ОБЧИСЛЮВАЛЬНІ ПРОЦЕСИ ТА СИСТЕМИ

DOI
УДК 681.326:519.713

В.І. Хаханов, д-р техн. наук,
e-mail: hahanov@icloud.com; orcid.org/0000-0001-5312-5841,
Е.І. Литвинова, д-р техн. наук,
orcid.org/0000-0002-9797-5271,
А.О. Воронов, аспірант,
orcid.org/0009-0004-0282-5775
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр-т Науки, 14

Векторна логіка для тестування функцій та структур

Векторні логічні обчислення — це економічний механізм інтелектуальних обчислень у пам'яті, який використовує транзакції читання-запису для вирішення практичних завдань аналізу та управління фізичними, соціальними та бізнес-процесами на основі моніторингу. Для будь-якої ситуації може бути побудована надлишкова модель цифрової структури і функціональності, яка позбавляє комп'ютинг алгоритму аналізу для отримання рішення. Дослідження спрямоване на скорочення часових та енергетичних витрат, пов'язаних з управлінням процесами та явищами у цифровому світі. Засобом досягнення цієї мети є використання векторно-логічних обчислювальних механізмів у пам'яті, які спрощують алгоритми за допомогою експоненціальної надлишковості структур даних. Аналіз функцій і структур проводиться з погляду алгоритмічної простоти з метою моделювання та симуляції. Запропоновано кілька механізмів, що базуються на декартовій логіці, для моделювання логічного вектора за структурою цифрової схеми. Логічний вектор використано для створення тестової карти за три матричні операції. Запропоновано хмарний сервіс MOSI, який моделює тестові набори та несправності як адреси таблиці істинності у цифрових схемах та функціональностях.

К л ю ч о в і с л о в а: функції та структурикомп'ютингу, векторно-логічна функціональність, векторна логіка, логічний вектор, декартова логіка, інтелектуальний комп'ютинг, карта тестування, моделювання несправностей, моделювання справної поведінки, промпт-інжиніринг.

Метрика структур та функцій комп'ютингу. Логіка — це структури та функції комп'ютингу. Алгоритм не потрібний, якщо модель функціональності сама формує висновки. Існують три типи моделей:

Стаття поширюється на умовах ліцензії відкритого доступу CC BY 4.0

1. Мінімальна модель — складний алгоритм її аналізу під час подачі вхідного набору даних.

2. Надлишкова модель — простий алгоритм аналізу при подачі вхідного набору даних.

3. Розумна надлишкова модель створює множину висновків на короткий запит (логічний вектор).

Розумний (константний) алгоритм на надлишковій моделі створює множину рішень і висновків. Проста логіка алгоритму забезпечує вичерпне рішення на простих надлишкових моделях. Векторна логіка — це ідеальна для комп'ютингуекспоненційно надлишкова форма представлення функцій та структур у вигляді послідовності 2^n бітів. Логіка містить універсум законів для формування висновків з урахуванням результатів моделювання ситуації. Структури та функції — це дві форми логіки комп'ютингу. Структура — це форма логіки спрямована на людину. Функція — це форма логіки, орієнтована на комп'ютинг. Вектор (двійковий, розмірністю 2^n) співставляє структуру та функції. У чому різниця цих двох понять? Структура — це сукупність взаємозалежних компонентів універсуму. Функція — це співвідношення між двома універсумами. Універсум та зв'язки дозволяють визначити метрику подібності структури та функції. Форма представлення структури та функції генерує різницю між ними. Універсум та зв'язки відображено у таблиці істинності або логічному векторі, як її скороченій моделі, що є найефективнішою для комп'ютингу.

Введемо визначення: логічний вектор – це експоненційно надлишкова модель для опису та аналізу функцій та структур. Логічний вектор конвертує функцію у структуру і навпаки структуру у функцію. Перша конвертація необхідна для прийняття рішення людиною. Друга конвертація необхідна комп'ютингу з метою зменшення обчислювальної складності алгоритму або його обнулення. У першому та другому випадку використовується логічний вектор як модель для функції та структури.

Моделі на основі штучного інтелекту використовують матрично-векторні структури, що описують багатошарові графи. Але вони нічого не мають спільного з тими внутрішніми надлишковими моделями, які використовують провідні компанії (NVIDIA) для створення інтелектуального промпт-інжинірингу. Можна з упевненістю сказати, що всі моделі, що використовуються у штучному інтелекті для обнулення алгоритмів їх аналізу — це моделі на рівні векторів та матриць [1]. Даний тип моделей має перевагу перед класичним комп'ютингом, яка полягає у збереженні енергії та часу при моделюванні для симуляції запитів. Це є наслідком використання архітектури in-memory комп'ютингу, яка вільна від інструкцій процесора і використовує лише транзакції читання-запису на пам'яті.

Структури та функції як форми логіки комп'ютингу є найпоширенішими формами моделі, які разом із алгоритмами їх аналізу складають механізми сучасного комп'ютингу [2—4]. Розглядаються їх відношення щодо заміни однієї форми іншою з метою економії часу й енергії з організацією обчислювального процесу у пам'яті. Цифрова схема історично виникла тоді, коли потрібно було з таблиці істинності синтезувати швидкодіючу комбінаційну схему в умовах економії повільноїта дорогої пам'яті. Наразі пам'ять — доступний дешевий енергоефективний продукт, швидкодія якої визначається в наносекундному діапазоні. Синтез логічної схеми стає непотрібним, якщо функціональність можна описати логічним вектором у пам'яті для організації обчислень без інструкцій процесора. До того ж логічна схема не тільки вимагає дорогих процедур синтезу, а й створює алгоритмічні проблеми її моделювання, тестування, верифікації і діагностування. Натомість запропоновано повернутися до таблиці істинності (для логічного вектора), які не вимагають синтезу, потрібно лише виконати процедури моделювання в будь-яку пам'ять. При цьому за допомогою логічного вектора вирішуються завдання верифікації алгоритмами лінійної обчислювальної складності за рахунок експоненційної надлишковості моделі. Що стосується численних цифрових схем, які вже існують у кіберпросторі, можна робити їх реінжиніринг шляхом перетворення схеми на функціональність, представлену логічним вектором, а далі вирішувати всі проблеми верифікації алгоритмами лінійної обчислювальної складності. Пропонується використання логічного вектора комбінаційної схеми для симуляції справної поведінки та несправностей як адрес в архітектурі комп'ютингу в пам'яті. Логічний вектор є технологічним, компактним та вичерпним уявленням схеми для економного вирішення всіх завдань проектування і тестування. Декартова логіка, за рахунок її експоненційної 2^{n+m} надлишковості, є ефективним інтелектуальним механізмом вирішення комбінаторних завдань (моделювання, симуляція, тестування, діагностування) алгоритмами, що характеризуються лінійною обчислювальною складністю. Декартова логіка є інструментом вирішення таких завдань:

1. Справне моделювання логічного вектора схеми без використання алгоритму моделювання справної поведінки.

2. Моделювання несправностей на карті тестування логіки без використання алгоритму моделювання несправностей.

Розглядаються актуальні для EDA-ринку питання зменшення вартості та часу верифікації цифрових проектів за рахунок моделювання логічного вектора цифрової схеми, який дозволяє суттєво спростити алгоритми справної симуляції, а синтез карти тестування звести до трьох матричних

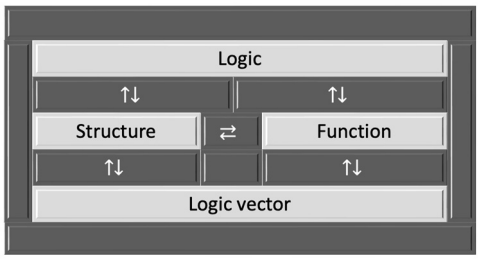


Рис. 1. Відношення між структурами та функціямикомп'ютингу

операцій. Практична значущість дослідження — це економічне (за витратами часу та енергії) векторно-логічне вирішення задач проєктування і тестування в архітектурі комп'ютингу в пам'яті на основі транзакцій читання-запису, вільної від інструкцій процесора. Теоретичний матеріал може бути корисним інженерам та студентам для розуміння процесів моделювання та симуляції при векторно-логічній верифікації цифрових проектів. Логічний вектор, таблиця істинності та матриця можуть розглядатись як масштабована форма визначення функціональності чи структури. Метрика логічного векторапрямована на явне завдання комбінаторної структури всіх відношень адресами бітів, що дозволяє моделювати вхідні умови без алгоритмів обробки. Логічний вектор є універсальною моделлю для аналізу фізичних та соціальних процесів відомими двійковими методами моделювання [4, 5]. Метрика функції та структури визначає їх схожість та відмінності (рис. 1), а також області їх можливого застосування як компонентів обчислювального механізму [6].

Аналогами відношень між функціями та структурами є функціональності та логічні схеми [7—9]. Перші наразі не потребують якоїсь додаткової обробки, пов'язаної із синтезом у дозволену систему технологічних елементів. Функціональність достатньо помістити в будь-яку пам'ять і використовувати як модель для симуляції вхідних двійкових наборів [10—13].

Друга модель (логічна схема) потребує засобів і часу для її синтезу, інструкцій процесора, мов опису апаратури та складних алгоритмів верифікації логічних схем, що синтезуються [14, 15]. Затримка отриманої логічної схеми в процесі її експлуатації може бути більшою, ніж час операцій запису та зчитування у пам'яті [16], необхідні для використання векторно-логічної моделі. Відсутність процесора для роботи векторно-логічної моделі робить її економічною з погляду часових та енергетичних витрат.

Комп'ютинг у пам'яті у поєднанні з векторною логікою є майбутнім мас-комп'ютингу [17], який можна швидко реалізувати за наявності матеріалу для організації пам'яті. Пропозиція відмовитися від логічних схем на користь логічних векторів — це виграш за часом та енергією комп'ютингу за рахунок експоненційних структур явних даних, представлених векторами, таблицями істинності або матрицями [17—20]. Можна припустити, що в найближчому десятилітті масово розвиватиметься in-memory інтелектуальний комп'ютинг на основі надлишкових векторно-матричних

структур даних, орієнтованих на формулювання коротких запитів до моделі генеративного штучного інтелекту для отримання вичерпної відповіді від розумних механізмів моделювання [10, 13—17]. Це особливо актуально для вирішення завдань у сфері проектування і тестування.

Виникають дві задачі:

1. Як побудувати логічний вектор процесу чи явища.

2. Як виконати моделювання вектора за існуючою логічною схемою [21—26].

Перший варіант — це унітарне кодування патернів процесу на універсумі знайдених примітивів з подальшим їх занесенням до рядків-змінних таблиць істинності. Після цього вирішуються всі комбінаторні завдання, пов'язані з аналізом великих даних отриманої таблиці істинності. Другий варіант — це пропозиція кількох механізмів обробки цифрових структур отримання логічного вектора схеми [27, 28], де запропонован рекурсивний алгоритм моделювання логічного вектора схеми за допомогою декартової логіки, що застосовується до елементів цифрової структури. Інший алгоритм заснований на використанні матриці паралельного моделювання кожної лінії схеми на наборах тих елементів, через які проходить сигнал на вихід, що моделюється. Третій алгоритм використовує логічні вектори елементів для заповнення таблиці справної поведінки всіх ліній схеми на вичерпному тесті. При цьому вхідні впливи елементів схеми розглядаються як адреси бітів логічного вектора, які формують стан лінії, що обробляється за допомогою транзакцій читання-запису [28]. Як приклад для верифікації запропонованих алгоритмів розглядається схема Шнейдера, яка представляє логічну структуру зі східними розгалуженнями (reconvergentfan-outs). Усі запропоновані алгоритми реалізовані в коді мови Python та розміщені як хмарний сервіс для вільного доступу студентів. Програмний застосунок Modeling for Simulation (MOSI) відкритий для користування студентами та фахівцями з метою навчання промпт-інжинірингу при вирішенні завдань проєктування і тестування.

Мета дослідження — зменшення часу верифікації цифрової схеми за рахунок побудови логічного вектора, що дозволяє будувати карту тестування функціональності трьома матричними операціями. Завдання дослідження:

1) розробка механізмів декартової логіки для побудови логічного вектора схеми;

2) розробка механізмів (моделі, алгоритми) побудови логічного вектора схеми, що базуються на використані векторних моделей елементів;

3) моделювання карти тестування логічної схеми, використовуючи побудований логічний вектор;

4) верифікація розроблених механізмів на прикладі схеми Шнейдера.

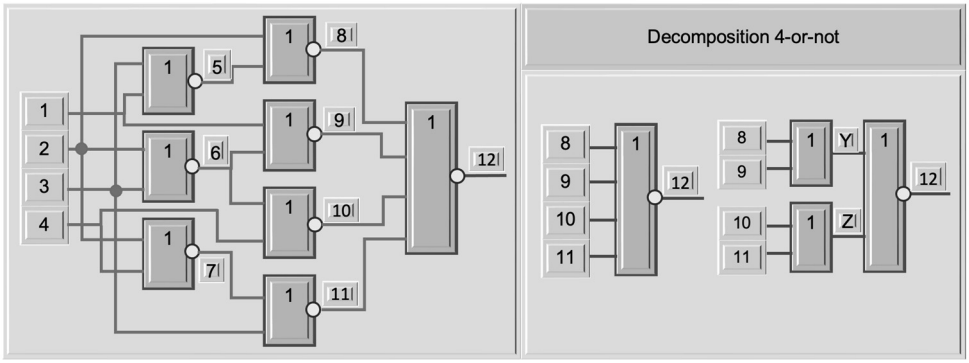


Рис. 2. Схема Шнейдера та декомпозиція чотиривходового елемента

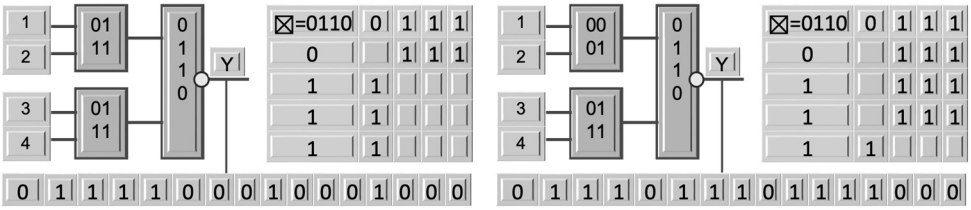


Рис. 3. Рефлексивні xor-відношення вектора 0111 та пари векторів 0001 та 0111

Побудова логічного вектора схеми методом декартової логіки. Процедура побудови логічного вектора розглядається на прикладі схеми Шнейдера (рис. 2), яка містить східні розгалуження на чотиривходовому елементі АБО-НЕ. Для того, щоб обробити цю схему декартовою логікою, необхідно привести всі елементи схеми до двовходових. Тому в правій частині рис. 2 представлено декомпозицію останнього елемента схеми на три елементи, два з яких елементи АБО, а третій елемент — НЕ. Логічний вектор — це уявлення процесу чи явища, функції чи структури послідовністю нулів і одиниць довжиною 2^n, де n — число бітів двійкових змінних для формування адрес бітів вектора. Це визначення робить логічний вектор незалежним від таблиці істинності. Але обов'язковим атрибутом вектора є стандартні адреси, які легко генеруються для будь-якого вектора. Декартовий добуток одного вектора формує рефлексивне (рис. 3) відношення $L = Y \boxtimes Y$, яке генерує декартову матрицю, в даному випадку активності, або логічний вектор L розмірністю 2^{n+1}.

Декартовий добуток двох різних векторів $0001 \boxtimes 0111 = 0111011101111000$ формує або синтезує новий вектор розмірністю $2^{n+1} = 2^n \times 2^m$. Схемотехнічне позначення такої операції може бути представлено трьома елементами. Декартова логіка є інваріантною до довжини логічних векторів. Тут використовуються таблиці істинності та логічні вектори. Цифрова схема взагалі не потрібна, виникають проблеми дорогого синтезу, складних

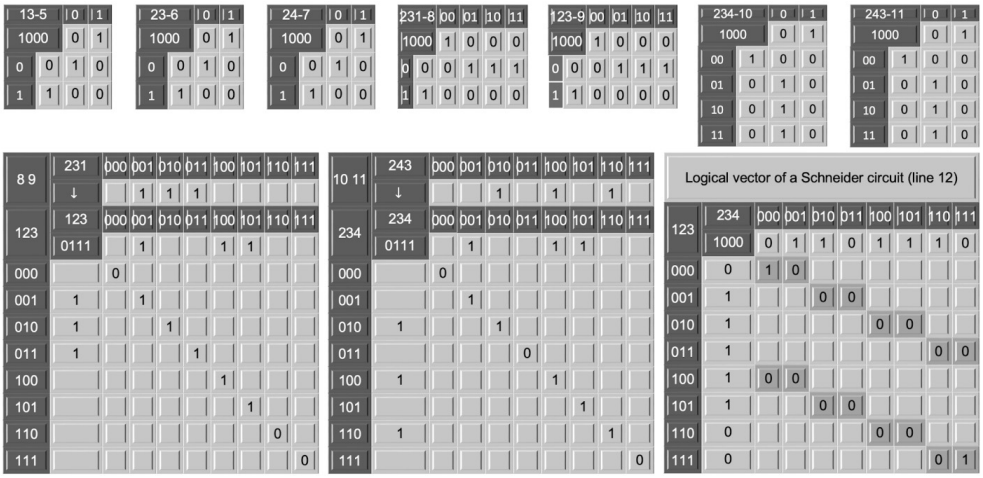


Рис. 4. Побудова логічного вектора за допомогою декартової логіки

алгоритмів верифікації. Потрібен лише стан виходу як реакція моделі на вхідне слово. Все це можна зробити в пам'яті без процесора на основі транзакцій читання-запису над бітами логічного вектора. Складність алгоритму (рис. 4) полягає у перекодуванні адрес таблиць істинності під нумерацію вхідних змінних (231-123, 243-234) для правильного розміщення одиничних значень логічного вектора. Останній елемент (чотиривходовий) розбивається на три двовходові елементи, де перші два є елементами АБО. Тому у двох декартових матрицях утворююча функція — це логічний вектор 0111. Надлишковість моделі визначається квадратом від довжини логічного вектора елемента. Власне, кожен елемент схеми обробляється окремо з урахуванням нумерації лінії схеми, а потім ітеративно всі елементи прив'язуються до вхідних змінних схеми. Алгоритм може будувати логічний вектор як для всієї схеми, так і її фрагментів, що закінчуються виходами елементів. Процедури побудови використовують лише транзакції читання-запису. Порожні клітини в матрицях позначають нулі. Вектори логічних елементів схеми зчитуються в матриці зліва-направо зверху-вниз за суттєвими координатами.

Недоліки запропонованого алгоритму побудови логічного вектора схеми:

1. Моделювання двовходових елементів схеми. Логіку з великою кількістю входів необхідно декомпозувати на двовходові елементи.

2. Необхідно виконувати перекодування адрес таблиці істинності та бітів логічного вектора кожного елемента відповідно до номерів зовнішніх входів схеми.

3. Складна структура даних для виконання алгоритму, що пов'язана з побудовою для кожного елемента декартової матриці.

Testing map																
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000		...1	..1.	..11	.1..	.1.1	.11.	.111	1...	1..1	1.1.	1.11	11..	11.1	111.	
0001		...0													111.	
0010			..0.											11.1		
0011				..00									11..			
0100					.0..							1.11				
0101						.0.0					1.1.					
0110							.00.			1..1						
0111								.000	1...							
1000								.111	0...							
1001							.11.			0..0						
1010						.1.1					0.0.					
1011					.1..							0.00				
1100				..11									00..			
1101			..1.											00.0		
1110		...1													000.	
1111		...0	..0.	..00	.0..	.0.0	.00.	.000	0...	0..0	0.0.	0.00	00..	00.0	000.	

Рис. 5. Моделювання карти тестування за дві матричні операції

Найголовніша перевага вектора полягає у тому, що його використання робить моделювання карти тестування функціональності вільним від алгоритму (рис. 5) [25, 28]. Для її побудови необхідно виконати лише дві матричні операції та логічний вектор схеми. Отримана карта тестування — це матриця відношень між тестовими наборами та комбінаціями вхідних несправностей цифрової функціональності, що перевіряються. На її основі можна вирішувати такі завдання:

1) визначити мінімальний тест для поодиноких константних несправностей;

2) визначити несправності, що перевіряються на заданих тестових наборах;

3) будувати схеми діагностування несправностей та тестування функціональності у режимі онлайн;

4) автоматично генерувати testbench(тестовий стенд)як мінімальну кількість тестових наборів, що перевіряють критичні режими функціональності.

У карті (матриці) тестування неперевірювані несправності на вхідних змінних представлені точками, а перевірювані — символами 0 і 1. Карта будується за три матричні операції [26—28]:

1) Xor — операція над бітами логічного вектора з метою отримання матриці активності;

2) отримання матриці дедуктивних векторів з урахуванням перекодування матриці активності за стандартними адресами таблиці істинності;

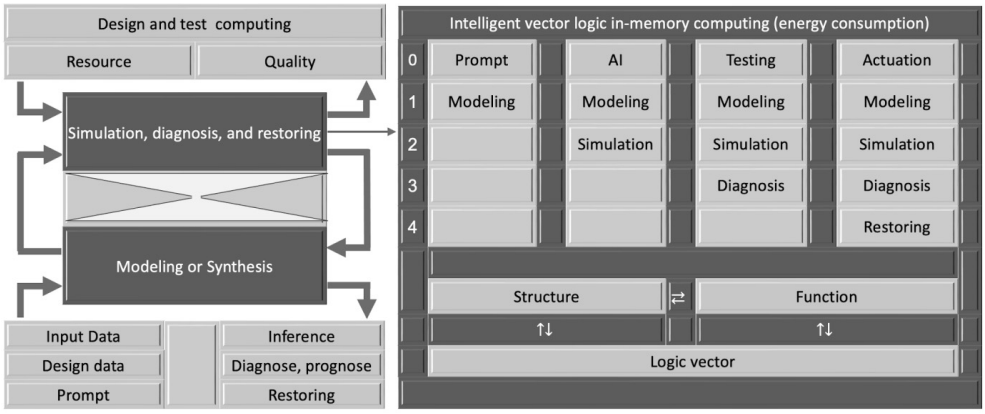


Рис. 6. Гармонія моделювання та синтезу в комп'ютингу

3) формування векторів несправностей, що перевіряються в 1-координатах карти тестування. Усі зазначені несправності інверсні до бітів тестових наборів.

Гармонія моделювання та синтезу в комп'ютингу. Інтелектуальний комп'ютинг (рис. 6) — це комбінація з трьох процедур: моделювання (створення або синтез моделі), симуляція (прогноз чи діагноз на створеній чи синтезованій моделі), актюація (управління об'єктами, процесами чи сервісами на створеній чи синтезованій моделі), що породжують чотири типи комп'ютингу. Взаємодія між комп'ютингом та людиною — це інтерфейс натуральної мови для користувача та двійкової мови для комп'ютингу. Це тренд п'ятої індустріальної революції. Інтелектуальний комп'ютинг проектування — це створення оцифрованої двійкової моделі процесу або явища з метою моделювання будь-яких вхідних ситуацій для отримання висновку (inference) у вигляді керуючих сигналів, прогнозу або діагнозу. По суті, в процесі моделювання використовують енергоефективні механізми, зручні для верифікації моделей процесів і явищ. В процесі синтезузастосовують механізми з низькою затримкою для моделювання та управління автономними агентами в реальному масштабі часу. Помилка в синтезі коштує мільйони доларів, помилка при моделюванні коштує сотні доларів. Тому взаємодія між моделюванням та синтезом має суто прагматичний характер, який визначається економікою комп'ютингу.

Моделювання допомагає синтезу позбутися помилок. У свою чергу синтез може бути корисним для моделювання в апаратній реалізації алгоритмів моделювання складних проектів для мінімізації часу їх верифікації. Ось такі гармонічні відношення існують між механізмами синтезу та моделювання. Структури і функції комп'ютингу перебувають у гармонічних відношеннях один з одним: коли зменшення обчислювальної складності

одного компонента призводить до збільшення іншого, і навпаки — теж правильно. Обидва компоненти описуються логічним вектором, що прийнятно для всіх алгоритмів аналізу великих даних, як адрес. У векторній моделі вирішуються всі завдання моделювання для симуляції як функціональностей, так і структур. Нижче наведено чотири моделі комп'ютингу, серед яких перші три орієнтовані на людину.

1. Промпт-моделювання карти тестування функціональності за три матричні операції без алгоритмів моделювання, що є найкращим рішенням.

2. Отримання рішення за допомогою штучного інтелекту — тривале навчання моделі для подальшого застосування алгоритмів моделювання.

3. Тест-комп'ютинг — створення моделі для подальшої симуляції з метою діагностування дефектів.

4. Актюаторнийкомп'ютинг, що є справжнімавтономнимкомп'ютингом без участі людини, який вирішує завдання створення моделі для виявлення та діагностування помилок з метою відновлення працездатності пристрою.

Логічний вектор, функція та структура. Існує деяке нерозуміння глибинних процесів, що відбуваються у сучасному комп'ютингу. Особливо це стосується відношень між функцією і структурою. Логічний вектор об'єднує ці два протилежні поняття, показуючи при цьому їх суттєві відмінності. Граф (структура) — це логіка, зручна для зорового сприйняття людиною, логіка в рисунку. Вектор (функція) — це логіка, зручна для комп'ютингу, логіка у векторі. Граф та вектор (структура та функція) — це дві форми логіки, які можна конвертувати одна в одну. Кожен граф — це вектор. Кожен вектор — це граф. Порожній граф — відсутність дуг. Повний та порожній граф не тестуються — це математичні константи, представлені векторами 1111 та 0000. Логічний вектор є універсальною, експоненційно надлишковою та простою моделлю опису структур та функцій, процесів та явищ, орієнтованою на обнулення алгоритмів її аналізу. Будь-яка таблиця істинності, матриця (квадратична чи прямокутна) є формою логічного вектора. Адресація бітів логічного вектора дозволяє вирішувати комбінаторні завдання комп'ютингу. Будь-яка скінченна дискретна структура може бути представлена логічним вектором. Будь-який логічний вектор конвертується у графову структуру. Матриця суміжностей графової структури є (неповним) логічним вектором. Декартова логіка — це $\boxtimes$ операція над двома логічними векторами $X \boxtimes Z$, довжиною 2^n, 2^m з метою отримання матриці або третього логічного вектора Y, довжиною 2^{n+m}. Тут n, m — будь-які цілі числа, $\boxtimes$ — будь-який вектор, завдовжки 4 біти. Будь-яка логічна декартова операція над векторами створює логічний вектор. Над графами, заданими логічними векторами, можна виконувати всі логічні операції (рис. 7). Це означає, що для графа за

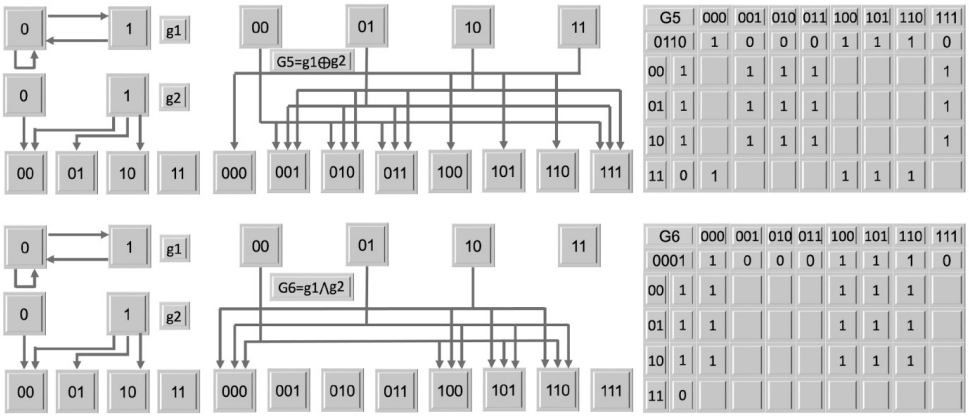
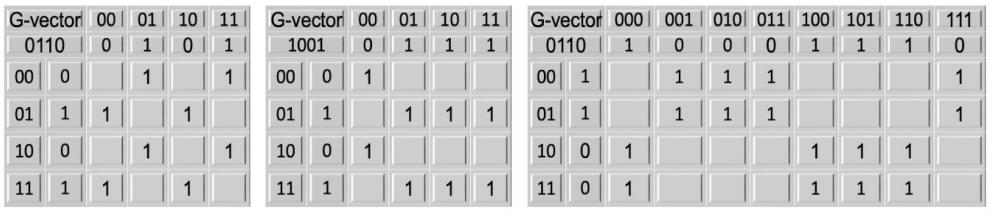


Рис. 7. Декартові та логічні операції над графами

логічним вектором можна будувати карту тестування несправностей та вирішувати всі завдання моделювання для симуляції вхідних наборів. Декартова операція з двох векторів різної довжини формує логічний вектор дводольного графа. Адреси бітів логічного вектора — це ідентифікатори вершин графа. Значення бітів логічного вектора — наявність (1) або відсутність (0) дуги в графі. Симуляція графа — це визначення двійкового значення виходу $Y(X) = \{0,1\}$ для вхідного набору X, що кодує дві вершини графа.

Комп'ютерна мова логіки — це моделювання для симуляції (modeling for simulation). Вектор — це будь-яка послідовність бітів. Логічний вектор — це послідовність з 2^n бітів, де n — ціле число (0, 1, 2, 3, …, n). Векторна логіка є експоненційнонадлишковою формою подання функцій і структур у вигляді послідовності 2^n бітів. Логіка векторного комп'ютингу — це структури та функції, представлені логічними векторами. Форми логіки — це вектор, граф (структура) і аналітичні вирази. Тількиекспоненційно надлишковий вектор є явною моделлю логіки, яка не потребує алгоритмів моделювання. Структурна та аналітична модель логіки потребує складних алгоритмів моделювання для отримання висновку (inference). Вектор — це самодостатня модель будь-якого процесу чи явища, функції чи структури. Будь-які логічні операції над векторами породжують вектор. Декартова логіка над векторами також породжує вектор із новими властивостями. Граф (структура) та функція — еквівалентні поняття. Це означає, що на векторній формі графів можна виконувати всі логічні та декартові операції. Обчислювальні рішення (карти тестування), отримані для функціональностей на логічних векторах, будуть дійсними та актуальними також для графів. Тут виникають два завдання:



G-vector	00	01	10	11
0110	0	1	0	1
00	0		1		1
01	1	1		1	
10	0		1		1
11	1	1		1	

G-vector	00	01	10	11
1001	0	1	1	1
00	0	1			
01	1		1	1	1
10	0	1			
11	1		1	1	1

G-vector	000	001	010	011	100	101	110	111
0110	1	0	0	0	1	1	1	0
00	1		1	1	1				1
01	1		1	1	1				1
10	0	1				1	1	1	
11	0	1				1	1	1	

Рис. 8. Декартова логіка на трьох прикладах

1. Візуалізація логічного вектора графом. Таке декоративне відображення графа процесу чи явища цікаве та приємне лише для очей людини, яка приймає рішення.

2. Побудова логічного вектору графа на основі його зображення. Це завдання орієнтоване на активний комп'ютинг і дозволяє «оживити» безліч графових моделей (схем, карт, скріншотів), що містяться в кіберпросторі, для практичних потреб бізнесу, транспорту та освіти. Суть полягає в побудові векторної моделі графа (карти скріншота) для симуляції (simulation) на ньому вхідних ситуацій з метою подальшого верифікованого управління процесами і явищами. Утворююча (декартова) логіка — це чотирибітовий ⊠-вектор, що створює матрицю (вектор, довжиною 2^{n+m}) на двох логічних векторах, розмірністю 2^n, 2^m. Конкатенація бітів двох векторів формує адресу комірки утворюючого вектора, вміст якої формує біти результуючого вектора або матриці (рис. 8). Декартова логіка, записана в дужках, породжує три вектори: 0101 (0110) 0101 = 0101101001011010; 0101 (1001) 0111 = =10000111100000111; 1100 (0110) 10001110 = 01110001011100011000111010001110.

Дводольний орієнтований граф — це дві множини вершин, які з'єднуються дугами, що йдуть в одному напрямку. Ідентифікатори вершин дводольного графа можуть мати різну кількість бітів. Утворююча (декартова) логіка — це ⊠-вектори, довжиною 4 біти, що створюють прямокутні матриці (вектори) дводольних графів на логічних векторах різної довжини. Утворювальна логіка створює стійкі переходи між двома множинами вершин графа, які визначаються одиничними координатами логічної матриці. Моделювання такого графа (матриці) — це налаштування зв'язків (тренінг) для вирішення конкретного завдання. Природно, будь-яка матриця (квадратична чи прямокутна) може бути представлена графом для візуалізації процесів, які відбуваються у матриці. Навпаки, теж вірно: будь-який граф може бути представлений у вигляді матриці, придатній для використання в алгоритмах комп'ютингу. Графи, при побудові карти тестування [25—28] відбивають результати матричних операцій:

1. *L*-matrix — це активність виходів функціональності $L = Y \oplus Y$, яка виходить з урахуванням декартової xor-операції над логічними вектором *Y*.

L1	1	1	0	1							00	01	10	11			00	01	10	11
	1			1			0	1	2	3	00			1			00		1.	
	1			1			1	0	3	2	01					1	01			10
	0	1	1		1		2	3	0	1	10		1	1	1		10	.1	0.	01
	1			1			3	2	1	0	11		1				11	.0		

L2	0	1	0	1							D	00	01	10	11		C	00	01	10	11
	0		1		1		0	1	2	3	00		1		1		00		.1		11
	1	1		1			1	0	3	2	01		1		1		01		.0		10
	0		1		1		2	3	0	1	10		1		1		10		.1		01
	1	1		1			3	2	1	0	11		1		1		11		.0		00

L3	1	0	0	0	1	1	1	0								D	000	001	010	011	100	101	110	111	C	000	001	010	011	100	101	110	111
	1		1	1	1				1		0	1	2	3	4	5	6	7	000			1		1					1	000		..1	.1.	.11				111
	0	1				1	1	1			1	0	3	2	5	4	7	6	001			1				1		1		001		..0			1..		1.0		110
	0	1				1	1	1			2	3	0	1	6	7	4	5	010				1		1			1	1	010			.0.		1..			10.	101
	0	1				1	1	1			3	2	1	0	7	6	5	4	011					1		1	1	1	011				.00		1.0	10.	100
	1		1	1	1				1		4	5	6	7	0	1	2	3	100					1		1	1	1	100				.11		0.1	01.	011
	1		1	1	1				1		5	4	7	6	1	0	3	2	101				1		1		1	1	101			.1.		0..		01.	010
	1		1	1	1				1		6	7	4	5	2	3	0	1	110			1			1	1		1	110		..1			0..	0.1		001
	0	1				1	1	1			7	6	5	4	3	2	1	0	111			1	1	1				1	111		..0	.0.	.00				000

Рис. 9. Три промпт-процедури побудови карт тестування

2. H — matrix — це константа перекодування матриці активності для отримання дедуктивної матриці $H = T \oplus A$.

3. D — matrix — це дедуктивна матриця активності $D = L_H$, впорядкована за стандартними HDL-адресами. Дана матриця містить множину дедуктивних рядків-векторів, які використовуються для транспортування несправностей вхідних змінних на вихід функціональності. Вона також містить множину похідних стовпців-векторів, які визначають умови для транспортування логічних активностей вхідних змінних на вихід функціональності.

4. C — matrix — це карта тестування всіх комбінацій несправностей F логічної функціональності на вичерпному тесті T: $C = T \leftrightarrow F$.

Тут ліві вершини представлені всіма вхідними наборами вичерпного тесту, праві вершини — це потенційні несправності, позначені одиничними значеннями бітів адрес. Ці адресні 1-біти завжди визначаються інверсними значеннями відповідних змінних у текстовому наборі. Результати такого довизначення формують вектори несправностей, які перевіряються для кожної правої вершини графа. Моделювання інших карт тестування наведено на рис. 9. Тут як запити (промпти) фігурують логічні вектори: $L1$=1101, $L2$=0101, $L3$=10001110. За логічними векторами за три матричні операції будуються карти тестування, які представлені відношеннями між вичерпним тестом і всіма комбінаціями вихідних несправностей, що перевіряються. У структурі даних, які тут використовуються, фігурують лише вектори та матриці, які теж є векторами, але більшої розмірності. Природно, що карта тестування функціональності є актуальною для верифікації графів або структур, заданих логічними векторами $L1$, $L2$, $L3$. Координати карти тестування — це вектори вхідних несправностей, що перевіряються, де символ «.» позначає неперевіряння несправностей щодо відповідної вхідної змінної. Несправності, що перевіряються, генеруються

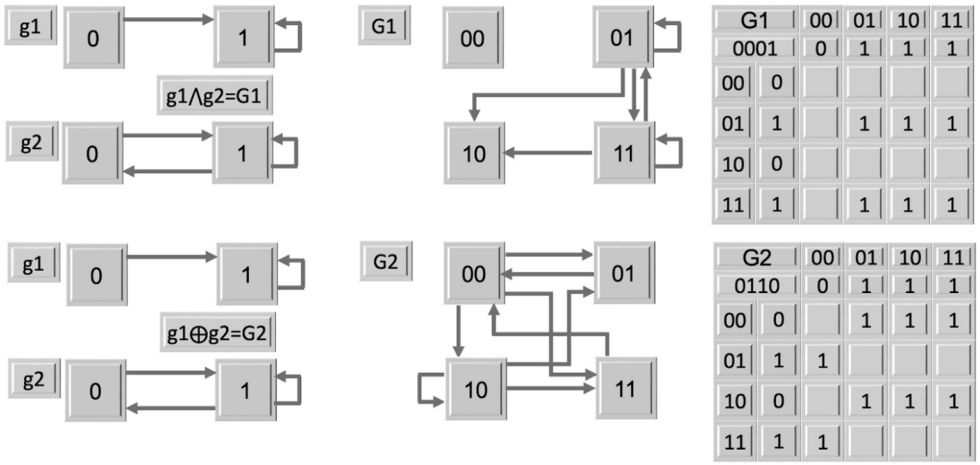


Рис. 10. Логіка векторів та графів комп'ютингу

за 1-координатами адрес бітів логічного вектора, який знаходиться у верхній частині матриці. Нульові координати адрес бітів формують точки. Вектори несправностей в *C*-карті тестування формують всі одиничні координати дедуктивної *D*-матриці. Пропонується моделювання графа (рис. 10) за утворювальною логічною операцією, заданою ⊠-вектором на основі двох підграфів, які також описуються логічними векторами.

Для цього використовуються такі аксіоми:

1. Будь-який граф — це логіка вектора, де поодинокі біти вказують на дуги між двома вершинами, що визначаються адресами бітів.

2. Будь-який логічний вектор є графом, де 1-координати ідентифікують дуги між двома вершинами, які становлять адресу цієї координати.

3. Два будь-які графи, що мають однакову розмірність логічних векторів, можуть формувати $2^n - 2$ $(n = 2)$ нетривіальних логічних відношень за допомогою декартової логіки.

4. Кількість вершин у графі, представленому логічним вектором, 2^n визначається як $k = 2^{n/2}$. Якщо $n = 4$, тоді кількість вершин у графі може бути більше 4.

5. Декартова логіка над двома векторами розмірності 2^n породжує логічний вектор довжиною 2^{2n}.

6. Декартова логіка — це логічна операція над бітами двох векторів, які розглядаються як адреси бітів утворювального логічного вектора, що формують біти третього вектора 2^{2n}.

7. Утворювальний логічний ⊠-вектор — це будь-який двійковий вектор розмірністю 4 біти.

8. Конкатенація адрес бітів логічних векторів створює адресу кожної комірки результуючого логічного вектора (матриці).

Адреси бітів вихідних векторів — це ідентифікатори вершин графа. Усі перетворення з логічними векторами знаходять своє відображення на графах. Навпаки, теж вірно, але алгоритмічно значно складніше. Граф G1 одержано за допомогою логічної декартової операції and (0001). Граф G2 отримано за допомогою логічної декартової операції xor (0110). Граф G3 отримано за допомогою логічної декартової операції or (0111). Граф G4 отримано за допомогою логічної декартової операції xor-not (1001). Якщо зробити, наприклад, логічну операцію and над двома векторами $0101 \wedge \wedge 0111=0101$, то отриманий результат якісно відрізняється від декартової and-операції над двома векторами $0101 \wedge 0111 = 0000011100000111$. Тут генеруються шість дуг за допомогою декартової and-операції. Потужність дуг результуючого графа визначається за формулою $a1 \times a2$.

Висновки

Представлено механізми моделювання логічного вектора цифрової схеми для побудови карти тестування схеми як логічної функціональності. Механізми вільні від використання інструкцій процесора і орієнтовані на обробку вхідних даних як адрес логічного вектора. При цьому використовується безпроцесорна архітектура комп'ютингу в пам'яті з урахуванням транзакцій читання-запису, що робить векторно-логічні механізми енерго- (36,3 %) і часо- (22,6 %) заощаджуючими [29]. Крім того, отриманий логічний вектор схеми є найпростішою і швидкодіючою паралельною моделлю для симуляції справної поведінки функціональності. Робиться висновок, що для сучасної швидкодіючої пам'яті (0,6 ns) [30] побудова вектора схеми є доцільним рішенням для імплементації функціональності в системі на кристалі SoC. Цифрова схема практично не має зараз переваг перед пам'яттю за швидкодією та енерговитратами.

Функціональність, записана у пам'яті у вигляді логічного вектора, не потребує синтезу та вільна від інструкцій процесора. Вона технологічна для вирішення задач моделювання, тестування, верифікації та діагностування. Запропоновані механізми моделювання логічного вектора цифрової схеми імплементовані у програмний застосунок MOSI, який підтримує будь-яку логіку, представлену у векторній формі. Він доступний для використанняяк хмарний сервіс або веб-додаток у навчальному процесі студентів напряму «Комп'ютерна інженерія».

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Markidis S. et al. NVIDIA Tensor Core Programmability, Performance & Precision. 2018.
2. IEEE Standard for Integrated Circuit (IC) Open Library Architecture (OLA). IEEE STD 1481-2009. 11 March 2010. P. 1-658.
3. Esmaieli E., SedaghatY., Peiravi A. Fanout-Based Reliability Model for SER Estimation in Combinational Circuits. IEEE Transactionson Circuitsand Systems I: Regular Papers. Jan. 2025. Vol. 72, no. 1. P. 228-240. doi: 10.1109/TCSI.2024.3458864 (дата звернення: 04.06.2026).
4. Hwang M.-E., Jung S.-O., Roy K. Slope Interconnect Effort: Gate-Interconnect Interdependent Delay Modeling for Early CMOS Circuit Simulation. IEEE Transactions on Circuits and Systems I: Regular Papers. July 2009. Vol. 56, no. 7. P. 1428-1441. doi: 10.1109/TCSI.2008.2006217 (дата звернення: 04.06.2026).
5. Huang W., Du J., Hua W., Bi K., Fan Q. A Hybrid Model-Based Diagnosis Approach for Open-Switch Faults in PMSM Drives. IEEE Transactions on Power Electronics. April 2022. Vol. 37, no. 4. P. 3728-3732. doi: 10.1109/TPEL.2021.3123144 (дата звернення: 04.06.2026).
6. Fu R., Wille R., Yoshikawa N., Ho T.-Y. Efficient Cartesian Genetic Programming-Based Automatic Synthesis Framework for Reversible Quantum-Flux-Parametron Logic Circuits. IEEE Transactionson Computer-Aided Design of Integrated Circuitsand Systems. 2025. doi: 10.1109/TCAD.2025.3546884 (дата звернення: 04.06.2026).
7. Berndt A. et al. Accuracy and Size Trade-offof a Cartesian Genetic Programming Flow for Logic Optimization. 2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI), Campinas, Brazil, 2021. P. 1-6. doi: 10.1109/SBCCI53441.2021.9529968 (дата звернення: 04.06.2026).
8. Dunaeva O., Autsou S., Kudelina K., Roosileht M. Utilizing the fuzzy logic algorithm for cartesian robot control system in conditions of mechanical damage transmission. IECON 2024 — 50th Annual Conference of the IEEE Industrial Electronics Society, Chicago, IL, USA. 2024. P. 1-5. doi: 10.1109/IECON55916.2024.10905445 (дата звернення: 04.06.2026).
9. Khan G. M., Zafari F., Mahmud S. A. Very Short-Term Load Forecasting Using Cartesian Genetic Programming Evolved Recurrent Neural Networks (CGPRNN). 2013 12th International Conference on Machine Learning and Applications, Miami, FL, USA.2013. P. 152-155. doi: 10.1109/ICMLA.2013.181 (дата звернення: 04.06.2026).
10. Inglese P., Vatajelu E. -I., Di Natale G. Side Channel and Fault Analyses on Memristor-Based Logic In-Memory. *IEEE Design & Test*, Vol. 41, no. 3. June 2024. P. 29-35. doi: 10.1109/MDAT.2023.3324522 (дата звернення: 04.06.2026).
11. Ye J., Zhu J., Cao J., Bi H., Ding Y., Chen B. A Novel Parallel In-Memory Logic Array Based on Programmable Diodes. *IEEE Journal of the Electron Devices Society*.Vol. 12. 2024. P. 738-744. doi: 10.1109/JEDS.2024.3457021 (дата звернення: 04.06.2026).
12. Chen X., Song T., Han Y. RRAM-based Analog In-Memory Computing: Invited Paper. *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, AB, Canada. 2021. P. 1-6. doi: 10.1109/NANOARCH53687.2021.9642235 (дата звернення: 04.06.2026).
13. Nguyen V.-T., Trinh Q.-K., Zhang R., Nakashima Y. XNOR-BSNN: In-Memory Computing Model for Deep Binarized Spiking Neural Network. *2021 International Conferenceon High Performance Big Data and Intelligent Systems (HPBD&IS)*, Macau, China. 2021. P. 17-21. doi: 10.1109/HPBDIS53214.2021.9658467 (дата звернення: 04.06.2026).
14. Hahanov V., Litvinova E., Davitadze Z., Chumachenko S., Devadze D., Hacimahmud A.V. Truth Table Based Intelligent Computing. *2024 31st International Conferenceon Mixed*

Design of Integrated Circuits and System (MIXDES), Gdansk, Poland. 2024. P. 199-204. DOI: 10.23919/MIXDES62605.2024.10614035 (дата звернення: 04.06.2026).
15. Privman V., Solenov D. Decoherence of Dynamically Manipulated Qubits. *2006 Sixth IEEE Conference on Nanotechnology*, Cincinnati, OH, USA. 2006. P. 842-845. DOI: 10.1109/NANO.2006.247790 (дата звернення: 04.06.2026).
16. Durmuş M. A., Demiralay K., Khan M. M., Atalay Ş. E., Sarpkaya İ. Moiré Localization Induced Enhancement in the Decoherence Time of Interlayer Excitonsin WSe2-MoSe2 Heterobilayers. *2024 Conference on Lasersand Electro-Optics (CLEO)*, Charlotte, NC, USA. 2024. P. 1-2.
17. 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*. Vol. 2. 2023. DOI: 10.34133/icomputing.0006 (дата звернення: 04.06.2026).
18. Tarek A., Alveed A., Farhan A. A Proof Theoretic Exploration of Mathematical Inductionin Computational Paradigm. *2023 Congressin Computer Science, Computer Engineering, & Applied Computing (CSCE)*, LasVegas, NV, USA. 2023. P. 963-972. DOI: 10.1109/CSCE60160.2023.00162 (дата звернення: 04.06.2026).
19. Kamide N. Natural Deduction with Explosion and Excluded Middle. *2023 IEEE 53rd International Symposium on Multiple-Valued Logic (ISMVL)*, Matsue, Japan. 2023. P. 24-29. DOI: 10.1109/ISMVL57333.2023.00016 (дата звернення: 04.06.2026).
20. Chang S.-H., Liu C.-N. J., Küster A. Behavioral Level Simulation Framework to Support Error-Aware CNN Trainingwith In-Memory Computing. *2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, Villasimius, Italy. 2022. P. 1-4. DOI: 10.1109/SMACD55068.2022.9816307 (дата звернення: 04.06.2026).
21. Li S., Zhang N., Zhang W., Yang R., Chang Y., Xiong B. Framework Independent Modeling for SRAM-Based In-Memory Computing. *2024 2nd International Symposium of Electronic Design Automation (ISEDA)*, Xi'an, China. 2024. P. 777-777. DOI: 10.1109/ISEDA62518.2024.10617799 (дата звернення: 04.06.2026).
22. Westphal J., Hardy J. Logicas a Vector System. *Journal of Logic and Computation*, Vol. 15, no. 5. Oct. 2005. P. 751-765. DOI: 10.1093/logcom/exi040 (дата звернення: 04.06.2026).
23. Mizraji E. Vector Logic: A Natural Algebraic Representation of the Fundamental Logical Gates. *Journal of Logic and Computation*. Vol. 18, no. 1. Feb. 2008. P. 97-121. DOI: 10.1093/logcom/exm057 (дата звернення: 04.06.2026).
24. Mizraji E. Vector logic allows counterfactual virtualization by the square root of NOT. *Logic Journal of the IGPL*. Vol. 29, no. 5. May 2020. P. 859-870. DOI: 10.1093/jigpal/jzaa026 (дата звернення: 04.06.2026).
25. Hahanov V., Gharibi W., Chumachenko S., Litvinova E. Vector Synthesis of Fault Testing Map for Logic. *IAES International Journal of Robotics and Automation (IJRA)*, Vol. 13, no. 3. 2024. P. 293-306. DOI:10.11591/ijra.v13i3.pp293-306 (дата звернення: 04.06.2026).
26. 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. *2024 IEEE East-West Design & Test Symposium (EWDTS)*, Yerevan, Armenia. 2024. P. 1-6. DOI: 10.1109/EWDTS63723.2024.10873615 (дата звернення: 04.06.2026).
27. Hahanov V., Devadze D., Hahanov I., Chumachenko S., Litvinova E., Obrizan V., Dmytro P., Mishchenko A., Maksymova N. Prompt-Testing of Logic. *2024 IEEE East-West Design & Test Symposium (EWDTS)*, Yerevan, Armenia. 2024. P. 1-5. DOI: 10.1109/EWDTS63723.2024.10873774 (дата звернення: 04.06.2026).

28. Hahanov V., Chumachenko S., Litvinova E., Hahanov I., Ponomarova V., Khakhanova H., Kulak G. Faults-as-address simulation. *IAES International Journal of Robotics and Automation*, Vol13, no 4. Dec. 2024. P. 452-468. DOI: http://doi.org/10.11591/ijra.v13i4.pp452-468 (дата звернення: 04.06.2026).
29. Ahn B., Jang J., Na H., Seo M., Son H., Song Y. H. AI Accelerator Embedded Computational Storage for Large-Scale DNN Models. *2022 IEEE 4th International Conference on Artificial Intelligence Circuitsand Systems (AICAS)*, Incheon, Korea, Republic of. 2022. P. 483-486. DOI: 10.1109/AICAS54282.2022.9869991 (дата звернення: 04.06.2026).
30. Wu B., Zhu H., Chen K., Yan C., Liu W. MLiM: High-Performance Magnetic Logic in-Memory Scheme With Unipolar Switching SOT-MRAM. *IEEE Transactions on Circuits and Systems I: Regular Papers*. Vol. 70, no. 6. June 2023. P. 2412-2424. DOI: 10.1109/TCSI.2023.3254607 (дата звернення: 04.06.2026).
31. Програмний застосунок MOSI. Електронний ресурс. mosilogic.github.io (дата звернення: 04.06.2026).

Стаття надійшла 11.03.2026;
Прорецензована 09.06.2026

REFERENCES

1. Markidis, S., et al. (2018). *NVIDIA Tensor Core programmability, performance & precision*.
2. Institute of Electrical and Electronics Engineers. (2010). *IEEE standard for integrated circuit (IC) open library architecture (OLA)* (IEEE Std 1481-2009, pp. 1-658). IEEE.
3. Esmaieli, E., Sedaghat, Y., & Peiravi, A. (2025). Fanout-based reliability model for SER estimation in combinational circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers, 72*(1), 228-240. https://doi.org/10.1109/TCSI.2024.3458864
4. Hwang, M.-E., Jung, S.-O., & Roy, K. (2009). Slope interconnect effort: Gate-interconnect interdependent delay modeling for early CMOS circuit simulation. *IEEE Transactions on Circuits and Systems I: Regular Papers, 56*(7), 1428-1441. https://doi.org/10.1109/TCSI.2008.2006217
5. Huang, W., Du, J., Hua, W., Bi, K., & Fan, Q. (2022). A hybrid model-based diagnosis approach for open-switch faults in PMSM drives. *IEEE Transactions on Power Electronics, 37*(4), 3728-3732. https://doi.org/10.1109/TPEL.2021.3123144
6. Fu, R., Wille, R., Yoshikawa, N., & Ho, T.-Y. (2025). Efficient Cartesian genetic programming-based automatic synthesis framework for reversible quantum-flux-parametron logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. https://doi.org/10.1109/TCAD.2025.3546884
7. Berndt, A., et al. (2021). Accuracy and size trade-off of a Cartesian genetic programming flow for logic optimization. In *2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)* (pp. 1-6). IEEE. https://doi.org/10.1109/SBCCI53441.2021.9529968
8. Dunaeva, O., Autsou, S., Kudelina, K., & Roosileht, M. (2024). Utilizing the fuzzy logic algorithm for Cartesian robot control system in conditions of mechanical damage transmission. In *IECON 2024 – 50th Annual Conference of the IEEE Industrial Electronics Society* (pp. 1-5). IEEE. https://doi.org/10.1109/IECON55916.2024.10905445
9. Khan, G.M., Zafari, F., & Mahmud, S.A. (2013). Very short-term load forecasting using Cartesian genetic programming evolved recurrent neural networks (CGPRNN). In *2013 12th International Conference on Machine Learning and Applications* (pp. 152-155). IEEE. https://doi.org/10.1109/ICMLA.2013.181

10. Inglese, P., Vatajelu, E.-I., & Di Natale, G. (2024). Side channel and fault analyses on memristor-based logic in-memory. *IEEE Design & Test, 41*(3), 29-35. https://doi.org/10.1109/MDAT.2023.3324522
11. Ye, J., Zhu, J., Cao, J., Bi, H., Ding, Y., & Chen, B. (2024). A novel parallel in-memory logic array based on programmable diodes. *IEEE Journal of the Electron Devices Society, 12*, 738-744. https://doi.org/10.1109/JEDS.2024.3457021
12. Chen, X., Song, T., & Han, Y. (2021). RRAM-based analog in-memory computing: Invited paper. In *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)* (pp. 1-6). IEEE. https://doi.org/10.1109/NANOARCH53687.2021.9642235
13. Nguyen, V.-T., Trinh, Q.-K., Zhang, R., & Nakashima, Y. (2021). XNOR-BSNN: In-memory computing model for deep binarized spiking neural network. In *2021 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)* (pp. 17-21). IEEE. https://doi.org/10.1109/HPBDIS53214.2021.9658467
14. Hahanov, V., Litvinova, E., Davitadze, Z., Chumachenko, S., Devadze, D., & Hacimahmud, A.V. (2024). Truth table based intelligent computing. In *2024 31st International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES)* (pp. 199-204). IEEE. https://doi.org/10.23919/MIXDES62605.2024.10614035
15. Privman, V., & Solenov, D. (2006). Decoherence of dynamically manipulated qubits. In *2006 Sixth IEEE Conference on Nanotechnology* (pp. 842-845). IEEE. https://doi.org/10.1109/NANO.2006.247790
16. Durmuş, M.A., Demiralay, K., Khan, M.M., Atalay, Ş.E., & Sarpkaya, İ. (2024). Moiré localization induced enhancement in the decoherence time of interlayer excitons in WSe2-MoSe2 heterobilayers. In *2024 Conference on Lasers and Electro-Optics (CLEO)* (pp. 1-2).
17. 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. (2023). *Intelligent computing: The latest advances, challenges, and future*. (2). https://doi.org/10.34133/icomputing.0006
18. Tarek, A., Alveed, A., & Farhan, A. (2023). A proof theoretic exploration of mathematical induction in computational paradigm. In *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)* (pp. 963-972). IEEE. https://doi.org/10.1109/CSCE60160. 2023.00162
19. Kamide, N. (2023). Natural deduction with explosion and excluded middle. In *2023 IEEE 53rd International Symposium on Multiple-Valued Logic (ISMVL)* (pp. 24-29). IEEE. https://doi.org/10.1109/ISMVL57333.2023.00016
20. Chang, S.-H., Liu, C.-N. J., & Küster, A. (2022). Behavioral level simulation framework to support error-aware CNN training with in-memory computing. In *2022 18th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)* (pp. 1-4). IEEE.https://doi.org/10.1109/SMACD55068.2022.9816307
21. Li, S., Zhang, N., Zhang, W., Yang, R., Chang, Y., & Xiong, B. (2024). Framework independent modeling for SRAM-based in-memory computing. In *2024 2nd International Symposium of Electronic Design Automation (ISEDA)* (p. 777). IEEE. https://doi.org/10.1109/ISEDA62518.2024.10617799
22. Westphal, J., & Hardy, J. (2005). Logic as a vector system. *Journal of Logic and Computation, 15*(5), 751-765. https://doi.org/10.1093/logcom/exi040
23. Mizraji, E. (2008). Vector logic: A natural algebraic representation of the fundamental logical gates. *Journal of Logic and Computation, 18*(1), 97-121. https://doi.org/10.1093/logcom/exm057
24. Mizraji, E. (2020). Vector logic allows counterfactual virtualization by the square root of NOT. *Logic Journal of the IGPL, 29*(5), 859-870. https://doi.org/10.1093/jigpal/jzaa026

25. Hahanov, V., Gharibi, W., Chumachenko, S., & Litvinova, E. (2024). Vector synthesis of fault testing map for logic. *IAES International Journal of Robotics and Automation, 13*(3), 293-306. https://doi.org/10.11591/ijra.v13i3.pp293-306
26. Hahanov, V., Litvinova, E., Hahanova, H., Chumachenko, S., Davitadze, Z., Hahanova, I., Kulak, H., Ponomarova, V., & Abdullayev, V. H. (2024). Vector-logical in-memory simulation of faults as truth table addresses. In *2024 IEEE East-West Design & Test Symposium (EWDTS)* (pp. 1-6). IEEE. https://doi.org/10.1109/EWDTS63723.2024.10873615
27. Hahanov, V., Devadze, D., Hahanov, I., Chumachenko, S., Litvinova, E., Obrizan, V., Dmytro, P., Mishchenko, A., & Maksymova, N. (2024). Prompt-testing of logic. In *2024 IEEE East-West Design & Test Symposium (EWDTS)*(pp. 1-5). IEEE. https://doi.org/10.1109/EWDTS63723.2024.10873774
28. Hahanov, V., Chumachenko, S., Litvinova, E., Hahanov, I., Ponomarova, V., Khakhanova, H., & Kulak, G. (2024). Faults-as-address simulation. *IAES International Journal of Robotics and Automation, 13*(4), 452-468. https://doi.org/10.11591/ijra.v13i4.pp452-468
29. Ahn, B., Jang, J., Na, H., Seo, M., Son, H., & Song, Y. H. (2022). AI accelerator embedded computational storage for large-scale DNN models. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (pp. 483-486). IEEE. https://doi.org/10.1109/AICAS54282.2022.9869991
30. Wu, B., Zhu, H., Chen, K., Yan, C., & Liu, W. (2023). MLiM: High-performance magnetic logic in-memory scheme with unipolar switching SOT-MRAM. *IEEE Transactions on Circuits and Systems I: Regular Papers, 70*(6), 2412-2424. https://doi.org/10.1109/TCSI.2023.3254607
31. MOSI. (n.d.). *MOSI software application*. Retrieved June 4, 2026, from https://mosilogic.github.io

Article received 11.03.2026;
Reviewed 09.06.2026

V.I. Hahanov, E.I. Litvinova, A.O. Voronov

VECTOR LOGIC FOR TESTING FUNCTIONS AND STRUCTURES

Vector logic computations are an efficient mechanism for in-memory intelligent computations that use read-write transactions to solve practical problems of analysis and control of physical, social, and business processes based on monitoring. For any given situation, an over-specified model of the digital structure and functionality can be constructed, which eliminates the need for computational analysis algorithms to arrive at a solution. The research aims to reduce the time and energy costs associated with managing processes and phenomena in the digital world. The means of achieving this goal is the use of vector-logical in-memory computing mechanisms, which simplify algorithms through the exponential redundancy of data structures. Functions and structures are analyzed from the perspective of algorithmic simplicity for modeling and simulation. Several mechanisms based on Cartesian logic are proposed for modeling a logical vector according to the structure of a digital circuit. The logical vector is used to generate a test map using three matrix operations. The MOSI cloud service is proposed, which models test sets and faults as truth table addresses in digital circuits and functionalities.

K e y w o r d s: computing functions and structures, vector logic functionality, vector logic, logical vector, Cartesian logic, intelligent computing, test map, fault simulation, good-value simulation, prompt engineering.

ХАХАНОВ Володимир Іванович, д-р техн. наук, професор, професор кафедри автоматизації проєктування обчислювальної техніки Харківського національного університету радіоелектроніки (ХНУРЕ). У 1978 р. закінчив Харківський інститут радіоелектроніки. Область наукових досліджень — комп'ютерні системи та мережі, квантовий, кіберфізичний, кіберсоціальний, векторно-логічний комп'ютинг.

ЛИТВИНОВА Євгенія Іванівна, д-р техн. наук, професор, професор кафедри автоматизації проєктування обчислювальної техніки Харківського національного університету радіоелектроніки (ХНУРЕ). У 1985 р. закінчила Харківський інститут радіоелектроніки. Область наукових досліджень — квантовий комп'ютинг, тестування та діагностика цифрових систем.

ВОРОНОВ Андрій Олександрович, аспірант кафедри автоматизації проєктування обчислювальної техніки Харківського національного університету радіоелектроніки (ХНУРЕ). У 2022 р. закінчив Харківський національний університет будівництва та архітектури. Область наукових досліджень — кіберфізичний комп'ютинг, проєктування і тестування систем на кристалах (SoC), машинне навчання.