логіки H -матриця використовується для розв'язання задач тестування цифрових схем і функціональних логічних елементів. Вона є центральною ланкою у структурі векторно-логічного моделювання, яка представлена на рис. 2.12.

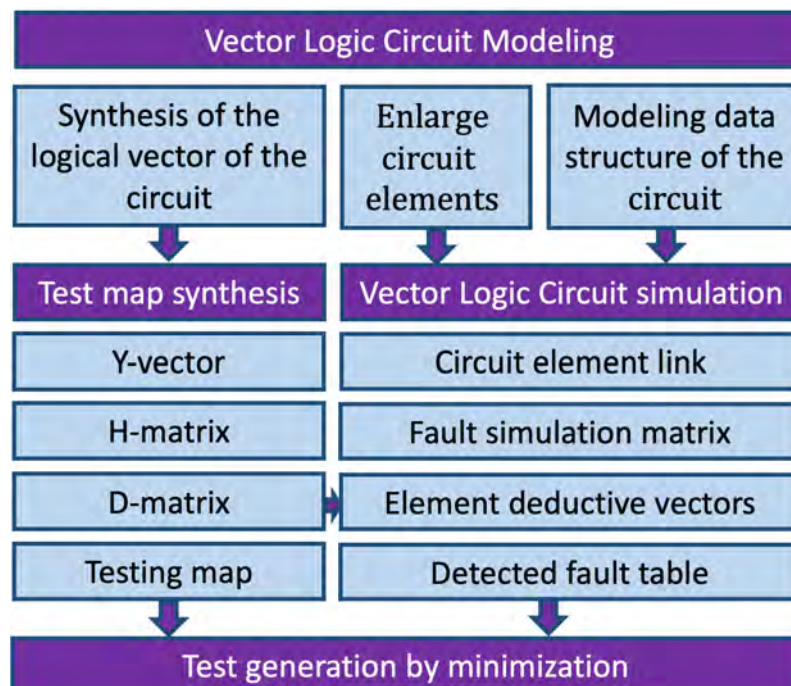


Рисунок 2.12 – Структура векторно-логічного моделювання елементів та симуляції схем

Н-матриця виконує роль моделі несправностей і механізму синтезу дедуктивних векторів. Йдеться про моделювання несправностей для синтезу тестів цифрових схем та обробки логічних елементів для побудови карт тестування на основі логічного вектора. Найшвидшим механізмом є побудова карти тестування для логічної функціональності. Тому якщо можна схему подати логічним вектором, це оптимальний варіант вирішення завдання тестування схемної структури. Найскладнішим варіантом є моделювання несправностей елементів схеми з урахуванням синтезу дедуктивних векторів, генерованих для тестових наборів. Моделювання несправностей схем або логічних елементів тут використовується для мінімізації тесту та оцінки якості покриття логічних несправностей.

Разом з логічною матрицею, яка є похідною від логічного вектора елемента, логічна константа бере участь у проектуванні розумних структур даних, які без моделювання дозволяють будувати карту тестування. Усі компоненти розумних структур даних похідні від логічного вектора. Усі задачі тестування вирішуються у двовимірному просторі, що організується на основі хог-операції над адресами таблиці істинності від n змінних.

2.6 Висновки до розділу 2

Таким чином, пропонується вирішувати задачі modeling, simulation and testing (рис. 2.13) за допомогою апарату векторно-логічного комп'ютингу шляхом істотного зменшення складності алгоритмів за рахунок надмірності розумних структур даних.

При цьому механізми моделювання та тестування реалізуються у пам'яті на основі read-write транзакцій без інструкції процесора. Це дозволяє реалізувати комп'ютинг тестування на енерго- та часозбережуваних механізмах, що дуже важливо для EDA-ринку. Для коректної взаємодії моделі L, T, F мають бути

представлені в одному форматі: логічного вектора, таблиці істинності чи дедуктивної матриці.



Рисунок 2.13 – Формалізація трьох процесів design and test комп'ютингу

Фактично виконується паралельне моделювання несправностей логіки на вичерпному тесті за константну складність алгоритму, завдяки чому вирішується задача: 1) моделювання несправностей і тестів, як адрес, розумних структур даних; 2) синтез карти тестування функціональностей з допомогою паралельно-матричних операцій з урахуванням логічного вектора; 3) побудова мінімізованого тесту та пошуку дефектів по карті тестування функціональності. Відмінність такого комп'ютингу полягає у зменшенні обчислювальної складності алгоритмів до нуля шляхом збільшення пам'яті для суперпозиції розумних та явних структур даних.

Логічний двійковий вектор розглядається як універсальна форма моделі уявлення процесів та явищ, структур та функціональностей. Логічний вектор на адресному просторі його бітів утворює таблицю істинності для n змінних, яка використовується як структури даних для вирішення комбінаторних завдань комп'ютингу. Адреса розглядається як двійковий n -розрядний вектор, що ідентифікує належність примітиву універсуму до кінцевого числа тестових n або

інших патернів. Тому адресний вектор використовується для кодування великих даних (несправностей) з метою обробки на таблиці істинності без програмування. На основі адресного кодування даних на таблиці істинності в паралельному режимі обробки n змінних вирішуються завдання: моделювання тестових наборів та несправностей, логічний аналіз графових структур, кластеризація великих даних, ідентифікація та розпізнавання патернів. На логічному векторі синтезується матриця відстаней між тестом та таблицею істинності, що формує матрицю активності логічної функціональності. Ці матриці зводять алгоритми моделювання несправностей та побудови тесту до разової паралельно-матричної процедури синтезу карти тестування логічної функціональності. При цьому зменшення складності алгоритмів синтезу відбувається за рахунок експоненціальних витрат пам'яті, необхідних для явної суперпозиції матриць і таблиць істинності.

Розглянуті моделі та методи тестування цифрових виробів орієнтовані на стандарт граничного сканування IEEE 1500 SECT [12], який зменшує складність обчислювального виробу завдяки розбиттю складної SoC на сукупність простих IP-core. Усі запропоновані моделі та алгоритми аналізу даних є інноваційними. Валідність отриманих результатів підтверджується обґрунтованим доказом основних теоретичних положень та численними експериментами на програмних додатках. Результати дослідження можуть бути корисними для фахівців та студентів, які займаються аналізом великих даних, а також розробкою, тестуванням та верифікацією цифрових виробів.

Новизна: розумні структури даних у вигляді логічних векторів та матриць (H) завдання комбінацій несправностей всіх логічних функцій від n -змінних.

Результати розділу опубліковані у роботах [14-16].

2.7 Перелік джерел посилання до розділу 2

1. Hahanov V. Cyber-Physical Computing для IoT-driven Services / V. Hahanov. – New York : Springer, 2018. – DOI:[10.1007/978-3-319-54825-8](https://doi.org/10.1007/978-3-319-54825-8)
2. Pretz K. The Maestro Behind Design-Software Behemoth Synopsys : CEO Aart de Geus створено логічний synthesis, який transformed how ICs are designed / Kathy Pretz // IEEE Spectrum. – Sep. 2023. – С. 52–53. – URL: <https://spectrum.ieee.org/aart-de-geus-profile> (дата звернення: 12.06.2026).
3. Hahanov V., Gharibi W., Chumachenko S., Litvinova E. Vector synthesis of fault testing map for logic / V. Hahanov, W. Gharibi, S. Chumachenko, E. Litvinova // International Journal of Robotics and Automation (IJRA). – 2024. – Vol. 13, No. 3. – P. 296–303. – DOI: <http://doi.org/10.11591/ijra.v13i3.pp293-306>.
4. Gharibi W., Hahanov V., Chumachenko S., Litvinova E., Hahanov I., Hahanova I. Vector-logic computing for faults-as-address deductive simulation / W. Gharibi, V. Hahanov, S. Chumachenko, E. Litvinova, I. Hahanov // IAES International Journal of Robotics and Automation (IJRA). – 2023. – Vol. 12, No. 3. – P. 274–288. – DOI: <http://dx.doi.org/10.11591/ijra.v12i3.pp274-288>.
5. Gharibi W., Hahanov V., Litvinova E., Hahanov I. Quantum Structures for Digital Systems Synthesis / W. Gharibi, V. Hahanov, E. Litvinova, I. Hahanov // Proceedings of IEEE East-West Design & Test Symposium (EWDTS-2015). – Batumi, 2015. – P. 115–122.
6. Hahanov V., Gharibi W., Chumachenko S., Litvinova E., Abdullayev V. H., Tariq Hama Salih T. H., Hahanov I. Vector-qubit models for SoC logic-structure testing and fault simulation / V. Hahanov, W. Gharibi, S. Chumachenko, E. Litvinova, V. H. Abdullayev, T. H. Tariq Hama Salih, I. Hahanov // 2021 IEEE 16th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), Lviv, Ukraine, 2021. – P. 24–28. – DOI: [10.1109/CADSM52681.2021.9385266](https://doi.org/10.1109/CADSM52681.2021.9385266).
7. Gharibi W., Hahanova A., Hahanov V., Chumachenko S., Litvinova E., Hahanov I. Vector-deductive memory-based transactions for fault-as-address

simulation / W. Gharibi, A. Hahanova, V. Hahanov, S. Chumachenko, E. Litvinova, I. Hahanov // Електронне моделювання. – 2023. – Т. 45, № 1. – С. 03–26. – DOI: <https://doi.org/10.15407/emodel.45.01>.

8. Gharibi W., Hahanova A., Hahanov V., Chumachenko S., Litvinova E., Hahanov I. Vector-Logic Synthesis of Deductive Matrices for Fault Simulation / W. Gharibi et al. // Електронне моделювання. – 2023. – Т. 45, № 2. – С. 16–33. – DOI: <https://doi.org/10.15407/emodel.45.02.016>.

9. Hahanov V., Chumachenko S., Litvinova Y., Hahanov I. I., Ponomarova V., Hahanova H., Kulak G. Faults-as-Address Simulation / V. Hahanov et al. // IAES International Journal of Robotics and Automation (IJRA). – 2024. – Vol. 13, No. 4. — P. 452–468. – DOI: <https://doi.org/10.11591/ijra.v13i4.pp452-468>.

10. Hahanov V., Chumachenko S., Litvinova Y., Hahanova I., Khakhanova A., Shkil A., Rakhlis D., Hahanov I., Shevchenko O. Vector-Logical Fault Simulation / V. Hahanov et al. // Radio Electronics, Computer Science, Control. – 2023. – № 2. – С. 37. – DOI: <https://doi.org/10.15588/1607-3274-2023-2-5>.

11. Hahanov V. et al. Logic for Bus Interconnect Testing / V. Hahanov et al. // 2025 IEEE East-West Design & Test Symposium (EWDTS), Tbilisi, Georgia, 2025. – P. 1–5. – DOI: 10.1109/EWDTS67441.2025.11303649.

12. IEEE Standard Testability Method for Embedded Core-based Integrated Circuits / IEEE Std 1500-2022 (Revision of IEEE Std 1500-2005). – 12 Oct. 2022. – P. 1–168. – DOI: 10.1109/IEEESTD.2022.9916221.

13. Hahanov V. Vector Logic Computing. / V. Hahanov. – New York : Springer, 2026. – ISBN 978-3-032-07001-2. – <https://doi.org/10.1007/978-3-032-07000-5> (Springer eBook, in Press). <https://link.springer.com/book/9783032070005>

14. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector logic for robotic system on chip design and test // IAES International Journal of Robotics and Automation (IJRA). – 2026. – Vol. 15, no. 2. – P. 415–426. – ISSN 2089-4856. – DOI: 10.11591/ijra.v15i2.pp415-426.

15. Хаханов В. І., Обрізан В. І., Рахліс Д., Хаханова Г. В., Хаханов І. В., Демченко О. І., Воронов А. О. Механізми схемотехнічного моделювання на

основі векторної логіки // Радіоелектроніка, інформатика, управління. – 2025. – № 4. – С. 219–232. – DOI: 10.15588/1607-3274-2025-4-20.

16. Hahanov V., Litvinova E., Chumachenko S., Hahanov I., Demchenko O., Voronov A. Vector Logic Modeling Self-Tested Circuits // 2025 IEEE East-West Design & Test Symposium (EWDTS), Tbilisi, Georgia, 12–15 Dec. 2025 : proceedings. – 2025. – 4 p. – DOI: 10.1109/EWDTS67441.2025.11303706.

3 МАТЕМАТИЧНИЙ АПАРАТ ДЕКАРТОВОЇ ТА ВЕКТОРНОЇ ЛОГІКИ ДЛЯ МЕХАНІЗМІВ МОДЕЛЮВАННЯ

3.1 Передмова

Як підкреслено вище, логічний вектор є найбільш технологічним, компактним та вичерпним представленням схеми для економного вирішення всіх задач дизайну та тестування. Розглядається декартова логіка, яка завдяки експоненційній надлишковості 2^{n+m} є ефективним інтелектуальним механізмом для вирішення комбінаційних задач (моделювання, simulation, тестування, діагностування) алгоритмами лінійної обчислювальної складності.

Пропонуються механізми побудови логічного вектора на основі матричних структур декартової логіки для вирішення задач Modeling for Simulation. Розглядається інженерний метод прямого паралельного good-value моделювання схеми на основі використання логічних векторів елементів та таблиць істинності.

Декартова логіка – це логічний вектор (матриця), як результат моделювання декартових логічних відношень між бітами логічних векторів або адресами таблиці істинності.

Декартова логіка вирішує задачі:

1. Good-value modeling circuit logic vector без алгоритму моделювання справної поведінки.
2. Fault modeling testing map of logic без алгоритму моделювання несправностей.

Пропонуються механізми створення карти тестування несправностей для структури шинних з'єднань SoC без використання алгоритмів моделювання.

3.2 Стан питання

Розгалуження в логічній схемі [1–4] породжують однойменні вхідні змінні (координати) на одному або кількох логічних елементах. Така особливість схеми призводить до мінімізації результуючого вектора (координат матриці) фрагмента структури за рахунок однойменних логічних змінних – шляхом залишення координат, які відповідають однаковим значенням вхідних сигналів (адресних бітів) цих змінних, та викреслення координат з різними вихідними сигналами. У результуючому векторі L завжди залишається кількість координат, що кратна степеню двійки (рис. 3.1).

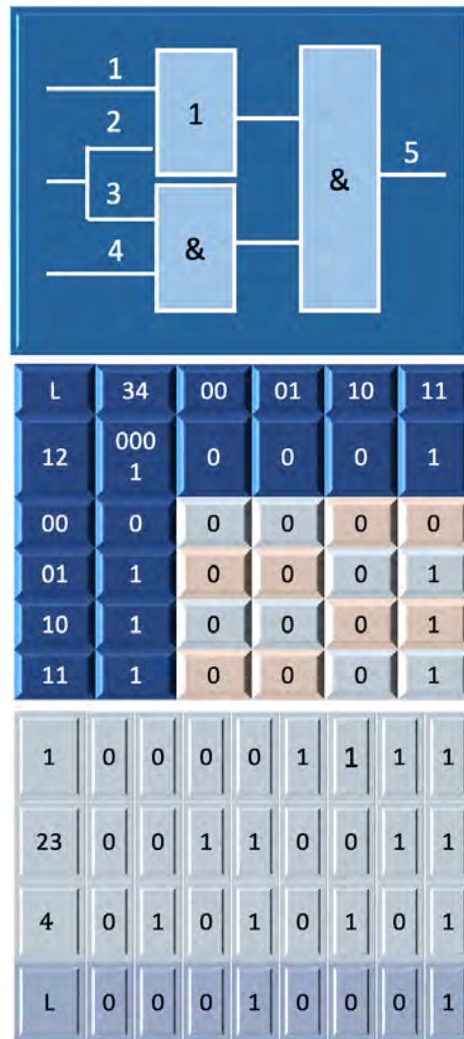


Рисунок 3.1 – Синтез логічного вектора для фрагмента схеми

Побудова результуючого логічного вектора двох взаємодіючих елементів на основі третього бінарного логічного вектора від двох змінних. Це універсальна процедура, вільна від інструкцій процесора, яка використовує лише read-write транзакції, орієнтовані на in-memory комп'ютинг. Тому при побудові карти тестування будь-якої логічної функціональності можна використовувати логічні вектори та read-write транзакції. Більше того, всі задачі design and test (як і будь-яке інше завдання) можна вирішувати на основі read-write транзакцій над логічними векторами без участі інструкцій процесора. Моделювання результуючого логічного вектора (у більше ніж двох вимірах) на основі третього логічного вектора від більш ніж двох змінних, застосовується для формування відношень між більш ніж двома векторами. Це паралельна процедура. Слід мати на увазі, що логічний вектор для будь-якої схеми може бути побудований шляхом рекурсивної суперпозиції двох векторів: поточного актуального вектора схеми та вектора наступного елемента. Побудову логічного вектора схеми можна розглядати як паралельне good-value пряме моделювання вичерпного тесту або вхідних наборів – це обов'язкова процедура при верифікації проєкту.

Якщо розглядати схему та її логічний вектор, то важко знайти пояснення, чому використовується саме схема замість логічного вектора для задання функціональності. Переваги схеми раніше визначалися як високошвидкісний механізм отримання значень на виходах при мінімальному обсязі пам'яті, яка тоді була дуже дорогою. Відомо, що read-write транзакції в пам'яті мають однакову затримку, що й логічні елементи: $\text{latency} = 0,6 \text{ нс}$. При цьому кожен користувач практично не обмежений у розмірах пам'яті, яку можна використати для розв'язання задачі. Тому можна стверджувати, що векторно-логічний комп'ютинг у пам'яті на основі read-write транзакцій є майбутнім для масових обчислень. Йому властиві мінімальні вартість створення моделі (яка не потребує синтезу), енергоспоживання, затримка під час експлуатації моделі. Компанії, що досліджують, розробляють та впроваджують штучний інтелект,

прагнуть перейти на прості двійкові моделі, розміщені в пам'яті, які подані логічними векторами або матрицями [5–10].

Схемотехніка потребує механізмів синтезу, які є гальмом на шляху створення інтелектуального in-memory комп'ютингу [11–13]. З іншого боку, існує альтернатива – будь-який процес або явище можна легко представити у вигляді векторів (патернів), двійково й унітарно закодованих на універсумі примітивних символів. При цьому синтез не потрібен, достатньо лише моделювання як процесу розміщення векторної або матричної моделі в пам'яті. Потім таку двійкову модель можна використати для розв'язання будь-яких задач штучного інтелекту методами логічного двійкового good-value моделювання на основі read-write транзакцій.

3.3 Векторна та декартова логіка

Логічний вектор – це представлення процесу або явища, функції чи структури у вигляді послідовності нулів і одиниць довжиною 2^n , де n – кількість бітів двійкових змінних для формування адрес бітів вектора. Це визначення робить логічний вектор незалежним від таблиці істинності. Водночас обов'язковим неявним атрибутом вектора є стандартні адреси, які легко згенерувати для будь-якого вектора. Декартовий добуток одного вектора з самим собою формує рефлексивне (рис. 3.2) відношення $L = Y \boxtimes Y$, яке генерує декартову матрицю – у цьому випадку матрицю активності або логічний вектор L розмірності 2^{n+1} .

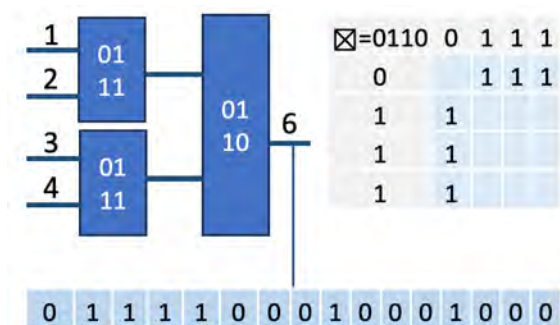


Рисунок 3.2 – Схема рефлексивного xor-відношення вектора 0111

Декартовий добуток двох різних векторів формує або синтезує новий вектор розмірності $2^{n+m} = 2^n \times 2^m$. Схемотехнічне позначення такої операції може бути представлене трьома елементами. Кожне розширення простору логічного вектора супроводжується появою нових властивостей, орієнтованих на практичне розв'язання задач аналізу.

Наприклад, матриця-вектор, що отримано у результаті декартової операції, вказує ті координати, які відрізняють два вектори (функціональності) – 1000 і 11011011 – між собою на вичерпному тесті (рис. 3.3).

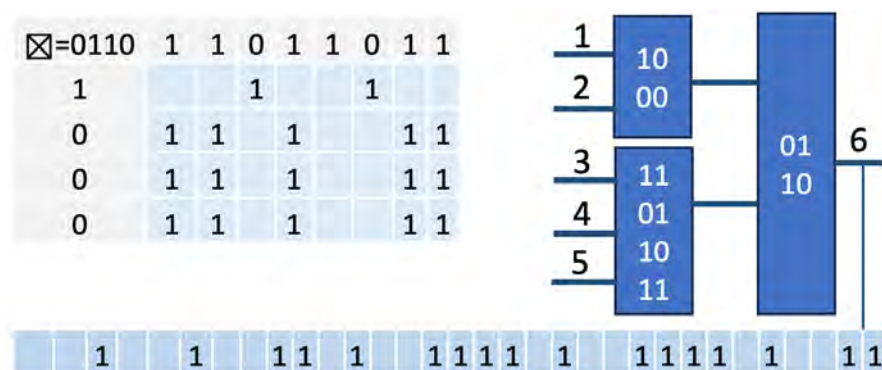


Рисунок 3.3 – Схема декартового добутку

Наведена схема є прикладом, що демонструє моделювання логічного вектора на основі векторів логічних елементів, які входять до складу схеми. Моделювання логічного вектора – це задача, актуальна для EDA-ринку, оскільки логічний вектор схеми – це найекономніша модель для розв'язання всіх задач design and test (і не тільки). Наведена схема є архітектурою (структурою) верифікації будь-якого цифрового проєкту на основі специфікації та актуальної моделі цифрового пристрою. Результат верифікації представлений логічним вектором, де одиничні координати показують невідповідність результатів тестування у просторі верифікації проєкту.

Векторна логіка подає логічні операції векторами станів таблиці істинності. Таблиця істинності має стандартизовані адреси бітів логічного

вектора вихідних станів, що дає змогу використовувати логічний вектор як самостійну модель, розміщену в пам'яті. У разі потреби для цієї векторної моделі завжди можна автоматично згенерувати адреси кожного біта або побудувати таблицю істинності. Генерація логічного вектора передбачає його розміщення в пам'яті, де доступ до кожного біта забезпечується архітектурою адресних дешифраторів.

Декартова логіка – це матриця відношень, побудована за допомогою логічної декартової операції ($\boxtimes$ -вектора) $L = X \boxtimes Y$ над бітами двох логічних векторів X і Y . Усі три вектори X , Y , $\boxtimes$ є логічними. Вектор $\boxtimes$ (G) виступає як вектор, що формує відношення між X та Y . І навпаки: будь-який з них (X , Y , G) може виступати як вектор, що встановлює відношення між двома іншими. Для формування декартового відношення необхідно дотримуватись таких умов для форматів векторів: $X = 2^n$, $Y = 2^m$, $G = 2^k$ (де n , m – будь-які, а $k = 2$).

Декартова логіка розглядається як розширення або надлишковість векторної логіки, яка дозволяє технологічно вирішувати ті задачі, які створюють проблеми для самої векторної логіки. Із теорії та практики відомо, що будь-яка надлишковість структур даних знижує обчислювальну складність алгоритму аналізу таких моделей. Оскільки модель і алгоритм утворюють єдиний механізм комп'ютингу, збільшення одного з компонентів призводить до зменшення іншого, і навпаки. Якщо необхідно розв'язати комбінаторну задачу лінійним алгоритмом, потрібно збільшити надлишковість моделі до експоненційної складності. Сьогодні виступає провідним трендом, коли інтелектуальні надлишкові моделі використовуються для вирішення всіх задач комп'ютингу лінійними або нульовими алгоритмами.

Прикладами практичного застосування декартової логіки для задач modeling for simulation виступають (рис. 3.4):

1	L-matrix: $L=Y\oplus Y$								2	H-matrix: $H=A\oplus A$							
Y	0	0	0	1	1	0	1	1	Y	000	001	010	011	100	101	110	111
0				1	1		1	1	000	0	1	2	3	4	5	6	7
0				1	1		1	1	001	1	0	3	2	5	4	7	6
0				1	1		1	1	010	2	3	0	1	6	7	4	5
1	1	1	1				1		011	3	2	1	0	7	6	5	4
1	1	1	1				1		100	4	5	6	7	0	1	2	3
0				1	1		1	1	101	5	4	7	6	1	0	3	2
1	1	1	1				1		110	6	7	4	5	2	3	0	1
1	1	1	1				1		111	7	6	5	4	3	2	1	0

3	D-matrix: $D=L_H$								4	L-matrix: $L=X\wedge Y$								
Y	T\L	000	001	010	011	100	101	110	111	Y	0	0	0	1	1	0	1	1
0	000				1	1		1	1	0								
0	001			1			1	1	1	0								
0	010		1			1	1	1		0								
1	011		1	1	1			1		1			1	1		1	1	
1	100		1			1	1	1		1			1	1		1	1	
0	101		1	1	1			1		0								
1	110				1	1		1	1	1			1	1		1	1	
1	111			1			1	1	1	1			1	1		1	1	

Рисунок 3.4 – Практичне використання розширеної декартової логіки

1. Моделювання L-матриці активності вихідних станів логічної функції Y: $L = Y \oplus Y$. Одиничні координати матриці визначають активність виходів при подачі вхідного набору, що задається адресами бітів логічного вектора.

2. Моделювання H-матриці перекодування $H = A \oplus A$, яка являє собою десяткові коди відстаней між адресами таблиці істинності будь-якої логічної функції. Це логічна константа для генерації всіх комбінацій несправностей.

3. Побудова матриці дедуктивних векторів $D = L_H$ для моделювання несправностей функціональності або схеми як адрес таблиці істинності.

4. Декартова AND-логіка для синтезу матриці, яка визначає властивість similarity (схожості) двох векторів у двовимірному просторі.

Побудова карти тестування схеми за вектором [14-16] – це один із потужних практичних аргументів на користь генерації логічного вектора, який

звільняє інженера від необхідності писати складні алгоритми синтезу тестів і моделювання несправностей (рис. 3.5).

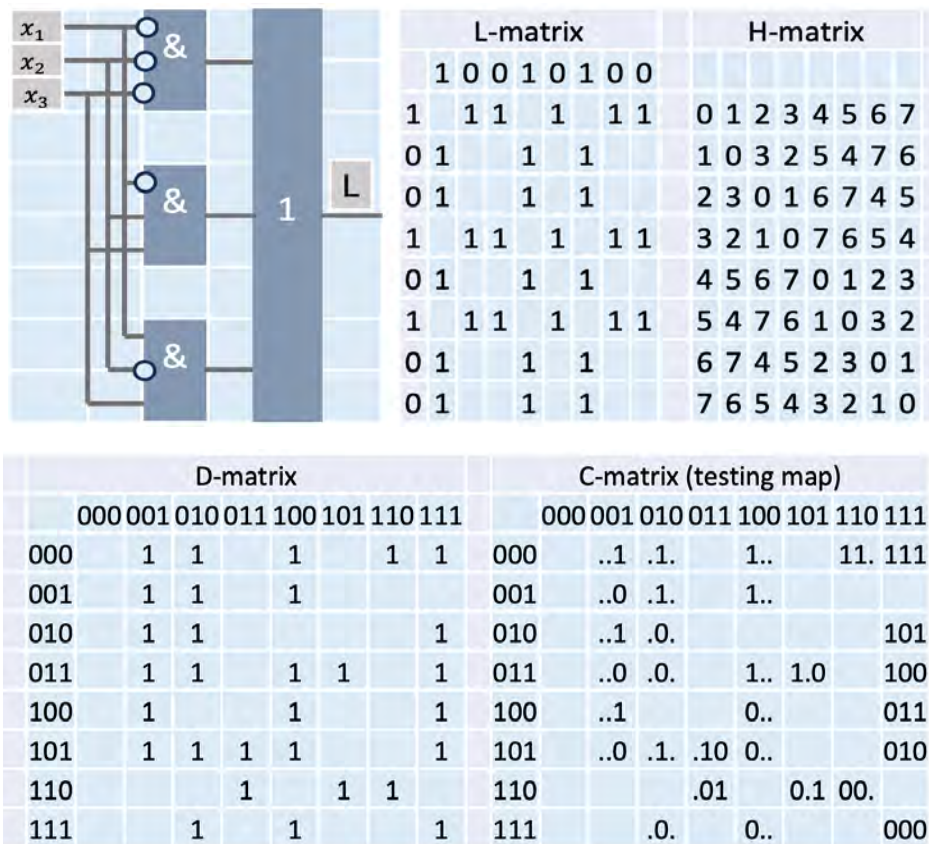


Рисунок 3.5 – Побудова карти тестування схеми за її логічним вектором

Механізм побудови карти тестування за логічним вектором 10010100 передбачає всього чотири матричні операції. Отриманий результат дозволяє вирішувати всі завдання: моделювання несправностей, тестування, верифікація, пошук і діагностування дефектів, оцінка якості тесту, вибір мінімального тесту. Константна складність матричних операцій на передбачуваному обсязі структур даних – це все, що потрібно для отримання карти тестування функціональності, заданої логічним вектором.

Можна розглядати дві метрики побудови карти тестування схеми:

1. Механізм моделювання несправностей на дедуктивних векторах елементів схеми:

$$C = k \times n \times 2^m \times n^2, \quad (3.1)$$

де k – довжина тесту, n – кількість логічних елементів у схемі, m – середня кількість входів в одному елементі, $n \times 2^m$ – складність генерації дедуктивних векторів, n^2 – складність моделювання несправностей лінії схеми на одному векторі.

2. Механізм моделювання карти тестування за логічним вектором схеми: $C = 3 \times 2^m$, де 3×2^m – довжина логічного вектора схеми, m – кількість зовнішніх змінних схеми, 3 – кількість матричних операцій для моделювання карти тестування.

3. Слід також враховувати складність алгоритмів і моделей, необхідних для програмування коду моделювання несправностей. У другому випадку код становить лише 100 операторів мови Python, у першому – близько 1000.

4. Варто враховувати алгоритмічну складність побудови логічного вектора схеми: $C = n \times 2^{2m}$, де n – кількість елементів у схемі, m – середня кількість входів у кожному елементі.

5. Інтегральна метрика складності побудови карти тестування для схеми через моделювання логічного вектора визначається як $C = n \times 2^{2m} + 3 \times 2^m$, що співмірно зі складністю алгоритму дедуктивного моделювання несправностей.

Практичний висновок: якщо специфікація функціональності задана логічним вектором, то немає потреби використовувати його для синтезу комбінаційної схеми, щоб потім моделювати цю схему задля отримання логічного вектора для подальшої верифікації функціональності. Слід безпосередньо використовувати логічний вектор специфікації в архітектурі інтегрованого комп'ютингу для тестування, верифікації, діагностування та експлуатації.

Карту тестування також можна отримати за одну декартову операцію над вектором логічної функції. Для цього необхідно виконати хог-операцію над бітами логічного вектора. Одиничні значення координат отриманої матриці

формують активність тестових наборів, на яких перевіряється комбінація несправностей, визначених одиничними координатами адрес таблиці істинності логічних несправностей. При цьому ознаки несправностей завжди будуть інверсними щодо бітів тестового набору. Синтез логічного вектора схеми передбачає на початковому етапі розбиття або декомпозицію багатовходових елементів на двовходові. Це дозволяє застосувати декартову логіку для отримання логічного вектора схеми або її фрагмента. Для регулярних функцій (and, or, xor) і їхніх похідних завжди можна побудувати таке розбиття.

Для довільної векторної логіки таке розбиття також завжди існує (рис. 3.6), оскільки будь-який логічний вектор може бути представлений у вигляді диз'юнктивної нормальної форми (ДНФ) або кон'юнктивної нормальної форми (КНФ), у яких використовуються регулярні логічні операції and, or, not.

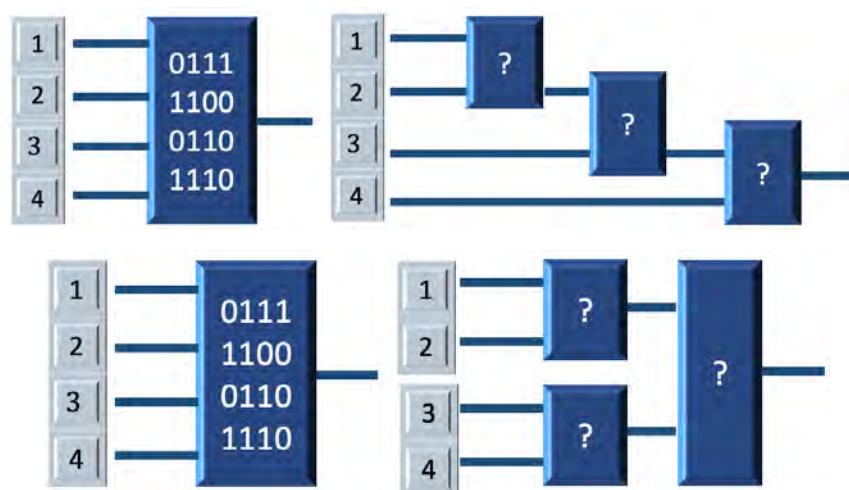


Рисунок 3.6 – Схема розбиття складних елементів на двовходові

У загальному випадку слід використовувати такий шаблон побудови логічного вектора за двома іншими векторами будь-якої розмірності, кратної 2^n , тобто ступеню двійки, та логічною функцією від двох змінних, яка є утворювальною (рис. 3.7). Можна стверджувати, що завжди формується

логічний вектор з кількістю бітів 2^{n+m} , де n і m – довжини двох векторів, які утворюють матричне декартове відношення.

$\boxtimes$	1	0	0	1	0	1	1	0
0								
1								
1								
0								

$\boxtimes=0111$	1	0	0	1	0	1	1	0
0	1			1		1	1	
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
0	1			1		1	1	

$\boxtimes=0110$	1	0	0	1	0	1	1	0
0	1			1		1	1	
1		1	1		1			1
1		1	1		1			1
0	1			1		1	1	

Рисунок 3.7 – Побудова логічного вектора-матриці фрагмента схеми на основі декартової логіки

3.4 Моделювання логічного вектора для цифрової структури на основі декартової логіки

Логічний вектор процесу або явища дозволяє розв’язувати всі задачі комп’ютингу в економному режимі за лінійний час виконання алгоритму. Тому надзвичайно важливо також будувати логічний вектор для цифрової схеми. Для цього *використовуємо рекурсивний алгоритм на основі декартової логіки*. Це метод отримання матриці логічного вектора L схеми шляхом декартового добутку бітів двох векторів $X \boxtimes Y$ за правилом логічної функції $G(\boxtimes)$. При цьому Y – це сукупний вектор уже оброблених елементів схеми, а X – це вектор

чергового елемента схеми, який підлягає обробці. У результаті фактично отримується логічний вектор схеми на повному вичерпному тесті, який можна інтерпретувати як суперпаралельне good-value моделювання знімної структури.

Декомпозиція логічного вектора (рис. 3.8) здійснюється на основі рівняння його синтезу

$$L = X \boxtimes Y. \quad (3.2)$$

Необхідно знати два компоненти рівняння: L та $\boxtimes$. Потрібно визначити вектори X і Y , які перебувають між собою у відношенні $\boxtimes$, де X і Y повинні відповідати певним співвідношенням довжин, що важливо для формування формату матриці вектора. Для визначення векторів X та Y працює правило:

$$X \boxtimes L_{ij} = Y. \quad (3.3)$$

У цьому процесі також може бути використана модель штучного інтелекту.

$\boxtimes=0001$	1	0	0	1	0	1	1	0
0								
1	1			1		1	1	
1	1			1		1	1	
0								
$L = X \boxtimes Y$								

Рисунок 3.8 – Декомпозиція логічного вектора

Для декомпозиції вектора немонотонної функції необхідно представити її у вигляді матриці на основі відомого утворювального G -вектора (рис. 3.9).

Initial form for decomposition								
$\boxtimes=0001$								
00=0								
01=0		1			1		1	1
10=0		1			1		1	1
11=1								
$L=X\boxtimes Y \rightarrow 1=1\boxtimes 1$								

Resulting decomposition								
$\boxtimes=0001$	1	0	0	1	0	1	1	0
0								
1	1			1		1	1	
1	1			1		1	1	
0								
$L=X\boxtimes Y$								

Initial form for decomposition								
$\boxtimes=0111$								
00=0		1			1		1	1
01=1		1	1	1	1	1	1	1
10=1		1	1	1	1	1	1	1
11=1		1			1		1	1
$L=X\boxtimes Y \rightarrow 0=0\boxtimes 0$								

Resulting decomposition								
$\boxtimes=0111$	1	0	0	1	0	1	1	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			1		1	1	
$L=X\boxtimes Y$								

Рисунок 3.9 – Декомпозиції логічних векторів

Мета цієї процедури – знайти два вектори, які формують матрицю шляхом конкатенації бітів двох векторів, які потрібно визначити. Конкатенація двох одиниць утворює адресу 11, яка формує всі одиничні координати в матриці логічного вектора. В іншому випадку – два нулі в адресі функціональності формують усі нульові координати матриці логічного вектора. У загальному випадку завдання формулюється так: знайти два логічні вектори, які на основі декартової логіки бітів генерують інтегральний вектор у формі матриці. Існує два умови для розв’язання цього завдання:

1) утворювальна функція відома. Алгоритмічна складність розв’язання задачі $A = 2^2 \times 2^n$;

2) якщо утворювальна функція невідома (рис. 3.10), то алгоритмічна складність розв’язання задачі становить: $A = 2^2 \times 2^2 \times 2^{n-2} = 2^4 \times 2^{n-2}$. Це завдання вирішується повним перебором усіх можливих підстановок усіх 16 функцій від двох змінних, за винятком двох тривіальних функцій: 0000 (завжди 0) і 1111 (завжди 1).

Initial form for decomposition								
$\boxtimes = 2^4 \times 2^{n-2}$								
00=0...1		1			1		1	1
01=0...1		1	1	1	1	1	1	1
10=0...1		1	1	1	1	1	1	1
11=1...0		1			1		1	1
$L = X \boxtimes Y$								

Рисунок 3.10 – Декомпозиція логічного вектора на основі перебору

Може існувати двійковий вектор, який неможливо отримати шляхом декартової суперпозиції двох векторів. У такому випадку залишається лише один шлях декомпозиції логічного вектора – представлення логічного вектора у вигляді диз'юнктивної нормальної форми (ДНФ) за одиничними координатами вектора. Наприклад, для вектора 10010100 розклад у вигляді ДНФ матиме вигляд (рис. 3.11):

$$L = \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 x_1 x_2 \vee x_1 \bar{x}_2 x_2. \quad (3.4)$$

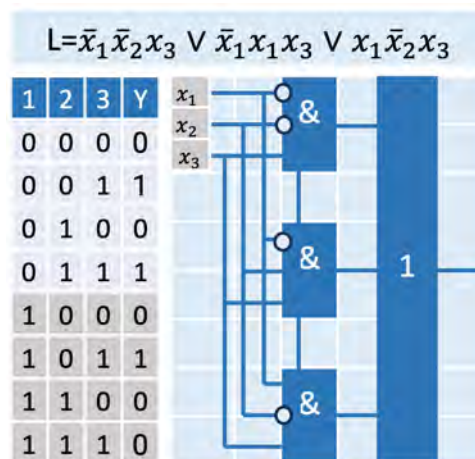


Рисунок 3.11 – Декомпозиції логічного вектора шляхом синтезу диз'юнктивної нормальної форми (ДНФ)

Кожен елемент схеми тут представлений логічним вектором монотонної логічної функції (and, or) з базису Поста. Тому немає труднощів у побудові логічного вектора цього фрагмента схеми, навіть якщо в ньому буде $2^n - 1$ одиничних значень бітів. Монотонна функція визначається логічним вектором, який має лише один перехід з одиниці в нуль або з нуля в одиницю в таблиці істинності.

Логічний вектор можна розділити на два за допомогою схеми комутації. Мета цієї процедури – отримати схемну структуру для немонотонного логічного вектора, що складається з двовходових елементів. Це цілком передбачуваний алгоритм, який працює для будь-якого нерегулярного вектора (рис. 3.12). Недолік такої процедури полягає в тому, що в результаті виникає певна надлишковість у вигляді схеми комутації, де елементи мають більше ніж два входи. Ця схема обслуговує розподіл вектора 11010101 на дві рівновеликі частини.

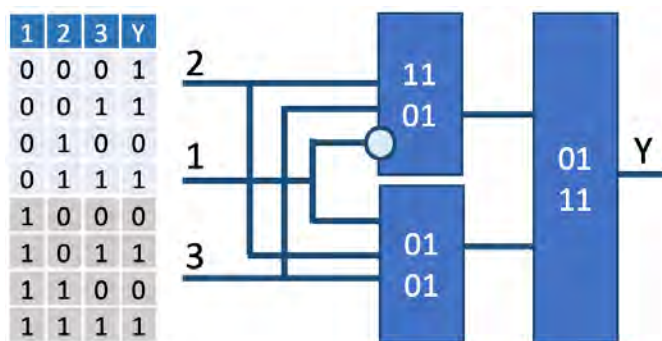


Рисунок 3.12 – Розподіл логічного вектора на два з використанням комутатора

3.5 Моделювання логічного вектора схеми шляхом good-value simulation (механізм good-value simulation цифрової схеми для синтезу логічного вектора)

Моделювання всіх 2^n вхідних наборів. Процедура є ефективною та технологічно дуже простою. Для обробки елементів схеми, представлених

логічними векторами L , розміщеними в пам'яті, використовується єдина формула:

$$M_i = L[M(X_i)]. \quad (3.5)$$

Для аналізу наведених структур даних схеми застосовуються read-write транзакції in-memory computing над бітами логічних векторів. Щоб визначити вихід оброблюваного елемента, необхідно сформувати вхідний двійковий вектор $M(X)$ відповідно до станів координат вектора моделювання M , номери яких вказані в полі Inputs (рис. 3.13).

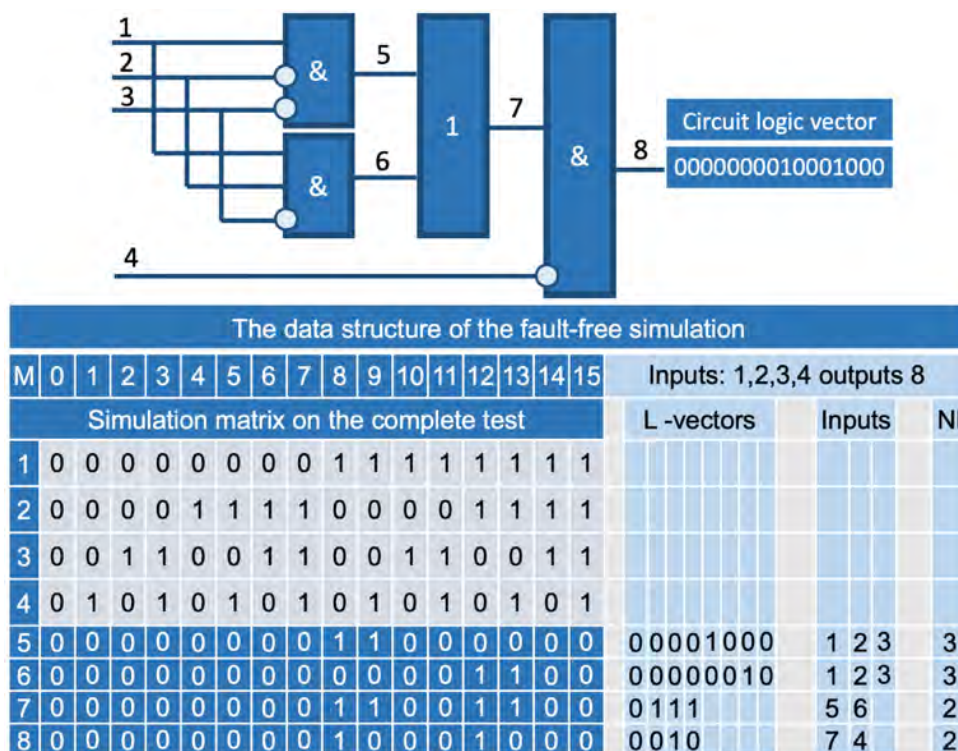


Рисунок 3.13 – Структура даних безпомилкового моделювання (fault-free simulation) схеми

Поле NI визначає кількість входів у кожному елементі. Після цього вхідний двійковий вектор $M(X)$ розглядається як адреса бітової комірки відповідного логічного вектора з поля L-vectors, яка формує вихідний стан

поточного елемента. Це моделювання дозволяє отримати всі стани ліній схеми на вичерпному тесті, включаючи лінію з виходом «вісім», на якій формується логічний вектор усієї схеми.

Моделювання логічного вектора схеми шляхом good-value simulation дозволяє отримати карту тестування всіх комбінацій вхідних несправностей (рис. 3.14) усього за три матричні операції [16].

	L-matrix													Testing map																
	0	0	0	0	0	0	0	0	1	0	0	0		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
0									1				1	0000								1...					11..			
0									1				1	0001									1..0					11.0		
0									1				1	0010										1.0.					110.	
0									1				1	0011											1.00				1100	
0									1				1	0100								1...				10..				
0									1				1	0101									1..0				10.0			
0									1				1	0110										1.0.				100.		
0									1				1	0111											1.00				1000	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1000	...	1	..1	..11		..1.1	..11.	..111	0...	0..1	0.1.	0.11	01..	01.1	011.	0111
0									1				1	1001	...	0				..1.0										
0									1				1	1010			..0.				..10.									
0									1				1	1011				..00				..100								
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1100	...	1	..1	..11		..0.1	..01.	..011	0...	0..1	0.1.	0.11	00..	00.1	001.	0011
0									1				1	1101	...	0				..0.0										
0									1				1	1110			..0.				..00.									
0									1				1	1111				..00				..000								

Рисунок 3.14 – Моделювання карти тестування за логічним вектором

0000000010001000 для схеми

Моделювання несправностей на основі схемної структури на порядок складніше, ніж good-value simulation. Саме цим пояснюється той факт, що під час верифікації цифрових проєктів зазвичай використовується лише good-value simulation схеми. Однак дедуктивне векторно-логічне моделювання несправностей [17] практично не поступається за обчислювальною складністю алгоритму good-value simulation.

Тут представлено всі комбінації логічних несправностей вхідних змінних комбінаційної схеми, що перевіряються на вичерпному тесті. Це приклад prompt-комп'ютингу, коли вся діагностична інформація для верифікації проєкту отримується на основі введення логічного вектора схеми. Складність

алгоритмів моделювання є лінійною завдяки експоненційній надлишковості структур даних, представлених логічними векторами, таблицями істинності та матрицями декартової логіки.

3.6 Метод декартової суперпозиції логічних векторів

Метод моделювання справної роботи для отримання логічного вектора має один суттєвий недолік: необхідність моделювання повного (вичерпного) тесту довжиною 2^n , де n – кількість вхідних змінних, для всіх елементів схеми. Більш оптимальним і мінімальним є метод, заснований на суперпозиції векторів за допомогою декартової логіки. За один прохід по всіх елементах схеми формується логічний вектор, який надалі можна використати для побудови карти тестування.

Розглянемо основні етапи алгоритму побудови логічного вектора схеми на основі декартової логіки на прикладі комбінаційної схеми (рис. 3.15).

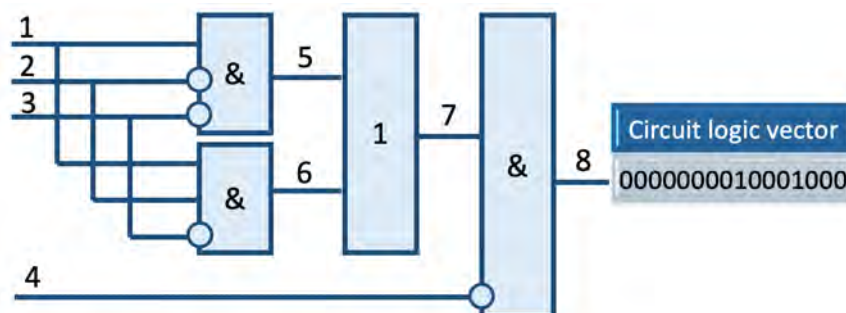


Рисунок 3.15 – Логічна схема зі розгалуженнями, що збігаються

Моделювання логічного вектора схеми методом декартової суперпозиції векторів (рис. 3.16):

12-5		0	1
	0001	1	0
	0	0	0
1	1	1	0

12-6		0	1
	0001	0	1
	0	0	0
1	1	0	1

56-7		000	001	010	011	100	101	110	111	
	0111	0	0	0	0	0	0	1	0	
	000	0	0	0	0	0	0	1	0	
	001	0	0	0	0	0	0	1	0	
	010	0	0	0	0	0	0	1	0	
	011	0	0	0	0	0	0	1	0	
	100	1	1	1	1	1	1	1	1	
	101	0	0	0	0	0	0	0	1	0
	110	0	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1	0	

74-8		0	1	
	0001	1	0	
	000	0	0	0
	001	0	0	0
	010	0	0	0
	011	0	0	0
	100	1	1	0
	101	0	0	0
	110	1	1	0
111	0	0	0	

123-5		0	1	
	0001	1	0	
	00	0	0	0
	01	0	0	0
	10	1	1	0
11	0	0	0	

123-6		0	1	
	0001	1	0	
	00	0	0	0
	01	0	0	0
	10	0	0	0
11	1	1	0	

Рисунок 3.16 – Отримання логічного вектора схеми на основі декартової логіки

1. Умовна розбивка всіх елементів на двовходові для спрощення процедури моделювання та уніфікування обробки елементів.

2. Побудова логічного вектора кожного двовходового елемента за допомогою декартової логіки.

3. Моделювання матриці двох векторів за допомогою утворювального вектора довжиною чотири біти. Кожна таблиця для обробки елемента представлена двома логічними векторами (вхідними), адресами бітів, утворювальним логічним вектором, які використовуються для виконання декартового добутку.

4. Видалення координат матриці, що суперечать вхідним наборам. Для елемента з виходом 7 залишаються лише координати головної діагоналі; решта вважаються суперечливими за адресами вхідних бітів.

5. Перевірка отриманого логічного вектора схеми за допомогою good-value simulation.

6. Отримання логічного вектора схеми: 0000000010001000.

3.7 Обговорення результатів та висновки

Особливості методу декартової суперпозиції логічних векторів полягають у наступному:

1) необхідно використовувати адреси таблиці істинності або біти логічного вектора для мінімізації кількості координат, які формують вектор чергового оброблюваного елемента;

2) якщо логічна схема не має східних розгалужень, то ця процедура мінімізації не застосовується;

3) недоліком методу є необхідність виконання процедури розбиття елемента на двовходові для їх рекурсивної обробки з метою побудови логічного вектора схеми;

4) метод є технологічно зручним для написання програмного коду, якщо всі елементи схеми – двовходові;

5) якщо не розбивати елемент на двовходові піделементи, то потрібно використовувати утворювальні логічні вектори довжиною більше ніж 4 біти.

Якщо специфікація функціональності задана логічним вектором, то відсутня потреба у синтезі комбінаційної схеми та подальшому моделюванні її, щоб знову отримати логічний вектор для верифікації. Замість цього логічний вектор специфікації слід безпосередньо використовувати в архітектурі in-memory комп'ютингу для тестування, верифікації, діагностування та експлуатації.

На основі теорії векторної логіки розроблено механізми та програмне забезпечення для економічного розв'язання задач design and test в архітектурі in-memory комп'ютингу (розділ 4).

Математичний апарат декартової логіки представлений у вигляді логічного вектора (матриці), що є результатом моделювання декартових логічних відношень між бітами логічних векторів або адресами таблиці істинності. Декартова логіка вирішує задачі: 1) Good-value modeling circuit logic vector – без алгоритму моделювання справної поведінки. 2) Fault modeling testing map of logic – без алгоритму моделювання несправностей.

Моделювання логічного вектора схеми дає змогу вдвічі зменшити час моделювання несправностей у порівнянні з моделюванням схеми, поданої у

вигляді логічних елементів. Моделювання займає лише 25% часу, необхідного для моделювання несправності схеми на основі логічного вектора.

Описано механізм *good-value simulation* цифрової схеми для синтезу логічного вектора з подальшим його використанням для побудови карти тестування несправностей на повному тесті лише за три матричні операції. Цей in-memory механізм є простим у реалізації, використовує лише read-write транзакції над бітами логічних векторів і не потребує інструкцій процесора.

Новизна дослідження полягає у використанні векторної логіки [14-17] в архітектурі in-memory комп'ютингу на основі read-write транзакцій, що дозволяє зменшити споживання ресурсів у вигляді часу та енергії.

Результати розділу відображено у роботах автора [18, 20].

3.8 Перелік джерел посилання до розділу 3

1. IEEE Std 1481-2009. Integrated Circuit (IC) Open Library Architecture (OLA) : IEEE Std 1481-2009. [Effective from 2010-03-11]. New York : IEEE, 2010. 658 p.

2. Esmaili E., Sedaghat Y., Peiravi A. Fanout-Based Reliability Model for SER Estimation in Combinational Circuits / E. Esmaili, Y. Sedaghat, A. Peiravi // IEEE Transactions on Circuits and Systems I: Regular Papers. 2025. Vol. 72, no. 1. P. 228–240. DOI: 10.1109/TCSI.2024.3458864.

3. Hwang M.-E., Jung S.-O., Roy K. Slope Interconnect Effort: Gate-Interconnect Interdependent Delay Modeling for Early CMOS Circuit Simulation / M.-E. Hwang, S.-O. Jung, K. Roy // IEEE Transactions on Circuits and Systems I: Regular Papers. 2009. Vol. 56, no. 7. P. 1428–1441. DOI: 10.1109/TCSI.2008.2006217.

4. Huang W., Du J., Hua W. et al. A Hybrid Model-Based Diagnosis Approach for Open-Switch Faults in PMSM Drives / W. Huang, J. Du, W. Hua et al. // IEEE Transactions on Power Electronics. 2022. Vol. 37, no. 4. P. 3728–3732. DOI: 10.1109/TPEL.2021.3123144.

5. Fu R., Wille R., Yoshikawa N. et al. Efficient Cartesian Genetic Programming-Based Automatic Synthesis Framework for Reversible Quantum-Flux-Parametron Logic Circuits / R. Fu, R. Wille, N. Yoshikawa et al. // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. DOI: 10.1109/TCAD.2025.3546884.

6. Berndt A. Accuracy and Size Trade-off of a Cartesian Genetic Programming Flow for Logic Optimization / A. Berndt et al. // 2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI), Campinas, Brazil, 2021 : proceedings. 2021. P. 1–6. DOI: 10.1109/SBCCI53441.2021.9529968.

7. Dunaeva O., Autsou S., Kudelina K. et al. Utilizing the fuzzy logic algorithm for cartesian robot control system in conditions of mechanical damage transmission / O. Dunaeva, S. Autsou, K. Kudelina et al. // IECON 2024 – 50th Annual Conference of the IEEE Industrial Electronics Society, Chicago, IL, USA, 2024 : proceedings. 2024. P. 1–5. DOI: 10.1109/IECON55916.2024.10905445.

8. Khan G. M., Zafari F., Mahmud S. A. Very Short-Term Load Forecasting Using Cartesian Genetic Programming Evolved Recurrent Neural Networks (CGPRNN) / G. M. Khan, F. Zafari, S. A. Mahmud // 2013 12th International Conference on Machine Learning and Applications, Miami, FL, USA, 2013 : proceedings. 2013. P. 152–155. DOI: 10.1109/ICMLA.2013.181.

9. Inglese P., Vatajelu E.-I., Di Natale G. Side Channel and Fault Analyses on Memristor-Based Logic In-Memory / P. Inglese, E.-I. Vatajelu, G. Di Natale // IEEE Design & Test. 2024. Vol. 41, no. 3. P. 29–35. DOI: 10.1109/MDAT.2023.3324522.

10. Ye J., Zhu J., Cao J. et al. A Novel Parallel In-Memory Logic Array Based on Programmable Diodes / J. Ye, J. Zhu, J. Cao et al. // IEEE Journal of the Electron Devices Society. 2024. Vol. 12. P. 738–744. DOI: 10.1109/JEDS.2024.3457021.

11. Chen X., Song T., Han Y. RRAM-based Analog In-Memory Computing: Invited Paper / X. Chen, T. Song, Y. Han // 2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), AB, Canada, 2021 : proceedings. 2021. P. 1–6. DOI: 10.1109/NANOARCH53687.2021.9642235.

12. Nguyen V.-T., Trinh Q.-K., Zhang R. et al. XNOR-BSNN: In-Memory Computing Model for Deep Binarized Spiking Neural Network / V.-T. Nguyen, Q.-K. Trinh, R. Zhang et al. // 2021 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS), Macau, China, 2021 : proceedings. 2021. P. 17–21. DOI: 10.1109/HPBDIS53214.2021.9658467.

13. Hahanov V., Litvinova E., Davitadze Z. et al. Truth Table Based Intelligent Computing / V. Hahanov, E. Litvinova, Z. Davitadze et al. // 2024 31st International Conference on Mixed Design of Integrated Circuits and System (MIXDES), Gdansk, Poland, 2024 : proceedings. 2024. P. 199–204. DOI: 10.23919/MIXDES62605.2024.10614035.

14. Hahanov V., Gharibi W., Chumachenko S. et al. Vector Synthesis of Fault Testing Map For Logic / V. Hahanov, W. Gharibi, S. Chumachenko et al. // IAES International Journal of Robotics and Automation (IJRA). 2024. Vol. 13, no. 3. P. 293–306. DOI: 10.11591/ijra.v13i3.pp293-306.

15. Hahanov V., Litvinova E., Hahanova H. et al. Vector-Logical In-Memory Simulation of Faults as Truth Table Addresses / V. Hahanov, E. Litvinova, H. Hahanova et al. // 2024 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, Armenia, 2024 : proceedings. 2024. P. 1–6. DOI: 10.1109/EWDTS63723.2024.10873615.

16. Hahanov V., Devadze D., Hahanov I. et al. Prompt-Testing of Logic / V. Hahanov, D. Devadze, I. Hahanov et al. // 2024 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, Armenia, 2024 : proceedings. 2024. P. 1–5. DOI: 10.1109/EWDTS63723.2024.10873774.

17. Hahanov V., Chumachenko S., Litvinova E. et al. Faults-as-address simulation / V. Hahanov, S. Chumachenko, E. Litvinova et al. // IAES International Journal of Robotics and Automation. 2024. Vol. 13, no. 4. P. 452–468. DOI: 10.11591/ijra.v13i4.pp452-468.

18. Хаханов В.І., Обрізан В.І., Рахліс Д., Хаханова Г.В., Хаханов І.В., Демченко О.І., Воронов А.О. Механізми схемотехнічного моделювання на основі векторної логіки / В.І. Хаханов, В.І. Обрізан, Д. Рахліс, Г.В. Хаханова,

I.B. Хаханов, О.І. Демченко, А.О. Воронов // Радіоелектроніка, інформатика, управління. – 2025. – № 4. – С. 219–232. – DOI: <https://doi.org/10.15588/1607-3274-2025-4-20>.

19. Vector Simulation Model [Електронний ресурс] : <https://vector-simulation-model.vercel.app/> (доступ 09.06.2026).

20. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector logic for robotic system on chip design and test / V. Hahanov, S. Chumachenko, E. Litvinova, A. Voronov, O. Demchenko, N. Maksymova // IAES International Journal of Robotics and Automation (IJRA). – 2026. – Vol. 15, no. 2. – P. 415–426. – ISSN 2089-4856. – DOI: [10.11591/ijra.v15i2.pp415-426](https://doi.org/10.11591/ijra.v15i2.pp415-426).

4 ВЕРИФІКАЦІЯ СТРУКТУР ДАНИХ, МЕХАНІЗМІВ, АЛГОРИТМІВ. ХМАРНИЙ СЕРВІС MOSI–HDL ДЛЯ МОДЕЛЮВАННЯ ТА ВІЗУАЛІЗАЦІЇ

4.1 Передмова

Розробляється хмарний сервіс для економічного вирішення задач MOdelling for SImulation (MOSI) на основі векторно–логічного in–memory комп'ютингу, орієнтованого на використання read–write транзакцій без інструкцій процесора.

Розробляється парсер–механізм для конвертації HDL коду логічної схеми у внутрішні векторно–логічні структури даних сервісу MOSI. Здійснюється modeling векторної логічної моделі цифрової схеми для good–value simulation вхідних тестових наборів, як адрес логічних векторів елементів. Генеруються дедуктивні вектори за векторно–логічною моделлю цифрової схеми для fault as address simulation вхідних тестових наборів. Створюється механізм modelling матриці моделювання несправностей, як бітів адрес дедуктивного вектора кожного елемента на тестовому наборі. Виконується рендеринг та синхронізація результатів good–value and fault as address simulation у таблицях good–value simulation, fault as address simulation, матриці моделювання несправностей на вхідному наборі та лініях логічної схеми, що виводиться на екран монітора. Виконується кодування механізмів моделювання та симуляції, а також їхня верифікація на прикладах логічних схем бібліотеки ISCAS.

Наукова новизна представлена векторно–логічними моделями елементів цифрової схеми, механізмами good–value simulation of test set as address та fault as address simulation цифрової схеми та fault as address simulation вхідного набору, використовуючи квадратичну матрицю симуляції. Реалізуються

механізми рендерингу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора.

Усі згадані інноваційні рішення орієнтовані на створення економічного масового комп'ютингу. Хмарні сервіси MOSI можуть бути корисними для студентів та інженерів, які займаються верифікацією цифрових проектів на основі логічних схем.

4.2 Стан питання

Промислові EDA [1] системи моделювання (Synopsys, Sentaurus TCAD; Cadence, Virtuoso, Spectre, Voltus, Quantus, Xcelium, Allegro; Aldec, Active-HDL, Riviera-PRO; Siemens EDA ModelSim, Intel Quartus Prime, PSIM; Xilinx ISE, Xilinx Vivado) орієнтовані на допомогу тестувальнику [2] або проектувальнику верифікувати [3] цифрові проекти на основі SoC, VLSI, ASIC, PLD, FPGA [4]. Ці системи оцінюються в десятки та сотні мільйонів доларів. Вони надають сервіси good-value simulation та fault simulation, які потрібні для генерації тестів та їх мінімізації, що є актуальним для цифрових проектів великої розмірності. Такі системи, як правило, недоступні невеликим компаніям, а також більшості університетів. При цьому в світі EDA design and test вважається, що сервіси моделювання несправностей більш складніші, ніж засоби good-value simulation, які використовуються для верифікації великих проектів, що налічують мільйони і мільярди вентилів. Але виявилося, що good-value simulation теж вирішує питання тестування таких проектів. Тому були розроблені стандарти [5] (IEEE 1500, P3405, 11.49, 1687) для обслуговування функціональностей SoC, ASIC, VLSI, FPGA шляхом розбиття на підсхеми, що тестуються на основі технології граничного сканування [6]. Ця технологія дозволяє використовувати testbench малої розмірності, для верифікації функцій та структур, як окремих частин великого проекту. Звичайно, після розбиття проекту на окремі функціональності стає можливим використовувати більш точні механізми моделювання

несправностей з метою мінімізації тесту та діагностування дефектів. Таким чином, розбиття великого проєкту на сегменти щодо функцій, структур, прийнятної кількості входів – це шлях до застосування точніших методів симулювання [5]. Після розбиття кожен сегмент обробляється за допомогою витратних механізмів моделювання несправностей. Так можна пояснити актуальність інноваційних механізмів моделювання несправностей та побудови карт тестування для функцій і структур великих цифрових проєктів, розбитих на сегменти. Слід зазначити, що це методи моделювання несправностей сьогодні використовують для введення схеми HDL мови [7] (VHDL, Verilog, SystemVerilog і SystemC), які передбачають використання потужних процесорів [8] до створення компілятивних моделей [9] великих проєктів і компілятивних алгоритмів симулювання дефектів. Компілятивні моделі подібні до жорсткої логіки, які не можна змінювати в процесі тестування цифрового проєкту. Компілятивні структури даних та алгоритми є енерговитратними та надзвичайно складними для їх кодування. Альтернативою таких рішень є використання простих явних векторно–логічних структур даних, які практично обнулюють складність алгоритмів їх аналізу при моделюванні несправностей як адрес таблиці істинності. Інтерпретативні або явні векторно–логічні моделі є гнучкими структурами [10] для їх зміни у процесі верифікації та налагодження проєкту [11]. При цьому векторно–логічні механізми (моделі та алгоритми), будучи поміщеними в архітектури in–memory computing [12], вимагають лише однієї read–write транзакції, вільні від інструкцій процесора [13]. Це робить векторно–логічний комп'ютинг [14] масовим у майбутньому та економічним з погляду витрат енергії та часу [15] для отримання Inference. Good–value simulation вирішує проблему – чи правильно працює пристрій на вичерпному чи обмеженому тесті [16]. Відповідає на питання, чи є у проєкті помилки.

Fault simulation вирішує проблеми [17, 18]:

- 1) чи правильно працює пристрій на вичерпному тесті. Він містить сервіс good–value simulation;
- 2) будує карту тестування несправностей на вичерпному тесті [19];

- 3) діагностує дефекти, що перевіряються на кожному тестовому наборі;
- 4) сприяє генерації мінімального тесту відносно поодиноких константних несправностей;
- 5) створює схеми безперешкодного контролю вхідних змінних на виходах функціональності. Цей сервіс забезпечує булева похідна по комбінації вхідних змінних;
- 6) діагностує дефекти вхідних змінних будь-якої кратності;
- 7) визначає тестові набори, що перевіряють комбінацію замовлених несправностей на входах функціональності;
- 8) все перераховане належить до результатів моделювання як логічних функцій, і будь-яких структур, представлених графами.

Мета – розробка сервісу для економічного вирішення задач modelling for simulation (MOSI) на основі векторно-логічного in-memory комп'ютингу шляхом використання read-write транзакцій без інструкцій процесора.

Завдання дослідження:

- 1) створення парсера механізму для конвертації HDL коду логічної схеми у внутрішні векторно-логічні структури даних сервісу MOSI;
- 2) Modeling векторно-логічної моделі цифрової схеми для good-value simulation вхідних тестових наборів, як адрес логічних векторів елементів;
- 3) Modeling дедуктивних векторів на векторно-логічній моделі логічної схеми для fault as address simulation вхідних тестових наборів;
- 4) Modelling матриці моделювання несправностей, як бітів адрес дедуктивного вектора кожного елемента на тестовому наборі;
- 5) рендеринг та синхронізація результатів good-value and fault as address simulation у таблицях good-value simulation, fault as address simulation, матриці моделювання несправностей на вхідному наборі та на лініях логічної схеми, що виводиться на екран монітора;
- 6) кодування механізмів моделювання та симуляції, а також їхня верифікація на прикладах логічних схем бібліотеки ISCAS [20, 21].

Наукова новизна:

1. Векторно–логічні моделі елементів цифрової схеми, що дозволяє здійснювати good–value simulation вхідних тестових наборів та симуляцію несправностей як адрес бітів логічного та дедуктивного вектора на основі read–write транзакцій.
2. Механізми good–value simulation of test set as address цифрової схеми без процесора.
3. Механізми fault as address simulation цифрової схеми без участі процесора.
4. Механізм fault as address simulation вхідного набору, використовуючи квадратичну матрицю симуляції.
5. Механізми рендерингу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора.

4.3 Рендеринг векторно-логічного моделювання цифрової схеми

Рендеринг [22] векторно-логічного моделювання цифрової схеми подано на рис. 4.1. Мета – modelling вхідних та вихідних даних для ергономічної візуалізації процесів та результатів симуляції.

Завдання:

- 1) виведення на екран HDL коду цифрової схеми;
- 2) виведення на екран режиму good–value simulation ліній схеми;
- 3) виведення на екран таблиці моделювання несправностей ліній схеми;
- 4) виведення на екран матриці симулювання несправностей на вхідному наборі;
- 5) виведення на екран векторно–логічних моделей елементів цифрової схеми та результатів good–value and fault simulation всіх ліній схеми.

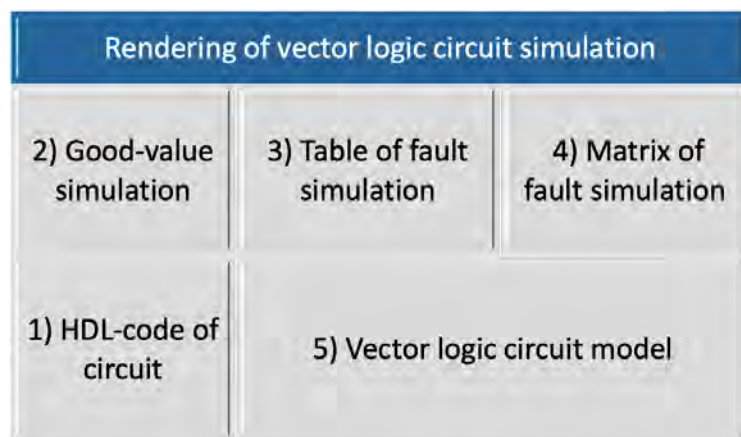


Рисунок 4.1 – Візуалізація векторно-логічного моделювання: 1 – HDL-код схеми; 2 – Good-value simulation; 3 – таблиця моделювання несправностей; 4 – матриця моделювання несправностей; 5 – векторно-логічне моделювання схеми

Метрика ергономічного рендерингу структур даних:

1. Всі вікна для відображення вхідних даних і результатів моделювання масштабуються від нуля до розміру монітора;
2. Всі процеси моделювання, що відображаються у вікнах 2–5, синхронізовані щодо вхідного тестового впливу;
3. Усі елементи схеми представлені логічними векторами, які роблять процес моделювання простим з урахуванням read–write транзакцій без інструкцій процесора;
4. Візуалізація якості вхідного набору та тесту спільно з good–value and detected fault на лініях схеми дає можливість прискорити процес верифікації та генерації мінімального тесту;
5. Rendering матриці моделювання несправностей на вхідному наборі дозволяє вивчити запропоновану теорію векторно–логічного моделювання за мінімальний час. Знання теорії важливе для пошуку помилок проектування цифрового пристрою та верифікації testbench. Промислові системи

моделювання цифрових схем не мають такого сервісу. Що знаходиться «під капотом», приховано від очей користувача і вартує великих грошей;

6. Інтегральною перевагою або новизною даних сервісів є синхронізація процесів моделювання по всіх вікнах, що представляють всі суттєві процедури для розуміння отриманого результату, що є важливим для здобувача освіти, що навчається технологіям, та інженера.

Поряд із цим пропонуються такі опційні сервіси *rendering for good-value and fault simulation*.

Перший режим – це підсвічування елементів та ліній схеми за інтегральним *inference* вхідного набору на матриці моделювання несправностей. Він дозволяє відстежити властивості вхідного тестового набору *good-value and fault simulation* на всіх лініях схеми. Перший режим використовується для ручної побудови тесту покриття логічних несправностей.

Другий режим – це підсвічування логічного елемента, його входів та виходу, який синхронізується з фіксованим рядком матриці моделювання несправності та HDL-кодом елемента. Рядки таблиць *good-value and fault simulation* є в даному режимі змінними, що дозволяє побачити *good-value operation and fault simulation* вибраного елемента на всіх тестових наборах. Другий режим використовується для точного діагностування помилок проектування, логічних несправностей чи дефектів у цифровому виробі.

4.4 Матриця моделювання несправностей

HDL code (VHDL, Verilog, SystemVerilog та SystemC) для опису цифрових проектів з'явився на EDA-ринку у вісімдесяті роки XX століття [7] і пов'язаний з наступними факторами:

1) стандарт спілкування між собою тестувальників та проектувальників, вчених та студентів;

2) стандарт спілкування між собою різних систем моделювання та проектування провідних компаній планети;

3) стандарт для порівняння існуючих EDA-механізмів modeling for simulation з точки зору швидкодії та точності;

4) організація бібліотек benchmark circuits [23] різних схемотехнічних рішень [20] (ISCAS 85, 89, 96, 99) для тестування існуючих та нових механізмів проектування та верифікації;

5) створення потужних компіляторів для моделювання цифрових проектів великої розмірності.

Всі ці фактори слід враховувати дослідникам на ринку EDA, щоб порівнювати свої результати існуючими. Тому достатньо взяти невелику схему C17 із бібліотеки ISCAS 85 [21], щоб пояснити на ній операції та переваги векторно–логічних механізмів modeling for simulation (лістинг 4.1).

Лістинг 4.1 – Опис схеми C17 мовою Verilog

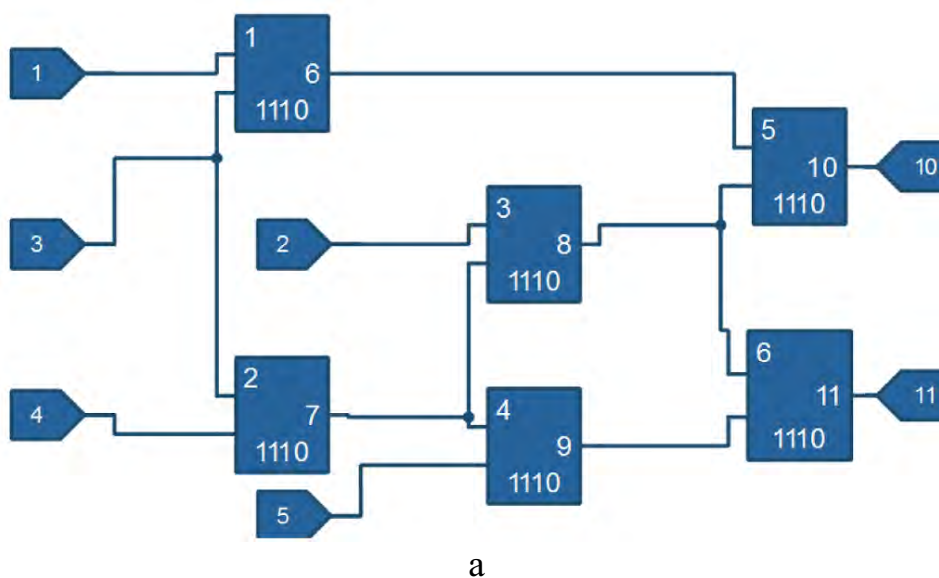
```
Verilog
module c17 (N1, N2, N3, N6, N7, N22, N23);
    input N1, N2, N3, N6, N7;
    output N22, N23;
    wire N10, N11, N16, N19;

    // Structural description on elements NAND
    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
```

Цей опис схеми Verilog використовувався для отримання:

1) зображення на моніторі векторно–логічної моделі;

2) modeling векторно-логічних структур даних – fault simulation matrix – для good-value as address and fault as address simulation на тестовому входному наборі (рис. 4.2).



Fault Simulation Matrix for 11111											Input	I-set	LV	DV
	1	2	3	4	5	6	7	8	9	10	11			
1	1													
2		1												
3			1											
4				1										
5					1									
6	1		1			1						1 3	1 1	1110 0111
7			1 1				1					3 4	1 1	1110 0111
8			1 1				1 1					2 7	1 0	1110 0100
9			1 1				1		1			7 5	0 1	1110 0010
10	1					1				1		6 8	0 1	1110 0010
11			1 1				1 1 1		1		1	8 9	1 1	1110 0111
sum	1		1 1			1	1 1 1		1	1	1	Date structure		
good	1	1	1 1	1	1	0	0 1 1	1	1	1	0			
fault	0		0 0			1	1 0 0	0	0	0	1			

б

Рисунок 4.2 – Логічна схема (а) та fault simulation matrix (б)

Fault simulation matrix [14] – це інтелектуальні ядро моделювання або розумні структури векторно-логічних даних для good-value as address and fault

as adress simulation на тестовому вхідному наборі. Розумно пов'язаними тут є такі компоненти:

1) квадратична матриця моделювання несправностей, що спочатку заповнено одиницями на координатах головної діагоналі;

2) Date structure, що містять номери вхідних ліній логічних елементів Input, вхідні тестові набори для кожного елемента схеми I-set, логічні вектори кожного елемента схеми LV для good-value as address simulation, дедуктивні вектори кожного елемента схеми DV для fault as address simulation;

3) inference результатів good-value as address and fault as adress simulation, sum – інтегральний вектор логічного OR-підсумовування всіх несправностей, що перевіряються на спостережуваних виходах схеми. Тут підсумовуються два вектори або два рядки

$$\text{sum} = 10V11=1011011111, \quad (4.1)$$

good – good-value as address simulation всіх ліній схеми, fault – fault as address simulation всіх несправностей лінії схеми на вхідному тестовому наборі.

Особливість fault simulation matrix – це унітарне кодування всіх несправностей одиничними координатами квадратичної матриці розмірністю $(n \times n)$, де n – число вхідних, вихідних і внутрішніх ліній логічної схеми. Кожен невхідний рядок матриці відповідає за транспортування несправностей вхідних ліній відповідного елемента на його вихід за допомогою дедуктивного вектора DV, який знаходиться в цьому ж рядку матриці праворуч. Адреса, яка формується одиничними та нульовими координатами матриці моделювання на входах Input, вибирає біт дедуктивного вектора, який формує (0) 1 значення на виході, що означатиме (не)перевірку комбінації вхідних несправностей, як сформованої адреси, на виході елемента. Наприклад, шостий рядок матриці має по координатах 1 і 3 одиничні значення, які формують адресу 11 для останнього біта дедуктивного вектора 0111. За цією адресою знаходиться 1. Це означає –

вхідні несправності координат 1 і 3 будуть перевірені на виході 6-го елемента. Вектор sum – це логічне підсумовування всіх векторів матриці, які відповідають вихідним лініям схеми, що спостерігаються. Ці одиниці формують несправності вектора fault, що перевіряються, які завжди будуть інверсні по відношенню good-value states цих ліній у good векторі або на тестовому вхідному наборі×

$$\text{fault} = (\text{sum}=1) \& \overline{\text{good}}. \quad (4.2)$$

Тут представлені структури даних (date structure), які показують формування дедуктивних векторів DV як функцію від вхідного набору та логічного вектора LV елемента $DV=f(I\text{-set}, LV)$. Modelling дедуктивного вектора в деталях буде показано нижче у підрозділі 4.5. Fault Simulation Matrix – це know how за простотою структур даних та алгоритмом їх обробки.

4.5 Моделювання дедуктивного вектора та тестової карти

Векторно-логічне моделювання цифрової схеми. Мета – суттєве зменшення витрат часу та енергії для modelling for simulation за рахунок використання read–write транзакцій in–memory computing на векторно–логічних моделях елементів цифрової схеми.

Метрика векторно–логічного моделювання (рис. 4.3) визначається за наступними правилами:

- 1) логічний вектор (01000010) задано для опису будь–якої функціональності або структури;
- 2) Read–write транзакція на основі явно заданої двійкової адреси біта логічного вектора, достатнього для обчислення стану будь–якого логічного елемента;
- 3) будь–яка двомісна операція задається логічним вектором довжиною 4 біти;

L-matrix									H-matrix								
	0	1	0	0	0	0	1	0									
0	1						1		0	1	2	3	4	5	6	7	
1	1		1	1	1	1		1	1	0	3	2	5	4	7	6	
0	1						1		2	3	0	1	6	7	4	5	
0	1						1		3	2	1	0	7	6	5	4	
0	1						1		4	5	6	7	0	1	2	3	
0	1						1		5	4	7	6	1	0	3	2	
1	1		1	1	1	1		1	6	7	4	5	2	3	0	1	
0	1						1		7	6	5	4	3	2	1	0	

D-matrix								
	000	001	010	011	100	101	110	111
000		1					1	
001		1	1	1	1	1	1	
010				1	1			
011			1			1		
100			1			1		
101				1	1			
110		1	1	1	1	1	1	
111		1					1	

C-matrix or testing map								
	000	001	010	011	100	101	110	111
000		..1					11.	
001		..0	.1.	.10	1..	1.0	11.	
010				.01	1..			
011			.0.			1.0		
100			.1.			0.1		
101				.10	0..			
110		..1	.0.	.01	0..	0.1	00.	
111		..0					00.	

Рисунок 4.3 – Матричні векторно–логічні механізми

4) двомісна операція над логічним вектором також породжує логічний вектор або матрицю довжини 2^{2n} ;

5) Xor–операція над логічним вектором породжує невпорядковану L –матрицю активності або повну множину невпорядкованих дедуктивних векторів і похідних;

6) упорядкування матриці активності за HDL–стандартом (H –matrix) проходження адрес логічних змінних породжує дедуктивну матрицю (D –matrix) $D=L_H$. Вона містить повну множину дедуктивних векторів у рядках і повну множину похідних в стовпцях матриці;

7) дедуктивний вектор – це логічний вектор, який визначає умови активізації (зміни) виходу елемента вхідними логічними несправностями на заданому вхідному наборі;

8) булева похідна – це логічний вектор, який визначає умови активізації виходу при зміні заданого числа вхідних змінних на вичерпному тесті;

8) C -matrix or testing map – це матриця відносин між вичерпним тестом і повною множиною всіх комбінацій несправностей вхідних змінних. Координати матриці визначаються F –векторами несправностей, що перевіряються в алфавіті (01.), де «точка» позначає неперевірку несправності на відповідній вхідній змінній. Кожна істотна (0,1) координата F -вектора визначається на місцях з 1-значенням бітів адрес, що знаходяться верхньої частини C -матриці, які визначаються інверсним значенням відповідних бітів вхідного тестового набору в лівій частині матриці.

Для побудови карти тестування будь-якої функції чи структури необхідні три матричні операції [15]. Але поставлена мета складніше – використовувати векторну логіку для моделювання несправностей всіх ліній вже побудованої схеми. Для досягнення цієї мети необхідно вирішити такі завдання (рис. 4.4):

1) створити парсер для HDL-коду опису логічної схеми у внутрішній структурі даних MOSI-сервісів;

2) Modeling векторно-логічних структур даних цифрової схеми для моделювання та відображення на екрані;



Рисунок 4.4 – Архітектура та векторно-логічні механізми для моделювання та рендерингу схеми [рисунок сформовано автором та оброблено за допомогою сервісу III Gemini]

- 3) Good-value simulation тестових наборів як адрес бітів логічних векторів елементів;
- 4) симуляцію несправностей як адрес бітів логічних векторів елементів;
- 5) рендеринг процесу good-value and fault simulation на векторно-логічних структурах даних логічної схеми з використанням скролінгу;
- 6) рендеринг векторно-логічної моделі схеми з використанням скролінгу та масштабування вікон;
- 7) рендеринг good-value and fault simulation на зображенні логічної схеми з використанням скролінгу та масштабування вікон;
- 8) Rendering input set fault simulation синтезу мінімального тесту логічної схеми;

9) Modeling – побудова моделі об'єкта чи явища у пам'яті комп'ютера.

Векторна логіка розглядається як теорія та практика моделінгу (modeling) структур даних та алгоритмів для безпроцесорного вирішення будь-якого завдання за допомогою read-write транзакцій у пам'яті комп'ютера. Simulation – як використання моделі процесу чи явища для визначення вихідних реакцій на задані вхідні набори. Rendering – оптимізація та ергономіка процесу візуалізації відносин на екрані між вхідними та вихідними даними програми з метою виходу EDA-ринок. Карта застосунку для верифікації та рендерингу структур функцій та цифрових логічних схем представлена вище (див. рис. 4.1). Така візуалізація процесів modeling for simulation на сьогоднішній день є інноваційною та привабливою не лише для університетів, але й для фахівців, які займаються тестуванням сучасних цифрових систем на основі SoC.

4.6 Моделювання таблиці істинності за допомогою дедуктивного вектора

Розглядається *механізм побудови дедуктивного вектора на вхідній множині з використанням явних адрес у таблиці істинності*. Цей механізм легко масштабується до будь-якої таблиці істинності з n змінних. Він також може бути використаний для виявлення несправностей у тестових наборах будь-якої функції чи структури. Вихідні дані (рис. 4.5): логічний вектор функціональності Y , тестова вхідна множина T .

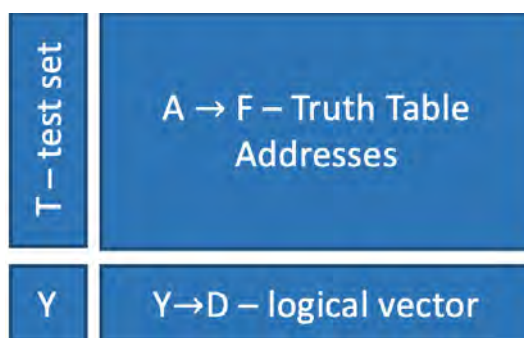


Рисунок 4.5 – Схема структур даних для моделювання дедуктивного вектора

Алгоритм моделювання дедуктивного вектора для тестового набору:

1) стан вихідної функції вираховується як

$$Y=Y_T; \quad (4.3)$$

2) векторні XOR операції

$$L=TY \oplus AY, \quad (4.4)$$

створюють активну неупорядковану таблицю істинності на тестовому наборі даних;

3) упорядкування отриманої таблиці істинності відповідно до адреси HDL, що йде за стандартом A, створює дедуктивний вектор $D_A=L$ за три операції.

Вихідні дані для моделювання (рис. 4.6): вхідний двійковий набір $T=11$ довжини n і логічний вектор $Y=0111$ довжини 2^n , що формує таблицю істинності 2^n вхідних наборів X , у прикладі змінних $x_1 x_2$. Вихідні дані: таблиця істинності розмірності $n \times 2^n$ несправностей, що виявляються. Розмір пам'яті для зберігання структур даних визначається виразом:

$$M = n \times 2^n + 2^n + n + 1, \quad (4.5)$$

де n – кількість змінних логічного елемента, що формує компактні та розумні структури даних: таблицю істинності, логічний (дедуктивний) вектор, вхідне двійкове слово $x=11$ і вихідні стани $Y=1$.

1	T	Y _T				1	T	Y _T				1	T	Y _T				1	T	Y _T									
x ₁	0	0	1	0	1	x ₁	1	0	1	0	1	x ₁	0	0	1	0	1	x ₁	1	0	1	0	1	x ₁	1	0	1	0	1
x ₂	0	0	0	1	1	x ₂	0	0	0	1	1	x ₂	1	0	0	1	1	x ₂	1	0	0	1	1	x ₂	1	0	0	1	1
Y	0	0	1	1	1	Y	1	0	1	1	1	Y	1	0	1	1	1	Y	1	0	1	1	1	Y	1	0	1	1	1

2	T	A=T⊕F				2	T	A=T⊕F				2	T	A=T⊕F				2	T	A=T⊕F									
x ₁	0	0	1	0	1	x ₁	1	1	0	1	0	x ₁	0	0	1	0	1	x ₁	1	1	0	1	0	x ₁	1	1	0	1	0
x ₂	0	0	0	1	1	x ₂	0	0	0	1	1	x ₂	1	1	1	0	0	x ₂	1	1	1	0	0	x ₂	1	1	1	0	0
L	0	0	1	1	1	L	1	1	0	0	0	L	1	1	0	0	0	L	1	1	0	0	0	L	1	1	0	0	0

3	T	D _A = L				3	T	D _A = L				3	T	D _A = L				3	T	D _A = L									
x ₁	0	0	1	0	1	x ₁	1	0	1	0	1	x ₁	0	0	1	0	1	x ₁	1	0	1	0	1	x ₁	1	0	1	0	1
x ₂	0	0	0	1	1	x ₂	0	0	0	1	1	x ₂	1	0	0	1	1	x ₂	1	0	0	1	1	x ₂	1	0	0	1	1
D	=	0	1	1	1	D	=	0	1	0	0	D	=	0	0	1	0	D	=	0	0	0	1	D	=	0	0	0	1

4	F _{ij} = $\overline{T}_i \leftarrow F_{ij} \wedge D_i = 1$					4	F _{ij} = $\overline{T}_i \leftarrow F_{ij} \wedge D_i = 1$					4	F _{ij} = $\overline{T}_i \leftarrow F_{ij} \wedge D_i = 1$					4	F _{ij} = $\overline{T}_i \leftarrow F_{ij} \wedge D_i = 1$										
x ₁	0	0	1	0	1	x ₁	1	0	0	0	0	x ₁	0	0	1	0	1	x ₁	1	0	1	0	0	x ₁	1	0	1	0	0
x ₂	0	0	0	1	1	x ₂	0	0	0	1	0	x ₂	1	0	0	0	1	x ₂	1	0	0	1	0	x ₂	1	0	0	1	0
D	=	0	1	1	1	D	=	0	1	0	0	D	=	0	0	1	0	D	=	0	0	0	1	D	=	0	0	0	1

Рисунок 4.6 – Схема моделювання дедуктивних векторів на основі вичерпного тесту

Приклад алгоритму моделювання:

1) Good-value simulation of logic element – обчислює вихідне значення Y при подачі на двійковий вхідний набір T, що розглядається як адреса біта логічного вектора Y_T;

2) далі виконується векторна матрична операція

$$A=T\oplus F, \quad (4.6)$$

щоби отримати таблицю істинності активності A. Тут стовпці відповідні змінні x₁, x₂ у стовпчиках обробляються як двійкові адреси 11, 01, 10, 00 для перекодування або зміни порядку бітів вектора L;

3) упорядкування координат L-вектора за вхідними двійковими адресами x₁, x₂ таблиць істинності для отримання дедуктивного вектора D_A=L. Ця

операція необхідна для приведення таблиці істинності до загальноприйнятого стандарту інкрементованих двійково–десяткових адрес на основі вхідних змінних, що потрібно для обробки елементів у цифровій структурі. Обчислювальна складність генерації дедуктивного вектору $2^n \times (n + 1)$.

Генерація виявлених вхідних несправностей логічного елемента за інверсними значеннями сигналів вхідних змінних x_1, x_2 на одиничних координатах стовпців таблиці істинності, покритих (активованих) одиничними значеннями координат дедуктивного вектору:

$$F_{ij} = \bar{T}_i \leftarrow F_{ij} \wedge D_i = 1. \quad (4.7)$$

Обчислювальна складність алгоритму визначається наступною формулою:

$$Q = 1 + 2^n \times (n + 1) + 2^n + 2^n \times n \quad (4.8)$$

виконання транзакцій читання–запису. Пропонується використовувати таблицю істинності для моделювання вхідних несправностей для будь–якого логічного елемента.

Формула моделювання: вхідний двійковий набір, що використовується як адреса комірки логічного вектору, містить вихідний стан компонента, який об'єднується з вхідним набором для формування вектору справних станів, призначеного для хог–взаємодії з усіма стовпцями таблиці істинності для створення невпорядкованого за адресами таблиці та вектору активності, до якого застосовується процедура впорядкування стовпців таблиці та бітів вектору активності за стандартними адресами у відповідних стовпцях таблиці, потім результуючий дедуктивний вектор активує одиничні координати у відповідних стовпцях таблиці істинності своїми одиничними значеннями, знаки яких визначаються інверсними станами біта вхідного набору.

4.7 Good-value та fault simulation цифрових схем

Векторно–логічне моделювання цифрових схем має таку метрику:

- 1) прості розумні та надмірні структури даних, які обнулюють алгоритм алгоритми моделювання good–value and fault as addresses simulation;
 - 2) в межах ці алгоритми використовують read–write транзакції без інструкцій процесора при реалізації їх в архітектурі in–memory computing;
 - 3) рендеринг результатів моделювання спрямований на допомогу тестувальнику вирішувати завдання верифікації функціональностей чи структур, мінімізації тестів та діагностування дефектів;
 - 4) синхронізація процесів моделювання за масштабованим вікном показує всі деталі моделювання схеми на вичерпному тесті та на окремих вхідних наборах, як на схемі, так і на окремому її елементі;
 - 5) вікна містять таку статистичну інформацію: час виконання окремих процедур моделювання, якості тестових вхідних наборів (Q) якість тесту Σ в цілому;
 - 6) зображення схеми на екрані монітора синхронізовано з матрицею моделювання несправностей та містить інформацію про перевірку логічних несправностей та good–value states всіх ліній схеми на кожному вхідному наборі.
- Всі результати good-value and fault simulation схеми C17 показані на рис. 4.7 у двох вікнах, які синхронізовані за рядками обох таблиць моделювання із матрицею моделювання та зображенням схеми на екрані монітора.

Test	Good-value simulation table											Test	Fault simulation table												
	1	2	3	4	5	6	7	8	9	10	11		Q	Σ	1	2	3	4	5	6	7	8	9	10	11
00000	0	0	0	0	0	1	1	1	1	0	0	00000	0.32	0.32	1			1	0		0	0	1	1	
00001	0	0	0	0	1	1	1	1	0	0	1	00001	0.36	0.50	1			0	0	0	0	1	1	0	
00010	0	0	0	1	0	1	1	1	1	0	0	00010	0.32	0.50	1			1	0		0	0	1	1	
00011	0	0	0	1	1	1	1	1	0	0	1	00011	0.41	0.55	1	1		0	0	0	0	1	1	0	
00100	0	0	1	0	0	1	1	1	1	0	0	00100	0.36	0.59	1	1		1	0		0	0	1	1	
00101	0	0	1	0	1	1	1	1	0	0	1	00101	0.45	0.64	1	1		1	0	0	0	0	1	1	
00110	0	0	1	1	0	1	0	1	1	0	0	00110	0.27	0.64	1				0		0	0	1	1	
00111	0	0	1	1	1	1	0	1	1	0	0	00111	0.41	0.77	1		0	0		0	1	0	0	1	
01000	0	1	0	0	0	1	1	0	1	1	1	01000	0.23	0.91	0					0	1		0	0	
01001	0	1	0	0	1	1	1	0	0	1	1	01001	0.23	0.91	0					0	1		0	0	
01010	0	1	0	1	0	1	1	0	1	1	1	01010	0.27	0.91	0	1				0	1		0	0	
01011	0	1	0	1	1	1	1	0	0	1	1	01011	0.27	0.91	0	1				0	1		0	0	
01100	0	1	1	0	0	1	1	0	1	1	1	01100	0.27	0.91	0		1			0	1		0	0	
01101	0	1	1	0	1	1	1	0	0	1	1	01101	0.27	0.91	0		1			0	1		0	0	
01110	0	1	1	1	0	1	0	1	1	0	0	01110	0.41	0.91	1		0	0		0	1	0	0	1	
01111	0	1	1	1	1	1	0	1	1	0	0	01111	0.41	0.91	1		0	0		0	1	0	0	1	
10000	1	0	0	0	0	1	1	1	1	0	0	10000	0.36	0.91	1	1		1	0		0	0	1	1	
10001	1	0	0	0	1	1	1	1	0	0	1	10001	0.41	0.91	1	1		0	0	0	0	1	1	0	
10010	1	0	0	1	0	1	1	1	1	0	0	10010	0.36	0.91	1	1		1	0		0	0	1	1	
10011	1	0	0	1	1	1	1	1	0	0	1	10011	0.41	0.91	1	1		0	0	0	0	1	1	0	
10100	1	0	1	0	0	0	1	1	1	1	0	10100	0.41	1.00	0	1	0		1	1		0	0	0	1
10101	1	0	1	0	1	0	1	1	0	1	1	10101	0.41	1.00	0		0	1	0	1	0		1	0	0
10110	1	0	1	1	0	0	0	1	1	1	0	10110	0.32	1.00	0		0		1		0	0	0	1	
10111	1	0	1	1	1	0	0	1	1	1	0	10111	0.41	1.00	0		0	0		1	1	0	0	1	
11000	1	1	0	0	0	1	1	0	1	1	1	11000	0.23	1.00	0					0	1		0	0	
11001	1	1	0	0	1	1	1	0	0	1	1	11001	0.23	1.00	0					0	1		0	0	
11010	1	1	0	1	0	1	1	0	1	1	1	11010	0.27	1.00	0	1				0	1		0	0	
11011	1	1	0	1	1	1	1	0	0	1	1	11011	0.27	1.00	0	1				0	1		0	0	
11100	1	1	1	0	0	0	1	0	1	1	1	11100	0.27	1.00	0		1			0	1		0	0	
11101	1	1	1	0	1	0	1	0	0	1	1	11101	0.18	1.00			1			0			0	0	
11110	1	1	1	1	0	0	0	1	1	1	0	11110	0.41	1.00	0		0	0		1	1	0	0	1	
11111	1	1	1	1	1	0	0	1	1	1	0	11111	0.41	1.00	0		0	0		1	1	0	0	1	

Рисунок 4.7 – Результати good-value and fault as address simulation

Візуалізація good-value стану та виявленої несправності лінії схеми на тестовому наборі (00111) суттєво допомагає спеціалісту у побудові тестового testbench (рис. 4.8). Ці сигнали на лініях схеми синхронізовані з рядками таблиць good-value та моделювання несправностей. У цьому випадку таблиця істинності несправностей елемента 10 синхронізована з лінією 10 матриці моделювання несправностей (див. рис. 4.2, б). Візуалізація таблиці несправностей допомагає

точно діагностувати поодинокі або множинні дефекти на входах кожного логічного елемента в схемі.

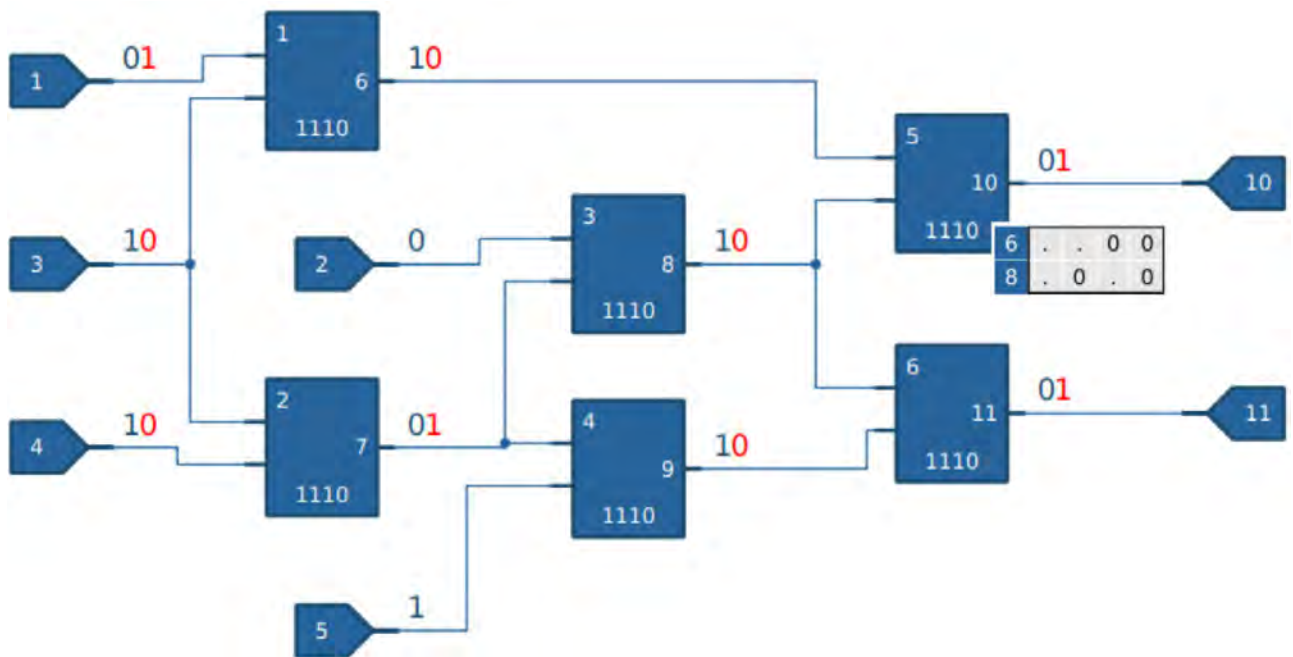


Рисунок 4.8 – Таблиця візуалізації несправностей, станів good-value та виявлених несправностей лінії схеми на тестовому наборі 00111

4.8 Обчислювальна складність алгоритмів моделювання

Складність алгоритмів моделювання good-value та fault simulation (справних значень та несправностей) як адрес на матриці моделювання визначається формулою

$$Q = n + n + \sum_{i=1}^n 2m_i \times 2^{m_i}, \quad (4.9)$$

де m_i – довжина логічного вектора i -го елемента. Тут перша складова – це складність вхідної множини справних значень моделювання, яка визначається кількістю елементів n у схемі, заданих логічними векторами. Другий член – це складність моделювання несправностей на вхідній множині, яка також визначається кількістю елементів схеми n , представлених дедуктивними

векторами. В обох випадках моделювання відбувається процес зчитування біта вектора, який визначає стани справних значень та несправностей елемента за сформованою бітовою адресою логічного або дедуктивного вектора.

Складність алгоритму моделювання дедуктивного вектора елемента визначається формулою $2^m \times 2^m$, якщо використовувати таблицю істинності розміром $m \times 2^m$ для m вхідних змінних та вхідний набір логічного елемента довжиною m .

Зменшення складності алгоритму дедуктивного векторного моделювання можна досягти, використовуючи Н-матрицю для кодування бітів логічного вектора в дедуктивний вектор. Обчислювальна складність побудови Н-матриці дорівнює $2^m \times 2^m$ (m – кількість вхідних змінних у логічному елементі). Н-матриця будується один раз для всіх логічних елементів схеми, орієнтуючись на довжину логічного вектора максимальної розмірності 2^m або кількість вхідних змінних m .

Наприклад, Н-матриця (Binary Distance) для трьох вхідних змінних (рис. 4.9) будується шляхом виконання декартової операції XOR над адресами відповідної таблиці істинності.

H-matrix (Binary Distance)									H-matrix (Decimal Distance)								
$\oplus$	000	001	010	011	100	101	110	111	Test	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111	000	0	1	2	3	4	5	6	7
001	001	000	011	010	101	100	111	110	001	1	0	3	2	5	4	7	6
010	010	011	000	001	110	111	100	101	010	2	3	0	1	6	7	4	5
011	011	010	001	000	111	110	101	100	011	3	2	1	0	7	6	5	4
100	100	101	110	111	000	001	010	011	100	4	5	6	7	0	1	2	3
101	101	100	111	110	001	000	011	010	101	5	4	7	6	1	0	3	2
110	110	111	100	101	010	011	000	001	110	6	7	4	5	2	3	0	1
111	111	110	101	100	011	010	001	000	111	7	6	5	4	3	2	1	0

а

б

Рисунок 4.9 – Н-матриця: двійкова (а) та десяткова (б) відстань

Десятковий еквівалент Н-матриці використовується для перекодування логічного вектора на вхідному наборі, представленого двійковим номером рядка Н-матриці (десяткова відстань), у дедуктивний вектор за формулою перетворення бітів

$$D=(L + Y)_H. \quad (4.10)$$

Тоді логічний вектор 01111110 (рис. 4.10) на вхідному тестовому наборі 011, $Y=1$, третього рядка стане дедуктивним вектором

$$01111110_{32107654}=00011000. \quad (4.11)$$

Логічний вектор 11100111 на вхідному тестовому наборі 011, $Y=0$, третього рядка перетвориться на дедуктивний вектор

$$11100111_{32107654}=01111110. \quad (4.12)$$

У випадку використання Н-матриці, складність алгоритму моделювання good-value та fault simulation як адрес для одного вхідного набору схеми буде визначатися формулою

$$Q=n+n+\sum_{i=1}^n 2^{m_i}, \quad (4.13)$$

де m_i – кількість входів логічного вектора i -го елемента.

Якщо, наприклад, усі елементи схеми представлені m вхідними змінними, то формула обчислювальної складності алгоритму моделювання good-value та fault simulation така:

$$Q=n+n+n \times 2^m = 2n+n \times 2^m = n(2+2^m). \quad (4.14)$$

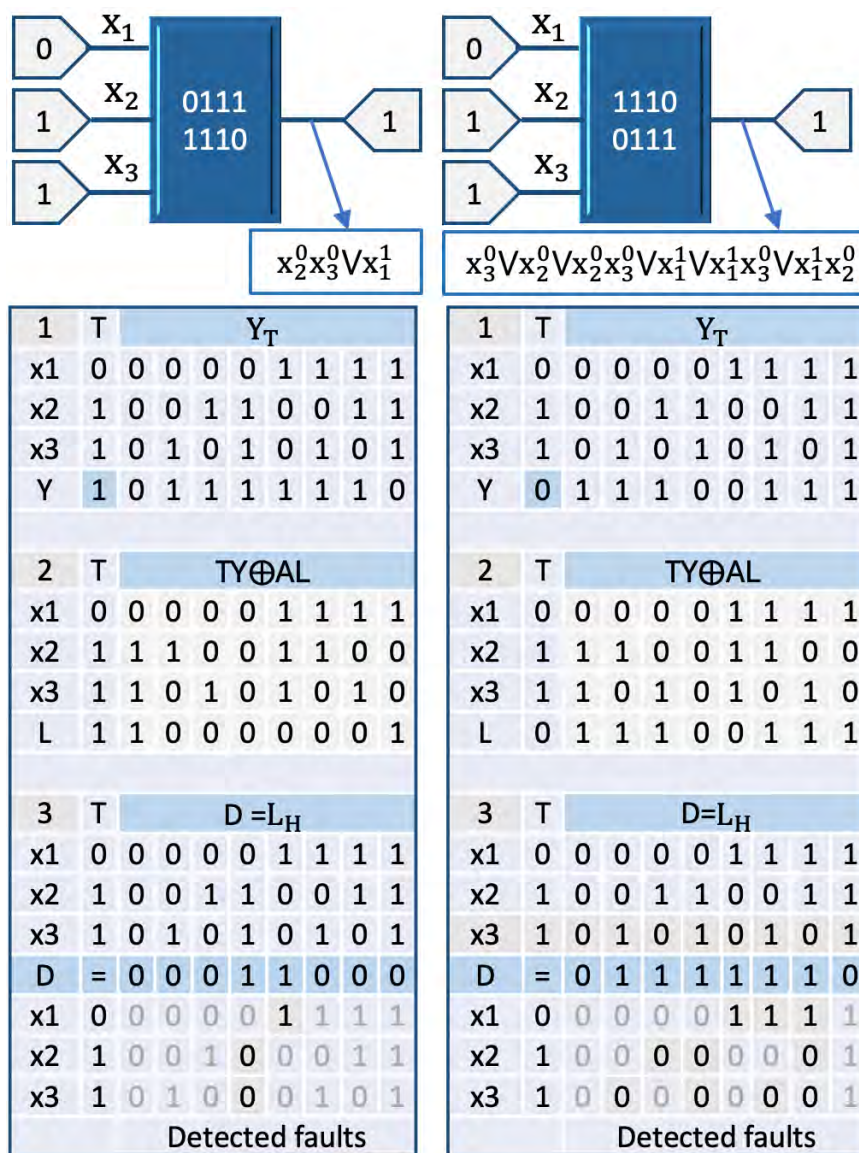


Рисунок 4.10 – Моделювання дедуктивних векторів та таблиць істинності для симуляції вхідних несправностей

Синхронізація матриці моделювання несправностей з логічним елементом під час моделювання вхідного набору на цифровій логічній схемі, показує комбінації вхідних несправностей, виявлених як 1-бітові явно задані адреси логічного дедуктивного вектора. За допомогою трьох логічних операцій дедуктивний вектор будується як функція XOR з вхідного набору та логічного вектора елемента. Дедуктивний вектор потім покриває 1-координати адрес таблиці істинності своїми 1-значеннями, які перетворюються на комбінації несправностей, що є оберненими до бітових значень вхідного тестового набору.

Приклад побудови дедуктивних векторів та визначення комбінацій вхідних несправностей для вхідного набору 011 для двох елементів, представлених логічними векторами 0111110 та 11100111 (див. рис. 4.9).

Тут таблиця істинності виступає повною та ідеальною моделлю для точної діагностики комбінації несправностей логічних елементів, виявлених на вхідному наборі. Матриця моделювання несправностей виявляє лише застрягли (stuck-at faults) несправності на лініях схеми. Але цей результат не завжди влаштовує фахівців з проєктування, тестування та діагностики. Аварії та катастрофи в історії траплялися саме через комбінацію багатьох несправностей або помилок у цифровому продукті. Тому для кожного елемента цифрової схеми важливо мати точний інструментарій для виявлення поодиноких та множинних несправностей та їх візуалізації. Візуалізація використовує таблицю істинності на виході елемента, яка синхронізована з відповідним рядком матриці моделювання несправностей. Таблиця істинності завжди відображає повний набір несправностей, які потрібно виявити, в 1-координатах адрес, що охоплюються 1-координатами дедуктивного вектора. Знаки таких несправностей (застряг-на-0 або 1) завжди будуть інверсними відносно відповідних бітів вхідного набору кожного елемента. Немає аналогів точної діагностики несправностей при моделюванні тестових вхідних наборів у промислових та академічних системах електронного візуалізації (EDA).

4.9 Зниження обчислювальної складності алгоритмів симуляції

Обчислювальну складність алгоритмів моделювання можна ще більше зменшити завдяки надмірності структур даних у матриці моделювання несправностей. Для цього матрицю дедуктивних векторів можна досить просто та з мінімальними технологічними зусиллями побудувати, використовуючи логічний вектор кожного елемента схеми у двох матричних операціях. Перша – це декартова операція XOR над логічним вектором елемента схеми $V=L\oplus L$ для отримання V-матриці активності. Друга – це матрична операція

переупорядкування координат матриці активності $D=V_H$ на основі використання H -матриці для отримання D -матриці дедуктивних векторів. Для логічного вектора 01111110 ці дві операції дають матрицю дедуктивних векторів (рис. 4.11), кількість яких завжди визначається повним набором вхідних даних.

V=L⊕L								
L	0	1	1	1	1	1	1	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
1	1							1
0		1	1	1	1	1	1	

H=A⊕A								
H	000	001	010	011	100	101	110	111
000	0	1	2	3	4	5	6	7
001	1	0	3	2	5	4	7	6
010	2	3	0	1	6	7	4	5
011	3	2	1	0	7	6	5	4
100	4	5	6	7	0	1	2	3
101	5	4	7	6	1	0	3	2
110	6	7	4	5	2	3	0	1
111	7	6	5	4	3	2	1	0

D=L _H								
D	000	001	010	011	100	101	110	111
000		1	1	1	1	1	1	
001		1					1	
010			1			1		
011				1	1			
100				1	1			
101			1			1		
110		1					1	
111		1	1	1	1	1	1	

Рисунок 4.11 – Моделювання матриць дедуктивних векторів

Попередня побудова матриці дедуктивних векторів для всіх елементів схеми зменшує обчислювальну складність алгоритмів моделювання good-value та fault simulation до оцінки $Q=n+n$. Це означає, що good-value та fault simulation на матриці моделювання мають однакову обчислювальну складність через надмірність структур даних у вигляді матриці дедуктивних векторів елементів схеми.

Надлишковість у дедуктивній векторній матриці для моделювання схем може підвищити продуктивність програмних застосунків, що моделюють схеми з подібними логічними елементами, шляхом створення бібліотеки таких матриць.

Проблема ефективного об'єднання адрес у матриці моделювання несправностей для транзакцій читання-запису між векторно-логічними структурами даних залишається невирішеною.

Обнулення алгоритмів моделювання. В обчислювальній техніці моделі та алгоритми підтримують гармонійні співвідношення, тобто коли виконується співвідношення балансу M (модель) + A (алгоритм) = 1. Інтелектуальні обчислення [23] використовують це гармонійне співвідношення між моделлю та алгоритмом, щоб компенсувати обмеження останнього за рахунок використання надмірності в моделі або її структурах даних. Моделювання (m) та симуляція (s) також взаємопов'язані: $m + s = 1$. Такі співвідношення дають досліднику-конструктору більше можливостей для проєктування, прагнучи мінімізувати час (затримку) чи простір (пам'ять), необхідні рішення завдань проєктування і тестування. Сьогодні в обчислювальних системах, що розвиваються, важливо мінімізувати час виконання алгоритму через моделювання експоненційної надмірності структур даних. Це дозволяє створювати надлишкові моделі чи структури даних без необхідності використання алгоритмів симуляції. Самі такі моделі є висновком. Тим часом, використання обчислень у пам'яті знижує затримку виведення, енергоспоживання та складність алгоритму (рис. 4.12).

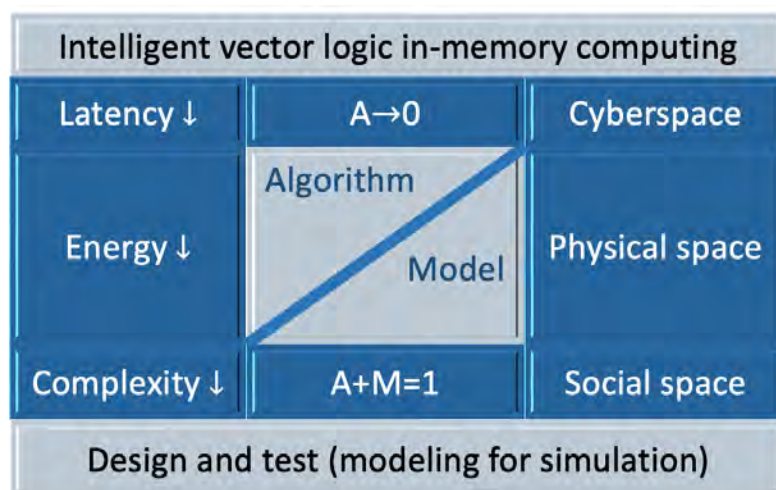


Рисунок 4.12 – Схема гармонічних відношень між алгоритмом та моделлю

Як приклад розглянемо моделювання тестової карти на функціональність або структуру, що використовується у вичерпному тесті без алгоритму моделювання несправностей. Вхідна інформація – це логічний вектор функціональності або структури, на якому будується матриця дедуктивних векторів.

Словесна формула моделювання тестової карти згідно з дедуктивною векторною матрицею: усі активні 1-координати дедуктивної векторної матриці визначаються F-адресами несправностей, що виявляються, де 1-координата адреси додатково визначається інверсними значеннями бітів вхідного T-множини тесту, а нульові координати додатково визначаються точками, що є індикаторами невиявлення несправностей за відповідними вхідними змінними.

Словесну формулу для моделювання несправностей можна представити наступним виразом:

$$C_{ij} = \bar{T}_i \leftarrow (D_{ij} = 1) \wedge (F_j = 1), \quad (4.15)$$

де C_{ij} – вектор вхідних несправностей, що виявляються в координаті тестової карти, T_i – тестовий набір для заданої функціональності, $D_{ij}=1$ – активна

координата матриці дедуктивних векторів, $F_j=1$ – суттєва координата адреси несправностей, що виявляються. Моделювання тестової карти з використанням матриці дедуктивних векторів для функціональності, заданої логічним вектором $L=01111110$, представлено на рис. 4.13. Тут наведено повний цикл моделювання тестової карти на основі логічного вектора функціональності, з використанням трьох матричних операцій.

$V=L \oplus L$								
L	0	1	1	1	1	1	1	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
0		1	1	1	1	1	1	

$H=A \oplus A$								
H	000	001	010	011	100	101	110	111
000	0	1	2	3	4	5	6	7
001	1	0	3	2	5	4	7	6
010	2	3	0	1	6	7	4	5
011	3	2	1	0	7	6	5	4
100	4	5	6	7	0	1	2	3
101	5	4	7	6	1	0	3	2
110	6	7	4	5	2	3	0	1
111	7	6	5	4	3	2	1	0

$D=V_H$								
D	000	001	010	011	100	101	110	111
000		1	1	1	1	1	1	
001		1					1	
010			1			1		
011				1	1			
100				1	1			
101			1			1		
110		1					1	
111		1	1	1	1	1	1	

$C_{ij} = \bar{T}_i \leftarrow (D_{ij} = 1) \wedge (F_j = 1)$								
T\F	000	001	010	011	100	101	110	111
000		..1	.1.	.11	1..	1.1	11.	
001		..0					11.	
010			.0.			1.1		
011				.00	1..			
100				.11	0..			
101			.1.			0.0		
110		..1					00.	
111		..0	.0.	.00	0..	0.0	00.	

Рисунок 4.13 – Моделювання тестової карти на основі дедуктивної матриці або логічного вектора

Цей приклад демонструє використання гармонійного зв'язку між моделюванням та симуляціями, в якому надмірність результуючих структур даних означає, що алгоритмам симуляції не потрібно створювати вичерпну

карту всіх несправностей під час тестування. Іншими словами, для побудови тестової карти алгоритм було спрощено завдяки експоненціальній надмірності логічного вектора та матричній надмірності інших важливих даних.

4.10 Обговорення результатів

Обговорюються питання використання нової технології для моделювання вхідних тестових наборів як адрес коректної поведінки та представлення несправностей як адрес у логічному векторі. Вона використовує архітектуру обчислень у пам'яті, засновану на транзакціях читання-запису, для зниження енергоспоживання та скорочення часу отримання висновків.

Modelling for Simulation (MOSI) HDL (<http://mosihdl.work>) [24, 25] включає в себе наступні сервіси:

- 1) моделювання структур даних цифрових схем для точної симуляції;
- 2) моделювання структур даних для симуляції несправностей як адрес на тестових наборах;
- 3) візуалізація всіх процесів, пов'язаних зі створенням моделей цифрових проєктів на рівні вентиля, регістра та системи як для точного моделювання, так і для симуляції несправностей на вхідних наборах.

Ці сервіси можуть бути використані для перевірки проєктів цифрової логіки та вивчення механізмів моделювання векторної логіки в технічних університетах.

Програма MOSI HDL (<http://mosihdl.work>) містить близько 13 тисяч рядків коду на C++ версії C++17, у тому числі невелика частина – модуль web, написаний на JavaScript + HTML. Сам модуль моделювання (model) становить понад 2 тисячі рядків коду, найбільший обсяг займає графічний модуль (view) – понад 8 тисяч рядків. Підтримуються Windows та Ubuntu.

Структура проєкту MOSI HDL у рядках коду MOSI HDL dataflow представлена на рис. 4.14.

Модульна структура MOSI HDL використовує патерн проектування MVC (Model – View – Controller). Кожна конкретна дія (парсинг, розміщення, трасування, генерація таблиць) реалізовано у вигляді окремих команд, які працюють з моделлю, а графіка відображає модель у різних формах (таблиці, матриці, текст, схема). Для побудови моделі даних використано перевагу Surelog – підтримка Verilog Procedural Interface (VPI), що дозволило в короткий термін розробки отримати результат парсингу.

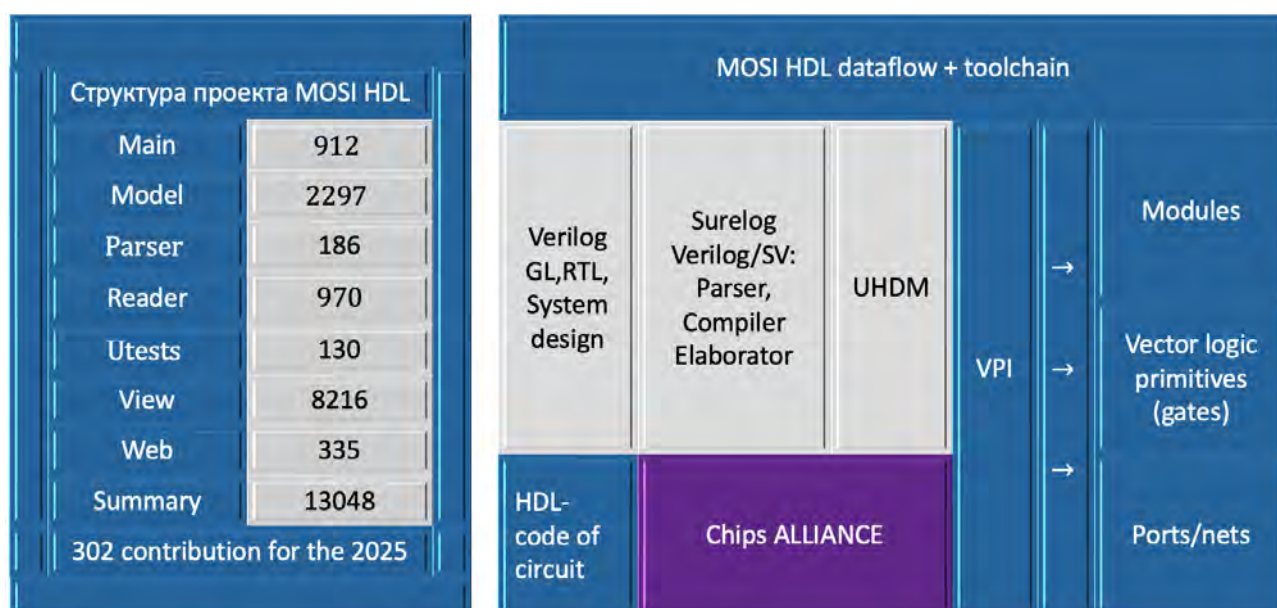


Рисунок 4.14 – Структура проекту у рядках коду та MOSI HDL dataflow

Програма MOSI HDL використовує зовнішні бібліотеки для вирішення завдань, які не пов'язані безпосередньо з моделюванням:

- 1) за парсинг файлів Verilog відповідає open-source парсер Surelog з його універсальною апаратною моделлю UHDM;
- 2) за розміщення елементів схеми (вузлів графа) відповідає бібліотека GraphViz;
- 3) за попереднє трасування з'єднань між елементами (ребер графа) відповідає пакет FLUTE (Fast Lookup Table based Rectilinear Steiner Minimal Tree algorithm, originally written by Chris Chu, Iowa State University);

4) остаточне трасування виконується одним із алгоритмів пошуку в лабіринті – A* search algorithm (розроблений Peter Hart, Nils Nilsson Bertram Raphael зі Stanford Research Institute, 1968);

5) графічні елементи (зокрема, іконки) використані із сервісу Flaticon;

6) графічний інтерфейс написаний із використанням бібліотеки Qt;

7) алгоритми та решта бізнес-логіки реалізовані з використанням Boost;

8) окремо розробляється web-модуль, який за допомогою docker-контейнерів та серверів дозволяє хостити десктопну програму як web-service;

9) для верифікації механізмів моделювання оброблені такі схеми ISCAS85: c17.v, c432.v, c499.v, c880.v, c1355.v, c1908.v, c3540.v, c6288.v. Статистика обробки схем має такий вигляд (рис. 4.15).

Actions	Circuits									
	Local Windows								Server Ubuntu	
	c17	c432*	c499*	c880*	c1355*	c1908*	c3540*	c6288*	c432*	c1355*
Verilog parsing using Surelog	0,343	1,3	1,4	2,8	4,7	4	9,7	20,3	1,6	4,5
Schematics placing using Graphviz	0,004	0,1	0,3	0,3	1,1	1,5	10,9	60,2	1,2	16,4
Schematics netline routing	0,003	0,5	0,8	0,6	1,8	4,6	10,3	4,5	0,6	1,8
Good value table generation	0,004	2,3	2,3	5,7	6,9	8,3	16,9	26,7	1,7	4,6
Fault simulation table generation	0,005	3,4	4,1	7,6	19,2	19	52	209,3	3,2	14,4
Total	0,359	7,7	8,8	17	33,7	37,4	99,9	320,9	8,3	41,8
Average fault matrix generation time	2E-04	0,003	0,004	0,007	0,019	0,019	0,051	0,204	0,003	0,014
* for 1024 vectors only										

Рисунок 4.15 – Результати затримки операції моделювання та симуляції

Обмеження MOSI HDL:

1) підтримується тільки комбінаційний Verilog Gate level-код без ієрархії (один модуль без інстанціювання інших модулів) та лише на скалярних провідниках;

2) трасування, хоч і показує високий рівень розведення (понад 90%), у багатьох випадках менше 100% і не гарантує оптимальних результатів;

3) схеми з числом входів в елементах більше 10 вимагають ручного введення тестових векторів;

- 4) розробка ведеться із 2025 року. Репозиторій містить 302 комміти коду, кожному комміту передуює 1–3 години розробки;
- 5) загальний обсяг робіт з кодування можна оцінити у 500 годин.

4.11 Архітектура програми MOSI HDL та приклади роботи

Програма побудована на основі архітектури MVC (Model-View-Controller), в основі якого лежить чітке розподілення ролей і відповідальності за певний функціонал (рис. 4.16). Джерелом даних виступає модуль Model який зберігає всі основні дані програми. Зміни чи маніпуляції над даними полягають в ролі відповідальності Controller, рендиринг і вивід даних для користувача відповідно є прерогативою View.

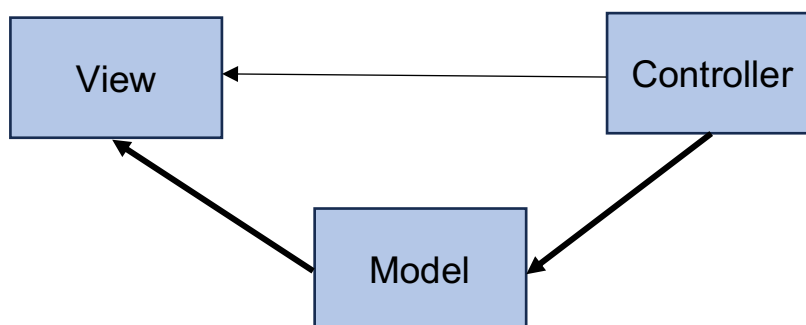


Рисунок 4.16 – Структурна схема архітектури MVC

Такий підхід через відсутність жорсткого зав'язування різних ролей дозволяє досягати масштабованості при обробці великого масиву даних. При необхідності змін чи додавання графічних елементів, таких як таблиць, діалогових вікон, документів, достатньо під'єднатись до існуючих механізмів в модулі View не порушуючи цілісність модулів з іншими ролями. Аналогічно додавання керівної дії (команди) обмежується модифікацією керуючого модуля Controller, а розробка бізнес логіки локалізована в модулі Model.

Архітектура MVC окрім чіткого розподілення ролей передбачає їх оркестрування через визначені сигнали управління. Так, між модулями Model і Controller існує зв'язок для заповнення моделі даними від користувача внаслідок дій останнього. Необхідний також зв'язок між модулями Controller та View для передачі сигналу оновлення даних в графічних елементах. При цьому самі дані для відображення модуль View отримує безпосередньо з модуля Model, чим і пояснюються зв'язок між ними.

Реалізації цих сигналів управління може бути реалізована в різний спосіб, але основний принцип залишається незмінним – кожен з модулів повинен обмежувати свою діяльність своєю роллю.

Для взаємодії між модулями Model та Controller (рис. 4.17) використано модифікований паттерн Command processor. В його основі лежить простий процесор, роль якого зводиться до перекерування назви команди разом з її аргументами до відповідних функцій Controller. Кожна з таких функцій виконує конкретну бізнес логіку через інтерфейс доступний в Document.

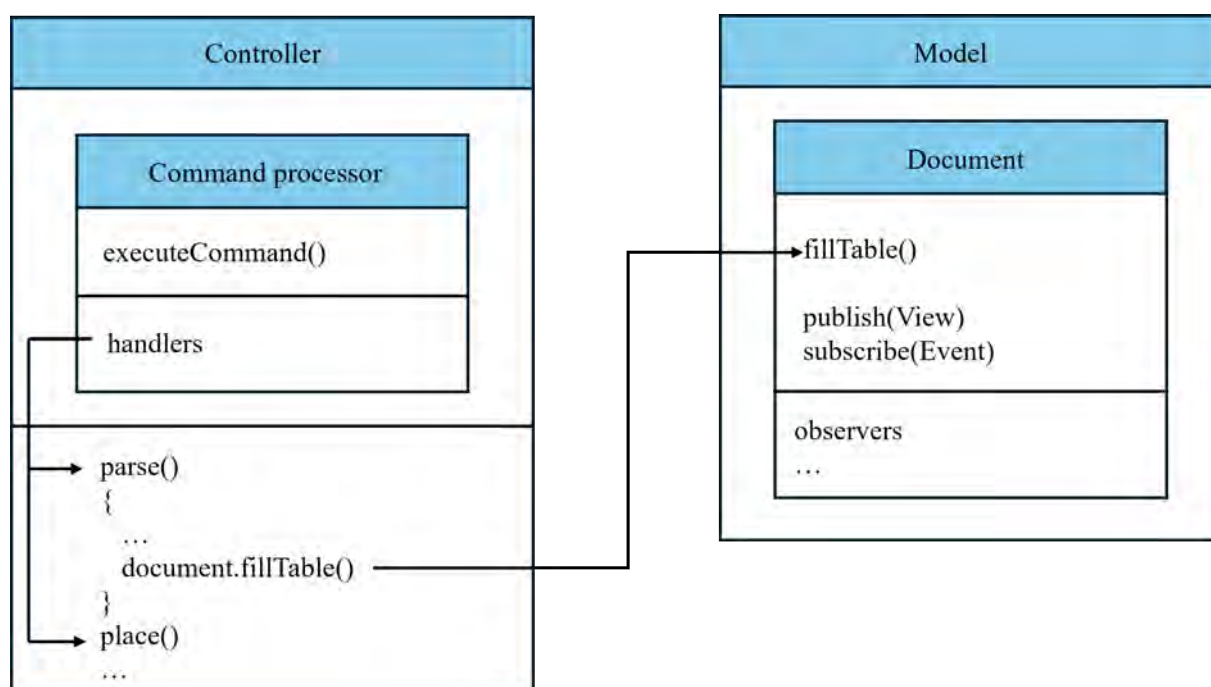


Рисунок 4.17 – Схема взаємодії модулів Controller та Model

Для реалізації зв'язку між модулями Model та Controller використано модифікований паттерн Publisher-Subscriber (рис. 4.18). В ролі Publisher виступає Document, а як Subscriber виступають всі графічні елементи модуля View. Після виконання бізнес логіки в Document виконується процедура publish з відповідним Event в залежності від виконуємої дії (заповнення таблиці істинності, матриці несправності, завантаження схеми тощо). Дана процедура оповіщує всіх своїх спостерігачів observers. З цього моменту модуль View, отримавши виклик процедури update(Event), диспетчеризує обробку до потрібного графічного елементу. Далі сам графічний елемент, маючи доступ на зчитування Document, виконує необхідні дії для свого завантаження.

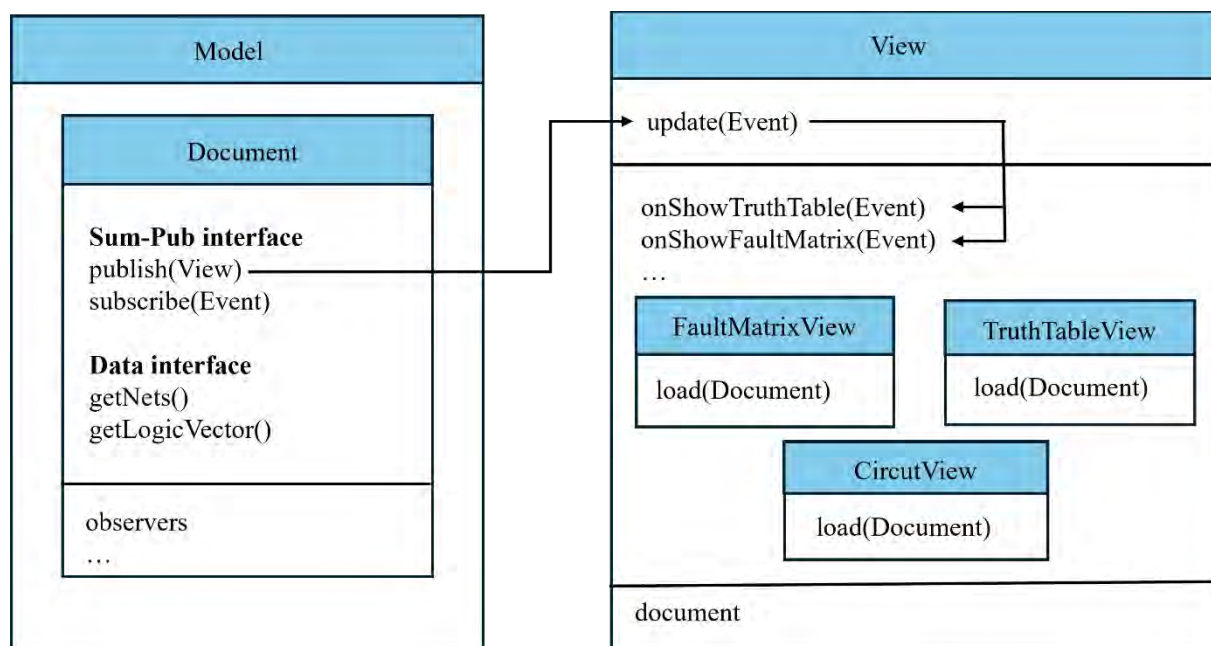


Рисунок 4.18 – Схема взаємодії модулів Model та Controller

Зв'язок між модулями Controller та View найменш завантажений з усіх так як його ціль зводиться до забезпечення керування і обв'язки модулів між собою, основний же потік даних проходить між Controller – Model та Model – View (рис. 4.19).

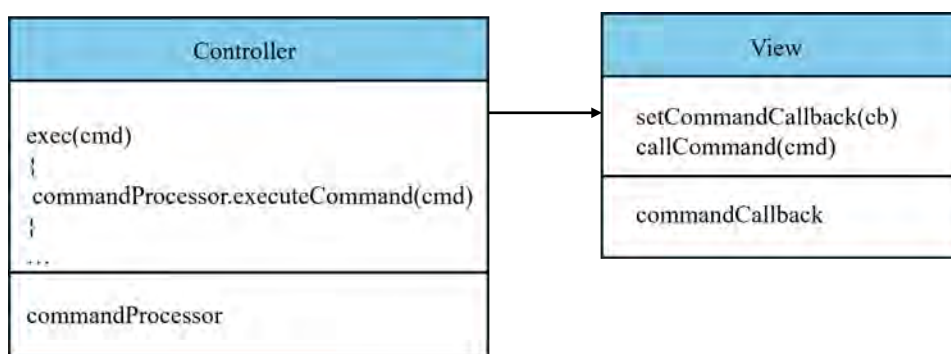


Рисунок 4.19 – Схема взаємодії модулів Controller та View

4.12 Обробка даних в MOSI HDL

Основним джерелом даних в MOSI-HDL (<http://mosihdl.work>) виступає Verilog файл, який описує комбінаційну схему за допомогою вентилів, з'єднаних скалярними провідниками без ієрархії. Як Verilog парсер використаний SureLog, який є open-source парсером, компілятором та елаборатором Verilog/SystemVerilog від ChipsAlliance [31].

Обробка даних запускається вибором файлу за допомогою діалогового вікна Open file (рис. 4.20), що в свою чергу запускає пакет з кількох команд. Перша з них – команда “parse”, яка відповідає за парсинг файлу. Розпарсувавши файл в SureLog, можливо побудувати універсальну апаратну модель бази даних (Universal Hardware Database Model – UHDM). Це в свою чергу дозволяє, використовуючи Verilog Procedural Interface (VPI), який є частиною IEEE 1364, прочитати тип вентилей в оригінальному файлі, а також їхні з'єднання (порти та провідники). Після цього відбувається створення Document як основа моделі даних в модулі Model.

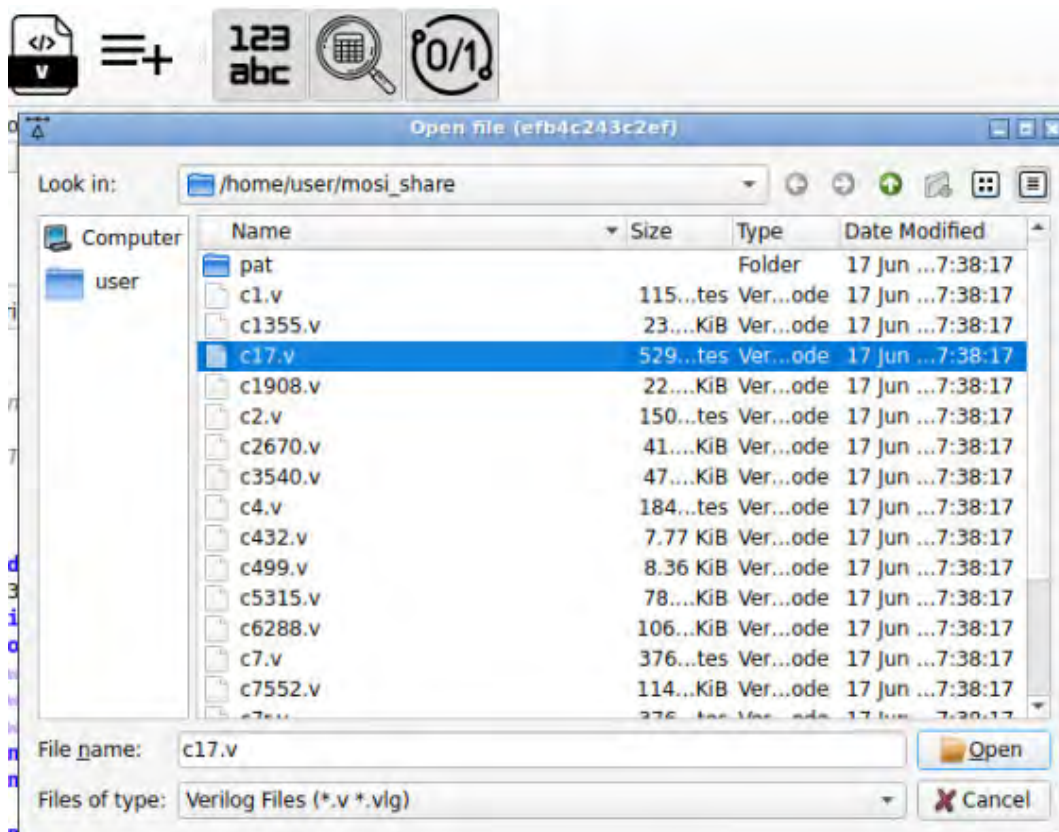


Рисунок 4.20 – Діалогове вікно вибору Verilog файлу

Після створення Document, починається виконання іншої команди – “place”, яка відповідає за створення пошарової репрезентації графа (стиль візуалізації Козо Сігуями) через засоби бібліотеки GraphViz (рис. 4.21). Пошарове розміщення елементів графа передбачає горизонтальне або вертикальне розміщення вершин таким чином щоб найближчі вершини знаходись поруч. Такий стиль розміщення найкраще відповідає візуалізації цифрових схем.

Маючи готовий граф розміщення, в якому кожній з вершин (вентиль або порт) присвоєна координата x, y, починається виконання третьої команди “route”. Її ціль полягає в побудові мінімального прямокутного дерева Штейнера (RSMT), іншими словами найкоротшої мережі що з’єднує кожен термінал вентиля в межах одного еквіпотенціального провідника, причому з’єднання проводяться під прямим кутом.

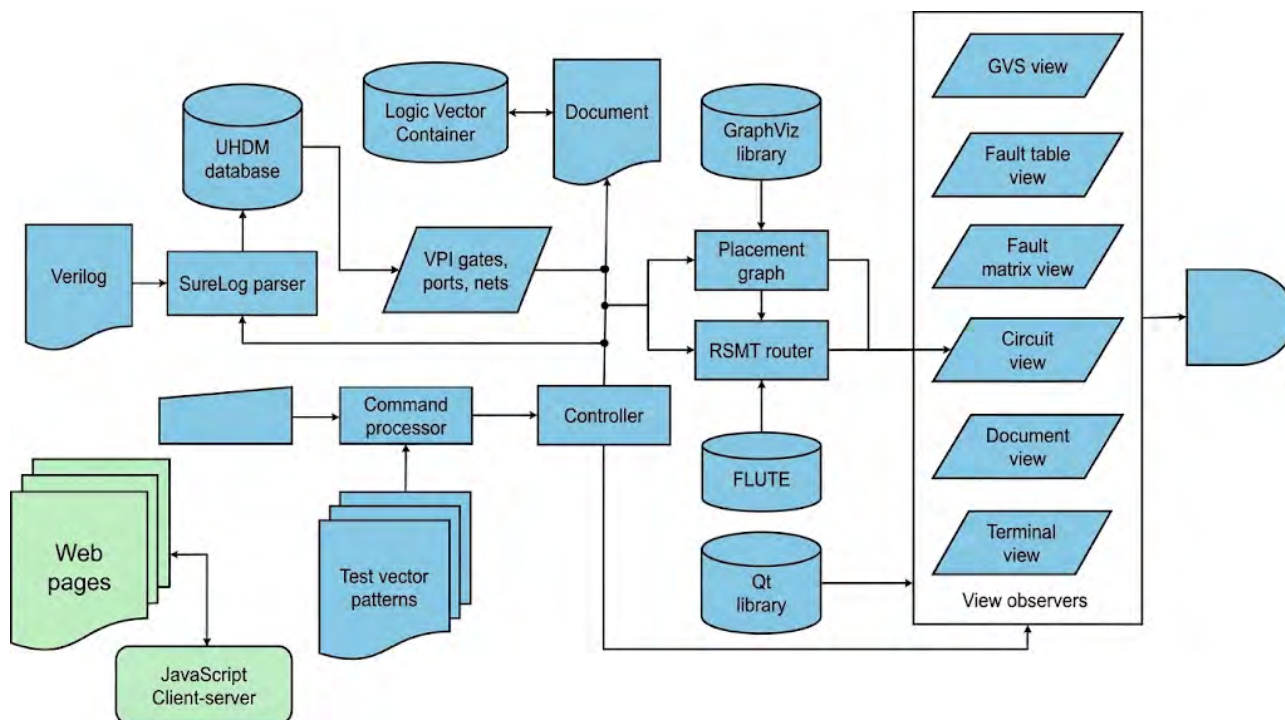


Рисунок 4.21 – Схема обробки даних в MOSI HDL

Для вирішення задачі трасування відбувається попереднє зонування відповідно до інтенсивності нагромадження вузлів. В результаті отримуємо певні менш заповнені зони (канали) які вигідно використати для першої фази трасування. Після отримання координат таких каналів і терміналів ці дані передаються пакету FLUTE [32] для швидкого трасування. Слід підкреслити, що це не гарантує безпомилкового трасування, провідник може потрапити до забороненої зони попередньо розведеного провідника. Тому після першої фази трасування відбувається перевірка на конфлікт з існуючими елементами. В разі виявлення конфліктів запускається друга фаза трасування, заснована на відомому алгоритмі пошуку в лабіринті A^* . Трасування для більших схем (ISCAS C432 і більші) хоча і виконуються на більше ніж 90%, тим не менш має існуючі проблеми, які плануються вирішити в майбутньому.

Після закінчення роботи команди “route” відбувається рендеринг вікон схеми, текстового вікна з файлом Verilog і статистична інформація у вікні

терміналу. На даному етапі є можливим візуально аналізувати схему, проводити звичайні віконні операції такі як зум чи скроллінг (рис. 4.22).

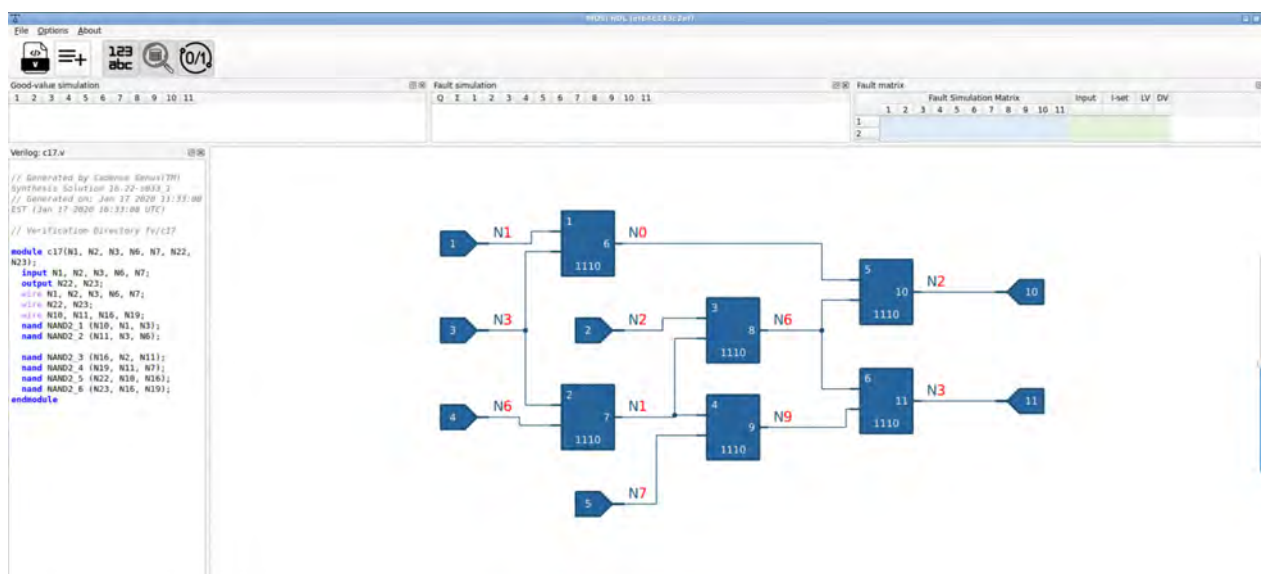


Рисунок 4.22 – Вікно MOSI HDL після завантаження файлу ISCAS C17

Подальша робота передбачає завантаження тестових векторів. З цією метою існує спеціалізоване діалогове вікно Test Vectors Generation. Дане діалогове вікно передбачає три способи вводу тестових векторів:

- 1) Sequential – генерування пакету векторів від початкового і до кінцевого вектору;
- 2) Manual – ручний ввід одного або декількох векторів;
- 3) File – зчитування файлу з готовими векторами.

Для невеликих схем, таких як ISCAS C17, можлива генерація повного пакету векторів, так як схема має всього 5 входів і відповідно обмежується 25 векторами (рис. 4.23).

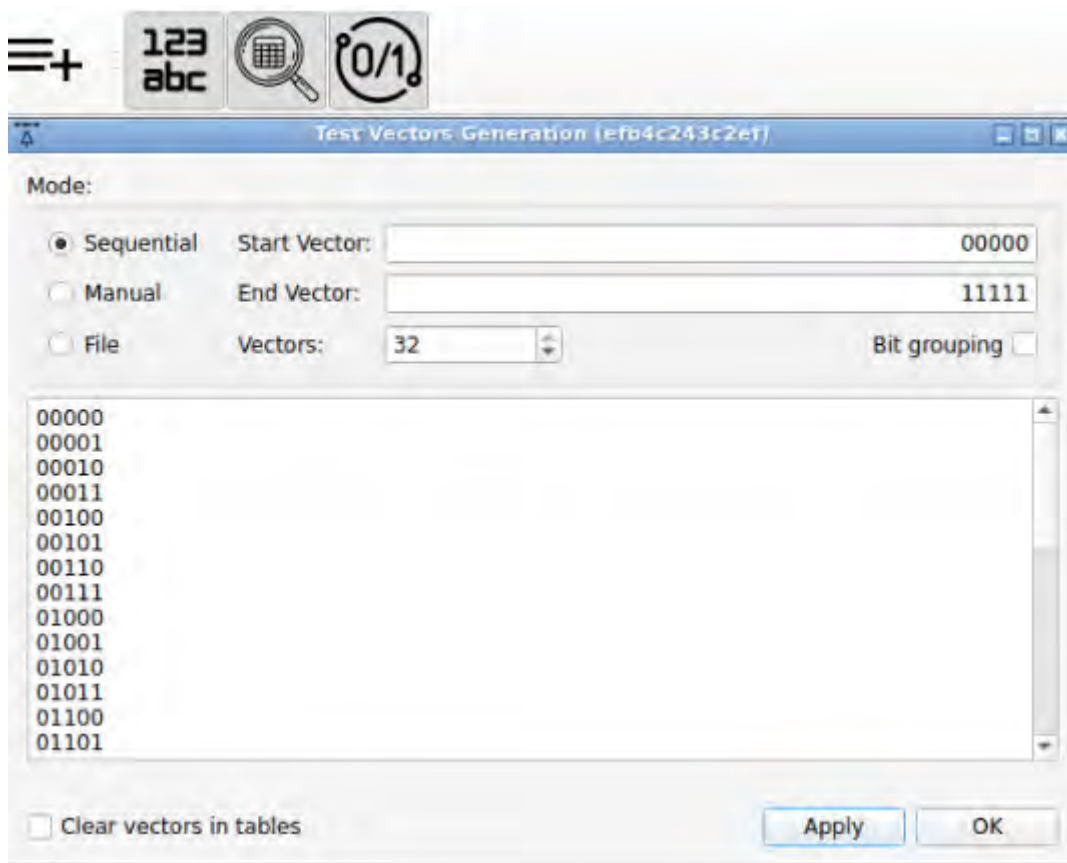


Рисунок 4.23 – Діалогове вікно Test Vectors Generation з 32 векторами

Прийняття даних векторів запускає пакет з кількох команд, які можна спостерігати в терміналі.

Зокрема, запускається команда `generatetruthtable`, яка відповідає за заповнення таблиці Good-value simulation значеннями справного моделювання для кожного з провідників у схемі.

Наприклад: `generatetruthtable 00000 00001 ..... 11110 11111`

Аналогічно для кожного з тесових векторів відбувається симуляцій несправностей з метою визначення типу несправності і на якому з провідників вона проявляється (рис. 4.24).

Good-value simulation												Fault simulation													
	1	2	3	4	5	6	7	8	9	10	11		Q	Σ	1	2	3	4	5	6	7	8	9	10	11
00000	0	0	0	0	0	1	1	1	1	0	0	00000	0.32	0.32		1			1	0		0	0	1	1
00001	0	0	0	0	1	1	1	1	0	0	1	00001	0.36	0.50		1			0	0	0	0	1	1	0
00010	0	0	0	1	0	1	1	1	1	0	0	00010	0.32	0.50		1			1	0		0	0	1	1
00011	0	0	0	1	1	1	1	1	0	0	1	00011	0.41	0.55		1	1		0	0	0	0	1	1	0
00100	0	0	1	0	0	1	1	1	1	0	0	00100	0.36	0.59	1	1			1	0		0	0	1	1
00101	0	0	1	0	1	1	1	1	0	0	1	00101	0.45	0.64	1	1		1	0	0	0	0	1	1	0
00110	0	0	1	1	0	1	0	1	1	0	0	00110	0.27	0.64	1					0		0	0	1	1
00111	0	0	1	1	1	1	0	1	1	0	0	00111	0.41	0.77	1		0	0		0	1	0	0	1	1
01000	0	1	0	0	0	1	1	0	1	1	1	01000	0.23	0.91		0					0	1		0	0
01001	0	1	0	0	1	1	1	0	0	1	1	01001	0.23	0.91		0					0	1		0	0
01010	0	1	0	1	0	1	1	0	1	1	1	01010	0.27	0.91		0	1				0	1		0	0
01011	0	1	0	1	1	1	1	0	0	1	1	01011	0.27	0.91		0	1				0	1		0	0
01100	0	1	1	0	0	1	1	0	1	1	1	01100	0.27	0.91		0		1			0	1		0	0
01101	0	1	1	0	1	1	1	0	0	1	1	01101	0.27	0.91		0		1			0	1		0	0
01110	0	1	1	1	0	1	0	1	1	0	0	01110	0.41	0.91	1		0	0		0	1	0	0	1	1
01111	0	1	1	1	1	1	0	1	1	0	0	01111	0.41	0.91	1		0	0		0	1	0	0	1	1
10000	1	0	0	0	0	1	1	1	1	0	0	10000	0.36	0.91		1	1		1	0		0	0	1	1
10001	1	0	0	0	1	1	1	1	0	0	1	10001	0.41	0.91		1	1		0	0	0	0	1	1	0
10010	1	0	0	1	0	1	1	1	1	0	0	10010	0.36	0.91		1	1		1	0		0	0	1	1
10011	1	0	0	1	1	1	1	1	0	0	1	10011	0.41	0.91		1	1		0	0	0	0	1	1	0
10100	1	0	1	0	0	0	1	1	1	1	0	10100	0.41	1.00	0	1	0		1	1		0	0	0	1
10101	1	0	1	0	1	0	1	1	0	1	1	10101	0.41	1.00	0		0	1	0	1	0		1	0	0
10110	1	0	1	1	0	0	0	1	1	1	0	10110	0.32	1.00	0		0			1		0	0	0	1
10111	1	0	1	1	1	0	0	1	1	1	0	10111	0.41	1.00	0		0	0		1	1	0	0	0	1
11000	1	1	0	0	0	1	1	0	1	1	1	11000	0.23	1.00		0					0	1		0	0
11001	1	1	0	0	1	1	1	0	0	1	1	11001	0.23	1.00		0					0	1		0	0
11010	1	1	0	1	0	1	1	0	1	1	1	11010	0.27	1.00		0	1				0	1		0	0
11011	1	1	0	1	1	1	1	0	0	1	1	11011	0.27	1.00		0	1				0	1		0	0
11100	1	1	1	0	0	0	1	0	1	1	1	11100	0.27	1.00		0		1			0	1		0	0
11101	1	1	1	0	1	0	1	0	0	1	1	11101	0.18	1.00				1			0			0	0
11110	1	1	1	1	0	0	0	1	1	1	0	11110	0.41	1.00	0		0	0		1	1	0	0	0	1
11111	1	1	1	1	1	0	0	1	1	1	0	11111	0.41	1.00	0		0	0		1	1	0	0	0	1

Рисунок 4.24 – Команда і таблиці Good-value simulation та Fault Simulation після загрузки 32 векторів для файла ISCAS C17

Так, для наведеного вище прикладу для вектору 01111 (див. рис. 4.24) можна виявити константу несправність 1 для ліній 1, 7, 10, 11 і константу несправність 0 для ліній 3, 4, 6, 8, 9. Така ж інформація дублюється на схемі у вигляді червоних 0 або 1, що означає виявлення константої несправності даного виду (рис. 4.25).

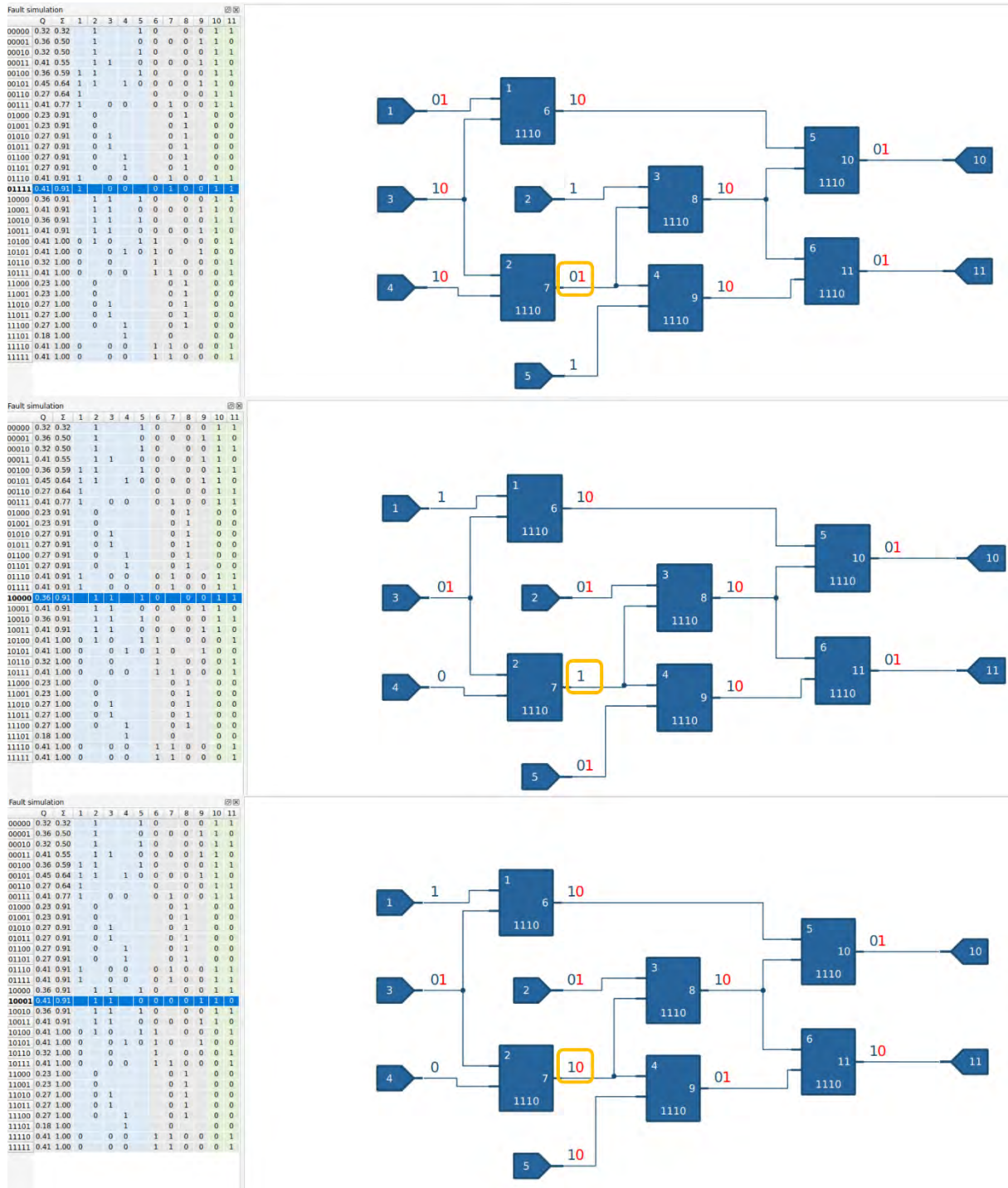


Рисунок 4.25 – Візуалізація константої несправності на схемі залежно від обраного тестового вектору

Слід звернути увагу, що для лінії 7 тестові вектори:

01111 виявляє константу несправність 1;

10000 не виявляє жодної констатної несправності;

10001 виявляє константу несправність 0.

Слід написати тестову процедуру у Verilog, яка буде вносити константи несправності на лінії 7. Для цього необхідно дізнатись, якій назві провідника відповідає лінія 7. Щоб це зробити, достатньо виключити опцію Show Logic/Fault і побачити на схемі, що лінія 7 відповідає провіднику N11. Аналогічно можна переключити опцію Nets as address у звичайний режим назв провідників і побачити у таблицях, що лінії 7 відповідає той самий провідник N11 (листинг 4.1).

Лістинг 4.1 – Тестбенч для ISCAS C17 на лінії 7 (N11)

для векторів 01111, 10000, 10001

```

1 module c17_tb;
2
3 reg [1:5] v;
4 wire o10, o11;
5
6 //module c17(N1, N2, N3, N6, N7, N22, N23);
7
8 c17 u1(.N1(v[1]), .N2(v[2]), .N3(v[3]), .N6(v[4]), .N7(v[5]), .N22(o10), .N23(o11));
9
10 task apply_vector_and_stuck(input [1:5] vec);
11 begin
12     $display("[%2t]: Start - GVS for %b", $time, vec);
13     v = vec;
14     #1;
15
16     force u1.N11 = 1'b1;
17     $display("[%2t]: S-a-1 at N11", $time);
18     #1;
19
20     release u1.N11;
21     force u1.N11 = 1'b0;
22     $display("[%2t]: S-a-0 at N11", $time);
23     #1;
24     release u1.N11;
25     $display("[%2t]: End - GVS for %b", $time, vec);
26 end
27 endtask
28
29 initial begin
30     $monitor("[%2t]: vector = %b outputs = %b%b",
31             $time, v, o10, o11);
32
33     #0;
34     apply_vector_and_stuck(5'b01111);
35     #10;
36
37     apply_vector_and_stuck(5'b10000);
38     #10;
39
40     apply_vector_and_stuck(5'b10001);
41     #10;
42 end

```

Скористаємось симулятором Verilog Icarus. Для цього зкомпілюємо файли c17 та c17_tb та виконаємо симуляцію (лістинг 4.2).

Лістинг 4.2 – Команди Icarus та відповідні результати тестбенчу для ISCAS C17 з несправностями на лінії 7 (N11)

```
iverilog.exe -o sim.vvp c17_tb.v c17.v
```

```
vvp.exe sim.vvp
```

```
[ 0]: Start - GVS for 01111
[ 0]: vector = 01111 outputs = 00
[ 1]: S-a-1 at N11
[ 1]: vector = 01111 outputs = 11
[ 2]: S-a-0 at N11
[ 2]: vector = 01111 outputs = 00
[ 3]: End - GVS for 01111
[13]: Start - GVS for 10000
[13]: vector = 10000 outputs = 00
[14]: S-a-1 at N11
[15]: S-a-0 at N11
[16]: End - GVS for 10000
[26]: Start - GVS for 10001
[26]: vector = 10001 outputs = 01
[27]: S-a-1 at N11
[28]: S-a-0 at N11
[28]: vector = 10001 outputs = 00
[29]: End - GVS for 10001
[29]: vector = 10001 outputs = 01
```

Проаналізуємо результати симуляції, використовуючи матрицю моделювання несправностей як референсне джерело.

В час 0 (див. лістинг 4.2) схема немає жодної несправності і демонструє істинні результати для вектору 01111 на вихідних лініях 10, 11 (N22, N23) значення 00.

В час 1 (див. лістинг 4.2) у схему згідно тестбенча вноситься одинична константа несправність 1 і ми помічаємо що вихідні лінії фіксують зміну з 00 на 11.

В час 2 (див. лістинг 4.2) у схему згідно тестбенча вноситься одинична константа несправність 0 і ми помічаємо що вихідні лінії фіксують стан 00 такий як і очікувалось би для істинного результату.

Проміжний висновок: для вектора 01111 результати симуляції доводять що поодинокі константа несправність 1 пропагується на виходи тоді як поодинокі константа несправність 0 маскується (не пропагується на виходи). Цей же висновок підтверджується матрицею несправностей яка показує для лінії 7 (N11) активні лінії 10, 11 (N22, N23) для несправності 1 (червоні квадрати з цифрами 1) – рис. 4.26.

Fault Simulation Matrix for 01111													Input		I-set		LV	DV
	N1	N2	N3	N6	N7	N10	N11	N16	N19	N22	N23							
N1	1																	
N2		1																
N3			1															
N6				1														
N7					1													
N10	1					1						1	3	0	1	1110 0010		
N11			1	1			1					3	4	1	1	1110 0111		
N16			1	1			1	1				2	7	1	0	1110 0100		
N19			1	1			1		1			7	5	0	1	1110 0010		
N22	1		1	1		1	1	1		1		6	8	1	1	1110 0111		
N23			1	1			1	1	1		1	8	9	1	1	1110 0111		
sum	1		1	1		1	1	1	1	1	1							
good	0	1	1	1	1	1	0	1	1	0	0							
fault	1		0	0		0	1	0	0	1	1							

Рисунок 4.26 – Матриця несправностей для вектора 01111

В час 13 схема немає жодної несправності і демонструє істинні результати для вектору 10000 на вихідних лініях 10, 11 (N22, N23) значення 00.

В час 14 і 15 в схему вносяться одиничні константа несправності 1 та 0, але зміни стану виходів не спостерігається впродовж дії даного вектору.

Проміжний висновок: для вектора 10000 результати симуляції доводять що одинична константа несправність як 1 так і 0 маскується (не пропагується на виходи). Це й же висновок підтверджується матрицею несправностей яка показує для лінії 7 (N11) відсутність активних ліній (пусті квадрати сірого кольору).

Fault matrix																
Fault Simulation Matrix for 10000												Input		I-set	LV	DV
	N1	N2	N3	N6	N7	N10	N11	N16	N19	N22	N23					
N1	1															
N2		1														
N3			1													
N6				1												
N7					1											
N10			1			1						1	3	1	0	
N11							1					3	4	0	0	
N16		1						1				2	7	0	1	
N19					1				1			7	5	1	0	
N22		1	1			1		1		1		6	8	1	1	
N23		1			1			1	1		1	8	9	1	1	
sum		1	1		1	1		1	1	1	1					
good	1	0	0	0	0	1	1	1	1	0	0					
fault		1	1		1	0		0	0	1	1					

Рисунок 4.27 – Матриця несправностей для вектора 10000

В час 26 схема немає жодної несправності і демонструє істинні результати для вектору 10001 на вихідних лініях 10, 11 (N22, N23) значення 01.

В час 27 у схему згідно тестбенча вноситься поодинокі константа несправність 1 але стан вихідних ліній не змінюється.

В час 28 у схему згідно тестбенча вноситься поодинокі константа несправність 0 і ми помічаємо що вихідна лінія 11(N23) фіксує зміну стану з 1 на 0.

Проміжний висновок: для вектора 10001 результати симуляції доводять що поодинокі константа несправність 1 маскується (не пропагується на виходи) тоді як поодинокі константа несправність 0 пропагується на вихід. Це й же висновок підтверджується матрицею несправностей, яка показує для лінії 7 (N11) тільки одну активну лінію 11 (N23) для несправності 0 (червоний квадрат з цифрою 1) – рис. 4.28.

Fault Simulation Matrix for 10001												Input	I-set	LV	DV
	N1	N2	N3	N6	N7	N10	N11	N16	N19	N22	N23				
N1	1														
N2		1													
N3			1												
N6				1											
N7					1										
N10			1			1						1	3	1	0
N11							1					3	4	0	0
N16		1						1				2	7	0	1
N19					1				1			7	5	1	1
N22		1	1			1				1		6	8	1	1
N23					1		1				1	8	9	1	0
sum		1	1		1	1	1	1	1	1	1				
good	1	0	0	0	1	1	1	1	0			0	1		
fault		1	1		0	0	0	0	1	1	0				

Рисунок 4.28 – Матриця несправностей для вектора 10001

Розглянемо приклад роботи з вже готовими тестовими послідовностями для схеми ISCAS C499. Як готові тестові послідовності використовується відома програма Atalanta-M в режимі автоматичного генератора тестових послідовностей. Atalanta-M не працює з файлами Verilog, для генерації потрібні файли в форматі BENCH. У випадку ISCAS референсні схеми вже існують в потрібному форматі (рис. 4.29-4.31).

atalanta-M -t c499.pat -W 1 c499.bench

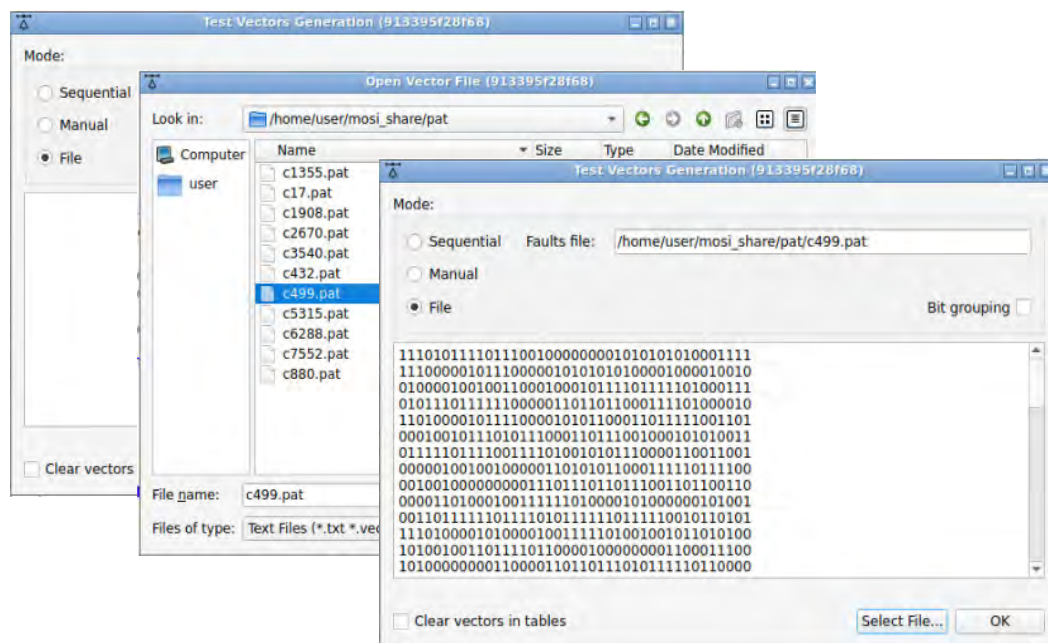


Рисунок 4.29 – Вибір паттерн файл з готовими тестовими послідовностями

Рисунок 4.30 – Результати симуляцій тестових векторів від Atalatna-M для схеми ISCAS C499

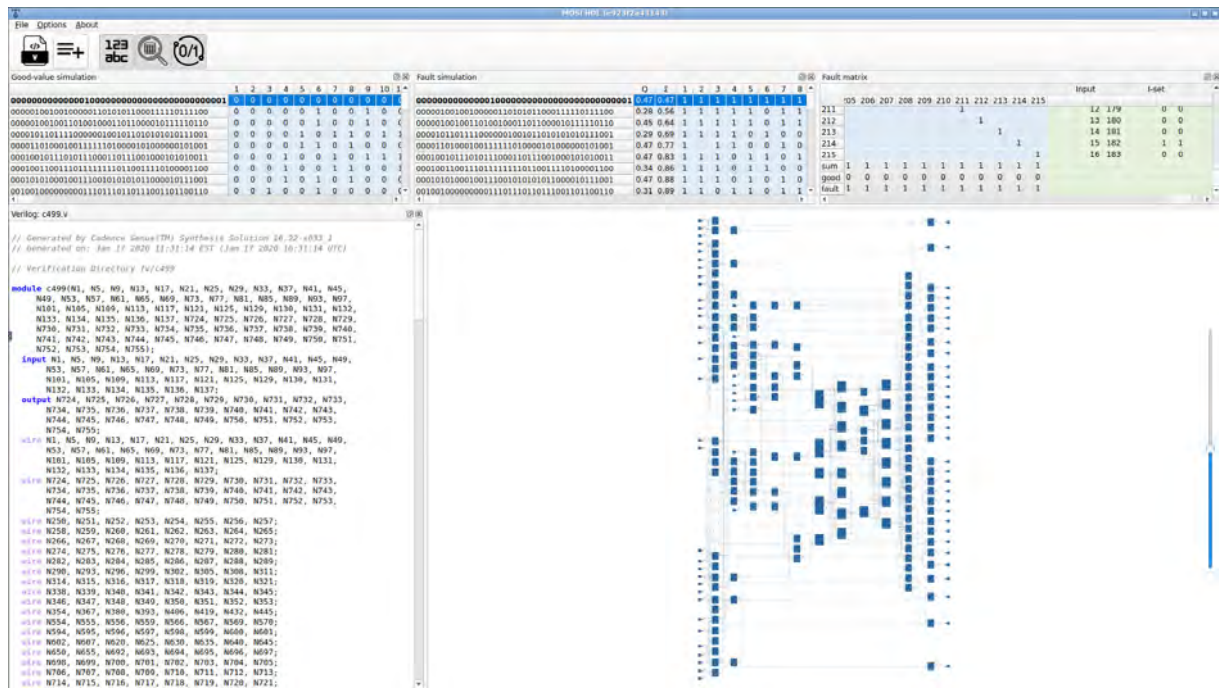


Рисунок 4.31 – MOSI HDL після загрузки файла ISCAS C499

Як видно з таблиці 4.1 з результатами симуляцій для схеми ISCAS C499 MOSI HDL підтверджує достатність зазначених векторів для 100% покриття несправностей.

Таблиця 4.1 – Час в секундах для кожної з дії в середовищі MOSI HDL

Дія	ISCAS85							
	c17	c432	c499	c880	c1355	c1908	c3540	c6288
Verilog парсинг за допомогою Surelog	0.343	1.3	1.4	2.8	4.7	4	9.7	20.3
Розміщення графа схеми за допомогою Graphviz	0.004	0.1	0.3	0.3	1.1	1.5	10.9	60.2
Трасування графа схеми	0.003	0.5	0.8	0.6	1.8	4.6	10.3	4.5
Генерація результатів справного моделювання	0.004	2.3	2.3	5.7	6.9	8.3	16.9	26.7
Генерація таблиці несправностей для перших 1024 векторів	0.005	3.4	4.1	7.6	19.2	19	52	209.3
Повний час	0.359	7.7	8.8	17	33.7	37.4	99.9	320.9
Середній час на генерування 1 вектора	0.0002	0.003	0.004	0.007	0.019	0.019	0.051	0.204

У межах подальшого розвитку системи розробляється web-модуль, який із використанням контейнеризації Docker дозволить розгортати програму у вигляді web-сервісу. Програмний комплекс підтримується на операційних системах Microsoft Windows, Linux та macOS

4.13 Висновки до розділу 4

Запропоновано сервіс MOSI-HDL (<http://mosihdl.work>) для економічного вирішення задач modelling for simulation (MOSI) на основі векторно-логічного in-memory комп'ютингу, що орієнтований на використання read-write транзакцій без інструкцій процесора.

Розроблено парсер-механізм для конвертації HDL коду логічної схеми у внутрішні векторно-логічні структури даних сервісу MOSI.

Здійснено modeling векторно-логічної моделі цифрової схеми для good-value simulation вхідних тестових наборів, як адрес логічних векторів елементів.

Запропоновано механізм генерування дедуктивних векторів за векторно-логічною моделлю цифрової схеми для fault as address simulation вхідних тестових наборів.

Створено механізм modelling матриці моделювання несправностей, як бітів адрес дедуктивного вектора кожного елемента на тестовому наборі.

Виконано оцінку складності алгоритмів моделювання та показано, що її можна знизити до лінійної.

Виконано рендеринг та синхронізацію результатів good-value and fault as address simulation у таблицях good-value simulation, fault as address simulation, матриці моделювання несправностей на вхідному наборі та на лініях логічної схеми, що виводиться на екран монітора.

Здійснено кодування механізмів моделювання та симуляції, а також їхню верифікацію на прикладах логічних схем бібліотеки ISCAS.

Представлено час обробки логічних схем та механізмів моделювання несправностей та good-value simulation.

Наукова новизна дослідження представлена векторно-логічними моделями елементів цифрової схеми, механізмами good-value simulation test set as address та fault as address simulation цифрової схеми і fault as address simulation вхідного набору, використовуючи квадратичну матрицю симуляції.

Реалізовано механізми ергономічного рендерингу та синхронізації результатів моделювання в чотирьох масштабованих вікнах, що виводяться на екран монітора.

Усі згадані інноваційні рішення спрямовані створення економічного масового комп'ютингу. Хмарні сервіси MOSI HDL [24] можуть бути корисними для студентів та інженерів, які займаються верифікацією цифрових проектів на основі векторно-логічних елементів.

Результати розділу відображено у роботах автора [26-30].

4.14 Перелік джерел посилання до розділу 4

1. Sifakis J. System design automation: challenges and limitations // Proceedings of the IEEE. – 2015. – Vol. 103, No. 11. – P. 2093–2103. – DOI: 10.1109/JPROC.2015.2484060.

2. Zhao B., et al. EDA-Q: Electronic Design Automation for Superconducting Quantum Chip // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2026. – Vol. 45, No. 1. – P. 266–279. – DOI: 10.1109/TCAD.2025.3580341.

3. Du H., Du X., Yang J. ASGF4EPD: An Automated Schematic Generation Framework for Electronic Products Design // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2026. – DOI: 10.1109/TCAD.2026.3652497.

4. Srilakshmi S., Madhumati G. L. A comparative analysis of HDL and HLS for developing CNN accelerators // 2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS), Coimbatore, India. – 2023. – P. 1060–1065. – DOI: 10.1109/ICAIS56108.2023.10073746.

5. Lee K.-J., Hsieh T.-Y., Chang C.-Y., Hong Y.-T., Huang W.-C. Platform design based on IEEE 1500 standard // IEEE Transactions on Very Large Scale Integration (VLSI) Systems. – 2010. – Vol. 18, No. 7. – P. 1134–1139. – DOI: 10.1109/TVLSI.2009.2019978.

6. Restuccia F., Kastner R. Cut and Forward: Safe and secure communication for FPGA System on Chips // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2022. – Vol. 41, No. 11. – P. 4052–4063. – DOI: 10.1109/TCAD.2022.3197343.

7. Kovachev D., Belchev S., Nedeva D., Uzuntonov T. Some guidelines on HDL selection for Digital Systems Design // 2021 International Conference on Biomedical Innovations and Applications (BIA), Varna, Bulgaria. – 2022. – P. 77–80. – DOI: 10.1109/BIA52594.2022.9831236.

8. Froehlich S., Drechsler R. LiM-HDL: HDL-Based Synthesis for In-Memory Computing // 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), Antwerp, Belgium. – 2022. – P. 1395–1400. – DOI: 10.23919/DATE54114.2022.9774627.

9. Nakamura S., Nakayama H., Onuma R., Tachibana J., Kaminaga H., Miyadera Y. A system for identification of stumbles in construction of program logic that does not appear as compilation errors // 2022 IEEE International Conference on Computing (ICOCO), Kota Kinabalu, Malaysia. – 2022. – P. 43–48. – DOI: 10.1109/ICOCO56118.2022.10031920.

10. Xiong Z.-Y., Yan Z.-X. Research on audio playing system design factors modeling based on interpretive structure model // 2021 International Conference on Machine Learning and Intelligent Systems Engineering (MLISE), Chongqing, China. – 2021. – P. 33–37. – DOI: 10.1109/MLISE54096.2021.00014.

11. Pouti N. Interpretive structural modeling of uncertainty reduction factors in social commerce // 2025 11th International Conference on Web Research (ICWR), Tehran, Iran. – 2025. – P. 88–99. – DOI: 10.1109/ICWR65219.2025.11006245.

12. Darabi N., Shukla P., Jayasuriya D., Kumar D., Stutts A. C., Trivedi A. R. Navigating the unknown: uncertainty-aware compute-in-memory autonomy of edge robotics // 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), Valencia, Spain. – 2024. – P. 1–6. – DOI: 10.23919/DAT58400.2024.10546628.

13. Shanbhag N. R., Roy S. K. Benchmarking In-Memory Computing Architectures // IEEE Open Journal of the Solid-State Circuits Society. – 2022. – Vol. 2. – P. 288–300. – DOI: 10.1109/OJSSCS.2022.3210152.

14. Hahanov V., Chumachenko S., Litvinova E., Hahanov I., Davitadze Z., Devadze D. Vector logic of computing // 2025 IEEE 13th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Gliwice, Poland. – 2025. – P. 1–5. – DOI: 10.1109/IDAACS68557.2025.11322309.

15. Hahanov V., Gharibi W., Chumachenko S., Litvinova E. Vector synthesis of fault testing map for logic // IAES International Journal of Robotics and Automation (IJRA). – 2024. – Vol. 13, No. 3. – P. 293–306. – DOI: <http://dx.doi.org/10.11591/ijra.v13i3.pp293-306>.

16. Sharma T., Chakraborty I., Ali M., Roy K. Evaluating compute in memory architectures for matrix multiplication: a dataflow-centric perspective // 2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Ghent, Belgium. – 2025. – P. 1–3. – DOI: 10.1109/ISPASS64960.2025.00056.

17. Guo H., Chen J., Jiao H. A 15T SRAM cell-based fully-digital computing-in-memory macro supporting high parallelism and fine-grained simultaneous read + write + MAC operations // IEEE Solid-State Circuits Letters. – 2025. – Vol. 8. – P. 153–156. – DOI: 10.1109/LSSC.2025.3567840.

18. Zhang S., Cui X., Wei F., Cui X. An area-efficient in-memory implementation method of arbitrary Boolean function based on SRAM array // IEEE Transactions on Computers. – 2023. – Vol. 72, No. 12. – P. 3416–3430. – DOI: 10.1109/TC.2023.3301156.

19. Lee H., Lee J., Kang S. An efficient test architecture using hybrid built-in self-test for processing-in-memory // IEEE Transactions on Very Large-Scale Integration (VLSI) Systems. – 2025. – Vol. 33, No. 5. – P. 1452–1456. – DOI: 10.1109/TVLSI.2024.3504539.

20. Kajihara S., Kinoshita K., Pomeranz I., Reddy S. M. Combinationally irredundant ISCAS-89 benchmark circuits // 1996 IEEE International Symposium on Circuits and Systems (ISCAS 96), Atlanta, GA, USA. – 1996. – Vol. 4. – P. 632–634. – DOI: 10.1109/ISCAS.1996.542103.

21. Bryan D. The ISCAS '85 Benchmark Circuits and Their Netlist Format. – Electronic resource: <https://www.scribd.com/doc/164149583/Iscas-85> – Accessed 09.02.2026.

22. Cuijie Y., Biyan L. Research on optimization strategies for GPU cluster cloud rendering virtual simulation resources // 2024 14th International Conference on Information Technology in Medicine and Education (ITME), Guiyang, China. – 2024. – P. 392–395. – DOI: 10.1109/ITME63426.2024.00084.

23. Zhu S., Yu T., Xu T., Chen H., Dustdar S., Gigan S., Gunduz D., Hossain E., Jin Y., Lin F., Liu B., Wan Z., Zhang J., Zhao Z., Zhu W., Chen Z., Durrani T. S., Wang H., Wu J., Zhang T., Pan Y. Intelligent Computing: The Latest Advances, Challenges, and Future / S. Zhu, T. Yu, T. Xu, H. Chen, S. Dustdar, S. Gigan, D. Gunduz, E. Hossain, Y. Jin, F. Lin, B. Liu, Z. Wan, J. Zhang, Z. Zhao, W. Zhu, Z. Chen, T. S. Durrani, H. Wang, J. Wu, T. Zhang, Y. Pan // Intelligent Computing. – 2023. – Vol. 2. – Art. 0006. – DOI: 10.34133/icomputing.0006.

24. MOSI HDL – Electronic resource: <http://www.mosihdl.work/>. – Accessed 09.06.2026.

25. Vector Simulation Model – Electronic resource: <https://vector-simulation-model.vercel.app/>. – Accessed 09.06.2026.

26. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector-logic Models of Digital Circuits for Simulation and Rendering / V. Hahanov, S. Chumachenko, E. Litvinova, A. Voronov, O. Demchenko, N. Maksymova // IAES International Journal of Robotics and Automation (IJRA). — 2026. — Vol. 15, no. 2. — P. 319–330. — DOI: 10.11591/ijra.v15i2.pp319-330.

27. Hahanov V., Litvinova E., Chumachenko S., Hahanov I., Demchenko O., Voronov A. Vector Logic Modeling Self-Tested Circuits / V. Hahanov, E. Litvinova, S. Chumachenko, I. Hahanov, O. Demchenko, A. Voronov // 2025 IEEE East-West Design & Test Symposium (EWDTS), Tbilisi, Georgia, 12–15 Dec. 2025 : proceedings. — 2025. — 4 p. — DOI: 10.1109/EWDTS67441.2025.11303706.

28. Демченко О.І. Програмне середовище MOSI HDL для моделювання векторно-логічних структур / О.І. Демченко // Радіоелектроніка та молодь у XXI столітті : матеріали XXX міжнар. молодіж. форуму, 22–24 квіт. 2026 р., Харків. — Харків : ХНУРЕ, 2026. — Т. 5. — С. 42–43.

29. Хаханов І.В., Кулак Г.К., Воронов А.О., Демченко О.І., Максимова Н.Г., Пономарьова В.І. (науковий керівник — д-р техн. наук, проф. Хаханов В.І.). Розробка «Хмарний сервіс MOdeling for Simulation (MOSI)» / І.В. Хаханов, Г.К. Кулак, А.О. Воронов, О.І. Демченко, Н.Г. Максимова, В.І. Пономарьова // XXIX Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Каталог виставки технічної творчості, 16–19 квіт. 2025 р. — 2025. — С. 23-24.

30. Демченко О.І., Воронов А.О., Максимова Н.Г. (науковий керівник — д-р техн. наук, проф. Хаханов В.І.). Розробка «HDL Modeling for Simulation Cloud Service (MOSI-HDL)» / О.І. Демченко, А.О. Воронов, Н.Г. Максимова // XXX Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Каталог виставки технічної творчості молоді, 22–24 квіт. 2026 р. — 2026. — С. 32. — (англійською мовою).

31. Dargelas A., Zeller H. Universal Hardware Data Model / A. Dargelas, H. Zeller // Proceedings of the Workshop on Open-Source EDA Technology (WOSET) 2020. — 2020. — <https://woset-workshop.github.io/PDFs/2020/a10.pdf>

32. Chu C., Wong Y.-C. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design / C. Chu, Y.-C. Wong // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2008. – Vol. 27, no. 1. – P. 70–83. – Jan. – DOI: 10.1109/TCAD.2007.907068.

ВИСНОВКИ

Дослідження було присвячене актуальній на EDA-ринку науково-практичній задачі щодо розробки економічно ефективного векторно-логічного modelling for simulation – моделінг в пам'яті (in-memory) для симуляції несправностей як адрес цифрових схем, заснованих на транзакціях читання-запису (read-write) без інструкцій процесора.

Досягнуто мету дослідження – зниження часових та енергетичних витрат fault/good-value as address simulation цифрових схем за рахунок надлишковості векторно-логічних моделей їх елементів. Виконання задач дослідження дозволило отримати такі результати, що відзначаються науковою новизною та практичною значущістю:

1. Вдосконалені розумні структури даних векторної логіки, які орієнтовані на безпроцесорний векторно-логічний modeling for fault/good-value as address simulation за допомогою read-write транзакцій, що дозволило зменшити обчислювальну складність алгоритмів симуляції до лінійної. Розумні структури даних у вигляді логічних векторів та матриць завдання комбінацій несправностей логічних функцій від n -змінних характеризуються паралелізмом моделювання тестових наборів, як адрес, що дає змогу ефективно обробляти вектори несправностей цифрової схеми та/або логічного елемента.

2. Векторно-логічна модель елементів цифрової схеми, що дозволяє здійснити good-value simulation вхідних тестових наборів та fault as address simulation як адрес бітів логічного та дедуктивного вектора на основі read-write транзакцій та знизити обчислювальну складність алгоритмів тестування логічних функціональностей до лінійної.

3. Архітектура для моделювання схеми та рендерингу процесів синхронізації векторно-логічного modeling for fault/good-value as address simulation, що містить побудову моделі об'єкта у пам'яті комп'ютера та дозволяє проєктувальнику візуально супроводжувати процес modeling for simulation

цифрової схеми (вручну будувати тести для покриття несправностей логічних схем).

4. Механізми векторно-логічного modeling for fault/good-value as address simulation, що відзначаються лінійною алгоритмічною складністю. Описано in-memory механізм good-value simulation цифрової схеми для синтезу логічного вектора з подальшим його використанням для побудови карти тестування несправностей на повному тесті за три матричні операції. Цей in-memory механізм є простим у реалізації, використовує лише read-write транзакції над бітами логічних векторів і не потребує інструкцій процесора. Описано механізм modeling матриці моделювання несправностей, як бітів адрес дедуктивного вектора кожного елемента на тестовому наборі, що масштабується до будь-якої таблиці істинності з n змінних та може бути використаний для виявлення несправностей у тестових наборах функції чи структури.

5. Здійснено верифікацію розумних структур даних та алгоритмів моделювання на десятках схем та функціональних елементів з бібліотеки ISCAS. Розроблено хмарний сервіс MOSI-HDL (<http://mosihdl.work>) для економічного вирішення задач modelling for simulation (MOSI) на основі векторно-логічного in-memory комп'ютингу, що орієнтований на використання read-write транзакцій без інструкцій процесора, який містить: векторно-логічні моделі елементів цифрової схеми для здійснення good-value simulation вхідних тестових наборів та симуляцію несправностей як адрес бітів логічного та дедуктивного вектора на основі read-write транзакцій; механізми good-value simulation of test set as address цифрової схеми без процесора; механізми fault as address simulation цифрової схеми без участі процесора; механізм fault as address simulation вхідного набору з використанням квадратичної матриці симуляції; механізми рендерингу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора.

Інтегральною перевагою або новизною даних сервісів є синхронізація процесів моделювання по всіх вікнах, що представляють всі суттєві процедури для розуміння отриманого результату.

ДОДАТОК А

Список публікацій здобувача, в яких опубліковані основні наукові результати дисертації

1. Демченко О.І., Чумаченко С.В. Технології прогресивного корпусування інтегральних мікросхем. *Системи управління, навігації та зв'язку*. 2025. Т. 3, № 81. С. 199–202. <https://doi.org/10.26906/SUNZ.2025.3.199> (Фахове видання. Категорія Б).

2. Хаханов В.І., Обрізан В.І., Рахліс Д., Хаханова Г.В., Хаханов І.В., Демченко О.І., Воронов А.О. Механізми схемотехнічного моделювання на основі векторної логіки. *Радіoeлектроніка, інформатика, управління*. 2025. № 4. С. 219-232. DOI: 10.15588/1607-3274-2021-0-0 (Фахове видання. Web of Science. Категорія А).

3. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector-logic Models of Digital Circuits for Simulation and Rendering. *IAES International Journal of Robotics and Automation (IJRA)*. 2026. Vol. 15, № 2. P. 319-330. (ISSN: 2089-4856). <http://doi.org/10.11591/ijra.v15i2.pp319-330> (Закордонне видання. Scopus, Q3).

4. Hahanov V., Chumachenko S., Litvinova E., Voronov A., Demchenko O., Maksymova N. Vector logic for robotic system on chip design and test. *IAES International Journal of Robotics and Automation (IJRA)*. 2026. Vol. 15, № 2. P. 415-426. (ISSN: 2089-4856). <http://doi.org/10.11591/ijra.v15i2.pp415-426> (Закордонне видання. Scopus, Q3).

ДОДАТОК Б

Результати, які засвідчують апробацію матеріалів дисертації

5. Vladimir Hahanov, Eugenia Litvinova, Svetlana Chumachenko, Ivan Hahanov, Oleh Demchenko, Andrii Voronov. Vector Logic Modeling Self-Tested Circuits. *2025 IEEE East-West Design & Test Symposium (EWDTS)*. Tbilisi, Georgia, 12–15 December 2025. 4 p. DOI: 10.1109/EWDTS67441.2025.11303706 (Scopus).

6. Чумаченко С. В., Демченко О. І. Технології прогресивного корпусування ІМС: основні виклики. *Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління (Current directions of development of information and communication technologies and control tools): матеріали 15ї міжнар. наук.-техн. конф., Баку – Харків – Жиліна. 24–25 квітня 2025 року. Т. 2. С. 26-27.* <https://repository.kpi.kharkov.ua/handle/KhPI-Press/93979>

7. Демченко О.І. Програмне середовище MOSI HDL для моделювання векторно-логічних структур. *Радіoeлектроніка та молодь у XXI столітті: матеріали XXX міжнар. молодіж. форуму, 22–24 квітня 2026 р. Харків: ХНУРЕ, 2026. Т. 5. С. 42–43.*

8. Хаханов І.В., Кулак Г.К., Воронов А.О., Демченко О.І., Максимова Н.Г., Пономарьова В.І. (науковий керівник – д-р техн. наук, проф. Хаханов В.І.) Розробка «Хмарний сервіс MOdeling for Simulation (MOSI)». XXIX Міжнародний молодіжний форум «Радіoeлектроніка та молодь у XXI столітті». Каталог виставки технічної творчості. 16-19 квітня 2025 року. С. 23-24.

9. Демченко О.І., Воронов А.О., Максимова Н.Г. (науковий керівник – д-р техн. наук, проф. Хаханов В.І.). Розробка «HDL Modeling for Simulation Cloud Service (MOSI-HDL)» (англійською мовою). XXX Міжнародний молодіжний форум «Радіoeлектроніка та молодь у XXI столітті». 22–24 квітня 2026 року. Каталог виставки технічної творчості молоді. С. 32.

ДОДАТОК В

Документи, що підтверджують впровадження результатів



АКТ

про впровадження в освітній процес ХНУРЕ результатів дисертаційної роботи Демченка Олега Івановича «Векторно-логічні моделі цифрових схем для симуляції та візуалізації», представленої на здобуття наукового ступеня доктора філософії за спеціальністю 123 – Комп'ютерна інженерія

Комісія у складі канд. техн. наук, доцента кафедри АПОТ Шкіля О.С., докт. техн. наук, професора кафедри АПОТ Хаханової Г.В., канд. техн. наук, доцента кафедри АПОТ Філіппенко І.В. розглянула матеріали дисертаційної роботи Демченка О.І., які використовуються в освітньому процесі кафедри АПОТ ХНУРЕ у 2025/2026 навчальному році, і прийшла до наступного висновку.

Розроблені у дисертаційній роботі векторно-логічні моделі цифрових схем та механізми для симуляції та візуалізації, а саме:

- векторно-логічні моделі елементів цифрової схеми, що дозволяє здійснювати good-value simulation вхідних тестових наборів та симуляцію несправностей як адрес бітів логічного та дедуктивного вектора на основі read-write транзакцій;
- механізми good-value simulation of test set as address цифрової схеми без процесора;
- механізми fault as address simulation цифрової схеми без участі процесора;
- механізм fault as address simulation вхідного набору з використанням квадратичної матриці симуляції;
- парсер-механізм для конвертації HDL коду логічної схеми у внутрішні векторно-логічні структури даних сервісу MOSI-HDL (MOdeling for SImulation HDL);
- механізми рендерингу та синхронізації результатів моделювання у чотирьох масштабованих вікнах, що виводяться на екран монітора;

впроваджені в освітній процес кафедри АПОТ ХНУРЕ та використовуються у таких навчальних дисциплінах:

1) «Розробка систем на кристалі (SoC)» для бакалаврів освітньої програми F7 Комп'ютерна інженерія у лекційному матеріалі за темами «Опис елементів комбінаційної логіки мовою HDL», «Технології проектування цифрових систем на кристалах»;

2) «Інтелектуальний комп'ютинг» для магістрантів ОПП «Спеціалізовані комп'ютерні системи» спеціальності F7 Комп'ютерна інженерія у лекційному матеріалі за темою «Векторно-логічні моделі структур та функцій» та лабораторному практикумі за темою «Справне моделювання схеми Шнейдера (good-value simulation) на вичерпному тесті».

Результати за висновками комісії внесено до протоколу № 13 від 11.06.2026 засідання кафедри АПОТ.

Підписи членів комісії:
15.06.2026

 Олександр ШКІЛЬ
 Ганна ХАХАНОВА
 Інна ФІЛІППЕНКО

ДОДАТОК Г

Дипломи переможців виставок науково-технічної творчості



**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ**



П О Ч Е С Н А ГРАМОТА

НАГОРОДЖУЄТЬСЯ

**Іван ХАХАНОВ,
Георгій КУЛАК, Андрій ВОРОНОВ,
Олег ДЕМЧЕНКО, Наталія МАКСИМОВА,
Вероніка ПОНОМАРЬОВА**

за здобуття Гран-прі на виставці технічної творчості молоді
у рамках ХХІХ Міжнародного молодіжного форуму
«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У ХХІ СТОЛІТТІ»

за напрямом «Програмно-апаратні розробки, прилади та пристрої»,
представленою інноваційною високотехнологічною розробкою –
хмарним сервісом MOdeling for SIMulation (MOSI)

Науковий керівник – Хаханов Володимир Іванович,
доктор технічних наук, професор, професор кафедри АПОТ ХНУРЕ

Голова конкурсної комісії,
в.о. ректора ХНУРЕ

19 квітня 2025 р.

Юрій РОМАНЕНКОВ

