

МЕХАНІЗМИ СХЕМОТЕХНІЧНОГО МОДЕЛЮВАННЯ НА ОСНОВІ ВЕКТОРНОЇ ЛОГІКИ

Хаханов В. І. – д-р техн. наук, професор кафедри автоматизації проектування обчислювальної техніки, Харківський національний університет радіоелектроніки, Україна.

Обрізан В. І. – канд. техн. наук, докторант кафедри автоматизації проектування обчислювальної техніки, Харківський національний університет радіоелектроніки, Україна.

Рахліс Д. Ю. – канд. техн. наук, доцент кафедри автоматизації проектування обчислювальної техніки, Харківський національний університет радіоелектроніки, Україна.

Хаханова Г. В. – д-р техн. наук, професор кафедри автоматизації проектування обчислювальної техніки, Харківський національний університет радіоелектроніки, Україна;

Хаханов І. В. – канд. техн. наук, асистент кафедри автоматизації проектування обчислювальної техніки, Харківський національний університет радіоелектроніки, Україна;

Демченко О. І. – аспірант кафедри автоматизації проектування обчислювальної техніки, Харківський національний університет радіоелектроніки, Україна.

Воронов А. О. – аспірант кафедри автоматизації проектування обчислювальної техніки, Харківський національний університет радіоелектроніки, Україна.

АНОТАЦІЯ

Актуальність. Розглядаються актуальні для EDA-ринку питання зменшення вартості та часу тестування й верифікації цифрових проєктів шляхом синтезу логічного вектора цифрової схеми, що дозволяє суттєво спростити алгоритми good-value simulation, а синтез карти тестування звести до трьох матричних операцій.

Мета. Мета дослідження – зменшення вартості та часу тестування і верифікації цифрових проєктів завдяки синтезу логічного вектора цифрової схеми, що дозволяє спростити алгоритм побудови карти тестування до трьох матричних операцій.

Метод. Пропонується синтез логічного вектора комбінаційної схеми для good-value та fault як адресів моделювання в архітектурі промпт-інтелектуального in-memoгу комп'ютигу. Логічний вектор є найбільш технологічним, компактним та вичерпним представленням схеми для економного вирішення всіх задач дизайну та тестування. Пропонується 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 процесу або явища, функції або структури на основі матричних структур декартової логіки для вирішення задач Modeling for Simulation. Пропонується 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.

Результати. Запропоновано 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. На основі теорії векторної логіки розроблено механізми та програмне застосування для економного розв'язання задач design and test в архітектурі in-memory комп'ютигу.

Висновки. Практична значущість дослідження орієнтована на вирішення всіх задач design and test на основі використання простих моделей векторної логіки, орієнтованої на економічний in-memory комп'ютиг на основі read-write транзакцій, вільний від інструкцій процесора. Пропонується використовувати логічний вектор специфікації в архітектурі in-memory комп'ютигу для тестування, верифікації, діагностування та експлуатації. In-memory vector logic computing – це економічне вирішення багатьох обчислювальних проблем з точки зору енергетичних, часових і вартісних витрат. Новизна дослідження полягає у використанні векторної логіки в архітектурі in-memory комп'ютигу на основі read-write транзакцій, що дозволяє зменшити споживання ресурсів у вигляді часу та енергії. Імплементація векторної логіки в in-memory комп'ютиг робить його масовим і енергоефективним, вільним від складних алгоритмів аналізу великих даних.

КЛЮЧОВІ СЛОВА: векторна логіка, логічний вектор, декартова логіка, інтелектуальний комп'ютиг, карта тестування, моделювання несправностей, good-value моделювання, in-memory моделювання для симуляції, prompt-інжиніринг.

АБРЕВІАТУРИ

EDA – Electronic Design Automation (Автоматизація проектування електроніки);

GUI – Graphical User Interface (Графічний інтерфейс користувача);

HDL – Hardware Description Language (Мова опису апаратури);

ДНФ (DNF) – диз'юнктивна нормальна форма;

КНФ (CNF) – кон'юнктивна нормальна форма;

AI – Artificial Intelligence (Штучний інтелект);

NI – Number of Inputs (Кількість входів);

MOSI – Modeling for Simulation.

НОМЕНКЛАТУРА

X – вхідні логічні вектори / черговий елемент схеми;

X_i – біт вхідного ефекту / координата вхідного набору;

Y – вихідні логічні вектори / результат декартового добутку;

L – логічний вектор (результат моделювання, матриця активності, специфікація);

L_{ij} – координата логічного вектора;

M – модель (пам'ять, матриця, структура даних);

M_i – біт вектора моделювання / координата пам'яті;

T – таблиця істинності;

A – алгоритм, зокрема перекодування / складність моделі;

F – помилки, несправності;

D – матриця дедуктивних векторів;

H – матриця перекодування;

G – утворювальний логічний вектор (функція декартової операції);

n – кількість бітів (змінних / входів);

m – середня кількість входів у логічному елементі;

k – довжина тесту;

$2^n, 2^m, 2^k$ – розмірність логічного вектора / кількість бітів;

$\otimes$ – декартова логічна операція (комбінаторна суперпозиція);

$\vee, \wedge, \neg$ – диз'юнкція (OR), кон'юнкція (AND), заперечення (NOT).

ВСТУП

Розглядаються актуальні для EDA-ринку питання зменшення вартості та часу тестування й верифікації цифрових проєктів шляхом синтезу логічного вектора цифрової схеми, що дозволяє суттєво спростити алгоритми good-value simulation, а синтез карти тестування звести до трьох матричних операцій. Структури даних для промпт-комп'ютигу (моделювання) логічного вектора схеми визначаються як: 1) два вхідних логічних вектори, довжина яких кратна ступеню двійки; 2) номери для ідентифікації вхідних змінних кожного вектора, які беруться з логічної схеми; 3) результуючий вектор схеми та вектор ідентифікаторів вхідних змінних схеми. Декартовий добуток двох векторів різної або однакової довжини, що відповідають фрагменту схеми з двох елементів, виконується на основі бінарної логічної операції (and, xor, or). Ця операція над векторами дозволяє отримати нові, раніше невідомі властивості в результуючому логічному векторі (матриці) для обробки даних.

Об'єкт дослідження – комп'ютигу на основі векторної і декартової логіки для оцифрованих процесів та явищ.

Предмет дослідження – механізми в моделюванні цифрових проєктів.

Мета дослідження – зменшення вартості та часу тестування і верифікації цифрових проєктів завдяки синтезу логічного вектору цифрової схеми, що дозволяє спростити алгоритм побудови карти тестування до трьох матричних операцій.

1 ПОСТАНОВКА ЗАДАЧІ

Розглядається задача моделювання логічного вектора для цифрової структури. Нехай задано логічні вектори X і Y . Практичне завдання полягає у побудові

матриці відношень за допомогою логічної декартової операції ($\otimes$ -вектора) $L = X \otimes Y$ над бітами двох логічних векторів X і Y з метою вирішення задач modeling for simulation: 1) моделювання L -матриці активності вихідних станів логічної функції Y ; 2) моделювання H -матриці перекодування; 3) побудова матриці дедуктивних векторів $D = L_H$ для моделювання несправностей функціональності або схеми як адрес таблиці істинності; 4) використання декартової AND-логіки для синтезу матриці, яка визначає властивість similarity (схожості) двох векторів у двовимірному просторі; 5) побудови карти тестування схеми за вектором.

Задачі дослідження: 1) аналіз існуючих підходів до побудови логічного вектора процесу або явища, функції чи структури; 2) визначення метрики алгоритмічної складності розв'язання задач моделювання схем на основі елементів або логічного вектора; 3) визначення матричних надлишкових структур декартової логіки для ефективного вирішення задач vector logic modeling and simulation [24–27]; 4) розробка методу прямого паралельного моделювання схеми на основі використання декартової логіки з метою отримання логічного вектора; 5) кодування та верифікація запропонованого методу на різних структурах логічних схем; 6) використання логічного вектора для prompt-інжинірингу інтелектуального комп'ютигу з метою побудови карти тестування.

2 ОГЛЯД ЛІТЕРАТУРИ

Розгалуження в логічній схемі [1–4] породжують однойменні вхідні змінні (координати) на одному або кількох логічних елементах. Така особливість схеми призводить до мінімізації результативного 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, кратна ступеню двійки (рис. 1).

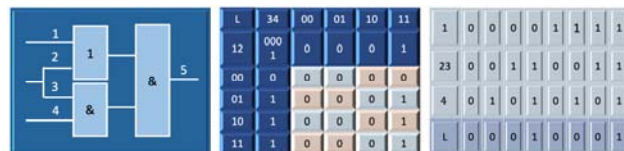


Рисунок 1 – Синтез логічного вектора для фрагмента схеми

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 інтеграцію комп'ютигу. Тому при побудові карти тестування будь-якої логічної функціональності можна використовувати логічні вектори та read-write транза-

кції. 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 для математичного гурмана. Слід мати на увазі, що логічний вектор для будь-якої схеми може бути побудований шляхом рекурсивної суперпозиції завжди 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 логічний вектор схеми, то дуже важко знайти пояснення, чому використовується саме схема замість простого 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.

Сьогодні read-write транзакції в пам'яті мають ту саму затримку (latency – 0,6 нс), що й логічні елементи. При цьому кожен користувач практично не обмежений у розмірах пам'яті, яку можна використати для розв'язання задачі. Висновок простий: векторно-логічний комп'ютинг у пам'яті на основі read-write транзакцій – це майбутнє масових обчислень. Він має мінімальні: вартість створення моделі (яка не потребує синтезу), енергоспоживання, затримку під час експлуатації моделі. До речі, всі компанії, що займаються штучним інтелектом, сьогодні переходять на прості двійкові моделі, розміщені в пам'яті, представлені у вигляді логічних векторів або матриць (а це одне й те саме) [5–10].

Схемотехніка вже 80 років 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. Синтез не потрібен. Потрібне лише моделювання – процес розміщення векторної або матричної моделі в пам'яті. Потім цю двійкову модель можна використати для розв'язання всіх задач штучного інтелекту методами логічного двійкового good-value моделювання на основі read-write транзакцій. Чи не час уже замислитися над масовим економічним in-memory комп'ютингом на основі таблиць істинності або логічних векторів як інженерної основи інтелектуального комп'ютингу?

Час декогеренції [14–15] – це період існування стійкого динамічного процесу або явища в мікро- або макросвіті, підтримуваного ззовні. У мікросвіті квантового комп'ютингу таку підтримку забезпечує тер-

мостат або додаткове охолодження апаратури. У макросоціальному світі – це зовнішні гроші або зброя. Для класичного або квантового комп'ютингу – це енергетична підтримка обчислень потужним процесором або апаратурою охолодження. В усіх випадках стабільність динамічного процесу зникає, коли припиняється зовнішня допомога. Тому in-memory комп'ютинг на основі read-write транзакцій виступає як майбутній масовий інтелектуальний комп'ютинг для обслуговування людства. Інтелектуальний комп'ютинг – це інтеграція класичних, квантових і штучно-інтелектуальних in-memory архітектур для формування гармонійних відносин між метою та засобами її досягнення [16]. Для кожної задачі потрібно обирати економічний механізм отримання ефективного розв'язання з точки зору енерговитрат і часу (рис. 2).

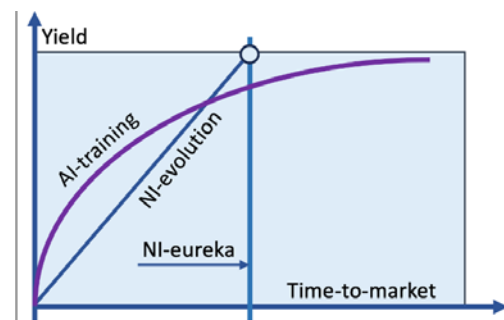


Рисунок 2 – Відносини між штучним та природним інтелектом

Індукція [17] – це шлях переходу від окремого до загального – мати штучного інтелекту (Artificial Intelligence), яка проходить довгий імовірнісний шлях навчання, що веде крізь море помилок. Дедукція [18] – це шлях від загального до конкретного – мати природного інтелекту (Natural Intelligence), – це множина великих і малих помилок, що призводять до великих і малих відкриттів. «Ймовірність (випадковість) – це фіговий листок на голому тілі нашого невігластва», – Альберт Ейнштейн. Імовірнісний процес із часом вивчення стає детермінованим. І навпаки – коли багато детермінованих явищ перетворюються на випадкові процеси. Штучний інтелект (Artificial Intelligence) гарантує рішення задачі з визначеною точністю за заданий час, але не може одразу відповісти на просте запитання: скільки буде 2×2 ? Для цього потрібні декілька ітерацій навчання AI-механізму. Природний інтелект (Natural Intelligence) може одразу дати точну відповідь на просте завдання або знайти розв'язок складної задачі за невизначений час (годину, день, місяць, рік) – або взагалі не знайти. Завжди потрібно обирати прийнятні – детерміновані або ймовірнісні – механізми розв'язання задач.

3 МАТЕРІАЛИ І МЕТОДИ

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) відношення $L = Y \otimes Y$, яке генерує декартову матрицю – у цьому випадку матрицю активності або логічний вектор L розмірності 2^{n+1} .

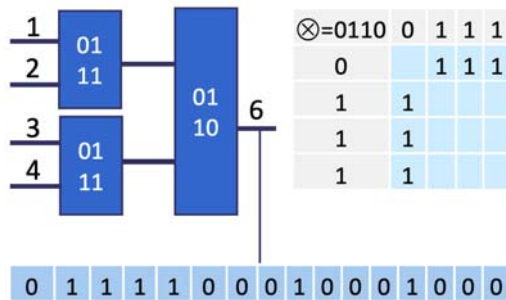


Рисунок 3 – Рефлексивне хог-відношення вектора 0111

Декартовий добуток двох різних векторів формує або синтезує новий вектор розмірності $2^{n+m} = 2^n \times 2^m$. Схематичне позначення такої операції може бути представлено трьома елементами (рис. 4). Кожне розширення простору логічного вектора супроводжується появою нових цікавих властивостей, орієнтованих на практичне розв'язання задач аналізу. Наприклад, матриця-вектор, отриманий у результаті декартової операції, вказує ті координати, які відрізняють два вектори (дві функціональності) – 1000 і 11011011 – між собою на вичерпному тесті. Наведена схема є хорошим 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 найекономішша модель для розв'язання всіх задач design and test (і не тільки).

Наведена схема є архітектурою (структурою) верифікації будь-якого цифрового проекту на основі специфікації та актуальної моделі цифрового пристрою. Результат верифікації представлений логічним вектором, де одиничні координати показують невідповідність результатів тестування у просторі верифікації проекту.

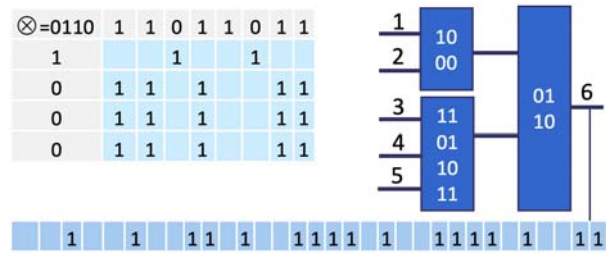


Рисунок 4 – Схематика декартового добутку

Розклад або декомпозиція логічного вектора – це практично орієнтоване завдання: як один елемент або вектор представити сукупністю векторів чи елементів із тими ж властивостями. Частково таке завдання вирішується шляхом синтезу схеми на основі логічного вектора. Проте це шлях назад до комп'ютингу вчорашнього дня. Фактично, теорія векторно-логічного комп'ютингу є дизраптором не лише для синтезу в традиційну систему елементів кремнієвого кристалу, а й для класичного комп'ютингу, що базується на архітектурі фон Неймана з центральним процесором. Крім того, логічний вектор – це ідеальна модель для представлення даних у кубітах квантового комп'ютингу. Логічним вектором і його похідними можна описати будь-які процеси, що існують сьогодні в кіберпросторі, фізичному просторі та соціальному просторі. Більше того, за допомогою логічного вектора можна керувати будь-якими об'єктами в цих трьох просторах на основі загального моніторингу й аналізу, які також можуть бути реалізовані за допомогою логічного вектора. Масштабування логічного вектора на будь-якому рівні ієрархії та розмірності залишає його вектором. Єдиним недоліком логічного вектора є експоненційний обсяг пам'яті, необхідної для його зберігання. Проте пам'ять у сучасному кіберпросторі вже не є дефіцитною. Оригінальність дослідження полягає у використанні векторної логіки в архітектурі in-метогу комп'ютингу на основі read-write транзакцій, що дозволяє зменшити споживання ресурсів у вигляді часу та енергії. Імплементация векторної логіки в in-метогу комп'ютинг робить його масовим та енергоощадним, вільним від складних алгоритмів аналізу структур даних.

Векторна логіка – це представлення логічних операцій 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 ($\otimes$ -вектора) $L = X \otimes Y$ над бітами двох логічних векторів X і Y . Усі три вектори X , Y , $\otimes$ є логічними. Вектор $\otimes$

(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 – це 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'ютигу, 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. Це сьогодні – тренд №1 у світі, коли інтелектуальні надлишкові моделі використовуються для вирішення всіх задач комп'ютигу лінійними або нульовими алгоритмами.

Кілька прикладів практичного застосування декартової логіки для задач modeling for simulation (рис. 5):

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 [24–26] – 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. 6).

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 говорити про дві метрики побудови карти тестування схеми:

1. Механізм моделювання несправностей на дедуктивних векторах елементів схеми:

$$C = k \times n \times 2^m \times n^2,$$

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.

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

Рисунок 5 – Практичне використання розширеної декартової логіки

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

Рисунок 6 – Побудова карти тестування схеми за її логічним вектором

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 має вигляд: $C = n \times 2^{2m} + 3 \times 2^m$, що співмірно зі складністю алгоритму дедуктивного моделювання несправностей у схемі.

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 in-методу комп'ютиру для тестування, верифікації, діагностування та експлуатації. In-memory vector logic комп'ютинг – це економічне вирішення багатьох обчислювальних проблем з погляду енергетичних, часових і вартісних витрат.

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. 7), 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.

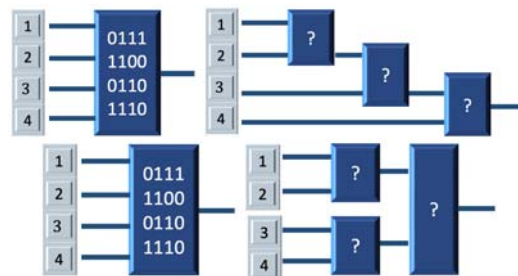


Рисунок 7 – Розбиття складних елементів на двовходові

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. 8).

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. Моделювання логічного вектора для цифрової структури. Логічний вектор процесу або явища дозволяє розв'язувати всі задачі комп'ютиру 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 $X \otimes Y$ за правилом логічної функції G ($\otimes$). При цьому вектор Y – це сукупний вектор уже оброблених елементів схеми, а вектор 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 змінної структури.

Декомпозиція логічного вектора (рис. 9) здійснюється на основі рівняння його синтезу $L = X \otimes Y$. Необхідно знати два компоненти рівняння: L та $\otimes$. Потрібно визначити вектори X і Y , які перебувають між собою у відношенні $\otimes$, де X і Y повинні відповідати певним співвідношенням довжин, що важливо для формування формату матриці вектора.

$\otimes$	1	0	0	1	0	1	1	0
0								
1								
1								
0								

$\otimes=0111$	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		

$\otimes=0110$	1	0	0	1	0	1
0	1			1		
1			1	1		1
1			1	1		1
0	1			1	1	1

Рисунок 8 – Побудова логічного вектора-матриці фрагмента схеми на основі декартової логіки

Для визначення векторів X та Y працює правило: $X \otimes L_{ij} = Y$. У цьому процесі також може бути використана модель штучного інтелекту.

$\otimes=0001$	1	0	0	1	0	1	1	0
0								
1	1			1		1	1	
1	1			1		1	1	
0								

$L=X \otimes Y$

Рисунок 9 – Декомпозиція логічного вектора

Для декомпозиції вектора немонотонної функції необхідно представити її у вигляді матриці на основі відомого утворювального G -вектора (рис. 10).

Initial form for decomposition								
$\otimes=0001$								
00=0								
01=0	1			1		1	1	
10=0	1			1		1	1	
11=1								

$L=X \otimes Y \rightarrow 1=1 \otimes 1$

Resulting decomposition								
$\otimes=0001$	1	0	0	1	0	1	1	0
0								
1	1			1		1	1	
1	1			1		1	1	
0								

$L=X \otimes Y$

Initial form for decomposition								
$\otimes=0111$								
00=0	1			1		1	1	
01=1	1	1	1	1	1	1	1	1
10=1	1	1	1	1	1	1	1	1
11=1	1			1		1	1	

$L=X \otimes Y \rightarrow 0=0 \otimes 0$

Resulting decomposition								
$\otimes=0111$	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	

$L=X \otimes Y$

Рисунок 10 – 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 формується 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. 11), 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								
$\otimes=2^4 \times 2^{n-2}$								
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

$L=X \otimes Y$

Рисунок 11 – 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 ДНФ за одиничними координатами вектора. Наприклад, для вектора 10010100 розклад у вигляді ДНФ матиме вигляд: $L = x_1 x_2 x_3 \vee x_1 x_1 x_2 \vee x_1 x_2 x_2$ (рис. 12).

Кожен елемент схеми тут представлений логічним вектором монотонної логічної функції (and, or) з бази-су 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.

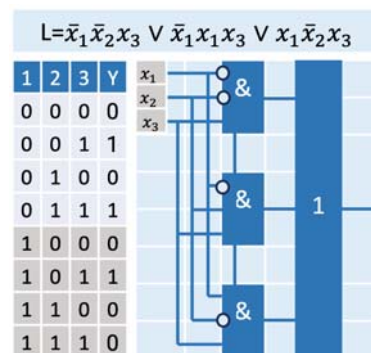


Рисунок 12 – Декомпозиції логічного вектора шляхом синтезу ДНФ

Розділити логічний вектор на два можна 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. 13). 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 поділ вектора 11010101 на дві рівновеликі частини.

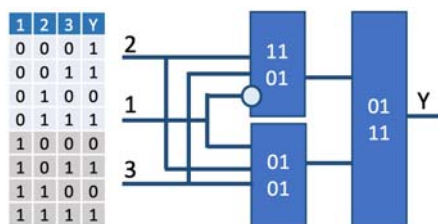


Рисунок 13 – Поділ логічного вектора на два з використанням комутатора

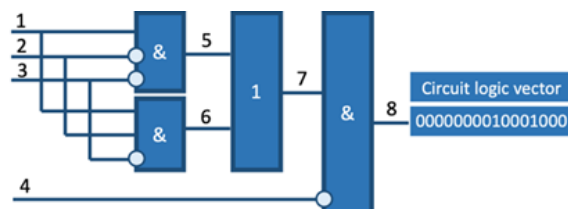
Моделювання логічного вектора схеми шляхом good-value simulation. Моделювання всіх 2^n вхідних наборів. Процедура є ефективною та технологічно дуже простою. Для обробки елементів схеми, представлених логічними векторами L , розміщеними в пам'яті, використовується єдина формула: $M_i = L[M(X_i)]$.

Для аналізу наведених структур даних схеми застосовуються read-write транзакції in-мемогу 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. 14).

Поле M 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. 15) усього за три матричні операції [26]. 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.



The data structure of the fault-free simulation																						
M	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Inputs: 1,2,3,4 outputs 8					
Simulation matrix on the complete test																	L -vectors		Inputs		NI	
1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1						
2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1						
3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1						
4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1						
5	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	00001000		1	2	3	3
6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	00000010		1	2	3	3
7	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0111		5	6	2	2
8	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0010		7	4	2	2

Рисунок 14 – Структура даних безпомилкового моделювання (fault-free simulation) схеми

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

Рисунок 15 – Моделювання карти тестування за логічним вектором 0000000010001000 для схеми

Однак дедуктивне векторно-логічне моделювання несправностей [27] практично не поступається за обчислювальною складністю алгоритму good-value simulation. Тут представлено всі комбінації логічних несправностей входних змінних комбінаційної схеми, що перевіряються на вичерпному тесті. Це яскравий приклад prompt-комп'ютингу, коли вся діагностична інформація для верифікації проекту 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.

Метод декартової суперпозиції логічних векторів. Метод моделювання справної роботи для отримання логічного вектора має один суттєвий недолік – це необхідність моделювання повного (вичерпного) тесту довжиною 2^n , де n – кількість входних змінних, для всіх елементів схеми. Більш оптимальним і мінімальним є метод, заснований на суперпозиції векторів за допомогою декартової логіки.

Рисунок 17 – Отримання логічного вектора схеми на основі декартової логіки

2. Виконуємо побудову логічного вектора кожного двохходового елемента за допомогою декартової логіки. 3. Моделюємо матрицю 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. Видаляємо координати матриці, що суперечать вхідним наборам. Для елемента з виходом 7 залишаються лише координати головної діагоналі; решта вважаються суперечливими за адресами вхідних бітів. 5. Перевіряємо отриманий логічний вектор схеми за допомогою good-value simulation. 6. У результаті отримуємо логічний вектор схеми: 0000000010001000.

4 ЕКСПЕРИМЕНТИ

Алгоритм forward modeling (прямого моделювання): кожен логічний елемент розглядається як вузол, у якому виконується декартова логіка (декартовий добуток) завжди двох векторів, що подаються на входи цього елемента. Декартова логіка кожного елемента – це його логічний вектор більшої розмірності 2^{n+1} .

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. 16).

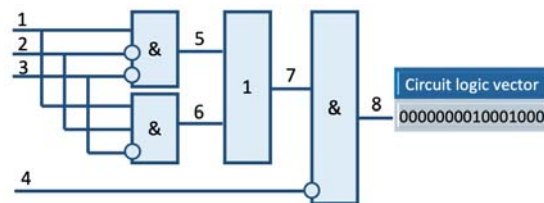


Рисунок 16 – Логічна схема зі зсидженими розгалуженнями

Моделювання логічного вектора схеми методом декартової суперпозиції векторів (рис. 17): 1. Умовно розбиваємо всі елементи на двохходові. Це спрощує процедуру моделювання та дозволяє уніфікувати обробку елементів.

Під час виконання декартової логіки враховуються особливості структури логічної схеми:

1) на елемент подані два зовнішніх входи – виконується декартова операція: $01 \ G \ 01$;

2) на елемент подані три зовнішніх входи – послідовно виконуються дві декартові операції $(01 \ G \ 01) \ G \ 01$. На елемент подано понад три зовнішніх входи, виконується рекурсія: $(01 \ G \ 01) \ G \ 01) \dots \ G \ 01$;

3) якщо на елемент подано один зовнішній вхід і вихід іншого елемента – виконується операція: $01 \ G \ Y$ або $X \ G \ 01$;

4) якщо на елемент подано два виходи з інших елементів – виконується: $X \ G \ Y$. Якщо на елемент подано понад два виходи з інших елементів – виконується рекурсія: $(X \ G \ Y) \ G \ Y) \dots \ G \ Y$ (рис. 18);

5) обмеження на пряме моделювання векторів схеми пов'язані з нерегулярним характером деяких логічних векторів, які містять більше ніж дві змінні. У такому разі потрібно виконати розклад або декомпозицію початкового вектора на кілька векторів регулярних логічних функцій від двох змінних, після чого застосовується рекурсивна процедура прямого моделювання багатовходового елемента на основі декартових операцій двохходових елементів.

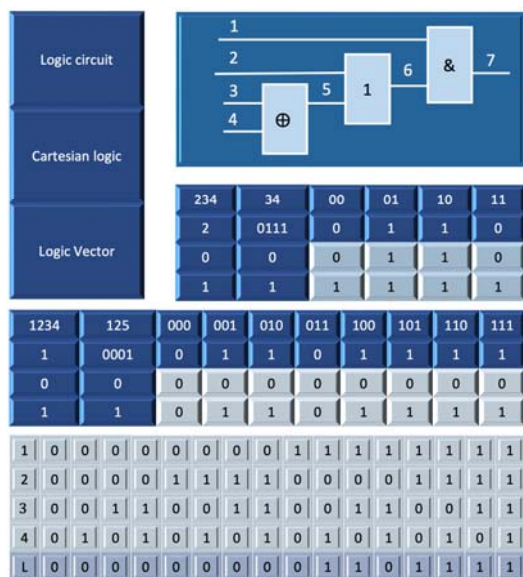


Рисунок 18 – Моделювання логічного вектора для ітеративної цифрової структури

Зворотнє моделювання (на відміну від forward modeling) передбачає обробку схеми від виходу до входів. Порядок обробки елементів: на кожному кроці вибирається той елемент, вихід якого з'єднаний із входом уже обробленого елемента. Обидва методи – forward і зворотнє моделювання – мають однакову обчислювальну складність. Вихідний результат – це таблиця істинності T за зовнішніми входами схеми разом із логічним вектором L і цифровими ідентифікаторами логічних входних змінних.

Хмарні сервіси modeling for simulation (MOSI) представлені такими компонентами: 1) синтез карти тестування за логічним вектором функціональності; 2) синтез логічного вектора цифрової комбінаційної схеми, заданої логічними векторами елементів; 3) моделювання справної поведінки логічної схеми на основі векторного представлення логічних елементів; 4) моделювання несправностей як адресів у логічній схемі на основі дедуктивних векторів логічних елементів; 5) пошук несправностей у логічній функціональності за її картою тестування; 6) графічний інтерфейс (GUI) для введення логічної схеми, де елементи представлені логічними векторами.

Інтерфейс застосунку GenLV. Вхідні дані: два логічні вектори X і Y однакової або різної довжини. Декартова логіка – це спосіб отримання матриці логічного вектора L шляхом декартового добутку бітів двох векторів за правилом логічної функції G : $L = X \otimes Y$, де $2^{n+m} = 2^n \times 2^m$. Утворювальний вектор G повинен мати довжину $G = 2^{n-2} = 4$.

1. Цифрові ідентифікатори логічних змінних: кількість визначається як двійковий логарифм довжини кожного з векторів: $\log_2 X$, $\log_2 Y$, $\log_2 G$, $\log_2 L$.

2. Інтерфейс вихідних даних декартової логіки: матриця M , що формує результат синтезу логічного вектора $L = X \otimes Y$, і містить усі три вектори (X , Y , L), ідентифікатори змінних і координати синтезованого вектора L у вигляді матриці, де $\otimes$ – будь-яка логіка, представлена вектором G .

3. Вихідний результат – це таблиця істинності T разом із логічним вектором L і цифровими ідентифікаторами логічних змінних (рис. 19).

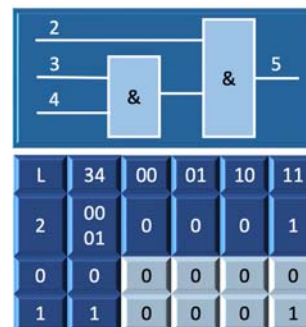


Рисунок 19 – Логічний вектор для фрагмента схеми без озгалужень

5 РЕЗУЛЬТАТИ

Запропоновано механізми побудови логічного вектора для цифрової комбінаційної схеми з метою подальшого синтезу карти тестування функціональності за допомогою лінійного алгоритму обробки векторно-матричних даних для розв'язання комбінаторних задач моделювання одиночних і кратних логічних несправностей [24–27]. Запропоновано апарат декартової логіки для розв'язання таких задач за допомогою алгоритмів лінійної обчислювальної складності. Декартова логіка, завдяки експоненційній надлишковості 2^{n+m} , є ефективним інструментом для задач моделювання, симуляції, тестування та діагностування. Наведено приклади практичного застосування векторної та декартової логіки для верифікації цифрових схем і функціональності. Представлено хмарні сервіси, що дозволяють вивчати та застосовувати ці механізми в моделюванні цифрових проєктів шляхом побудови тестів на основі моделювання несправностей. За простотою та оригінальністю ці механізми не мають аналогів.

На основі теорії векторної логіки розроблено механізми та програмне забезпечення для економного розв'язання задач design and test в архітектурі інтегрованого комп'ютерного (рис. 20).

Хмарні сервіси cloud modeling for simulation (MOSI) представлені такими компонентами:

1. Синтез карти тестування за логічним вектором функціональності.

2. Синтез логічного вектора цифрової комбінаційної схеми, заданої логічними векторами її елементів.

MOSI – vector logic in-memory MOdeling for Simulation test and fault as address			
Design and test service	Complexity	Limitation	Novelty
GUI logic circuit	n^2 (line number)	Screen Area	Logic vector
Good value circuit simulation	n – linear	Screen Area	Logic vector
Circuit fault simulation	2^n (exponential)	Screen Area	fault as address
Modeling circuit logic vector	n^2 (line number)	Screen Area	Cartesian logic
Modeling logic testing map	n – linear	$2^{n=9}$ (input<10)	fault as address
Logic fault detection on test	n – linear	$2^{n=9}$ (input<10)	fault as address
The services are focused on processor-free in-memory computing via read-write transactions			
Service economics – low latency, low energy, low code, low time-to-market, high yield			

Рисунок 20 – Шість сервісів хмарної функціональності на основі векторної логіки

3. Моделювання справної поведінки логічної схеми на основі векторного представлення логічних елементів.

4. Моделювання несправностей як адрес у логічній схемі на основі дедуктивних векторів логічних елементів.

5. Діагностика функціональності з використанням її карти тестування.

6. Графічний інтерфейс користувача (GUI) для введення логічної схеми, в якій елементи представлені логічними векторами.

6 ОБГОВОРЕННЯ

Представлені сервіси, що показані на рисунку 20, мають одну важливу спільну рису – це логічний вектор, який використовується як модель для побудови механізмів Modeling for Simulation. Сервіси реалізовані мовами програмування Python і JavaScript з орієнтацією на архітектури майбутнього масового in-memory (quantum) комп'ютингу, що базується на read-write транзакціях без інструкцій центрального процесора. Така реалізація робить механізми Modeling for Simulation перспективними для EDA-ринку з точки зору часових та енергетичних витрат. Представлені сервіси орієнтовані на інженерне розуміння розв'язання задач design and test за технологією prompt-комп'ютингу, що є надзвичайно важливим для навчання студентів комп'ютерній інженерії у технічних вишах світу.

Особливості запропонованого 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 сходження розгалужень, то ця процедура мінімізації не застосовується; 3) недоліком методу є необхідність виконання процедури розбиття елемента

на двохходові для їх рекурсивної обробки з метою побудови логічного вектора схеми; 4) метод є технологічно зручним для написання програмного коду, якщо всі елементи схеми – двохходові; 5) якщо не розбивати елемент на двохходові піделементи, то потрібно використовувати утворювальні логічні вектори довжиною більше ніж 4 біти.

Якщо специфікація функціональності задана логічним вектором – немає потреби синтезувати комбінаційну схему, а потім моделювати 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. In-memory vector logic комп'ютинг – це економічне рішення багатьох обчислювальних задач з погляду енерговитрат, часу та вартості.

Запропоновано 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.

Хмарний сервіс з GUI-інтерфейсом для введення схеми на основі логічних елементів у вигляді векторів дозволяє в 3–5 разів скоротити час створення моделі у порівнянні з описом схеми мовою HDL.

Моделювання логічного вектора схеми дає змогу вдвічі зменшити час моделювання несправностей у порівнянні з моделюванням схеми, поданої у вигляді логічних елементів. Саме 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-мемогу механізм є простим у реалізації, використовує лише read-write транзакції над бітами логічних векторів і не потребує інструкцій процесора.

ВИСНОВОК

Практичне значення дослідження – впровадження в EDA-ринку інженерного підходу до розв’язання задач design and test на основі використання простих моделей векторної логіки, орієнтованої на економічний in-мемогу комп’ютинг [19–20], що базується на read-write транзакціях і вільний від інструкцій процесора.

«Векторна логіка» представлена переважно у вигляді матрично-векторного формалізму для логічних функцій і значень істинності/хибності [21–23]. Проте це майже нічого не означає з практичної точки зору детермінованої комп’ютерної інженерії.

Новизна дослідження полягає у використанні векторної логіки [24–27] в архітектурі in-мемогу комп’ютингу на основі read-write транзакцій, що дозволяє зменшити споживання ресурсів у вигляді часу та енергії. Імплементація векторної логіки в in-мемогу комп’ютинг робить його масовим і енергоефективним, вільним від складних алгоритмів аналізу великих даних.

ЛІТЕРАТУРА

1. 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. Fanout-Based Reliability Model for SER Estimation in Combinational Circuits [Text] / 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. Slope Interconnect Effort: Gate-Interconnect Interdependent Delay Modeling for Early CMOS Circuit Simulation [Text] / 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. A Hybrid Model-Based Diagnosis Approach for Open-Switch Faults in PMSM Drives [Text] / [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. Efficient Cartesian Genetic Programming-Based Automatic Synthesis Framework for Reversible Quantum-Flux-Parametron Logic Circuits [Text] / [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. Accuracy and Size Trade-off of a Cartesian Genetic Programming Flow for Logic Optimization [Text] / [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. Utilizing the fuzzy logic algorithm for cartesian robot control system in conditions of mechanical damage transmission [Text] / [O. Dunaeva, S. Autou, 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. Very Short-Term Load Forecasting Using Cartesian Genetic Programming Evolved Recurrent Neural Networks (CGPRNN) [Text] / 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. Side Channel and Fault Analyses on Memristor-Based Logic In-Memory [Text] / 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. A Novel Parallel In-Memory Logic Array Based on Programmable Diodes [Text] / [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 [Text] / 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. XNOR-BSNN: In-Memory Computing Model for Deep Binarized Spiking Neural Network [Text] / [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. Truth Table Based Intelligent Computing [Text] / [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. Decoherence of Dynamically Manipulated Qubits [Text] / V. Privman, D. Solenov // 2006 Sixth IEEE Conference on Nanotechnology, Cincinnati, OH, USA, 2006 : proceedings. – 2006. – P. 842–845. – DOI: 10.1109/NANO.2006.247790.
15. Moiré Localization Induced Enhancement in the Decoherence Time of Interlayer Excitons in WSe₂-MoSe₂ Heterobilayers [Text] / [M. A. Durmuş, K. Demiralay, M. M. Khan et al.] // 2024 Conference on Lasers and Electro-Optics (CLEO), Charlotte, NC, USA, 2024 : proceedings. – 2024. – P. 1–2.
16. Intelligent Computing: The Latest Advances, Challenges, and Future [Text] / [Shiqiang Zhu, Ting Yu, Tao Xu et al.] // Intelligent Computing. – DOI: 10.34133/icomputing.0006.
17. Tarek A. A Proof Theoretic Exploration of Mathematical Induction in Computational Paradigm [Text] / A. Tarek, A. Alveed, A. Farhan // 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE), Las Vegas, NV, USA, 2023 : proceedings. – 2023. – P. 963–972. – DOI: 10.1109/CSCE60160.2023.00162.
18. Kamide N. Natural Deduction with Explosion and Excluded Middle [Text] / N. Kamide // 2023 IEEE 53rd International Symposium on Multiple-Valued Logic (ISMVL), Matsue, Japan, 2023 : proceedings. – 2023. – P. 24–29. – DOI: 10.1109/ISMVL57333.2023.00016.
19. Behavioral Level Simulation Framework to Support Error-Aware CNN Training with In-Memory Computing [Text] / S.-H. Chang, C.-N. J. Liu, A. Küster // 2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), Villasimius, Italy, 2022 : proceedings. – 2022. – P. 1–4. – DOI: 10.1109/SMACD55068.2022.9816307.

20. Framework Independent Modeling for SRAM-Based In-Memory Computing [Text] / [S. Li, N. Zhang, W. Zhang et al.] // 2024 2nd International Symposium of Electronic Design Automation (ISED), Xi'an, China, 2024 : proceedings. – 2024. – P. 777. – DOI: 10.1109/ISED62518.2024.10617799.
21. Westphal J. Logic as a Vector System [Text] / J. Westphal, J. Hardy // Journal of Logic and Computation. – 2005. – Vol. 15, No. 5. – P. 751–765. – DOI: 10.1093/logcom/exi040.
22. Mizraji E. Vector Logic: A Natural Algebraic Representation of the Fundamental Logical Gates [Text] / E. Mizraji // Journal of Logic and Computation. – 2008. – Vol. 18, No. 1. – P. 97–121. – DOI: 10.1093/logcom/exm057.
23. Mizraji E. Vector logic allows counterfactual virtualization by the square root of NOT [Text] / E. Mizraji // Logic Journal of the IGPL. – 2020. – Vol. 29, No. 5. – P. 859–870. – DOI: 10.1093/jigpal/jzaa026.
24. Vector Synthesis of Fault Testing Map For Logic [Text] / [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.
25. Vector-Logical In-Memory Simulation of Faults as Truth Table Addresses [Text] / [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.
26. Prompt-Testing of Logic [Text] / [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.
27. Faults-as-address simulation [Text] / [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.
- Received 14.07.2025.
Accepted 14.10.2025.

UDC 681.326

MECHANISMS OF SCHEMATIC MODELING BASED ON VECTOR LOGIC

Hahanov V. I. – Dr. Sc., Professor of the Design Automation Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.

Obrizan V. I. – PhD, Post-Doctoral Student of the Design Automation Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.

Rakhlis D. Y. – PhD, Associate Professor of the Design Automation Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.

Khakhanova H. V. – Dr. Sc., Professor of the Design Automation Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.

Hahanov I. V. – PhD, Assistant of the Design Automation Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.

Demchenko O. I. – Post-graduate student of the Design Automation Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.

Voronov A. O. – Post-graduate student of the Design Automation Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine.

ABSTRACT

Context. This paper addresses issues relevant to the EDA market – reducing the cost and time of testing and verification of digital projects by synthesizing the logic vector of a digital circuit, which significantly simplifies the algorithms of good-value simulation and reduces the synthesis of the test map to three matrix operations.

Objective. The aim of the study is to reduce the cost and time of testing and verification of digital projects by synthesizing the logic vector of a digital circuit, which allows for simplifying the algorithm for constructing the test map to three matrix operations.

Method. The synthesis of a logic vector for a combinational circuit is proposed for good-value and fault as modeling addresses within the architecture of prompt-intelligent in-memory computing. The logic vector is the most technological, compact, and exhaustive representation of the circuit for efficiently solving all design and testing tasks. Cartesian logic is proposed as an effective intelligent mechanism for solving combinatorial problems (modeling, simulation, testing, diagnostics) using algorithms of linear computational complexity, thanks to its exponential

redundancy 2^{n+m} . Mechanisms are proposed for constructing a logic vector of a process or phenomenon, function or structure, based on matrix structures of Cartesian logic for solving Modeling for Simulation tasks. An engineering method of direct parallel good-value modeling of the circuit is proposed, based on the use of logic vectors of elements and truth tables. Cartesian logic is a logic vector (matrix) as the result of modeling Cartesian logical relations between the bits of logic vectors or truth table addresses. Cartesian logic solves the following tasks: 1. Good-value modeling of a circuit logic vector without the need for a functional behavior modeling algorithm. 2. Fault modeling testing map of logic without a fault simulation algorithm.

Results. A mathematical apparatus of Cartesian logic is proposed, represented as a logic vector (matrix), which is the result of modeling Cartesian logical relations between the bits of logic vectors or addresses of the truth table. Based on the theory of vector logic, mechanisms and software have been developed for efficiently solving design and test tasks within the in-memory computing architecture.

Conclusions. The practical value of the study lies in addressing all design and test tasks using simple models of vector logic, oriented toward economical in-memory computing based on read-write transactions and free from processor instructions. It is proposed to use the specification logic vector within the in-memory computing architecture for testing, verification, diagnostics, and operation. In-memory vector logic computing is an economical solution to many computational problems in terms of energy, time, and cost efficiency. The novelty of the study lies in the use of vector logic within the in-memory computing architecture based on read-write transactions, which reduces resource consumption in terms of time and energy. Implementing vector logic in in-memory computing makes it scalable and energy-efficient, and frees it from complex big data analysis algorithms.

KEYWORDS: vector logic, logic vector, Cartesian logic, intelligent computing, test map, fault modeling, good-value modeling, in-memory modeling for simulation, prompt engineering.

REFERENCES

1. 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 [Text], *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2025, Vol. 72, No. 1, pp. 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 [Text], *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2009, Vol. 56, No. 7, pp. 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 [Text], *IEEE Transactions on Power Electronics*, 2022, Vol. 37, No. 4, pp. 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 [Text], *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. DOI: 10.1109/TCAD.2025.3546884.
6. Berndt A. et al. Accuracy and Size Trade-off of a Cartesian Genetic Programming Flow for Logic Optimization [Text], 2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI), Campinas, Brazil, 2021 : proceedings, 2021, pp. 1–6. DOI: 10.1109/SBCCI53441.2021.9529968.
7. Dunaeva O., Autou S., Kudelina K. et al. Utilizing the fuzzy logic algorithm for cartesian robot control system in conditions of mechanical damage transmission [Text], *IECON 2024 – 50th Annual Conference of the IEEE Industrial Electronics Society, Chicago, IL, USA, 2024 : proceedings*, 2024, pp. 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) [Text], 2013 12th International Conference on Machine Learning and Applications, Miami, FL, USA, 2013 : proceedings, 2013, pp. 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 [Text], *IEEE Design & Test*, 2024, Vol. 41, No. 3, pp. 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 [Text], *IEEE Journal of the Electron Devices Society*, 2024, Vol. 12, pp. 738–744. – DOI: 10.1109/JEDS.2024.3457021.
11. Chen X., Song T., Han Y. RRAM-based Analog In-Memory Computing: Invited Paper [Text], 2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), AB, Canada, 2021 : proceedings, 2021, pp. 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 [Text], 2021 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS), Macau, China, 2021 : proceedings, 2021, pp. 17–21. DOI: 10.1109/HPBDIS53214.2021.9658467.
13. Hahanov V., Litvinova E., Davitadze Z. et al. Truth Table Based Intelligent Computing [Text], 2024 31st International Conference on Mixed Design of Integrated Circuits and System (MIXDES), Gdansk, Poland, 2024 : proceedings, 2024, pp. 199–204. DOI: 10.23919/MIXDES62605.2024.10614035.
14. Privman V., Solenov D. Decoherence of Dynamically Manipulated Qubits [Text], 2006 Sixth IEEE Conference on Nanotechnology, Cincinnati, OH, USA, 2006 : proceedings, 2006, pp. 842–845. DOI: 10.1109/NANO.2006.247790.
15. Durmuş M. A., Demiralay K., Khan M. M. et al. Moiré Localization Induced Enhancement in the Decoherence Time of Interlayer Excitons in WSe₂-MoSe₂ Heterobilayers [Text], 2024 Conference on Lasers and Electro-Optics (CLEO), Charlotte, NC, USA, 2024 : proceedings, 2024, pp. 1–2.
16. Zhu Shiqiang, Yu Ting, Xu Tao et al. Intelligent Computing: The Latest Advances, Challenges, and Future [Text], *Intelligent Computing*. DOI: 10.34133/icomputing.0006.
17. Tarek A., Alveed A., Farhan A. A Proof Theoretic Exploration of Mathematical Induction in Computational Paradigm [Text], 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE), Las Vegas, NV, USA, 2023 : proceedings, 2023, pp. 963–972. DOI: 10.1109/CSCE60160.2023.00162.
18. Kamide N. Natural Deduction with Explosion and Excluded Middle [Text], 2023 IEEE 53rd International Symposium on Multiple-Valued Logic (ISMVL), Matsue, Japan, 2023 : proceedings, 2023, pp. 24–29. DOI: 10.1109/ISMVL57333.2023.00016.
19. Chang S.-H., Liu C.-N. J., Küster A. Behavioral Level Simulation Framework to Support Error-Aware CNN Training with In-Memory Computing [Text], 2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), Villasimius, Italy, 2022 : proceedings, 2022, pp. 1–4. DOI: 10.1109/SMACD55068.2022.9816307.
20. Li S., Zhang N., Zhang W. et al. Framework Independent Modeling for SRAM-Based In-Memory Computing [Text], 2024 2nd International Symposium of Electronic Design Automation (ISED), Xi'an, China, 2024 : proceedings, 2024, P. 777. DOI: 10.1109/ISED62518.2024.10617799.
21. Westphal J., Hardy J. Logic as a Vector System [Text] // *Journal of Logic and Computation*, 2005, Vol. 15, No. 5, pp. 751–765. DOI: 10.1093/logcom/exi040.
22. Mizraji E. Vector Logic: A Natural Algebraic Representation of the Fundamental Logical Gates [Text], *Journal of Logic and Computation*, 2008, Vol. 18, No. 1, pp. 97–121. DOI: 10.1093/logcom/exm057.
23. Mizraji E. Vector logic allows counterfactual virtualization by the square root of NOT [Text], *Logic Journal of the IGPL*, 2020, Vol. 29, No. 5, pp. 859–870. DOI: 10.1093/jigpal/jzaa026.
24. Hahanov V., Gharibi W., Chumachenko S. et al. Vector Synthesis of Fault Testing Map For Logic [Text], *IAES International Journal of Robotics and Automation (IJRA)*, 2024, Vol. 13, No. 3, pp. 293–306. DOI: 10.11591/ijra.v13i3.pp293-306.
25. Hahanov V., Litvinova E., Hahanova H. et al. Vector-Logical In-Memory Simulation of Faults as Truth Table Addresses [Text], 2024 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, Armenia, 2024 : proceedings, 2024, pp. 1–6. DOI: 10.1109/EWDTS63723.2024.10873615.
26. [Hahanov V., Devadze D., Hahanov I. et al. Prompt-Testing of Logic [Text], 2024 IEEE East-West Design & Test Symposium (EWDTS), Yerevan, Armenia, 2024 : proceedings, 2024, pp. 1–5. DOI: 10.1109/EWDTS63723.2024.10873774.
27. Hahanov V., Chumachenko S., Litvinova E. et al. Faults-as-address simulation [Text], *IAES International Journal of Robotics and Automation*, 2024, Vol. 13, No. 4, pp. 452–468. DOI: 10.11591/ijra.v13i4.pp452-468.