МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Кваліфікаційна наукова
праця на правах рукопису

КИРИЛЮК ТАРАС ПЕТРОВИЧ

УДК 004.4:004.8

ДИСЕРТАЦІЯ

СИНТЕЗ ВІДМОВОСТІЙКИХ ЗВОРОТНИХ ЛОГІЧНИХ ПРИСТРОЇВ МЕТОДАМИ ШТУЧНОГО ІНТЕЛЕКТУ

121 – Інженерія програмного забезпечення
12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.
_____ Т. П. Кирилюк

Науковий керівник **Дейбук Віталій Григорович**, доктор фізико-математичних наук, професор

Чернівці - 2026

АНОТАЦІЯ

Кирилюк Т. П. Синтез відмовостійких зворотних логічних пристроїв методами штучного інтелекту. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 – «Інженерія програмного забезпечення» – Чернівецький національний університет імені Юрія Федьковича, Чернівці, 2026.

Дисертаційна робота присвячена вирішенню науково-прикладної задачі підвищення ефективності автоматизованого проєктування зворотних та квантових логічних схем з використанням методів штучного інтелекту, зокрема розробці інформаційної технології для структурного синтезу відмовостійких пристроїв, які характеризуються мінімальною квантовою вартістю та здатністю до збереження парності сигналів на базі узагальнених вентилів Фредкіна. Також вивчено та вдосконалено моделі еволюційного пошуку, які завдяки використанню об'єктно-орієнтованого представлення хромосом та адаптивних операторів мутації є основним алгоритмічним інструментом при генерації оптимальної топології схеми великої розмірності.

Результати роботи є підґрунтям для подальших теоретичних і практичних наукових розробок у галузі автоматизації проєктування квантових обчислювачів та криптографічних систем на їх основі.

Дисертація складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків. У **вступі** обґрунтовано актуальність теми дослідження, сформульовано мету, завдання, предмет, об’єкт та методи дослідження, вказано наукову новизну, теоретичне та практичне значення отриманих результатів, подано та проаналізовано зв’язок роботи з науковими темами. Зазначено особистий внесок здобувача, а також наведено відомості про апробацію та публікації основних результатів дисертації. Описано структуру та обсяг дисертаційної роботи.

Перший розділ містить ключові відомості з теорії зворотної логіки, опис основних напрямів досліджень та завдання, якими займається сучасна інженерія квантових зворотних схем. Тут проведено огляд та опис існуючих підходів до синтезу (детерміновані алгоритми, класичні еволюційні стратегії), виявлено їхні обмеження, пов'язані з експоненційним зростанням обчислювальної складності. Розглянуто та проаналізовано базові логічні елементи: вентилі Фейнмана, Тоффолі, які є основою для побудови енергоефективних обчислювачів, та їх особливості в контексті відмовостійкості. Окремо описано перевагу вентиля Фредкіна через його унікальну топологію яка забезпечує ефект збереження парності сигналів. Здійснено класифікацію та огляд методів однієї з важливих технік штучного інтелекту – еволюційних обчислень, та обґрунтовано доцільність використання генетичних алгоритмів для вирішення багатокритеріальних оптимізаційних задач синтезу.

У **другому розділі** систематизовано теоретичні засади проєктування зворотних схем та проведено класифікацію логічних вентилів, від класичних бінарних до квантових вентилів. Розглянуто матричне представлення логічних елементів, що дозволило формалізувати процеси перетворення інформації у векторному просторі станів. Проаналізовано умови функціональної повноти для універсальних базисів, що є необхідною умовою для синтезу довільної булевої функції. Визначено ключові критерії якості синтезованих рішень: квантову вартість, глибину схеми та апаратну складність. Окрему увагу приділено проблемі надлишковості виходів та питанням відмовостійкості, зокрема методам збереження парності сигналів для забезпечення надійності обчислень. Здійснено огляд існуючих метаевристичних підходів до синтезу, таких як метод симульованого відпалу, алгоритми мурашиних колоній та класичні генетичні алгоритми. Виявлено їхні спільні недоліки: схильність до передчасної збіжності,

застрягання в локальних екстремумах та низьку ефективність при роботі зі схемами великої розмірності.

Основні результати даного розділу можна підсумувати наступним чином:

– формалізовано математичний апарат зворотної логіки для подальшої автоматизації синтезу;

– обґрунтовано вибір метрик оптимізації, які враховують як мінімізацію ресурсів, так і вимоги до відмовостійкості;

– на основі порівняльного аналізу доведено необхідність розробки вдосконаленого генетичного методу, здатного подолати обмеження існуючих аналогів.

Третій розділ присвячений розробці та програмній реалізації вдосконаленого генетичного методу структурного синтезу зворотних схем. Особливу увагу приділено розширенню бази шляхом синтезу нових реконфігурованих пристроїв на основі узагальненого вентиля Фредкіна. Використання таких вентилів дозволяє створювати схеми з внутрішньою апаратною відмовостійкістю, завдяки збереженню парності сигналів між входами та виходами, що є сильною стороною запропонованого підходу. Детально описано запропоновану модифікацію алгоритму, яка, на відміну від класичних підходів, використовує об'єктно-орієнтовану модель представлення хромосом. Це дозволило перейти від «сліпої» оптимізації бітових масивів до оперування функціональними об'єктами вентилів і схем з визначеними властивостями, що суттєво зменшує простір пошуку рішень. Розроблено та обґрунтовано комплексний оператор багатокомпонентної мутації, який інтегрує три типи структурних змін: додавання, видалення, та модифікація параметрів та забезпечує баланс між дослідженням простору рішень та експлуатацією знайдених оптимумів. Для вирішення проблеми маршрутизації сигналів впроваджено механізм динамічної перестановки вихідних ліній, що дозволяє алгоритму ефективно виходити з локальних

екстремумів. Окрему увагу приділено програмній реалізації методу. Описано архітектуру розробленого програмного комплексу мовою C#, який здійснює повний цикл синтезу. Реалізовано модуль автоматичної трансляції схем у формат OpenQASM, що забезпечило пряму сумісність із квантовим фреймворком IBM Qiskit для верифікації результатів.

Основні результати даного розділу можна підсумувати наступним чином:

– запропоновано розширений базис на основі реконфігурованих узагальнених вентилів Фредкіна, що забезпечує апаратну відмовостійкість синтезованих схем;

– розроблено вдосконалений генетичний метод із використанням об'єктно-орієнтованої моделі хромосоми;

– реалізовано адаптивні еволюційні оператори для оптимізації топології та маршрутизації зворотних схем;

– створено автоматизовану систему синтезу зворотних пристроїв, інтегровану з сучасними засобами квантового моделювання.

У **четвертому розділі** наведено результати експериментальних досліджень та практичної апробації розробленого математичного і програмного забезпечення. Здійснено синтез еталонних тестових наборів зворотних функцій різної розмірності та проведено порівняльний аналіз отриманих схем з результатами роботи відомих аналогів. Статистичний аналіз підтвердив, що запропонований вдосконалений генетичний метод дозволяє зменшити квантову вартість схем при збереженні прийнятного часу генерації. Особливу увагу приділено практичному застосуванню розробленої технології. Описано процес синтезу відмовостійкого зворотного потокового шифратора на базі реконфігурованих елементів RRG2 та RRG3 та його подальшу апаратну реалізацію на базі FPGA сімейства Altera Cyclone IV. Для верифікації проєкту застосовано метод вичерпного

тестування, який підтвердив повну функціональну еквівалентність синтезованої схеми вихідній специфікації.

Основні результати даного розділу можна підсумувати наступним чином:

– експериментально підтверджено ефективність розробленого методу на стандартних наборах тестових функцій;

– здійснено успішну апаратну імплементацію синтезованого криптографічного пристрою з підтвердженою відмовостійкістю, досягнуто високих показників швидкодії (максимальна частота $F_{max} \approx 310$ МГц);

– доведено придатність розроблених засобів для автоматизованого проєктування високопродуктивних компонентів систем на кристалі.

У **висновках** підсумовано основні теоретичні та практичні результати дисертаційного дослідження, які в сукупності вирішують актуальну науково-прикладну задачу підвищення ефективності синтезу зворотних відмовостійких логічних пристроїв.

У **додатках** подано наукові публікації, в яких відображено основні результати роботи, відомості про апробацію – акти про впровадження результатів дослідження, лістинги ключових модулів розробленого програмного забезпечення, приклади синтезованих схем у форматі OpenQASM та фрагменти Verilog-коду для FPGA.

Теоретичне значення. Результати теоретичних досліджень, а саме розроблена математична модель структурного синтезу, розширений базис реконфігурованих вентилів Фредкіна та модифікований генетичний метод з об’єктно-орієнтованою хромосомою, поглиблюють теорію проєктування квантових обчислювачів. Вони можуть використовуватися для подальших наукових досліджень у галузі автоматизації проєктування, а також у навчальних курсах кафедр програмного забезпечення комп’ютерних систем та комп’ютерних систем та мереж Чернівецького національного

університету імені Юрія Федьковича, пов'язаних з теорією алгоритмів, архітектурою комп’ютерних систем та квантовими обчисленнями.

Практичне значення. Розроблений у дисертаційній роботі програмний комплекс автоматизованого синтезу, модуль трансляції схем у формат OpenQASM та реалізований на FPGA відмовостійкий зворотний шифратор на базі узагальнених вентилів Фредкіна можуть в подальшому використовуватися для створення енергоефективних компонентів систем на кристалі. Запропоновані підходи до інтеграції з IBM Qiskit дозволяють верифікувати складні квантові алгоритми.

Ключові слова: оптимізація, програмне забезпечення, продуктивність, інформаційна система/інформаційні технології, модель, відмовостійкість, автоматизація, моделювання, генетичний алгоритм, швидка еволюція, машинне навчання, інтелектуальна система, мутація параметрів, штучний інтелект, алгоритм.

ABSTRACT

Kyrylyuk T. Synthesis of fault-tolerant reversible logic devices using artificial intelligence methods. – Qualification scientific work as a manuscript.

Dissertation for the degree of Doctor of Philosophy in specialty 121 – “Software Engineering” – Yuriy Fedkovych Chernivtsi National University, Chernivtsi, 2026.

The dissertation is devoted to solving the scientific and applied problem of increasing the efficiency of automated design of reversible and quantum logic circuits using artificial intelligence methods, in particular, the development of information technology for the structural synthesis of fault-tolerant devices, which are characterized by minimal quantum cost and the ability to preserve signal parity based on generalized Fredkin gates. Evolutionary search models have also been studied and improved, which, thanks to the use of object-oriented representation of chromosomes and adaptive mutation operators, are the main algorithmic tool in generating the optimal topology of a high-dimensional circuit.

The results of the work are the basis for further theoretical and practical scientific developments in the field of automation of the design of quantum computers and cryptographic systems based on them.

The dissertation consists of an introduction, four sections, conclusions, a list of sources used and appendices. The **introduction** substantiates the relevance of the research topic, formulates the goal, objectives, subject, object and methods of the research, indicates the scientific novelty, theoretical and practical significance of the results obtained, presents and analyzes the connection of the work with scientific topics. The personal contribution of the applicant is indicated, and information about the testing and publication of the main results of the dissertation is provided. The structure and scope of the dissertation work are described.

The **first chapter** contains key information on the theory of reversible logic, a description of the main areas of research and the tasks that modern quantum reversible circuit engineering deals with. Here, a review and description of existing approaches to synthesis (deterministic algorithms, classical evolutionary strategies) are conducted, their limitations associated with the exponential growth of computational complexity are identified. The basic logical elements are considered and analyzed: Feynman and Toffoli gates, which are the basis for building energy-efficient computers, and their features in the context of fault tolerance. The advantage of the Fredkin gate due to its unique topology that provides the effect of preserving signal parity is separately described. The methods of one of the important techniques of artificial intelligence - evolutionary computing - are classified and reviewed, and the feasibility of using genetic algorithms for solving multi-criteria optimization problems of synthesis is substantiated.

In the **second chapter**, the theoretical principles of designing reversible circuits are systematized and a classification of logic gates is carried out, from classical binary to quantum gates. The matrix representation of logic elements is considered, which allowed formalizing the processes of information transformation in the vector space of states. The conditions of functional completeness for universal bases, which is a

necessary condition for the synthesis of an arbitrary Boolean function, are analyzed. The key criteria for the quality of synthesized solutions are determined: quantum cost, circuit depth, and hardware complexity. Special attention is paid to the problem of output redundancy and fault tolerance issues, in particular, methods for preserving signal parity to ensure the reliability of calculations. A review of existing metaheuristic approaches to synthesis, such as the simulated annealing method, ant colony algorithms, and classical genetic algorithms, is carried out. Their common shortcomings are identified: a tendency to premature convergence, getting stuck in local extrema, and low efficiency when working with high-dimensional circuits.

The main results of this section can be summarized as follows:

– the mathematical apparatus of reversible logic is formalized for further automation of synthesis;

– the choice of optimization metrics is justified, which take into account both resource minimization and fault tolerance requirements;

– based on a comparative analysis, the need to develop an improved genetic method capable of overcoming the limitations of existing analogues is proven.

The **third chapter** is devoted to the development and software implementation of an improved genetic method for structural synthesis of reversible circuits. Particular attention is paid to expanding the base by synthesizing new reconfigurable devices based on the generalized Fredkin gate. The use of such gates allows creating circuits with internal hardware fault tolerance, due to the preservation of signal parity between inputs and outputs, which is a strong point of the proposed approach. The proposed modification of the algorithm is described in detail, which, unlike classical approaches, uses an object-oriented model of chromosome representation. This allowed us to move from "blind" optimization of bit arrays to operating on functional objects of gates and circuits with defined properties, which significantly reduces the solution search space. A complex multicomponent mutation operator has been developed and substantiated, which integrates three types of structural changes: addition, deletion, and modification of parameters and provides a balance between exploring the solution space and

exploiting the found optima. To solve the signal routing problem, a mechanism for dynamic permutation of output lines has been introduced, which allows the algorithm to effectively exit local extrema. Special attention is paid to the software implementation of the method. The architecture of the developed software complex in C#, which performs a full synthesis cycle, is described. A module for automatic translation of schemes into the OpenQASM format has been implemented, which provided direct compatibility with the IBM Qiskit quantum framework for verifying results.

The main results of this section can be summarized as follows:

– an extended basis based on reconfigured generalized Fredkin gates has been proposed, which provides hardware fault tolerance of synthesized schemes;

– an improved genetic method has been developed using an object-oriented chromosome model;

– adaptive evolutionary operators are implemented to optimize the topology and routing of reversible circuits;

– an automated system for the synthesis of reversible devices is created, integrated with modern quantum modeling tools.

The **fourth chapter** presents the results of experimental research and practical testing of the developed mathematical and software. The synthesis of reference test sets of reversible functions of various dimensions is carried out and a comparative analysis of the obtained circuits with the results of the work of known analogues is carried out. Statistical analysis confirmed that the proposed improved genetic method allows to reduce the quantum cost of circuits while maintaining an acceptable generation time. Special attention is paid to the practical application of the developed technology. The process of synthesizing a fault-tolerant reversible stream encoder based on reconfigurable elements RRG2 and RRG3 and its subsequent hardware implementation based on the FPGA of the Altera Cyclone IV family is described. The exhaustive testing method was used to verify the project, which confirmed the full functional equivalence of the synthesized circuit to the original specification.

The main results of this section can be summarized as follows:

– the effectiveness of the developed method was experimentally confirmed on standard sets of test functions;

– a successful hardware implementation of a synthesized cryptographic device with confirmed fault tolerance was carried out, high performance indicators were achieved (maximum frequency F_{max}≈310 MHz);

– the suitability of the developed tools for automated design of high-performance components of systems on a crystal was proven.

The **conclusions** summarize the main theoretical and practical results of the dissertation research, which together solve the current scientific and applied problem of increasing the efficiency of synthesis of reversible fault-tolerant logic devices.

The **appendices** contain scientific publications that reflect the main results of the work, information on testing - acts on the implementation of the research results, listings of key modules of the developed software, examples of synthesized circuits in OpenQASM format and fragments of Verilog code for FPGA.

Theoretical significance. The results of theoretical research, namely the developed mathematical model of structural synthesis, the extended basis of reconfigurable Fredkin gates and the modified genetic method with an object-oriented chromosome, deepen the theory of quantum computer design. They can be used for further scientific research in the field of design automation, as well as in training courses of the departments of computer systems software and computer systems and networks of Yuriy Fedkovych Chernivtsi National University related to the theory of algorithms, architecture of computer systems and quantum computing.

Practical significance. The software complex of automated synthesis developed in the dissertation work, the module for translating schemes into the OpenQASM format and the fault-tolerant reversible encoder based on generalized Fredkin gates implemented on FPGA can be further used to create energy-efficient components of systems-on-Chip. The proposed approaches to integration with IBM Qiskit allow for the verification of complex quantum algorithms.

Keywords: optimization, software, performance, information system/information technology, model, fault tolerance, automation, modeling, genetic algorithm, rapid evolution, machine learning, intelligent system, parameter mutation, artificial intelligence, algorithm.

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці у виданнях, включених до переліку наукових фахових видань України та проіндексованих у наукометричних базах даних Web of Science Core Collection та/або Scopus:

1. Kyryliuk T., Palahuta M., Deibuk V. Using artificial intelligence methods for the optimal synthesis of reversible networks. *Radioelectronic and Computer Systems*. 2024. Vol. 2024, no. 4. P. 112–122. URL:https://doi.org/10.32620/reks.2024.4.10 (Scopus, Q3)

   Внесок авторів: **Кирилюк Т.** – розроблення алгоритму, проведення експериментів, підготовка основного рукопису; Палагута М. – розроблення алгоритму, проведення експериментів, підготовка основного рукопису; Дейбук В. – рецензування та редагування рукопису.

Наукові праці у виданнях, проіндексованих у наукометричній базі даних Scopus:

2. Deibuk V., Dovhaniuk O., Kyryliuk T. The Extended Fredkin Gates with Reconfiguration in NCT Basis. *Lecture Notes on Data Engineering and Communications Technologies*. 2023. Vol. 181 : Advances in Computer Science for Engineering and Education VI : ICCSEEA 2023 (Warsaw, Poland, 17–19 March 2023). Warsaw, P. 95–105. URL: https://doi.org/10.1007/978-3-031-36118-0_9. (Scopus)

   Внесок авторів Дейбук В. – наукове керівництво, концептуальна перевірка, рецензування та редагування рукопису; Довганюк О. – написання рукопису,

проведення експериментів та аналіз результатів; **Кирилюк Т.** – проведення експериментів, перевірка результатів та доопрацювання рукопису

3. Dovhaniuk O., Kyryliuk T., Deibuk V. Reversible Fault-Tolerant Encryption Using Extended Fredkin Gate with Reconfiguration. *Advances in Transdisciplinary Engineering*. 2025. Vol. 65 : Artificial Intelligence, Medical Engineering and Education : AIMEE 2024 (Huangshi, China, 26–27 Oct. 2024). Huangshi, P. 69–76. DOI: https://doi.org/10.3233/atde250113. (Scopus).

   Внесок авторів: Довганюк О. – проведення експериментів; **Кирилюк Т.** – основне написання рукопису, верифікація експериментів, підготовка презентації; Дейбук В. – рецензування рукопису та наукове керівництво.

Наукові праці, які засвідчують апробацію матеріалів дисертації;

4. Kyryliuk T., Deibuk V., Kyryliuk S. Synthesis of reversible circuits by genetic algorithm with multicomponent mutation. *Інформаційні технології та комп'ютерне моделювання* : матеріали міжнар. наук.-практ. конф. (м. Івано-Франківськ, 6–8 лип. 2023 р.). Івано-Франківськ, 2023. С. 150–151.

   Внесок авторів: **Кирилюк Т.** – проведення експериментів, розроблення алгоритму, підготовка основного рукопису; Дейбук В. – наукове керівництво, доопрацювання рукопису; Кирилюк С. – проведення експериментів, фіксація експериментальних результатів

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- NCT – бібліотека вентилів NOT, CNOT, Toffoli;
- CAD – автоматизація проєктування електронних пристроїв;
- GT – узагальнений вентиль Toffoli;
- FPGA – програмовані логічні матриці;
- BDD – бінарні діаграми прийняття рішень;
- GA – генетичні Алгоритми;
- ACO – мурашиний алгоритм;
- QMDD – квантові діаграми багатозначних рішень;
- QC – квантова вартість;
- SA – алгоритм симульованого відпалу;
- OpenQASM – відкрита мова квантового асемблера;
- ASCII – 7-бітовий стандарт кодувWання символів;
- PS/2 – стандарт апаратного послідовного інтерфейсу для підключення клавіатур;
- VESA – асоціація стандартизації відеоелектроніки, що регламентує часові параметри розгортки екрана;
- VGA – стандарт відеоадаптерів та моніторів.

ВСТУП

Актуальність та обґрунтування теми дисертаційного дослідження.

Характерною рисою сучасного етапу розвитку інформаційних технологій є широке застосування методів штучного інтелекту, а особливо еволюційних обчислень, для розв'язання надскладних задач в різних галузях науки і техніки. Разом з тим, нагальна потреба практичного створення квантових комп'ютерів, які є фізично і логічно зворотними системами, вимагає нових підходів до моделювання їх основних елементів. Теоретичною та практичною основою таких досліджень стають інтелектуальні методи синтезу та аналізу, що дозволяють оперувати поняттями зворотності та енергоефективності.

Не дивлячись на існуючі аналітичні та діаграмні методи (бінарні діаграми прийняття рішень, таблиці істинності), основним недоліком сучасних моделей залишається експоненційне зростання складності зі збільшенням простору рішень. Крім того, залишаються недостатньо дослідженими питання забезпечення надійності обчислень, зокрема автоматизованого синтезу схем, стійких до відмов.

Зважаючи на наведене вище, розробка та вдосконалення методів оптимального синтезу зворотних відмовостійких логічних пристроїв на підставі використання еволюційних методів є актуальною задачею.

Об'єктом дослідження є відмовостійкі зворотні логічні пристрої обробки інформації та процеси їх синтезу з використанням методів штучного інтелекту.

Предметом дослідження є методи, моделі та апаратно-програмні засоби оптимального синтезу зворотних відмовостійких логічних пристроїв на основі алгоритмів еволюційних обчислень.

Метою дисертаційного дослідження є розробка уніфікованої еволюційно-орієнтованої методології для ефективного синтезу та оптимізації зворотних відмовостійких логічних пристроїв шляхом створення математичних моделей реконфігуровних компонентів, вдосконалення генетичних методів структурного пошуку та побудови апаратно-програмного комплексу для

верифікації та моделювання схем з мінімізацією квантової вартості та апаратної складності.

Відповідно до поставленої мети в дисертаційній роботі розв’язуються такі **основні задачі**:

– провести аналітичний огляд існуючих аналітичних, діаграмних та евристичних методів синтезу, виявити їх недоліки, пов'язані з експоненційним зростанням складності;

– удосконалити еволюційні методи синтезу, націлені на використання узагальнених вентилів Фредкіна, для забезпечення оптимальності та відмовостійкості зворотних схем;

– розробити математичні моделі та алгоритми для структурного синтезу логічних пристроїв, що враховують фізичну та логічну зворотність;

– створити програмну складову комплексу для автоматизованого моделювання та генерації зворотних схем;

– розробити апаратну-програмну частину комплексу (на базі FPGA) та провести експериментальні дослідження синтезованих пристроїв обробки інформації;

– здійснити порівняльний аналіз розроблених методів з існуючими аналогами та довести їх ефективність.

Методи дослідження. В основу дисертаційної роботи покладені методи штучного інтелекту та комп'ютерного моделювання, а саме: теорія множин та булева алгебра (для формалізації логічних функцій та операцій зворотних вентилів); теорія графів (для моделювання топології схем та маршрутизації сигналів); еволюційні обчислення (для структурної оптимізації схем, мінімізації квантової вартості та забезпечення відмовостійкості); об'єктно-орієнтоване проєктування (для розробки архітектури програмного комплексу); методи математичного моделювання (для верифікації синтезованих пристроїв та аналізу їх характеристик).

Наукова новизна. В дисертаційній роботі

1) *вперше:*
   - запропоновано модифікацію генетичного методу структурного синтезу, яка, на відміну від існуючих бітових представлень, використовує об'єктно-орієнтовану модель хромосоми, де гени представлені як функціональні блоки з визначеними властивостями, що дозволило формалізувати правила комутації та суттєво скоротити простір пошуку рішень;
   - розроблено метод синтезу відмовостійких зворотних логічних пристроїв, що базується на інтеграції метрик надійності та контролю збереження парності безпосередньо в фітнес-функцію еволюційного алгоритму, що дозволяє отримувати схеми, стійкі до одиничних помилок, без необхідності їх пост-синтезної модифікації;
   - обґрунтовано комплексний оператор структурної мутації, який поєднує топологічні зміни (додавання, видалення та модифікації вентилів) із параметричною оптимізацією керуючих ліній, що забезпечує вихід із локальних екстремумів при синтезі функцій великої розмірності.
2) *набуло подальшого розвитку:*
   - застосування узагальнених вентилів Фредкіна для побудови універсальних логічних елементів, що дозволило реалізовувати 24 нових адаптивних функцій шифрування та комутації без зміни фізичної архітектури зв'язків;
   - метод багатокритеріальної оптимізації квантових схем, який, на відміну від аналогів, одночасно мінімізує параметри: логічну похибку та квантову вартість;
   - математичне забезпечення автоматизованої верифікації синтезованих структур шляхом їх трансляції у формат квантових інструкцій OpenQASM, що забезпечує інтероперабельність з сучасними квантовими симуляторами такими як IBM Qiskit.

Практичне значення одержаних результатів.

Розроблено програмний комплекс (CAD-систему), який реалізує повний цикл синтезу зворотних схем, їх оптимізацію та автоматичну трансляцію у формат OpenQASM, що забезпечує пряму сумісність та можливість верифікації на квантовому фреймворку IBM Qiskit.

Засобами розробленого програмного забезпечення синтезовано зворотний потоковий шифратор, який успішно імплементовано на FPGA сімейства Altera Cyclone IV. Експериментально підтверджено його високу швидкодію (максимальна частота Fmax≈310 МГц) та стабільність роботи, що доводить придатність запропонованого підходу для створення реальних апаратних засобів захисту інформації.

Зв’язок роботи з науковими програмами, планами, темами.

Дисертаційне дослідження виконано на кафедрі програмного забезпечення комп’ютерних систем Чернівецького національного університету імені Юрія Федьковича. Її зміст відповідає тематиці науково-дослідних робіт: «Дослідження, моделювання та розробка програмного забезпечення складних динамічних систем» (Державний реєстраційний номер 0121U109232).

Публікації. За темою дисертації опубліковано 5 робіт, в яких подано результати досліджень. З них 4 статі у рецензованих виданнях (2 з яких – в виданні, включеного до переліку наукових фахових видань України та проіндексовані у наукометричній базі даних Scopus (Q3), та 2 – у періодичних наукових виданнях, проіндексованих у наукометричних базах даних Scopus), у збірниках матеріалів міжнародних та всеукраїнських наукових конференцій – 1 робіта.

Особистий внесок здобувача. Дисертант брав активну участь в постановці задачі, виборі методів досліджень (еволюційні обчислення, структурний синтез), обговоренні та інтерпретації результатів, підготовці матеріалів до опублікування в усіх працях. Так, в роботі [1] особисто розробив математичну модель синтезу та запропонував об'єктно-орієнтоване

представлення хромосоми для зменшення простору пошуку рішень. У працях [2, 3] здійснив програмну реалізацію модифікованого генетичного алгоритму, розробив модуль трансляції схем у формат OpenQASM та провів верифікацію результатів у середовищі IBM Qiskit. Експериментальними дослідженнями та аналізом його часових характеристик, дисертант займався у [4].

Апробація результатів. Основні положення та результати дисертаційної роботи доповідались та обговорювались на наукових семінарах кафедри програмного забезпечення комп'ютерних систем Чернівецького національного університету імені Юрія Федьковича, а також на міжнародних наукових і науково-практичних конференціях:

- The 6th International Conference on Computer Science, Engineering and Education Applications (ICCSEEA2023) яка проходила у Варшаві, Польща 17–19 березня 2023 року;
- Міжнародна науково-практична конференція «Інформаційні технології та комп'ютерне моделювання» (ITCM-2023) яка проходила в місті Івано-Франківськ, 6–8 липня 2023 року;
- The 8th International Conference on Artificial Intelligence, Medical Engineering, Education (AIMEE2024) – Huangshi, China, October 26–27, 2024.

Структура та обсяг роботи. Дисертаційна робота є завершеним дисертаційним дослідженням, загальним обсягом 243 сторінки (з них: 189 сторінки основного тексту, 14 сторінок – література, 41 сторінок – додатки). Дисертація складається зі вступу, чотирьох розділів з підрозділами, висновків, списку використаних джерел (115 позиції) та додатків.

РОЗДІЛ 1. АНАЛІЗ ЕНЕРГОЕФЕКТИВНОСТІ ТА СКЛАДНОСТІ ПРОЄКТУВАННЯ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням вимог до обчислювальних потужностей, що спонукає до пошуку нових архітектурних та схемотехнічних рішень. В умовах, коли традиційні підходи до побудови цифрових систем стикаються з проблемою зростання енергоспоживання, критично важливим стає дослідження альтернативних парадигм обробки інформації, які дозволяють докорінно змінити принципи енергообміну в логічних елементах.

Одним із найбільш перспективних напрямків вирішення цих задач є перехід до зворотної логіки, яка розглядається як фундаментальна основа для побудови енергоефективних систем майбутнього, включаючи квантові комп'ютери [5]. Використання принципів зворотності дозволяє розглядати обчислювальний процес як інформаційно безвтратний, що відкриває шлях до створення пристроїв із теоретично нульовою дисипацією енергії.

Проте впровадження цієї технології вимагає вирішення низки складних інженерних та наукових завдань, пов'язаних насамперед із проєктуванням відповідних схем. Специфіка зворотного базису суттєво відрізняється від класичної булевої логіки, що робить традиційні методи автоматизованого проєктування CAD малоефективними. Простір пошуку рішень при синтезі зворотних схем характеризується високою комбінаторною складністю, що зумовлює необхідність залучення евристичних та еволюційних методів оптимізації [6].

У даному розділі дисертаційної роботи проводиться комплексний аналіз передумов переходу до зворотної логіки, розглядаються особливості простору пошуку при проєктуванні таких пристроїв, а також обґрунтовується доцільність застосування еволюційних стратегій як інструменту для автоматизованого синтезу оптимальних схем.

1.1 Фізичні та технологічні межі мініатюризації цифрових пристроїв

Сучасна мікроелектроніка досягла етапу розвитку, коли подальше збільшення продуктивності обчислювальних систем традиційними екстенсивними методами стає неможливим через фундаментальні фізичні обмеження. Протягом останніх десятиліть індустрія рухалася шляхом постійного зменшення розмірів транзисторів, що дозволяло інтегрувати все більше логічних елементів на одиницю площі кристала. Цей процес, відомий як масштабування Деннарда [7], забезпечував зростання тактових частот при збереженні енергоспоживання на прийнятному рівні.

Однак сьогодні ця тенденція зіткнулася з жорсткими бар'єрами. Еволюція напівпровідникових технологій, що тривалий час підпорядковувалася емпіричному закону Мура [8], наближається до свого природного насичення. Зменшення топологічних норм до нанометрових масштабів призвело до того, що класичні фізичні моделі перестають коректно описувати поведінку носіїв заряду, а на перший план виходять квантові ефекти, які раніше можна було ігнорувати.

Ключовою проблемою подальшої мініатюризації стає не стільки технологічна складність виробництва, скільки енергетичні аспекти функціонування надщільних схем. При наближенні розмірів транзисторів до атомних величин виникають паразитні ефекти, зокрема струми витоку через підзатворний діелектрик відоме як квантове тунелювання, що призводить до значного зростання статичного енергоспоживання навіть у режимі простою.

Зростання щільності компонентів неминуче веде до збільшення питомої потужності тепловиділення. Сучасні процесори за густиною теплового потоку вже наблизилися до критичних значень, що вимагає складних систем відведення тепла і обмежує можливість підвищення тактових частот — явище, відоме як «power wall» або енергетична стіна. Тепловий шум, що зростає внаслідок нагрівання кристала, починає впливати на надійність обробки сигналів, підвищуючи ймовірність бітових помилок [9].

Таким чином, індустрія опинилася перед необхідністю перегляду базових принципів обробки інформації. Оскільки подальше зменшення розмірів класичних КМОН-транзисторів обмежене термодинамічними факторами [10], актуальним стає дослідження фізики обчислювальних процесів з точки зору мінімізації енергетичних витрат на фундаментальному рівні, що безпосередньо пов'язано з поняттями ентропії та зворотності обчислень.

1.1.1 Еволюція напівпровідникових технологій та межа закону Мура

Історичний розвиток обчислювальних систем протягом останніх десятиліть визначався емпіричним законом Мура, який став фундаментальною метрикою для прогнозування продуктивності апаратного забезпечення. З точки зору системної інженерії, ця закономірність описує експоненційне зростання кількості транзисторів на кристалі, що дозволяло програмним системам нарощувати функціональність без зміни базових архітектурних принципів.

Динаміку зростання інтеграційної щільності можна описати базовою залежністю:

$$N(t) \approx N_0 \cdot 2^{\frac{t}{T}}, \qquad (1.1.1.1)$$

де:

- $N(t)$ — кількість елементів на кристалі в момент часу t;
- N_0 — початкова кількість елементів;
- T — період подвоєння (історично складає близько 18–24 місяців).

Тривалий час це зростання супроводжувалося ефектом масштабування Деннарда, який дозволяв підвищувати тактову частоту при збереженні незмінної густини потужності. Однак, на сучасному етапі індустрія зіткнулася з так званою «стіною енергоспоживання».

Для розробників програмно-апаратних комплексів [11-12] критичним показником стає динамічне енергоспоживання цифрової КМОН-схеми, яке визначається формулою [13]:

$$P_{dynamic} \approx \alpha \cdot C_{load} \cdot V^2 \cdot f, \quad (1.1.1.2)$$

де:

- α — коефіцієнт активності перемикання (визначається логікою роботи алгоритму та архітектурою схеми);
- C_{load} — ємність навантаження;
- V — напруга живлення;
- f — робоча частота.

Аналіз цієї залежності показує, що подальше підвищення продуктивності шляхом збільшення частоти f стало енергетично невигідним через квадратичну залежність від напруги. Більше того, при зменшенні технологічних норм суттєву роль почала відігравати статична потужність витоку P_{static}, яка споживається незалежно від виконання корисних обчислень.

Сумарна потужність системи описується як:

$$P_{total} = P_{dynamic} + P_{static}, \quad (1.1.1.3)$$

Зростання P_{static} призвело до феномену «темного кремнію» (Dark Silicon), коли значна частина транзисторів на кристалі повинна залишатися вимкненою, щоб уникнути перегріву. Це створює фундаментальну проблему для подальшого масштабування класичної архітектури фон Неймана.

Таким чином, криза закону Мура є не лише технологічною, але й алгоритмічною проблемою. Оскільки ми більше не можемо покладатися на зростання частоти процесорів, виникає потреба у зміні обчислювальної парадигми. Це актуалізує перехід до зворотної логіки, яка дозволяє мінімізувати коефіцієнт втрат енергії на рівні виконання елементарних логічних операцій, відкриваючи шлях до енергоефективних обчислень нового покоління [14, 15].

1.1.2. Термодинаміка обчислень: ентропія фон Неймана та принцип Ландауера

Розуміння енергетичних процесів у сучасних обчислювальних системах неможливе без аналізу інформаційної термодинаміки [16]. Ключовим поняттям тут виступає зв'язок між логічною незворотністю операцій та фізичною дисипацією тепла. Якщо класична теорія інформації оперує ентропією Шеннона для опису невизначеності стану системи, то для фізичних реалізацій обчислювальних процесів, особливо на квантовому рівні, фундаментальною є ентропія фон Неймана [17]:

$$S(\rho) = -Tr(\rho \ln \rho), \qquad (1.1.2.1)$$

де ρ — матриця густини, що описує квантовий стан системи. Для чистих станів ентропія фон Неймана дорівнює нулю, що відповідає повній визначеності системи. Однак у процесі незворотних класичних обчислень, наприклад, при використанні вентилів AND або OR, відбувається втрата інформації про попередній стан, що еквівалентно зростанню ентропії.

Цей фундаментальний зв'язок був сформульований Рольфом Ландауером у 1961 році [18]. Принцип Ландауера стверджує, що будь-яка логічно незворотна операція, така як стирання біта інформації або об'єднання двох шляхів обчислень в один, неминуче супроводжується виділенням теплової енергії в навколишнє середовище. Це обмеження не залежить від технології виготовлення транзисторів, а випливає з законів термодинаміки.

Згідно з принципом Ландауера, мінімальна кількість енергії E, необхідна для стирання одного біта інформації, тобто зменшення ентропії інформаційної системи на один біт, визначається нерівністю [19]:

$$E \geq k_B T \ln 2, \qquad (1.1.2.2)$$

де:

- k_B — стала Больцмана (1.380649×10^{-23} Дж/К);
- T — абсолютна температура середовища (у Кельвінах);

При кімнатній температурі (T≈300 К) ця фундаментальна термодинамічна межа, відома як межа Ландауера, становить [20]:

$$E_{min} \approx 2.804 \times 10^{-23} \text{Дж} \approx 0.0175 \text{ еВ}, \qquad (1.1.2.3)$$

Хоча це значення видається мізерним, у сучасних високопродуктивних процесорах, які виконують мільярди операцій за секунду на мільярдах транзисторів, сумарні втрати енергії стають критичними. Більше того, реальні КМОН-транзистори працюють на енергетичних рівнях, що на кілька порядків перевищують цю межу, однак тенденція мініатюризації невпинно наближає нас до неї.

Справедливість принципу Ландауера була підтверджена експериментально низкою незалежних наукових груп. Зокрема, у дослідженнях Y. Jun, M. Gavrilov та J. Bechhoefer (2014) [21] було продемонстровано, що маніпуляції з мікроскопічними частинками, які імітують операцію стирання біта пам'яті, дійсно потребують витрат енергії, що асимптотично наближаються до теоретичного мінімуму. Також вагомий внесок зробили A. Bérut, A. Petrosyan та S. Ciliberto, які у своїй роботі «Information and thermodynamics: experimental verification of Landauer’s erasure principle» (2015) [19] надали пряме експериментальне підтвердження того, що середня теплова енергія, яка виділяється при стиранні одного біта інформації, не може бути меншою за $k_B T \ln 2$, остаточно верифікувавши термодинамічну природу інформаційних втрат.

Отже, принцип Ландауера встановлює жорстку нижню межу енергоспоживання для будь-якої обчислювальної системи, побудованої на

незворотній логіці. Єдиним теоретично обґрунтованим шляхом подолання цього бар'єру є перехід до зворотних обчислень, де логічні операції є бієктивними і не призводять до втрати інформації, а отже, теоретично можуть виконуватися з нульовою дисипацією енергії.

1.1.3. Аналіз енергетичних витрат при незворотних обчисленнях

У класичних цифрових системах, побудованих на базі архітектури фон Неймана, переважна більшість логічних операцій є незворотними. Це означає, що кількість вхідних сигналів не дорівнює кількості вихідних, або ж стан входів неможливо однозначно відновити за станом виходів. Типовим прикладом є двовходовий вентиль І (AND), який відображає чотири можливі вхідні комбінації (00, 01, 10, 11) в одну з двох вихідних (0 або 1). Така втрата інформаційної ентропії на фізичному рівні трансформується у теплову енергію.

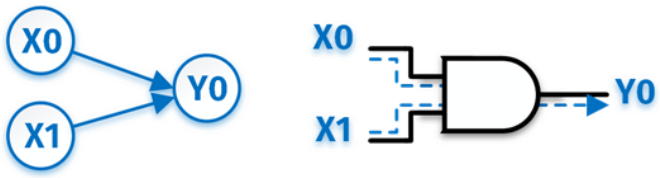


Рис. 1.1.3.1. Логічний вентиль І

У сучасній напівпровідниковій схемотехніці (технологія КМОН) механізм розсіювання енергії має складнішу структуру, ніж теоретична межа Ландауера, і складається з двох основних компонентів: енергії перемикання (динамічна складова) та енергії витоку (статична складова).

При виконанні незворотної логічної операції відбувається перезарядка ємності навантаження C_{load}. Енергія, що забирається від джерела живлення для зарядки ємності до напруги V_{dd}, дорівнює $C_{load}\ V_{dd}^{\ 2}$. Половина цієї енергії запасається в електричному полі конденсатора, а інша половина розсіюється у вигляді тепла на активному опорі каналу p-транзистора. Однак критичний

момент настає при стиранні інформації (перехід 1 → 0): запасена енергія ($0.5\mathrm{C}_{load}\ \mathrm{V}_{dd}{}^2$) повністю перетворюється на тепло при розрядці через n-транзистор на «землю».

Таким чином, повна енергія, що розсіюється за один цикл перемикання вентиля, визначається рівнянням:

$$E_{cycle} \approx \mathrm{C}_{load}\ \mathrm{V}_{dd}{}^2. \qquad (1.1.3.1)$$

Важливо зазначити, що в сучасних технологіях ця величина на кілька порядків перевищує термодинамічну межу Ландауера описану формулою (1.1.2.2). Проте співвідношення між реальною енергією комутації та теоретичним мінімумом ($E_{cycle}/E_{Landauer}$) стрімко скорочується зі зменшенням технологічних норм. Це свідчить про те, що можливості оптимізації за рахунок зниження напруги живлення V_{dd} та паразитних ємностей C_{load} практично вичерпані.

Проблема посилюється при зростанні тактової частоти *f*. Потужність тепловиділення *P* для масиву з *N* вентилів зростає лінійно:

$$P = \alpha \cdot \mathrm{N} \cdot \mathrm{J} \cdot \mathrm{C}_{load} \cdot \ \mathrm{V}_{dd}{}^2, \qquad (1.1.3.1)$$

де α — коефіцієнт активності перемикання. Оскільки незворотна логіка вимагає постійного скидання заряду в землю для зміни логічних станів, щільність тепловиділення у сучасних процесорах досягає значень (понад 100 Вт/см²), які вимагають громіздких систем охолодження.

Єдиним фундаментальним способом усунення цього домінуючого фактора енерговитрат є відмова від принципу «заряд-розряд» зі знищенням інформації. Перехід до адіабатичних та зворотних схем дозволяє зберігати енергію, накопичену в ємностях, замість її розсіювання, що робить аналіз

методів синтезу таких схем критично важливим завданням сучасної комп'ютерної інженерії [22].

1.2. Зворотні обчислення як парадигма подолання енергетичних обмежень

Вичерпання можливостей екстенсивного розвитку напівпровідникових технологій зумовило необхідність пошуку альтернативних підходів до організації обчислювальних процесів. Якщо традиційна булева логіка базується на незворотних операціях, що неминуче призводить до втрати інформації та виділення тепла, то зворотні обчислення пропонують принципово іншу філософію обробки даних. Ця парадигма розглядає обчислювальний процес не як послідовність незворотних змін станів, а як фізичну еволюцію системи, в якій зберігається повна інформація про її попередні стани.

Перехід до зворотної логіки дозволяє трансформувати архітектуру обчислювальних пристроїв з дисипативних систем у консервативні. У такій моделі комп'ютер розглядається як механізм, що змінює інформаційний стан без фізичного знищення носіїв інформації. Це відкриває шлях до створення адіабатичних схем, у яких енергія, витрачена на перемикання транзисторів, не розсіюється у вигляді тепла, а зберігається для використання у наступних тактах роботи пристрою.

Окрім вирішення проблеми енергоефективності класичних КМОН-структур, зворотні обчислення є фундаментом для квантової інформатики. Квантова механіка, яка описує еволюцію замкнених квантових систем, є унітарною і, отже, зворотною за своєю природою. Тому розробка методів синтезу та оптимізації зворотних схем має подвійне призначення: вона є критично необхідною як для проєктування наднизькоенергетичних класичних процесорів, так і для створення алгоритмічної бази квантових комп'ютерів.

Впровадження цієї парадигми вимагає докорінного перегляду методів проєктування цифрових систем. Класичні методи мінімізації логічних функцій

стають непридатними, оскільки вони спрямовані на зменшення кількості вентилів без урахування вимоги збереження інформації. Це створює новий клас інженерних задач у галузі автоматизованого проєктування CAD, де ключовими критеріями оптимізації стають не лише швидкодія, але й мінімізація надлишкових виходів та квантової вартості схеми.

1.2.1. Концепція логічної зворотності та нульова дисипація енергії

Фундаментальною відмінністю зворотної логіки від класичної є вимога взаємно-однозначної відповідності між вхідними та вихідними векторами станів системи [23]. У булевій алгебрі функція *f* вважається логічно зворотною, якщо вона є бієкцією, тобто відображенням «один-в-один». Це означає, що для будь-якого вихідного вектора *Y* існує єдиний вхідний вектор *X*, такий що $f(X) = Y$, і навпаки: $f^{-1}(Y) = X$.

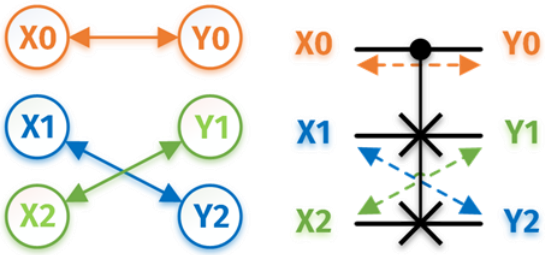


Рис. 1.2.1.1. Бієктивний вентиль або функція

Формально, зворотний логічний елемент з n входами та n виходами (так звана схема $n \times n$) описується функцією перестановки P над множиною з 2^n можливих станів:

$$f: \{0,1\}^n \to \{0,1\}^n, \qquad (1.2.1.1)$$

Така властивість накладає жорсткі структурні обмеження на проєктування схем:

1. Збереження розрядності: кількість вхідних ліній повинна дорівнювати кількості вихідних.
2. Заборона розгалуження [24] (Fan-out): вихід одного вентиля не може безпосередньо подаватися на входи кількох інших вентилів. У квантовому контексті це пов'язано з теоремою про заборону клонування.
3. Відсутність зворотних зв'язків: комбінаційні зворотні схеми є ациклічними.

Важливим теоретичним обґрунтуванням можливості побудови обчислювальних систем на таких принципах стала робота Чарльза Беннета (1973) [25]. Він довів, що будь-який алгоритм, який може бути виконаний на класичній машині Тюрінга, може бути реалізований на логічно зворотній машині Тюрінга. Це спростувало поширену думку про те, що обчислення неможливі без енергетичних витрат.

Зв'язок між логічною зворотністю та енергоспоживанням базується на термодинаміці. Якщо логічний процес не знищує інформацію, тобто ентропія системи не змінюється ($\Delta S=0$), то теоретично він може бути реалізований як термодинамічно зворотний процес. У такому процесі система проходить через послідовність рівноважних станів, і робота, витрачена на зміну стану бітів, не перетворюється на тепло, а зберігається в системі, наприклад, у реактивних компонентах схеми, і може бути використана для виконання зворотної операції.

Отже, умова нульової дисипації енергії $E_{diss} \to 0$ виконується тоді, коли обчислювальний процес є квазістатчиним та інформаційно безвтратним.

Проте більшість корисних функцій, наприклад, додавання або множення у своїй природній формі є незворотними. Операція $c = a + b$ не дозволяє відновити a і b, знаючи лише c). Для реалізації таких функцій у зворотному базисі застосовується техніка занурення: незворотна функція $f(x)$ доповнюється додатковими входами (ancilla inputs) та надлишковими виходами (garbage outputs), щоб утворити розширену зворотну функцію $F(x, c)$.

Мінімізація кількості цих додаткових ліній та надлишкової інформації є одним із ключових завдань синтезу, оскільки саме вони визначають складність та вартість реалізації пристроїв, у теоретичній межі, з нульовою дисипацією енергії.

1.2.2. Перспективи застосування зворотної логіки в IoT-пристроях та системах з обмеженим живленням

Стрімкий розвиток концепції Інтернету речей (IoT) та граничних обчислень призвів до появи класу пристроїв, для яких енергоефективність є більш критичним параметром, ніж гранична швидкодія [26]. Автономні сенсори, медичні імплантати, "розумний пил" та носимі гаджети функціонують в умовах жорстких енергетичних лімітів, часто покладаючись на батарейне живлення або системи збору енергії з навколишнього середовища.

У класичних КМОН-реалізаціях навіть у режимі очікування відбувається значний витік енергії, а активні обчислення швидко виснажують джерело живлення. Застосування зворотної логіки в цьому контексті відкриває перспективи створення адіабатичних схем [27]. Це клас мікроелектронних пристроїв, схемотехніка яких базується на принципах зворотності, що дозволяє не розсіювати енергію, накопичену в паразитних ємностях, а повертати її назад у джерело живлення для повторного використання.

Основними напрямками застосування зворотних схем у системах з обмеженим живленням є [27]:

1. Наднизькоенергетичні мікроконтролери: Використання зворотних арифметико-логічних пристроїв (ALU)[28, 29], та чипів пам’яті [30] дозволяє знизити динамічне енергоспоживання на порядок порівняно з традиційними рішеннями. Це є критичним для IoT-вузлів, які повинні працювати роками без заміни елемента живлення [31].

2. Легковагова криптографія: Забезпечення конфіденційності даних у IoT-мережах є складним завданням [32], оскільки стандартні алгоритми шифрування (наприклад, AES) є надто енергоємними. Зворотні схеми ідеально підходять для реалізації потокових шифрів та блоків перестановки, оскільки операції шифрування та дешифрування у зворотному базисі часто є симетричними або легко реалізованими на одній апаратній структурі [33, 34]. Це дозволяє створювати захищені канали зв'язку з мінімальними накладними витратами енергії.
3. Цифрова обробка сигналів: Первинна обробка даних із сенсорів [35] (фільтрація, швидке перетворення Фур'є) вимагає виконання великої кількості операцій множення та додавання. Реалізація цих блоків на базі зворотних вентилів, наприклад таких як вентиль Фредкіна або Тоффолі дозволяє виконувати попередню обробку інформації "на краю" з мінімальним тепловиділенням, зменшуючи обсяг даних, що передаються по бездротовому каналу.
4. Відмовостійкість у важкодоступних середовищах: Багато IoT-пристроїв розміщуються у місцях, де обслуговування неможливе. Наприклад, вбудовані в будівельні конструкції або розгорнуті в агресивних середовищах. Зворотна логіка має природну здатність до виявлення помилок. Оскільки зворотні вентилі часто мають властивість збереження парності, це дозволяє реалізовувати апаратний самоконтроль без введення значної апаратної надлишковості, що підвищує надійність системи в цілому.

Таким чином, зворотна логіка перестає бути виключно теоретичною абстракцією для квантових комп'ютерів і стає реальною технологічною основою для наступного покоління енергоефективної електроніки Інтернету речей.

1.2.3. Роль зворотних схем у квантовій криптографії та захисті інформації

Розвиток квантових обчислень створює двояку ситуацію в сфері інформаційної безпеки. З одного боку, алгоритми Шора та Гровера [36] ставлять під загрозу стійкість класичних криптосистем з відкритим ключем (RSA, ECC). З іншого боку, квантова механіка пропонує інструменти для створення абсолютно захищених каналів зв'язку та нових методів шифрування [37, 38]. У цьому контексті зворотні схеми виступають не просто як допоміжний елемент, а як фундаментальна апаратна база для реалізації криптографічних примітивів майбутнього[39].

Оскільки квантова еволюція описується унітарними перетвореннями, усі квантові логічні операції є за своєю природою зворотними. Це означає, що будь-який алгоритм квантового шифрування або протокол розподілу ключів (наприклад, BB84 або E91) на найнижчому рівні реалізується через каскади зворотних вентилів таких як CNOT, Тоффолі та Фредкін.

Роль зворотних схем у захисті інформації можна систематизувати за трьома ключовими напрямками, які є актуальними як для квантових, так і для класичних реалізацій [40]:

1. Структурна бієктивність як основа шифрування. Основна властивість зворотної логіки — взаємно-однозначне відображення входів на виходи — робить її ідеальним кандидатом для побудови симетричних блокових шифрів [41]. У такій схемі ключ шифрування визначає налаштування керуючих ліній зворотних вентилів, що змінює шлях проходження сигналів. Оскільки схема є зворотною, процес дешифрування реалізується тією ж самою апаратною структурою, але зі зворотним порядком виконання операцій, або тим самим порядком у випадку самозворотних або інволютивних архітектурах. Це дозволяє суттєво зменшити апаратні витрати на реалізацію криптомодулів в IoT-пристроях та FPGA [33, 34].

2. Апаратна відмовостійкість та збереження парності. У класичній криптографії атаки методом внесення помилок є серйозною загрозою. Зловмисник навмисно змінює стан біта, щоб проаналізувати реакцію системи та отримати ключ. Зворотні схеми дозволяють реалізувати захист від таких атак на рівні логіки. Використання вентилів, що зберігають парність, наприклад, модифіковані вентилі Фредкіна, де $P_{in} = P_{out}$, дає можливість миттєво детектувати помилки в процесі обчислень без складних зовнішніх схем контролю. Якщо парність вхідного і вихідного векторів не співпадає, система блокує видачу результату, запобігаючи витоку інформації.
3. Динамічна реконфігурація логіки. Перспективним напрямком є використання реконфігурованих зворотних елементів [42, 43], які здатні змінювати свою логічну функцію в залежності від стану керуючих сигналів. Це дозволяє створювати поліморфні шифратори, внутрішня структура яких змінюється динамічно під час роботи. Для зовнішнього спостерігача це створює ефект невизначеності, що значно ускладнює зворотний інжиніринг та криптоаналіз схеми.

Таким чином, проєктування зворотних схем виходить за межі суто енергетичної оптимізації і стає критично важливою задачею для створення захищених обчислювальних систем [44]. Розробка ефективних методів автоматизованого синтезу таких схем є необхідною умовою для впровадження постквантової криптографії та надійних апаратних шифраторів.

1.3. Проблема автоматизованого синтезу зворотних пристроїв

Незважаючи на теоретичну привабливість зворотної логіки з точки зору термодинаміки та квантових обчислень, її практичне впровадження стримується значним технологічним розривом у засобах автоматизованого проєктування CAD. Якщо для класичних цифрових схем існують потужні інструментальні

засоби синтезу, верифікації та топологічного трасування, які вдосконалювалися десятиліттями, наприклад, методи мінімізації на основі карт Карно або алгоритму Куайна-Маккласкі [45], то для зворотної схемотехніки такі стандартні підходи є непридатними [5].

Фундаментальна проблема полягає у тому, що класичні методи мінімізації орієнтовані на зменшення кількості вентилів у базисі {*AND, OR, NOT*} без урахування збереження інформації. Спроба прямої адаптації цих методів до зворотних схем призводить до створення неймовірно громіздких структур з величезною кількістю надлишкових виходів, що нівелює будь-які енергетичні переваги.

Задача синтезу зворотного пристрою формулюється як пошук такої послідовності, тобто каскаду, елементарних зворотних вентилів з фіксованої бібліотеки, наприклад NCT, яка реалізує задану функцію перестановки вхідних векторів. При цьому синтезована схема повинна задовольняти жорстким структурним обмеженням:

1. Лінійність структури: відсутність зворотних зв'язків та розгалужень.
2. Фіксована розрядність: кількість ліній у схемі залишається незмінною від входу до виходу.
3. Мінімізація ресурсів: схема повинна бути оптимальною за декількома конфліктуючими критеріями (кількість вентилів, квантова вартість, кількість додаткових входів).

На даний момент не існує універсального детермінованого алгоритму, який міг би за поліноміальний час синтезувати оптимальну зворотну схему довільної складності. Існуючі евристичні методи добре працюють лише для невеликих функцій, але зі збільшенням розрядності схеми вони наштовхуються на проблему так званого "комбінаторного вибуху"[46]. Це перетворює задачу синтезу зворотних схем на одну з найскладніших проблем у сучасній комп'ютерній інженерії, що вимагає застосування інтелектуальних методів пошуку та оптимізації.

1.3.1. Особливості простору пошуку рішень у зворотному базисі

Процес синтезу зворотної схеми можна формалізувати як задачу пошуку декомпозиції заданої фітнес-функції перестановки F у каскад елементарних вентилів з фіксованої бібліотеки L. На відміну від класичного синтезу в базисі Буля [47], де простір рішень структурується через мінімізацію диз'юнктивних нормальних форм, простір пошуку у зворотному базисі має природу дискретної комбінаторної оптимізації на групах перестановок.

Ключовою особливістю цього простору є його факторіальна розмірність. Для системи з n вхідними змінними таблиця істинності містить 2^n рядків. Оскільки зворотна функція є бієкцією, кожному вхідному набору відповідає унікальний вихідний набір. Отже, загальна кількість можливих зворотних функцій N_{rev} для n змінних визначається як кількість перестановок множини з 2^n елементів:

$$N_{\text{rev}} = (2^n)! \tag{1.3.1.1}$$

Ця величина демонструє вибухове зростання складності так званий "комбінаторний вибух" [48]. Наприклад:

1. Для n=3 кількість функцій становить 8!=40,320.
2. Для n=4 це значення зростає до $16! \approx 2.09 \times 10^{13}$.
3. Для n=5 простір налічує близько 2.63×10^{35} варіантів, що вже робить повний перебір brute-force неможливим навіть на суперкомп'ютерах.

Другою критичною особливістю є некомутативність операцій. У класичній логіці порядок паралельних гілок часто не впливає на результат, але у каскадних зворотних структурах зміна порядку слідування вентилів (наприклад, $Gate_A$ і $Gate_B$) може кардинально змінити результуючу функцію ($Gate_A \cdot Gate_B \neq Gate_B \cdot Gate_A$), якщо вони діють на спільні лінії. Це означає, що

простір пошуку є чутливим до послідовності елементів, що ускладнює застосування методів локального пошуку.

Третім фактором, що ускладнює топологію простору пошуку, є використання додаткових ліній – ancilla inputs [49]. Часто оптимальна реалізація функції $n \times n$ неможлива без введення k додаткових ліній, що розширює простір пошуку від $(2^n)!$ до $(2^{n+k})!$. Визначення мінімального необхідного k заздалегідь є нетривіальною задачею, тому алгоритм синтезу змушений працювати в умовах невизначеної розмірності простору.

Крім того, простір рішень є "лабіринтом" з великою кількістю локальних екстремумів. Оскільки фітнес-функція є дискретною і недиференційовною, стандартні градієнтні методи тут незастосовні. Невелика зміна структури схеми як мутація одного вентиля може призвести до повної втрати функціональної еквівалентності [50], що робить ландшафт фітнес-функції вкрай нерівномірним і складним для навігації евристичними алгоритмами.

1.3.2. Комбінаторна складність синтезу оптимальних схем

Синтез зворотної схеми з алгоритмічної точки зору можна розглядати як задачу пошуку шляху на графі станів, де вузлами є перестановки, а ребрами — застосування елементарних вентилів з бібліотеки [51]. Основна складність цієї задачі полягає не стільки у знахожденні будь-якого рішення [52], схоже до проблеми комівояжера [53], що гарантується універсальністю базису, скільки у знахожденні оптимального рішення за критеріями квантової вартості – Quantum Cost та глибини схеми.

Якщо розглядати задачу як перебір варіантів, то кількість можливих схем зростає експоненційно зі збільшенням глибини. Нехай L — це розмір бібліотеки доступних вентилів, наприклад, для 3-х змінних у базисі NCT це може бути декілька десятків варіацій NOT, CNOT та Toffoli з різними комбінаціями керуючих ліній. Тоді кількість можливих схем глибиною d оцінюється як:

$$N_{circuits} \approx |L|^d \quad (1.3.2.1)$$

Оскільки для реалізації складних функцій перестановки глибина d може досягати десятків або сотень вентилів, простір пошуку стає трансчисленним. Наприклад, навіть для невеликої схеми з 10 вентилів при бібліотеці з 20 елементів кількість варіантів становить 20^{10}.

Задача синтезу мінімальної зворотної схеми формально належить до класу NP-складних задач. Для $n > 3$ змінних не існує ефективних детермінованих алгоритмів, які гарантовано знаходять глобальний оптимум за прийнятний час. Точні методи, такі як алгоритми на основі таблиць пошуку (lookup tables) або SAT-солвери, стають обчислювально непридатними через вичерпання оперативної пам'яті або експоненційний час роботи.

Додатковим фактором, що ускладнює синтез, є багатокритеріальність оптимізації. Часто покращення одного параметру (наприклад, зменшення кількості вентилів) призводить до погіршення іншого (збільшення кількості надлишкових ліній або квантової вартості). Це означає, що замість одного глобального оптимуму існує множина Парето-оптимальних рішень.

Ще однією проблемою є відсутність явного "градієнта" в просторі пошуку. Невелика зміна в структурі схеми, заміна одного вентиля або зміна порядку, призводить до кардинальної зміни реалізованої функції перестановки. Це унеможливлює використання простих жадібних алгоритмів, оскільки локально оптимальний крок часто не веде до глобального оптимуму.

Таким чином, комбінаторна складність задачі синтезу зворотних схем вимагає застосування стохастичних методів пошуку, які здатні ефективно досліджувати величезні простори рішень, не перевіряючи кожен варіант. Це обумовлює доцільність використання еволюційних стратегій, які імітують природний відбір для знаходження субоптимальних, але достатньо ефективних рішень за поліноміальний час.

1.4. Еволюційні стратегії як інструмент оптимального синтезу

Зіткнувшись із експоненційною складністю простору пошуку рішень у зворотному базисі, дослідники змушені відмовитися від класичних детермінованих алгоритмів на користь методів штучного інтелекту. У цьому контексті еволюційні стратегії виступають як найбільш ефективний клас метаевристичних алгоритмів глобальної оптимізації. Вони пропонують підхід, що базується не на прямому переборі чи аналітичному виведенні структури схеми, а на ітеративному покращенні популяції потенційних рішень шляхом імітації природних процесів селекції та адаптації [54].

Ключовою перевагою еволюційних стратегій є їхня здатність працювати в умовах "чорної скриньки". Алгоритму не потрібно знати внутрішню математичну структуру операцій перестановки; йому достатньо мати метрику – фітнес-функцію, яка оцінює якість конкретного рішення. Це дозволяє абстрагуватися від складності окремих квантових вентилів і зосередитися на мінімізації параметрів макроскопічного рівня — квантової вартості, глибини схеми та кількості додаткових ліній.

Застосування еволюційного підходу до синтезу логіки дозволяє вирішити дві фундаментальні проблеми, що блокують роботу точних методів:

1. Подолання локальних екстремумів: Завдяки стохастичній природі пошуку еволюційні алгоритми здатні "вистрибувати" з локальних пасток ландшафту оптимізації, досліджуючи віддалені області простору станів, які могли б бути пропущені градієнтними або жадібними методами.
2. Багатокритеріальність: Синтез схеми часто вимагає знаходження компромісу між суперечливими вимогами, наприклад, мінімальна кількість вентилів проти мінімальна кількість надлишкових виходів). Еволюційні стратегії природним чином адаптуються до пошуку

множини Парето-оптимальних рішень, надаючи розробнику вибір між різними варіантами реалізації.

У сучасному проєктуванні цифрових систем еволюційні методи, зокрема генетичні алгоритми, метод рою частинок, мурашині алгоритми, розглядаються не як альтернатива, а як єдиний можливий шлях для синтезу складних зворотних схем, розмірність яких перевищує можливості повного перебору. Замість того, щоб шукати ідеальне рішення нескінченно довго, еволюційна стратегія дозволяє знайти субоптимальне, але інженерно прийнятне рішення за поліноміальний час.

Таким чином, перехід до біоінспірованих алгоритмів знаменує зміну парадигми в CAD-системах: від алгоритмічного конструювання схеми за суворими правилами до її вирощування та еволюції під тиском критеріїв оптимізації.

1.4.1. Огляд можливостей еволюційних алгоритмів для задач CAD

У сфері автоматизації проєктування електронних пристроїв (EDA/CAD) еволюційні обчислення зайняли нішу розв’язання задач [55], що не піддаються формалізації класичними математичними методами. Якщо традиційні алгоритми синтезу наприклад, BDD – Binary Decision Diagrams [56] ефективні для мінімізації площі кристала у стандартному булевому базисі, то еволюційні алгоритми демонструють перевагу в умовах невизначеності, жорстких топологічних обмежень та нових фізичних базисів реалізації, таких як зворотна логіка.

Основним інструментом у цьому класі задач є Генетичні Алгоритми (GA) [58]. Їх застосування в CAD-системах базується на аналогії між структурою електронної схеми та генотипом біологічного організму.

1. Кодування: схема представляється у вигляді хромосоми — вектора, де кожен ген кодує тип вентиля, його цільові та керуючі лінії.

2. Фітнес-функції: якість схеми оцінюється кількісно. Для зворотних схем це зазвичай зважена сума кількості невідповідних бітів на виході – відстань Геммінга, та апаратних витрат – квантова вартість.
3. Генетичні оператори: оператор кросоверу – crossover дозволяє комбінувати вдалі фрагменти двох батьківських схем наприклад, обмін каскадами вентилів, а оператор мутації – mutation вносить випадкові зміни: додавання, видалення або заміна вентиля, що запобігає передчасній збіжності алгоритму до локального оптимуму.

Іншим потужним підходом є алгоритми рою частинок – Swarm Algorithm, Мурашиний алгоритм – Ant Colony Optimization (ACO) [59]. На відміну від GA, який оперує готовими варіантами схем, ACO використовує конструктивний підхід. "Мурахи" (агенти) будують схему поетапно, додаючи вентиль за вентилем, рухаючись по графу можливих станів. Вибір наступного вентиля визначається ймовірнісно, базуючись на "феромонному сліду" – досвіді попередніх успішних ітерацій та евристичній інформації. Цей метод є особливо ефективним для знаходження найкоротших шляхів синтезу, тобто схем з мінімальною глибиною.

Ключові можливості еволюційних алгоритмів для задач синтезу зворотних схем включають:

1. Робота з "чорною скринькою": Алгоритм не потребує аналітичного знання про те, як саме вхідні змінні перетворюються на вихідні. Це дозволяє синтезувати схеми для частково визначених функцій або функцій, заданих лише таблицею істинності [57].
2. Багатокритеріальна оптимізація – multi-objective optimization: У CAD-задачах часто виникає конфлікт інтересів, наприклад, зменшення затримки проти зменшення енергоспоживання. Еволюційні методи дозволяють явно знаходити фронт Парето-множину рішень, де покращення одного параметра неможливе без погіршення іншого, надаючи інженеру вибір.

3. Адаптивність до бібліотеки елементів: зміна технологічного базису (наприклад, перехід від базису NCT до базису, що включає вентилі Переса або Фредкіна) в еволюційному алгоритмі вимагає лише зміни алфавіту кодування [58], тоді як для детермінованих методів це часто означає повну переробку алгоритму синтезу [57].

Проте, пряме застосування стандартних еволюційних стратегій у CAD має свої обмеження, пов'язані з необхідністю валідації кожного рішення (симуляція схеми займає час) та проблемою масштабованості. Це зумовлює необхідність розробки спеціалізованих модифікацій алгоритмів, адаптованих до специфіки зворотної логіки.

1.4.2. Аналіз обмежень класичних еволюційних підходів

Попри доведену ефективність еволюційних стратегій у задачах глобальної оптимізації, їх безпосереднє застосування до синтезу зворотних схем виявляє низку суттєвих обмежень. Специфіка простору перестановок та жорсткі структурні вимоги до схем призводять до того, що класичні реалізації GA [60] та методів рою частинок часто демонструють недостатню продуктивність або низьку якість отриманих рішень.

Основним недоліком класичних еволюційних підходів є схильність до передчасної збіжності. У процесі еволюції популяція рішень швидко втрачає генетичне різноманіття, тобто виродження, концентруючись навколо локального екстремуму, який може бути далеким від глобального оптимуму. У ландшафті зворотних схем, який характеризується великою кількістю локальних пасток, стандартні оператори мутації, наприклад, випадкова зміна одного вентиля, часто виявляються неспроможними вивести систему з цього стану стагнації. Це призводить до синтезу схем, які функціонально коректні, але мають надмірну кількість вентилів, що призводить до збільшення загальної квантової вартості, що і робить їх непридатними для фізичної реалізації.

Другим критичним обмеженням є висока обчислювальна вартість оцінки фітнес-функції. Для визначення пристосованості кожної хромосоми-схеми необхідно виконати повну симуляцію її роботи на всіх 2^n вхідних векторах. Зі збільшенням розрядності схеми час симуляції зростає експоненційно. Класичні алгоритми, які потребують тисячі поколінь та великих популяцій для знаходження рішення, стають часозатратними для $n > 4$. Відсутність механізмів скороченої оцінки або локальної оптимізації в класичних підходах суттєво сповільнює процес синтезу.

Також варто відзначити неефективність стандартних генетичних операторів для домену зворотної логіки.

1. Деструктивний кросовер: Класичний одноточковий або двоточковий кросовер, запозичений з бінарної оптимізації, при застосуванні до послідовності вентилів часто руйнує корисні функціональні блоки, сформовані в батьківських хромосомах. Це призводить до того, що нащадки часто мають гіршу пристосованість, ніж батьки, що гальмує еволюційний прогрес.
2. Слабкість сліпих мутацій: Випадкова заміна параметрів вентиля без урахування контексту схеми має низьку ймовірність покращення результату. Класичні алгоритми не використовують знання про структуру помилки (наприклад, на яких саме вхідних векторах схема працює неправильно), діючи методом "спроб і помилок". Цей недолік можна виправити використовуючи напрямлені мутації.

Окремою проблемою є масштабованість. Ефективність класичних еволюційних методів стрімко падає зі зростанням розмірності задачі. Якщо для 3-бітних функцій вони знаходять оптимальні рішення за секунди, то для 4-х та 5-бітних функцій час пошуку може зростати до годин, а гарантія знаходження рішення зникає. Це зумовлено тим, що простір пошуку росте факторіально, а розвідувальні можливості алгоритмів залишаються фіксованими.

Таким чином, наявні обмеження класичних підходів диктують необхідність розробки вдосконалених модифікацій еволюційних алгоритмів. Зокрема, актуальним є впровадження адаптивних операторів мутації [61], які здатні вносити значні структурні зміни для виходу з локальних екстремумів, та гібридних стратегій пошуку [62], що і є предметом досліджень у наступних розділах даної роботи.

Висновки до розділу 1

У першому розділі дисертаційної роботи проведено аналіз сучасного стану проблеми синтезу цифрових пристроїв та обґрунтовано необхідність переходу до нової обчислювальної парадигми. Основні результати розділу зводяться до наступного:

1. Вичерпання можливостей класичної мікроелектроніки. Показано, що подальше збільшення щільності інтеграції транзисторів, відповідно до закону Мура, наштовхнулося на фундаментальні фізичні обмеження. Термодинамічна межа Ландауера доводить, що незворотність логічних операцій призводить до неминучого розсіювання енергії у вигляді тепла ($k_B T \ln 2$ Дж на біт). Це робить проблему енергоефективності ключовим викликом для сучасних обчислювальних систем.
2. Обґрунтування переходу до зворотної логіки. Встановлено, що єдиним способом подолання термодинамічного бар'єру є застосування зворотної логіки, яка дозволяє виконувати обчислення без втрати інформації. Визначено перспективи застосування зворотних схем не лише як базису для квантових комп'ютерів, але і як основи для наднизькоенергетичних IoT-пристроїв та криптостійких систем захисту інформації.
3. Комбінаторна складність задачі синтезу. Виявлено, що автоматизований синтез оптимальних зворотних схем належить до

класу NP-складних задач. Простір пошуку рішень, що являє собою групу перестановок розмірності $(2^n)!$, характеризується факторіальним зростанням "комбінаторним вибухом". Це робить класичні детерміновані методи проєктування такі як перебір та таблиці пошуку непридатними для схем із кількістю змінних $n > 4$.

4. Обмеженість існуючих методів синтезу. Проведений аналіз існуючих засобів автоматизованого проєктування показав, що стандартні евристичні алгоритми часто генерують схеми, далекі від оптимальних за критеріями квантової вартості та глибини. Класичні еволюційні стратегії страждають від передчасної збіжності до локальних максимумів та низької швидкодії через відсутність механізмів адаптації до специфіки зворотних вентилів.
5. Необхідність розробки гібридних інтелектуальних методів. На основі виявлених проблем сформульовано основне завдання дослідження: розробка та вдосконалення методів еволюційного синтезу, зокрема створення гібридних стратегій пошуку. Такий підхід має поєднувати глобальні механізми розвідки простору станів – еволюційні алгоритми з локальними евристиками оптимізації структури такими як багатокомпонентна мутація як оптимізація перестановки виходів. Це дозволить автоматизувати проєктування конкурентних, відмовостійких зворотних пристроїв прийнятної складності.

РОЗДІЛ 2. ОГЛЯД МЕТОДІВ СИНТЕЗУ ЗВОРОТНИХ СХЕМ

Проєктування зворотних схем є самостійною та швидкозростаючою галуззю комп'ютерної інженерії, яка вимагає принципово нових алгоритмічних підходів порівняно з традиційним логічним синтезом. Якщо в класичній схемотехніці основна задача зводиться до мінімізації булевих виразів у базисі {AND, OR, NOT}, то в зворотній логіці задача формулюється як декомпозиція глобальної перестановки станів на послідовність елементарних зворотних операцій.

Специфіка фізичних реалізацій (квантові точки, адіабатична CMOS-логіка, оптичні системи) накладає жорсткі обмеження на структуру схем: фіксована кількість ліній, заборона розгалуження сигналів та відсутність зворотних зв'язків. Це робить стандартні маршрути проєктування та наявні CAD-інструменти непридатними для прямого використання.

У цьому розділі проведено систематизацію та порівняльний аналіз існуючих методів синтезу зворотних схем. Розглянуті підходи можна умовно поділити на три великі класи:

1. Точні методи: базуються на повному переборі або використанні SAT-солверів. Вони гарантують знаходження мінімальної схеми, але обмежені малим числом змінних.
2. Евристичні та конструктивні методи: використовують набори правил або декомпозицію функцій для швидкого синтезу, не гарантуючи при цьому оптимальності отриманого рішення.
3. Методи штучного інтелекту: застосовують еволюційні стратегії та алгоритми рою частинок для пошуку субоптимальних рішень у багатовимірних просторах.

Метою даного розділу є детальний аналіз математичного апарату, базових логічних примітивів та алгоритмічних стратегій, що використовуються в сучасних системах автоматизованого проєктування. Такий аналіз дозволить

виявити недоліки існуючих рішень та сформувати вимоги до вдосконалених методів синтезу, розробка яких є предметом даної дисертаційної роботи.

2.1. Математичні моделі та елементна база зворотної логіки

Формалізація процесу проєктування зворотних пристроїв вимагає переходу від класичної булевої алгебри, що оперує незворотними функціями (наприклад, AND:$\{0,1\}^2 \to \{0,1\}$), до алгебри перестановок. Математичною моделлю зворотної логічної схеми з *n* вхідними та *n* вихідними лініями є функція *F*, яка здійснює взаємно-однозначне відображення (бієкцію) простору вхідних станів на простір вихідних станів.

Нехай $B = \{0,1\}$. Множина всіх можливих вхідних векторів X та вихідних векторів Y визначається як B^n. Функція *F:* $B^n \to B^n$ називається зворотною, якщо для кожного вектора $Y \in B^n$ існує єдиний вектор $X \in B^n$, такий що:

$$F(X) = Y \text{ та } F^{-1}(Y) = X, \tag{2.1.1}$$

У контексті квантової інформатики та лінійної алгебри, поведінку такої системи зручно описувати не через таблиці істинності, а за допомогою матричного формалізму. Кожному логічному стану системи k (де $0 \leq k < 2n$) ставиться у відповідність базисний вектор e_k у гільбертовому просторі розмірності 2^n.

Тоді будь-який зворотний логічний елемент або ціла схема може бути представлена унітарною матрицею перестановки M розмірності $2^n \times 2^n$. Ця матриця складається виключно з нулів та одиниць, причому в кожному рядку і кожному стовпці міститься рівно одна одиниця. Дія схеми на вхідний вектор X описується матричним множенням:

$$Y_{state} = M \cdot X_{state}, \tag{2.1.2}$$

Ключовою властивістю такої моделі є те, що послідовне з'єднання логічних елементів математично відповідає добутку їхніх матриць. Якщо схема складається з послідовності вентилів $g_1, g_2, \dots g_k$, що описуються матрицями $M_1, M_2, \dots M_k$, то результуюча матриця схеми M_{sys} визначається як:

$$M_{sys} = M_k \cdot M_{k-1} \cdot \dots \cdot M_1, \tag{2.1.3}$$

Важливо зазначити, що множення матриць є некомутативним ($A \cdot B \neq B \cdot A$). Це накладає фундаментальне обмеження на методи синтезу: зміна порядку розташування вентилів у схемі призводить до зміни реалізованої логічної функції.

Таким чином, задача синтезу зворотної схеми з математичної точки зору формулюється як задача факторизації або декомпозиції заданої матриці перестановки P на добуток матриць елементарних вентилів із заданої бібліотеки, при мінімізації кількості множників k або загальної вартості реалізації.

2.1.1. Класифікація зворотних вентилів: від класичних (NOT, CNOT) до квантових операторів

У задачах автоматизованого синтезу вибір бібліотеки елементів – Gate Library є першим кроком, що визначає ефективність майбутньої схеми. Зворотні вентилі класифікують за кількістю входів/виходів, логічною функцією та вартістю реалізації.

Універсальною основою для побудови будь-якої зворотної булевої функції є бібліотека NCT (NOT, CNOT, Toffoli) [63]. Проте для оптимізації вартості часто використовують розширені бібліотеки, що включають вентилі Фредкіна який можна легко отримати з двох CNOT та одного Toffoli [64].

Розглянемо детально основні типи вентилів, розглядаючи їх як оператори над вектором станів $X = (x_1, x_2, \dots, x_n)$.

1. Вентилі сімейства NCT та GT. Це основний клас вентилів, що інвертують значення на цільовій лінії (target line) тоді і тільки тоді, коли на всіх керуючих лініях (control lines) встановлено значення логічної одиниці.

- Вентиль NOT (Inverter): Найпростіший елемент *1×1*. Зображений на рисунку 2.1.1.1

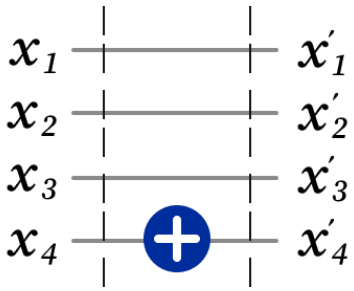


рис. В.1. Квантовий вентиль NOT

Змінює стан біта на протилежний. У квантовій механіці відповідає оператору Паулі-X σ_x.

$$x' = x \oplus 1, \qquad (2.1.1.1)$$

- Вентиль CNOT (Controlled-NOT, вентиль Фейнмана): елемент 2×2 з одним керуючим входом x_1 та одним цільовим x_2 зображений на рисунку 2.1.1.2. Якщо $x_1 = 1$, то x_2 інвертується.

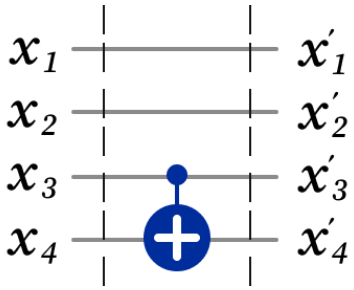


рис. 2.1.1.2. Квантовий вентиль CNOT

Цей вентиль є критично важливим, оскільки дозволяє копіювати інформацію, що заборонено у чистому вигляді, та створювати заплутані стани у квантових системах. Описується функцією

$$\begin{cases} x_1' = x_1 \\ x_2' = x_1 \oplus x_2 \end{cases}, \tag{2.1.1.2}$$

- Вентиль Toffoli (CCNOT): елемент *3×3* з двома керуючими входами (рисунок 2.1.1.3).

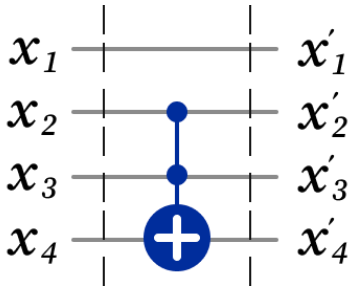


рис. 2.1.1.3. Квантовий вентиль Toffoli

Це універсальний вентиль, на базі якого можна реалізувати класичні операції AND та NAND, що робить його достатнім для побудови будь-якої логічної схеми.

$$\begin{cases} x_1' = x_1 \\ x_2' = x_2 \\ x_3' = x_3 \oplus (x_1 \cdot x_2) \end{cases}, \tag{2.1.1.3}$$

- Узагальнений Тоффолі (Generalized Toffoli — GT): Вентиль з k керуючими лініями $k > 2$(рисунок 2.1.1.4).

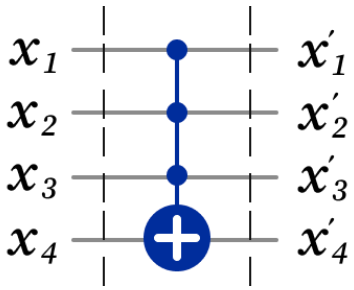


Рис. 2.1.1.4 Узагальнений вентиль Тоффолі

Він інвертує цільовий біт, якщо активні всі k контролів. Хоча такі вентилі спрощують логічну структуру схеми на етапі проєктування, їх фізична реалізація є дуже дорогою і вимагає декомпозиції на простіші елементи.

2. Вентилі сімейства Swap (Fredkin). Цей клас вентилів змінює не значення бітів, а їх розташування (маршрутизацію).

- Вентиль Фредкіна (Controlled-SWAP): елемент 3×3, який міняє місцями значення двох цільових ліній, якщо керуюча лінія активна.

$$\begin{cases} x_1' = x_1 \\ x_2' = (\overline{x_1} \cdot x_2) \oplus (x_1 \cdot x_3), \\ x_3' = (\overline{x_1} \cdot x_3) \oplus (x_1 \cdot x_2) \end{cases} \quad (2.1.1.4)$$

Вентиль Фредкіна, зображений на рисунку 2.1.1.5, є консервативним [23], тобто вага Геммінга вхідного вектора дорівнює вазі вихідного $\sum x_i = \sum x_i'$. Це робить його ідеальним для побудови відмовостійких схем [43], де помилка може бути виявлена через порушення балансу одиниць та нулів.

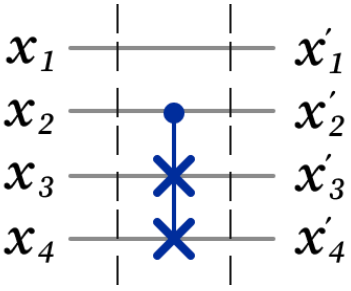


Рис. 2.1.1.5 Квантовий вентиль Фредкіна

- Комбіновані вентилі

Створені для зменшення квантової вартості схем. Вентиль Переса (Peres Gate — PG): Елемент *3×3*, який часто називають "новим універсальним вентилем", зображений на рисунку 2.1.1.6.

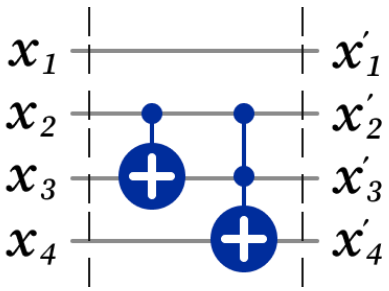


Рис. 2.1.1.6 Квантовий вентиль Переса

Він функціонально еквівалентний послідовному включенню вентиля Toffoli та CNOT, але реалізується фізично дешевше.

$$\begin{cases} x_1' = x_1 \\ x_2' = x_1 \oplus x_2 \\ x_3' = x_3 \oplus (x_1 \cdot x_2) \end{cases}, \qquad (2.1.1.5)$$

Використання вентиля Переса [65] дозволяє суттєво зменшити кількість елементарних квантових операцій порівняно з чистим NCT-базисом.

Кожен із наведених вентилів є зворотним $G \cdot G^{-1} = I$. При цьому вентилі NOT, CNOT, Toffoli, Fredkin, Peres є самозворотними – унітарним, тобто $G^{-1} = G$. Це означає, що застосування одного й того ж вентиля двічі поспіль повертає систему у вихідний стан, що значно спрощує процедуру дешифрування у криптографічних застосуваннях. Квантова вартість описаних вентилів представлена в таблиці 2.1.1.1

Таблиця 2.1.1.1. Квантова вартість зворотних вентилів [66]

Вентиль	Квантова вартість
NOT	1
CNOT	1
Тоффолі	5
Узагальнений Тоффолі	$2^n - 3$, де n – кількість ліній
Фредкіна	5
Переса	4

Окрім класичних вентилів з позитивною полярністю керування, де активація цільової лінії відбувається виключно за умови наявності логічної одиниці на керуючих входах, у задачах автоматизованого синтезу зворотних схем активно застосовуються вентилі з негативною полярністю активації [64, 22, 67].

У таких вентилях зміна стану цільового біта ініціюється тоді, коли відповідна керуюча лінія знаходиться у стані логічного нуля. На графічних схемах керуючі входи з негативною полярністю традиційно позначаються незафарбованим колом ∘, на відміну від суцільно зафарбованого кола ·, яке використовується для позитивної полярності.

Математично функціонування такого вентиля описується через інверсію відповідного керуючого сигналу. Наприклад, для узагальненого вентиля Тоффолі з одним негативним керуючим входом x_1 та одним позитивним x_2, функція переходу для цільової лінії x_3 набуде наступного вигляду:

$$\acute{x_1} = x_3 \oplus (\overline{x_1} \cdot x_2) \qquad (2.1.1.5)$$

Введення до бібліотеки елементів вентилів із змішаною або суто негативною полярністю має критичне значення для структурної оптимізації. У класичному базисі NCT для реалізації активації за нулем необхідно розмістити вентиль NOT перед керуючим входом і ще один вентиль NOT після нього. Застосування ж вентилів з вбудованою негативною полярністю дозволяє логічно поглинути вентилі NOT. Це суттєво зменшує загальну кількість логічних елементів у каскаді, мінімізуючи як затримку поширення сигналу, так і сумарну квантову вартість результуючої схеми.

2.1.2 Матричне представлення логічних вентилів

Для формального аналізу, верифікації та симуляції роботи зворотних схем використовується апарат лінійної алгебри. У цьому представленні кожен логічний стан системи n змінних розглядається як одиничний вектор у 2^n-вимірному гільбертовому просторі H^{2^n}.

Базисні стани кодуються як ортогональні вектори-стовпчики. Наприклад, для одного кубіта (біта):

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}, \quad (2.1.2.1)$$

Будь-який зворотний логічний вентиль, що діє на n змінних, представляється унітарною матрицею U розмірності $2^n \times 2^n$. Оскільки ми розглядаємо класичну зворотну логіку (без суперпозиції станів), ці матриці є матрицями перестановки: вони містять лише нулі та одиниці, і є розрідженими.

Матриці базових вентилів:

1. Вентиль NOT: Діє на 1 змінну. Матриця *2×2* міняє місцями базисні вектори $|0\rangle$ та $|1\rangle$:

$$C_{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad (2.1.2.2)$$

2. Вентиль CNOT: Діє на 2 змінні. У базисі $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ матриця має розмірність *4×4*. Вона залишає перші два стани без змін (коли керуючий біт = 0), а останні два міняє місцями (коли керуючий біт = 1):

$$C_{NOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad (2.1.2.3)$$

3. Вентиль Toffoli: Діє на 3 змінні. Матриця *8×8* є одиничною матрицею – Identity Matrix для перших 6 рядків, а останні два рядки, що відповідають станам $|1100\rangle$ та $|111\rangle$ переставлені:

$$Toffoli = \begin{pmatrix} I_6 & 0 \\ 0 & X \end{pmatrix}, \text{де } X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad (2.1.2.4)$$

Масштабування системи та тензорний добуток

Критично важливим аспектом матричного моделювання є опис дії вентиля меншої розмірності на систему більшої розмірності. Якщо вентиль *G* діє лише на частину ліній схеми, то повна матриця операції M_{op} обчислюється як тензорний добуток (добуток Кронекера) матриць, що діють на кожну лінію.

Нехай схема має 3 лінії (x_1, x_2, x_3), і ми застосовуємо вентиль NOT до лінії x_2. Тоді на лініях x_1 та x_3 діє тотожне перетворення (Identity matrix, $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$). Повна матриця перетворення:

$$M_{sys} = I \otimes C_{NOT} \otimes I, \quad (2.1.2.5)$$

Розрахунок:

$$M_{sys} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad (2.1.2.6)$$

Саме операція тензорного добутку призводить до експоненційного зростання розмірності матриць $2^n \times 2^n$, що робить метод прямих матричних обчислень непридатним для симуляції схем з великою кількістю змінних n>15…20 на класичних комп'ютерах, вимагаючи застосування більш ефективних структур даних, таких як QMDD (Quantum Multiple-Valued Decision Diagrams) [115].

2.1.3. Універсальні базиси (NCT, Toffoli, Fredkin) та умови повноти

Поняття функціональної повноти у зворотній логіці суттєво відрізняється від класичного визначення у булевій алгебрі. Якщо у традиційній схемотехніці набір вентилів вважається тим який складає повний базис (універсальним), якщо за його допомогою можна реалізувати будь-яку булеву функцію (наприклад: штрих Шеффера NAND), то у зворотній логіці повнота визначається здатністю генерувати групу всіх можливих перестановок [48].

Множина всіх зворотних функцій над n змінними утворює симетричну групу S_{2^n}. Набір зворотних вентилів є універсальним, якщо композицією цих вентилів можна отримати будь-який елемент групи S_{2^n} (для парних n) або знакозмінної групи A_{2^n} (для $n \geq 3$).

Найбільш поширеним та дослідженим є базис NCT [68]. З точки зору теорії груп, він є універсальним для реалізації всіх парних перестановок вхідних векторів.

1. Вентиль Toffoli забезпечує нелінійність, необхідну для обчислень (аналог AND).

2. Вентилі NOT та CNOT дозволяють здійснювати лінійні перетворення та змінювати парність перестановки [69].

Відомо, що для $n \geq 3$ набір NCT є повним. Це означає, що будь-яка зворотна функція може бути реалізована без використання додаткових ліній, якщо вона належить до класу парних перестановок. Для непарних перестановок необхідне додавання однієї надлишкової лінії.

Існує сильніше твердження: множина вентилів Toffoli $TOFF = \{T_k \mid k = 0, 1, \dots, n\}$, де T_k – вентиль з k керуючими лініями (при цьому $T_0 = NOT, T_1 = CNOT, T_2 = Toffoli$), також є універсальною [9]. Це дозволяє будувати схеми, використовуючи лише один тип логічного примітиву різної розмірності, що спрощує структуру генотипу в еволюційних алгоритмах [70].

Вентиль Фредкіна є функціонально повним для консервативної логіки (де кількість одиниць на вході дорівнює кількості на виході). Проте, сам по собі він не може реалізувати функції, що не зберігають вагу Геммінга (наприклад, інвертор) [23]. Для досягнення універсальності у широкому сенсі (реалізація довільної булевої функції), базис вентилів Фредкіна вимагає доступу до джерел констант (0 та 1) та використання додаткових ліній. За наявності таких ліній вентиль Фредкіна може моделювати роботу всіх класичних вентилів (AND, OR, NOT, FAN-OUT) [31, 71].

Важливою особливістю синтезу є те, що багато логічних функцій неможливо реалізувати зворотньо без введення додаткових ліній. Наприклад, функція $y = x_1 \cdot x_2$ (класичне AND) не є зворотною, оскільки втрачає інформацію. Щоб реалізувати її у зворотному базисі, необхідно збільшити розмірність простору з 2 до 3 змінних, де третя лінія слугує для запису результату, зберігаючи вхідні операнди незмінними:

$$(x_1, x_2, 0) \rightarrow (x_1, x_2, x_1 \cdot x_2) \qquad (2.1.3.1)$$

Таким чином, умова повноти базису завжди розглядається у контексті доступних ресурсів:

1. Strict Universality: можливість реалізувати функцію на n лініях без додаткової пам'яті (можливо лише для парних перестановок).
2. Universality with Ancilla: можливість реалізувати будь-яку функцію за умови додавання k допоміжних ліній, гарантується для базисів NCT та Fredkin.

Для задач автоматизованого синтезу це означає, що алгоритм повинен вміти динамічно визначати необхідність додавання нових ліній, якщо цільова перестановка не може бути знайдена у поточному просторі станів.

2.2. Критерії якості та властивості зворотних схем

Процес синтезу зворотних схем принципово відрізняється від класичного проєктування тим, що простір допустимих рішень є значно вужчим через жорсткі структурні обмеження. Перед тим як оцінювати ефективність отриманого пристрою, необхідно визначити набір фундаментальних властивостей, яким повинна відповідати будь-яка коректна зворотна схема.

Якщо в класичній схемотехніці якість визначається переважно площею кристала та затримкою сигналу, то у зворотній логіці на перший план виходять структурна коректність та інформаційна цілісність.

Для того щоб цифрова схема вважалася зворотною та придатною для фізичної реалізації на квантових або адіабатичних обчислювачах, вона повинна задовольняти наступним умовам:

1. Бієктивність відображення. Схема повинна реалізувати взаємно-однозначне відображення вхідних векторів у вихідні. Це означає, що кожному можливому вхідному стану відповідає один і лише один вихідний стан. Порушення цієї умови призводить до незворотної втрати

інформації та виділення тепла, що суперечить меті створення енергоефективних пристроїв.

2. Каскадна структура. Зворотна схема моделюється як послідовність логічних вентилів, розташованих один за одним. Виходи попереднього вентиля є входами для наступного. Це спрощує матричний опис схеми, дозволяючи представляти її як добуток матриць окремих каскадів.
3. Заборона розгалуження. У класичній електроніці вихід одного логічного елемента може бути поданий на входи декількох інших елементів. У зворотній логіці пряме копіювання сигналу шляхом розгалуження провідника є неможливим – що також підтверджується теоремою про заборону клонування [72]. Для дублювання сигналу необхідно використовувати спеціальні вентилі (наприклад, CNOT або Feynman gate), які явно прописуються в структурі схеми.
4. Відсутність зворотних зв'язків. У рамках базової задачі синтезу комбінаційних схем заборонено використання петель зворотного зв'язку. Потік інформації спрямований виключно від входів до виходів.

Окремою проблемою є оцінка якості реалізації функцій, які за своєю природою не є зворотною, наприклад, суматор, де з суми неможливо однозначно відновити доданки. У таких випадках оцінка якості схеми нерозривно пов'язана з ефективністю вкладення незворотної функції у зворотну. Це вимагає розширення простору станів шляхом додавання:

1. Константних входів: ліній, на які подаються фіксовані значення (0 або 1) для налаштування роботи вентилів.
2. Надлишкових виходів: ліній, які не несуть корисної інформації про результат фітнес-функції, але необхідні для забезпечення зворотності – збереження унікальності станів.

Таким чином, "ідеальна" зворотна схема — це така схема, яка реалізує задану таблицю істинності, дотримуючись усіх топологічних обмежень, і при

цьому використовує мінімальну кількість ресурсів. Кількісний вимір цих ресурсів здійснюється через систему спеціалізованих метрик.

2.2.1. Метрики оптимізації: квантова вартість, глибина схеми, апаратна складність, додаткові входи та надлишкові виходи

Оцінка ефективності синтезованих зворотних схем є багатофакторною задачею. На відміну від традиційної CMON-логіки, де основними критеріями є площа кристала та енергоспоживання, у зворотній логіці "вартість" обчислень визначається складністю фізичної взаємодії кубітів та необхідними ресурсами для підтримання когерентності системи.

Для кількісного порівняння схем та побудови фітнес-функції в еволюційних алгоритмах використовуються наступні метрики.

1. Квантова вартість [48] (Quantum Cost — QC)

Квантова вартість є найбільш репрезентативною метрикою складності. Вона визначається як кількість елементарних квантових операцій, необхідних для реалізації заданого логічного вентиля або схеми.

За одиницю вартості ($QC = 1$) приймаються примітивні операції, що фізично легко реалізуються: інверсія одного кубіта – 1×1 або контрольована операція над двома кубітами – наприклад, CNOT, 2×2.

Вартість складніших вентилів розраховується на основі їх декомпозиції на елементарні примітиви [66]:

- Вентиль Toffoli (3×3): потребує 5 елементарних вентилів - $QC = 5$.
- Вентиль Peres (3×3): має $QC = 4$. Це робить його більш вигідним ресурсом для синтезу, оскільки він виконує складну логіку з меншими витратами.
- Вентиль Fredkin (3×3): має QC=5.

Загальна квантова вартість схеми - $QC_{circuit}$ дорівнює сумі вартостей усіх її компонентів. Мінімізація саме цього параметру є пріоритетною задачею синтезу.

2. Кількість вентилів (Gate Count — GC)

Це метрика, що визначає загальне число логічних блоків у схемі, незалежно від їхньої внутрішньої складності.

$$GC = N_{gates}, \tag{2.2.1.1}$$

Хоча GC є грубішою оцінкою, ніж квантова вартість, її мінімізація важлива для зменшення загальної ймовірності відмови системи, оскільки кожен фізичний вентиль є потенційним джерелом шуму та помилок.

3. Апаратна складність: додаткові та надлишкові лінії

Відмінністю зворотного логічного синтезу є вимога збереження розмірності простору станів: кількість входів схеми повинна строго дорівнювати кількості виходів $n \times n$. Однак більшість практично значущих логічних функцій, що використовуються в цифровій техніці такі як суматори, мультиплексори, компаратори є незворотними і описуються відображенням $f: \{0,1\}^n \to \{0,1\}^m$, де зазвичай n>m.

Неможливість прямої реалізації таких функцій у зворотному базисі породжує проблему структурної надлишковості. Для забезпечення умови бієктивності необхідне розширення схеми шляхом введення допоміжних ліній, що призводить до появи двох типів ресурсних витрат: додаткових входів (Ancilla Inputs) та надлишкових виходів (Garbage Outputs).

Додатковими входами називаються лінії, які не несуть вхідних змінних заданої функції, а ініціалізуються фіксованими константними значеннями (логічним 0 або 1). Вони необхідні у двох випадках:

- Для збереження результату: Оскільки зворотні вентилі змінюють вхідні дані "на місці", для збереження результату операції без втрати операндів

необхідна вільна лінія. Наприклад, класична операція $y = a \cdot b$ (AND) вимагає переходу від 2 змінних до 3: $(a, b, 0) \rightarrow (a, b, a \cdot b)$.

- Для керування логікою: У деяких випадках, наприклад, у вентилі Фредкіна, константа на вході дозволяє перетворити універсальний вентиль на специфічний (наприклад, реалізувати NOT або FAN-OUT).

Збільшення кількості ліній експоненційно розширює простір пошуку для алгоритмів синтезу – $N!$, що робить мінімізацію кількості додаткових ліній пріоритетною задачею [73, 74].

Надлишковими виходами називаються вихідні лінії схеми, значення яких не потрібні користувачеві (не входять до множини фітнес-функцій), але є побічним продуктом забезпечення зворотності. Формально, якщо ми реалізуємо функцію з n входами та m виходами, використовуючи k додаткових ліній, то загальна ширина схеми становить $w = n + k$. Тоді кількість надлишкових виходів g визначається як:

$$g = w - m = (n + k) - m \qquad (2.2.2.1)$$

Проблема "очищення" надлишковості: У класичній схемотехніці невикористані виходи можна просто ігнорувати. У квантових обчисленнях наявність надлишкових кубітів, які заплутані з корисними даними, є критичною проблемою. Якщо виміряти або відкинути ці кубіти, це може зруйнувати суперпозицію всієї системи. Тому для коректної роботи часто використовується техніка Bennett's embedding [75] або зворотного обчислення, коли після отримання результату запускається дзеркальна копія схеми для повернення сміттєвих ліній у стан констант 0 чи 1 відповідно. Це подвоює кількість вентилів, що робить кожну одиницю надлишкового виходу надзвичайно "дорогою".

4. Глибина схеми (Circuit Depth)

Глибина схеми визначається як максимальна кількість вентилів на критичному шляху від входу до виходу. Ця метрика безпосередньо корелює зі

швидкодією (затримкою). У контексті квантових реалізацій глибина схеми є критичним параметром через явище декогеренції: квантовий стан існує обмежений час, тому обчислення має завершитися до того, як система втратить інформацію.

Важливо зазначити, що наведені метрики часто є конфліктними. Наприклад, зменшення кількості надлишкових виходів часто призводить до різкого зростання квантової вартості через необхідність складнішої логіки перестановки. Тому задача синтезу формулюється як пошук Парето-оптимальних рішень, де покращення одного параметру неможливе без погіршення іншого. У рамках даного дослідження основна увага приділяється мінімізації квантової вартості як інтегральному показнику якості.

2.2.2. Відмовостійкість та методи збереження парності

Однією з ключових проблем фізичної реалізації зворотних та квантових обчислювачів є їхня висока чутливість до зовнішніх завад [26]. На відміну від класичної CMON-логіки, де рівні напруги – логічний 0 та 1 мають значні запаси завадостійкості, квантові стани є крихкими, і будь-яка взаємодія з середовищем може призвести до спонтанної інверсії біта.

Традиційні методи підвищення надійності, такі як потрійне апаратне резервування, є занадто витратними для зворотних схем через високу вартість кожного кубіта. Тому пріоритетним напрямком є проєктування схем з вбудованою властивістю самоперевірки [76, 77].

Основними підходами до забезпечення відмовостійкості [78, 79] на етапі логічного синтезу є використання збереження парності [80] (Parity Preservation).

Схема називається такою, що зберігає парність, якщо парність вхідного вектора завжди дорівнює парності вихідного вектора:

$$x_1 \oplus x_2 \oplus \ldots \oplus x_n = y_1 \oplus y_2 \oplus \ldots \oplus y_n \qquad (2.2.3.1)$$

Ця властивість дозволяє детектувати будь-яку непарну кількість помилок у режимі реального часу. Якщо на виході схеми сума за модулем 2 не співпадає з вхідною, це сигналізує про збій.

Проблема синтезу: Більшість стандартних універсальних вентилів не зберігають парність.

- Наприклад, вентиль Toffoli ($A, B, C \rightarrow A, B, C \oplus AB$) змінює парність, оскільки вносить нелінійність без компенсації.
- Для побудови таких схем необхідно використовувати спеціалізовані відмовостійкі зворотні вентилі зі збереженням парності, такі як вентиль CNOT або вентиль Фредкіна.

Більш суворим критерієм відмовостійкості є збереження парності. Логічний вентиль, або схема, називається тим який зберігає парність, якщо вага Геммінга вхідного вектора дорівнює вазі Геммінга вихідного вектора:

$$\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \qquad (2.2.3.2)$$

Ця властивість є природною для фізичних процесів, що описують переміщення носіїв заряду або енергетичних пакетів без їх генерації чи знищення.

1. Вентиль Фредкіна є базовим елементом консервативної логіки. Оскільки він лише переставляє біти місцями (SWAP), кількість одиниць завжди зберігається.
2. Будь-яка схема, побудована виключно з вентилів Фредкіна, є автоматично консервативною, що робить її ідеальною для тестування: помилка типу "bit-flip" (0 замість 1) порушує баланс ваги Геммінга і миттєво детектується.

Врахування вимог відмовостійкості накладає додаткові обмеження (constraints) на простір пошуку еволюційного алгоритму:

2.3. Аналіз підходів до синтезу та реконфігурації вентильних структур

Задача синтезу зворотних схем формально визначається як пошук [82] послідовності елементарних вентилів $G_1, G_2,\ldots,G_k$ з фіксованої бібліотеки, композиція яких реалізує задану цільову перестановку P:

$$P = G_k \circ G_{k-1},\circ \ldots \circ G_1 \tag{2.3.1}$$

У сучасній теорії проєктування зворотних пристроїв виділяють три основні класи методів, що розрізняються за стратегією пошуку та якістю отримуваних рішень [83]:

- Точні методи: Забезпечують знаходження схеми з мінімально можливою квантовою вартістю або кількістю вентилів. Вони базуються на алгоритмах пошуку у просторі станів (наприклад, A*, BFS) або використанні SAT-солверів (Satisfiability solvers). Головний недолік — неможливість роботи зі схемами розмірністю понад 4-5 змінних через експоненційну складність.
- Евристичні та конструктивні методи [84]: Ці методи не гарантують оптимальності, але здатні синтезувати схеми для великої кількості змінних ($n > 10$) за прийнятний час. Вони базуються на декомпозиції складної перестановки на простіші (наприклад, алгоритми на основі перетворень, такі як MMD (Miller-Maslov-Dueck Algorithm) [85] або алгоритми на основі діаграм рішень QMDD. Часто результатом роботи таких методів є схеми з великою кількістю надлишкових вентилів.

- Метаевристичні або еволюційні методи [86]: Це клас стохастичних алгоритмів, які імітують природні процеси для пошуку субоптимальних рішень. Вони займають проміжну нішу: дозволяють знаходити кращі рішення, ніж прості евристики, і працюють зі складнішими схемами, ніж точні методи. Саме цей клас методів є предметом дослідження даної роботи.

2.3.1. Огляд методів синтезу логічних функцій на основі таблиць істинності

Синтез на основі таблиць істинності є найбільш прямим підходом до проєктування зворотних схем. Вхідними даними для таких методів є вектор перестановки P довжиною $N = 2^n$, де $P[i]$ вказує, у який вихідний стан переходить вхідний стан i.

Загальна ідея цих методів полягає в декомпозиції складної перестановки P на послідовність простіших перестановок, що відповідають базовим вентилям (NOT, CNOT, Toffoli).

Найбільш відомим та широко вживаним представником цього класу є алгоритм MMD (Miller-Maslov-Dueck). Це детермінований алгоритм, який гарантує знаходження рішення для будь-якої зворотної функції n змінних.

Принцип роботи: Алгоритм працює ітеративно, рядок за рядком наближаючи таблицю істинності до вигляду $y_i = x_i$.

1. Розглядається перший вхідний набір, наприклад, 00 ... 0. Якщо вихід не співпадає з входом, додаються вентилі NOT/CNOT, щоб перетворити вихідне значення на потрібне.
2. Отримані вентилі фіксуються, і алгоритм переходить до наступного рядка таблиці.

3. При переході до наступних рядків використовуються вентилі з більшою кількістю керуючих ліній (Toffoli, Generalized Toffoli), щоб не порушити вже впорядковані попередні рядки (властивість керування).

Перевага цього методу – швидкодія та гарантована збіжність. Алгоритм здатний синтезувати схеми для $n > 20$ за лічені секунди.

Недолік – це отримані схеми часто є далекими від оптимальних. Вони мають характерну "східчасту" структуру та надмірну квантову вартість, що вимагає застосування процедур пост-оптимізації.

Синтез на основі розкладання на цикли (Cycle-Based Synthesis). Згідно з теорією груп, будь-яку перестановку можна представити як добуток незалежних циклів. Наприклад, перестановка (2,1,3) містить цикл (1,2), де елементи міняються місцями, а 3 залишається на місці. Методи цього класу спочатку розкладають задану функцію на множину неперетинних циклів. Далі кожен цикл реалізується окремо за допомогою спеціальних каскадів вентилів.

- Цикли довжиною 2 (транспозиції) реалізуються найпростіше.
- Цикли довжиною 3 є базовими "цеглинками" для побудови парних перестановок. Цей підхід дозволяє краще контролювати структуру схеми, ніж MMD, але також схильний генерувати довгі ланцюжки вентилів.

Точні методи перебору (Exact Synthesis). Для схем малої розмірності використовують методи, що гарантують знаходження мінімально можливої схеми.

1. Пошук у ширину (BFS): Алгоритм будує дерево всіх можливих схем, починаючи з 1 вентиля, потім 2, 3 і т.д., доки не буде знайдено схему, що реалізує потрібну функцію. Через експоненційний ріст дерева цей метод застосовний лише для $n \leq 3$.
2. Алгоритм A* (A-Star): Вдосконалена версія BFS, яка використовує евристичну функцію оцінки відстані до мети, щоб відсікати безперспективні гілки пошуку. Дозволяє працювати з $n = 4$, але вимагає значних обсягів оперативної пам'яті.

3. SAT-Based синтез: Задача синтезу формулюється як задача здійсненності булевої формули (SAT), яка потім вирішується спеціальними солверами.

Методи на основі таблиць істинності є надійним інструментом для отримання початкового рішення. Однак, MMD генерує надлишкові схеми, а точні методи не масштабуються. Це створює потребу в еволюційних методах, які здатні знаходити рішення кращі за MMD, працюючи з розмірностями, недосяжними для точних методів.

2.3.2. Існуючі підходи до динамічної реконфігурації

У контексті проєктування зворотних та квантових обчислювальних пристроїв під динамічною реконфігурацією розуміють здатність фіксованого логічного блоку змінювати вид реалізованої булевої функції під впливом зовнішніх керуючих сигналів. На відміну від структурної реконфігурації, де змінюються фізичні зв'язки між елементами, функціональна реконфігурація базується на використанні універсальних логічних блоків або програмованих зворотних вентилів [42, 87].

Ідея реконфігурації [88] полягає у розділенні вхідних ліній пристрою на два класи:

1. Інформаційні входи ($x_1, \dots, x_n$): змінні, над якими виконується операція.
2. Керуючі або налаштувальні входи ($c_1, \dots, c_m$): лінії, на які подаються константи (0 або 1).

Змінюючи вектор значень на керуючих входах, один і той самий апаратний блок на виході формує результати різних логічних операцій (AND, OR, XOR, NAND тощо).

$$F(x, c) = \begin{cases} x_1 \cdot x_2, & \text{якщо } c = 0 \\ x_1 + x_2, & \text{якщо } c = 1 \end{cases}, \qquad (2.3.2.1)$$

Цей підхід дозволяє створювати гнучкі архітектури, схожі за принципом дії на класичні FPGA, але адаптовані до вимог зворотності.

У науковій літературі виділяють кілька основних напрямків реалізації реконфігурованих структур [89]:

1. *Реалізація на основі мультиплексорів* – класичним підходом є використання теореми розкладання Шеннона, згідно з якою будь-яку булеву функцію можна реалізувати за допомогою мультиплексорів. Оскільки базовий зворотний вентиль Фредкіна (CSWAP) функціонально є керованим перемикачем – мультиплексором 2:1, він історично став першим елементом для побудови реконфігуровної логіки. Існуючі рішення пропонують каскади вентилів Фредкіна, де частина входів зафіксована константами. Однак, відомі топології часто страждають від надмірної кількості надлишкових виходів, оскільки кожен вентиль Фредкіна генерує два невикористаних виходи на одну корисну операцію.
2. *Спеціалізовані програмовані вентилі* - низка дослідників пропонує використовувати спеціально розроблені складні вентилі, наприклад: – HNG, TSG, MKG [90], які спроєктовані таким чином, щоб забезпечити максимальну кількість логічних функцій при мінімальній кількості вентилів. Такі елементи здатні реалізовувати повний набір класичних булевих операцій в одному блоці. Перевага – це компактність на логічному рівні. Недолік - висока квантова вартість реалізації. Фізично такий "один" вентиль розпадається на велику кількість елементарних квантових операцій, що робить його складним для практичного застосування в умовах декогеренції.
3. *Реконфігурація на основі бібліотек.* Цей підхід передбачає створення бібліотеки фіксованих підсхем з різною вартістю, які підставляються в основну схему в залежності від режиму роботи. Хоча це забезпечує гнучкість, такий метод вимагає значних ресурсів пам'яті для зберігання варіантів конфігурації та складних алгоритмів комутації сигналів.

Аналіз сучасних публікацій показує, що більшість існуючих підходів до реконфігурації мають спільні недоліки:

- Надлишковість – для забезпечення універсальності, можливості реалізувати будь-яку функцію, схеми містять багато зайвих елементів, які не працюють у конкретний момент часу.
- Відсутність систематизованого синтезу: Більшість рішень є евристичними ("ручними") розробками конкретних блоків, а не результатом роботи автоматизованих алгоритмів.

Це обумовлює необхідність розробки нових методів автоматизованого синтезу реконфігурованих структур, які б знаходили оптимальний баланс між універсальністю блоку та його квантовою вартістю.

2.4. Порівняльний аналіз еволюційних методів синтезу схем

Як було показано в попередніх підрозділах, простір пошуку зворотних схем зростає факторіально зі збільшенням кількості змінних ($N!$). Це робить задачу синтезу NP-складною, що унеможливлює використання точних детермінованих методів для схем практичної значущості ($n > 5$). З іншого боку, існуючі евристики (наприклад, MMD) дають гарантований результат, але часто генерують субоптимальні схеми з надлишковою квантовою вартістю.

Для подолання цих обмежень у сучасній теорії синтезу застосовуються метаевристичні алгоритми. Це клас методів стохастичної оптимізації, які не гарантують знаходження ідеального (глобального) екстремуму, але здатні знаходити достатньо хороші рішення за прийнятний час, ефективно досліджуючи величезні простори пошуку.

На відміну від класичних задач оптимізації, де фітнес-функція є неперервною і диференційованою, ландшафт пошуку зворотних схем є дискретним і "різким". Невелика зміна в структурі схеми (наприклад, видалення

одного вентиля) може кардинально змінити реалізовану функцію, роблячи її повністю непридатною.

Тому еволюційні методи для цієї задачі адаптуються за трьома напрямками:

1. Кодування: Схема подається не як бітовий рядок, а як упорядкований список вентилів або інструкцій (хромосома змінної довжини).
2. Фітнес-функція – Fitness Function: Вона є складною і багатокритеріальною, враховуючи не лише коректність таблиці істинності аналізовану за допомогою ваги Геммінга різниці векторів, але й квантову вартість схеми, кількість вентилів та надлишкових виходів.
3. Оператори пошуку: Використовуються специфічні мутації, що відповідають алгебраїчним властивостям групи перестановок.

Для синтезу зворотних структур найбільш широко застосовуються три класи метаевристик, які розрізняються стратегією дослідження простору рішень:

1. Алгоритми на основі траєкторії: Працюють з одним рішенням, намагаючись покращити його на кожному кроці. Представником якого є алгоритм симульованого відпалу.
2. Популяційні алгоритми: Оперують множиною рішень – популяцією, які схрещуються та конкурують між собою. Це дозволяє підтримувати різноманітність і уникати застрягання в локальних екстремумах. Представником є – генетичні алгоритми (GA) та генетичне програмування.
3. Алгоритми ройового інтелекту: Моделюють колективну поведінку агентів (мурах, бджіл, часток), які обмінюються інформацією про знайдені перспективні ділянки простору пошуку. Представником наступних є: метод рою часток (PSO) та мурашині алгоритми (ACO) [91, 92].

У подальших пунктах буде проведено детальний аналіз кожного з цих підходів з точки зору їх ефективності для мінімізації квантової вартості та здатності до реконфігурації схем.

2.4.1. Метод симульованого відпалу: переваги та недоліки в задачах оптимального синтезу

Метод симульованого відпалу (Simulated Annealing — SA) — це стохастичний алгоритм оптимізації, натхненний фізичним процесом відпалу в металургії, де матеріал нагрівають, а потім повільно охолоджують для формування кристалічної решітки з мінімальною енергією – без дефектів.

У контексті синтезу зворотних схем цей метод інтерпретується наступним чином:

1. Стан системи – поточна послідовність вентилів логічної схеми.
2. Енергія E: Значення фітнес-функції, яка зазвичай є сумою відстані Геммінга – помилки таблиці істинності та квантової вартості схеми.
3. Температура T: Керуючий параметр, що визначає ймовірність прийняття погіршуючих рішень.

Принцип роботи алгоритму [57]:

Алгоритм працює ітеративно з одним поточним рішенням. На кожному кроці відбувається випадкова модифікація схеми – мутація: додавання, видалення або заміна вентиля.

Ключовою особливістю SA є механізм уникнення локальних екстремумів. Якщо нове рішення краще за поточне $\Delta E < 0$, воно приймається безумовно. Якщо ж нове рішення гірше $\Delta E > 0$, воно все одно може бути прийняте з ймовірністю P, що залежить від поточної температури:

$$P(\Delta E, T) = e^{-\frac{\Delta E}{T}} \qquad (2.4.1.1)$$

На початку процесу (висока температура) алгоритм часто приймає рішення так як температура висока, активно досліджуючи простір і часто приймаючи погані рішення, щоб вистрибнути з локальних ям. З часом температура знижується за певним законом, і алгоритм переходить до жадібної стратегії, затримуючись у найближчому мінімумі.

Переваги SA:

- Простота реалізації – алгоритм не вимагає складних структур даних. Він оперує лише однією схемою, що робить його вкрай економним з точки зору використання пам'яті, на відміну від генетичних алгоритмів, що зберігають великі популяції.
- Гнучкість - метод легко адаптується до різних фітнес-функцій, можна оптимізувати глибину, вартість або кількість надлишкових виходів, просто змінивши формулу розрахунку енергії.
- Ефективність для локальної оптимізації – SA показує відмінні результати як метод дооптимізації. Якщо взяти вже працюючу схему, наприклад, згенеровану методом MMD, і застосувати до неї SA з низькою температурою, можна суттєво зменшити її квантову вартість.

Попри свою популярність, метод симульованого відпалу має критичні недоліки при синтезі складних зворотних схем "з нуля":

- Повільна збіжність – для знаходження глобального оптимуму теоретично необхідне нескінченно повільне охолодження. На практиці це виливається у величезну кількість ітерацій, що робить синтез схем великої розмірності ($n > 10$) часомістким.
- Чутливість до параметрів – ефективність методу критично залежить від графіку охолодження (початкова температура, коефіцієнт зниження). Неправильний вибір параметрів призводить або до "застрягання" в локальному мінімумі, якщо охолодження занадто швидке, або до марного блукання – якщо занадто повільне.

- Траєкторний характер пошуку – оскільки алгоритм працює лише з одним рішенням, він не використовує інформацію про інші перспективні області простору пошуку. Це обмежує його здатність знаходити принципово нові структурні рішення порівняно з популяційними методами.

Отже метод SA є потужним інструментом для спрощення вже існуючих схем. Однак для задачі синтезу він поступається еволюційним алгоритмам у швидкості та здатності знаходити глобальні оптимуми у складних ландшафтах пошуку.

2.4.2. Алгоритми мурашиних колоній: аналіз ефективності феромонних моделей для дискретної оптимізації

Алгоритми мурашиних колоній [59] (Ant Colony Optimization — ACO) належать до класу методів ройового інтелекту [93, 94] і базуються на імітації харчової поведінки реальних мурах. Ключовою особливістю цього підходу є використання стигмергії — непрямої взаємодії агентів через зміни у навколишньому середовищі.

У задачах дискретної оптимізації, до яких належить синтез логічних схем, роль середовища виконує граф пошуку, а роль комунікаційного сигналу — віртуальний феромон.

*Принцип роботи феромонної моделі.*Задача синтезу моделюється як пошук найкоротшого шляху на графі, де:

- Вузли відповідають поточним станам таблиці істинності (або перестановки).
- Ребра відповідають застосуванню елементарного логічного вентиля (Toffoli, Fredkin тощо), що переводить систему з одного стану в інший.

Пошук рішення виконується популяцією штучних агентів "мурах". Кожен агент будує своє рішення ітеративно, додаючи вентилі у схему крок за кроком.

Вибір наступного вентиля відбувається імовірнісно, базуючись на двох факторах:

1. Феромонний слід (τ) – пам'ять колонії – показує, наскільки успішним був вибір цього вентиля іншими мурахами в минулому.
2. Евристична інформація (η) – жадібна оцінка – локальна метрика, наприклад, наскільки даний вентиль зменшує відстань Геммінга до фітнес-функції.

Ймовірність вибору j-го вентиля після і-го стану визначається формулою [59]:

$$P_{ij} = \frac{\left[\tau_{ij}\right]^{\alpha} \cdot \left[n_{ij}\right]^{\beta}}{\sum([\tau]^{\alpha} \cdot [n]^{\beta})}, \qquad (2.4.2.1)$$

де α та β — вагові коефіцієнти, що регулюють баланс між використанням накопиченого досвіду та евристикою.

Динаміка оновлення феромонів. Критично важливим елементом ACO є механізм оновлення феромонів, який забезпечує навчання системи:

- Підсилення: Після завершення ітерації мурахи, що знайшли коректні схеми, або схеми з найменшою помилкою, додають феромон на пройдений шлях. Кількість феромону обернено пропорційна вартості знайденої схеми $Q = 1/QC$. Це підвищує ймовірність того, що наступні покоління оберуть саме ці вентилі.
- Випаровування: На кожному кроці частина феромону на всіх ребрах зникає $\tau \leftarrow (1-\rho)\tau$. Це запобігає необмеженому накопиченню інтенсивності сліду та дозволяє колонії, забувати старі субоптимальні шляхи, уникаючи передчасної збіжності в локальних мінімумах.

Перевагами ACO є:

1. Конструктивна природа – ACO ідеально підходить для задач, де рішенням є впорядкована послідовність елементів. Мураха природним чином формує схему вентиль за вентилем.
2. Пошук найкоротших шляхів – оскільки алгоритм максимізує щільність феромону на коротких шляхах, він природним чином схильний до мінімізації кількості вентилів та квантової вартості.

Недоліки виявляються в наступному:

1. Висока обчислювальна складність – на кожному кроці кожна мураха повинна переоцінювати ймовірності для великої кількості можливих кандидатів вентилів. Для схем з великою кількістю змінних $n > 10$ граф стає занадто розгалуженим.
2. Проблема стагнації – існує ризик, що всі мурахи почнуть ходити однією й тією ж стежкою (субоптимальною), якщо початкове накопичення феромону на ній було занадто швидким.

Як бачимо з описаного ACO демонструють високу ефективність у знаходженні схем з мінімальною глибиною. Однак через високі вимоги до обчислень на кожній ітерації, вони часто використовуються в гібридних підходах або для задач середньої розмірності. Для масштабування на більші схеми більш перспективними виглядають генетичні алгоритми, які ми розглянемо далі.

2.4.3. Генетичні алгоритми: аналіз існуючих рішень та джерел

Генетичні алгоритми – GA на сьогодні є найбільш дослідженим класом метаевристик для синтезу зворотних схем [58, 95]. Це обумовлено їхньою здатністю працювати з популяцією рішень одночасно, що забезпечує кращий баланс між розвідкою простору пошуку та експлуатацією знайдених перспективних областей порівняно з методами однієї точки (як SA).

Аналіз наукових джерел дозволяє виділити кілька домінуючих підходів до адаптації GA під специфіку квантових схем, а також ідентифікувати ключові проблеми, що залишаються невирішеними.

1. Проблема кодування хромосоми. У класичних GA хромосома — це бітовий рядок [96]. Однак для синтезу схем таке кодування є неефективним через складну структуру вентилів. В існуючих рішеннях переважно використовується пряме цілочисельне кодування. Хромосома являє собою масив структур, де кожен ген кодує один вентиль з параметрами: тип вентиля (NOT, CNOT, Toffoli, Fredkin), керуючі лінії, цільові лінії.

Недолік існуючих підходів: Більшість стандартних методів використовують хромосоми фіксованої довжини. Це змушує дослідників або задавати максимальну глибину схеми "із запасом" – що веде до появи надлишкових вентилів, які нічого не роблять, але споживають ресурси алгоритму, або запускати алгоритм багаторазово, поступово збільшуючи довжину, що є обчислювально неефективним.

2. *Проблема операторів кросоверу.* Оператор кросовера є основним рушієм еволюції в класичних ГА. Однак у синтезі зворотних схем його ефективність є предметом дискусій.

Зворотна схема — це не просто набір вентилів, а сувора послідовність. Вентиль, що стоїть на початку каскаду, впливає на всі наступні.

Руйнівний вплив: Стандартний одноточковий кросовер, обмін половинками схем між батьками, часто призводить до того, що нащадок втрачає корисні властивості обох батьків. Схема, яка була "майже правильною", після кросоверу з іншою схемою часто стає абсолютно непрацездатною – радикально змінюється таблиця істинності.

Через це в ряді сучасних робіт спостерігається тенденція до відмови від кросовера на користь стратегій, що базуються виключно на мутаціях (Mutation-based Evolution), або використання спеціалізованих "розумних" кросоверів, які обмінюються лише функціонально незалежними блоками.

3. *Ландшафт фітнес-функції.* Аналіз джерел [97] вказує на те, що фітнес-функції зазвичай будується як зважена сума:

$$Fitness = \omega_1 \cdot Error + \omega_2 \cdot Cost \qquad (2.4.3.1)$$

де Error — помилка реалізації функції (відстань Геммінга), а Cost — квантова вартість.

Проблема "плато" (Plateau Problem): Основна проблема існуючих підходів полягає в тому, що простір пошуку містить величезні "плато", де зміна одного вентиля не змінює відстань Геммінга, але змінює структуру схеми. Стандартні GA часто "блукають" на цих плато, не маючи градієнта для покращення. Це призводить до передчасної збіжності — алгоритм знаходить локальний екстремум і не може з нього вибратися.

4. *Проблема "роздування" коду*. Характерною вадою еволюційних методів, описаною в літературі, є неконтрольоване зростання розміру схеми. Алгоритму простіше додати 10 нових вентилів, щоб виправити одну помилку, ніж знайти елегантну заміну одного існуючого вентиля. В результаті, навіть якщо GA знаходить рішення, воно часто містить величезну кількість надлишкових елементів, наприклад, пари $G \cdot G^{-1}$, що вимагає застосування агресивних методів пост-оптимізації.

Як свідчить проведений аналіз, хоча існуючі реалізації генетичних алгоритмів демонструють здатність знаходити рішення для задач середньої складності, пряме перенесення класичних еволюційних операторів на квантову логіку є малоефективним. Для підвищення якості синтезу необхідна глибша інтеграція специфіки предметної області в структуру алгоритму. У даній роботі пропонується новий підхід, що базується на об'єктно-орієнтованому представленні схеми. На відміну від простих числових масивів, таке кодування дозволяє розглядати кожен вентиль як самостійний об'єкт із властивостями, що дає змогу розробити спеціалізовані генетичні оператори.

2.4.4. Відкриті проблеми існуючих метаевристичних підходів

Узагальнюючи проведений аналіз методів синтезу: MMD [85, 98], симуляція відпалу [57], мурашині [59, 93, 92] та генетичні алгоритми [70, 58, 96, 61] можна виділити ряд системних проблем, які залишаються відкритими і суттєво обмежують ефективність автоматизованого проєктування зворотних схем великої розмірності [106].

Ці проблеми можна класифікувати за чотирма основними напрямками:

1. Проблема ландшафту пошуку та "ефект плато". Фітнес-функції, що базується на відстані Геммінга, має суттєвий недолік — дискретність. У просторі пошуку існують великі області – плато, де зміна структури схеми – мутація вентиля не призводить до зміни значення відстані Геммінга. Тому класичні градієнтні та еволюційні методи на таких ділянках втрачають "напрямок" руху, оскільки не отримують зворотного зв'язку про те, чи покращило нове рішення ситуацію. Це призводить до марного блукання алгоритму і значних витрат процесорного часу без покращення результату.

2. Передчасна збіжність (Premature Convergence) [6]. Це типова проблема для генетичних алгоритмів при синтезі складних структур. Якщо в популяції з'являється одна схема, яка трохи краща за інші – локальний оптимум, вона швидко витісняє інші варіанти через механізм селекції.

Наслідком є – різноманітність популяції втрачається, і алгоритм застрягає в локальному екстремумі, не досягнувши глобально оптимального рішення. Існуючі методи підтримки різноманітності, наприклад, висока ймовірність мутації, часто діють деструктивно, руйнуючи вже знайдені правильні фрагменти схеми.

3. Ігнорування доменної специфіки в операторах. Більшість існуючих реалізацій використовують стандартні оператори кросоверу та мутації, запозичені з класичної теорії оптимізації числових функцій. "Сліпі" оператори

розглядають схему як набір чисел, не враховуючи алгебраїчні властивості зворотних вентилів. Як наслідок – значна частина обчислювальних ресурсів витрачається на генерацію та перевірку завідомо неефективних або абсурдних схем, наприклад, таких, що містять послідовності взаємно обернених вентилів, які потім доводиться видаляти на етапі пост-оптимізації.

4. Конфлікт критеріїв оптимізації. Синтез схеми є багатокритеріальною задачею: необхідно одночасно досягти коректності функціонування $Error = 0$ та мінімізувати вартість $Cost \rightarrow min$. Ці цілі часто суперечать одна одній. Зменшення кількості вентилів може призвести до порушення логіки роботи, а виправлення логічної помилки часто вимагає додавання нових вентилів. Існуючі методи часто зводять задачу до однокритеріальної (зваженої суми), що не дозволяє гнучко керувати процесом пошуку компромісних рішень.

Зазначені проблеми свідчать про те, що потенціал класичних метаевристик у задачі синтезу зворотних схем вичерпано. Подальший прогрес можливий лише шляхом розробки гібридних підходів та модифікації базових еволюційних операторів. Зокрема, перспективним є перехід до об'єктно-орієнтованого представлення, яке дозволить інтегрувати знання про властивості вентилів безпосередньо в механізм генерації рішень, що і є метою даної роботи.

Висновки до розділу 2

У другому розділі проведено комплексний аналіз теоретичних основ, математичних моделей та алгоритмічних підходів до синтезу та реконфігурації зворотних логічних схем. За результатами дослідження зроблено наступні висновки:

1. Базис синтезу: встановлено, що для побудови ефективних зворотних та відмовостійких схем найбільш доцільним є використання бібліотеки вентилів NCT для реалізації логічних функцій та вентилів Фредкіна для задач реконфігурації та маршрутизації сигналів. Вентилі Фредкіна,

завдяки властивості збереження парності, створюють передумови для апаратної діагностики помилок.

2. Критерії оптимізації: визначено, що основним критерієм ефективності синтезованих схем є квантова вартість – QC, яка відображає сумарну складність елементарних квантових операцій. Додатковими цільовими параметрами є мінімізація кількості вентилів та надлишкових виходів, що дозволяє знизити енергоспоживання та підвищити надійність пристрою.
3. Обмеження існуючих методів: аналіз точних алгоритмів синтезу, зокрема, MMD та методів на основі таблиць істинності, показав, що вони гарантують отримання робочої схеми, проте часто генерують надлишкові рішення, далекі від оптимальних. Крім того, через експоненційну складність вони є непридатними для схем практичної розмірності (n>4 змінних.
4. Проблеми класичних метаевристик: розгляд еволюційних методів симуляція відпалу, мурашині та генетичні алгоритми виявив, що їх «сліпе» застосування до задачі зворотного синтезу є малоефективним. Стандартні оператори кросоверу та мутації часто руйнують структуру блоків, а дискретність простору пошуку призводить до передчасної збіжності алгоритмів у локальних екстремумах.
5. Напрямок подальших досліджень: обґрунтовано необхідність розробки вдосконаленого методу еволюційного синтезу, який базується на векторному представленні станів системи та об'єктно-орієнтованій моделі схеми. Такий підхід дозволить:
    - Розробити спеціалізовані генетичні оператори такі як багатокомпонентна мутація, що враховують алгебраїчні властивості вентилів;
    - Реалізувати ефективну реконфігурацію на базі розширених вентилів Фредкіна;

- Забезпечити вихід із локальних екстремумів для досягнення глобального оптимуму.

Отримані висновки є підґрунтям для розробки власного методу автоматизованого синтезу та реконфігурації, опис якого наведено у наступному розділі.

РОЗДІЛ 3. РОЗРОБКА МЕТОДУ СИНТЕЗУ НОВИХ РЕКОНФІГУРОВАНИХ ПРИСТРОЇВ

Як було показано у попередньому розділі, класичні детерміновані алгоритми та стандартні метаевристики, такі як симульований відпал або класичні генетичні алгоритми, стикаються із суттєвими обмеженнями при синтезі зворотних схем великої розмірності. Основними проблемами є передчасна збіжність до локальних оптимумів, неконтрольоване зростання квантової вартості та ігнорування специфічних алгебраїчних властивостей зворотних вентилів під час генерації рішень.

Даний розділ присвячено розробці та опису нового методу автоматизованого синтезу, спрямованого на вирішення цих проблем. В основі запропонованого підходу лежить поєднання еволюційних обчислень із об'єктно-орієнтованим представленням структури схеми. Такий підхід дозволяє перейти від маніпулювання абстрактними числовими масивами до роботи з об'єктами, які інкапсулюють правила комутації та логіку функціонування квантових вентилів.

Ключовою метою розробки є створення ефективного інструментарію для синтезу зворотних схем та їх компонентів таких як реконфігуровані вентилі— універсальні логічні блоки, здатні змінювати свою функціональність під дією керуючих сигналів. Це вимагає не лише модифікації еволюційних операторів, але й перегляду базису елементарних елементів, що використовуються для побудови схем.

У цьому розділі послідовно розглядаються: математична модель розширеного базису елементів, архітектура запропонованого генетичного алгоритму, специфіка розроблених операторів мутації та кросоверу, а також деталі програмної реалізації системи синтезу.

3.1. Розширений базис реконфігурованих пристроїв

Традиційні підходи до синтезу часто жорстко прив'язані до конкретних бібліотек, наприклад, тільки NCT. Натомість, для ефективного дослідження реконфігурованих структур необхідний інструмент, який дозволяє гнучко варіювати набір будівельних блоків.

Основою методу є оперування векторами станів системи. Нехай схема має N ліній. Вхідний вплив описується вектором *Input*, що містить повну послідовність станів у лексикографічному порядку (від 0 до $2^N - 1$). Завдання полягає у знаходженні такої послідовності вентилів, яка трансформує вхідний вектор *Input* у заданий цільовий вектор *Target*:

$$Target = [y_0, y_1, y_2, \dots, y_{2^n-1}], \tag{3.1.1}$$

де y_i — очікуване двійкове значення виходу схеми, що відповідає і-му вхідному набору.

Пошук рішення виконується розробленим модифікованим генетичним алгоритмом [100]. Алгоритм не здійснює повний перебір, а використовує еволюційну стратегію:

1. Генерує популяцію схем-кандидатів із доступних у бібліотеці вентилів.
2. Оцінює кожну схему за відхиленням її вихідного вектора від *Target* – компонент фітнес-функція.
3. Виконує селекцію найперспективніших рішень.
4. Застосовує оператори мутації та кросоверу для мінімізації помилки та вартості схеми.
5. Перевіряється критерій зупинки і повторюються кроки 2 – 5.

Важливою відмінністю розробленого методу є підтримка динамічної бібліотеки примітивів Ω включаючи розширені вентилі Фредкіна [43]. Користувач може визначати склад цієї бібліотеки перед запуском процесу синтезу, що дозволяє проводити різні типи експериментів:

1. Однорідні базиси: Синтез схем виключно на основі вентилів Фредкіна $\Omega = \{CSWAP\}$ або тільки на основі CNOT $\Omega=\{CNOT\}$.
2. Класичні повні базиси: Використання бібліотеки NCT $\Omega=\{NOT,CNOT,Toffoli\}$.
3. Гібридні базиси: Поєднання комутаційних та логічних елементів $\Omega=\{NOT,CNOT,Toffoli,CSWAP\}$.

Варто окремо відзначити структурний взаємозв'язок між цими базисами. Як відомо, комутаційний вентиль Фредкіна не є елементарним з точки зору апаратної реалізації і може бути декомпозований у класичному базисі NCT. На рівні квантової або зворотної логічної схеми його еквівалентне представлення складається з послідовного каскаду трьох вентилів: вхідного вентиля CNOT, центрального вентиля Тоффолі та вихідного вентиля CNOT.

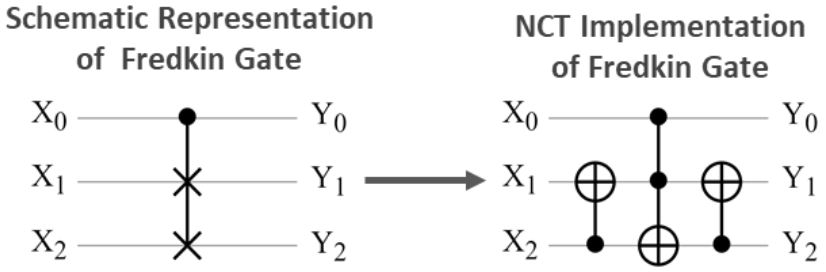


Рис. 3.1.1. Еквівалентна декомпозиція вентиля Фредкіна в базисі NCT.

Саме завдяки підтримці динамічних бібліотек розроблений інструмент дозволяє системно аналізувати подібні структурні альтернативи. Такий підхід дозволяє емпірично визначити, який саме набір елементів є найбільш ефективним (з точки зору вартості та кількості вентилів) для реалізації конкретних реконфігурованих компонентів, що розглядаються в даному розділі.

3.1.1. Структурна реконфігурація на основі узагальненого вентиля Фредкіна

У рамках розробленого методу ключовим елементом для побудови реконфігурованих структур обрано вентиль Фредкіна. Цей вибір обумовлений фундаментальною відмінністю в логіці його роботи порівняно з класичними елементами базису NCT. Якщо вентилі Тоффолі реалізують алгебраїчні операції над бітами (інверсія, додавання за модулем 2), то вентиль Фредкіна реалізує логіку маршрутизації, що є природною основою для побудови перемикальних схем.

Вентиль Фредкіна діє у просторі станів трьох кубітів. Його оператор F може бути представлений як унітарна матриця перестановки розміром 8×8:

$$F = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad (3.1.1.1)$$

З матричного представлення видно, що вентиль діє як тотожне перетворення для всіх станів, окрім випадків, де перший – керуючий, біт дорівнює 1. У цих випадках відбувається обмін амплітуд, або значень, між станами $|101\rangle \leftrightarrow |110\rangle$. Це підтверджує його роль як керованого комутатора, який не змінює інформаційний зміст сигналів, а лише їх фізичне розташування.

Суть запропонованого підходу полягає у трактуванні вентиля Фредкіна як зворотного аналога класичного мультиплексора 2-в-1 (2:1 MUX). Аналітично вихідні сигнали на інформаційних лініях описуються рівняннями:

$$\begin{cases} Out_1 = \overline{c} \cdot x_1 \oplus \mathrm{c} \cdot x_2 \\ Out_2 = c \cdot x_1 \oplus \overline{c} \cdot x_2 \end{cases}, \quad (3.1.1.2)$$

де c — сигнал на керуючій лінії, x_1, x_2 — вхідні інформаційні сигнали.

Якщо розглядати лінію c як конфігураційний вхід, то вихід Out_2 фактично обирає функцію: при $c = 0$ схема прозоро пропускає сигнал x_1, а при $c = 1$ — перемикається на сигнал x_2. Ця властивість дозволяє реалізувати динамічну зміну логіки роботи пристрою без зміни його фізичної топології, що є основою для FPGA-подібних квантових структур [89].

Для формалізації задачі синтезу в нашому алгоритмі вводиться суворе розмежування вхідних змінних на два класи:

- Інформаційні змінні $X = \{x_1, \dots, x_n\}$ – вектор даних, над якими виконуються обчислення. Ці змінні проходять крізь каскад вентилів, змінюючи свої позиції або значення.
- Конфігураційні змінні $P = \{p_1, \dots, p_k\}$ – вектор параметрів, що визначає поточну функцію схеми. У запропонованому підході ці змінні використовуються переважно як керуючі для вентилів Фредкіна, визначаючи шляхи проходження інформаційних сигналів.

Використання вентиля Фредкіна як базового примітиву для реконфігурації має ряд переваг перед класичним базисом NCT:

1. Збереження парності – кількість одиниць вхідного вектора дорівнює кількості одиниць вихідного. Це спрощує фізичну реалізацію в технологіях, де енергія залежить від кількості збуджених станів.
2. Зменшення глибини схеми – реалізація функції перемикання (MUX) в базисі вентилів Тоффолі вимагає мінімум 3 вентилів та додаткових надлишкових ліній. Вентиль Фредкіна виконує цю операцію за 1 такт, що суттєво зменшує обчислювальну складність та час синтезу, а також квантову вартість синтезованих реконфігурованих блоків.

3. Універсальність – комбінуючи мультиплексорні властивості вентилю Фредкіна з фіксованими константами 0 та 1, можна отримати повний спектр булевих функцій, включаючи нелінійні AND та OR, що детально розглядається в наступних підрозділах.

3.1.2. Синтез нових реконфігураційних функцій

Базуючись на властивостях вентиля Фредкіна як керованого комутатора, у роботі ставиться завдання синтезу класу нових логічних компонентів — універсальних реконфігурованих блоків RRG — Reversible reconfigurable gates.

Однак, використання виключно класичного вентиля Фредкіна, з позитивним керуванням та прямими цільовими виходами, суттєво обмежує простір пошуку рішень. Для забезпечення максимальної гнучкості синтезу, в рамках даного дослідження класичний вентиль Фредкіна був істотно розширений. Завдяки комбінуванню прямої та інверсної полярності на керуючій лінії, а також застосуванню інверторів на цільових інформаційних лініях, було синтезовано узагальнене сімейство розширених вентилів Фредкіна – Extended Fredkin Gates.

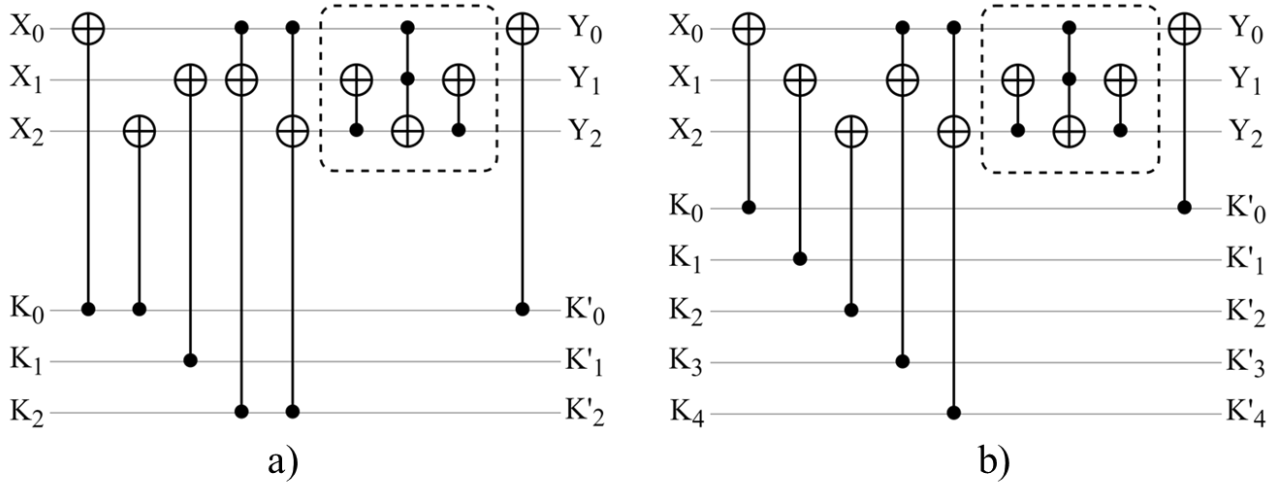


Рис. 3.1.2.1. Варіанти схемотехнічної реалізації універсального RRG на основі вентилів Фредкіна.

Таблиця 3.1.2.1 — Повний базис розширених реконфігурованих вентилів Фредкіна в базисі RRG (b) залежно від ключів активації.

K_0	K_1	K_2	K_3	K_4	Функція	K_0	K_1	K_2	K_3	K_4	Функція
0	0	0	0	0	$Y_1 = \overline{X_0}\,X_1 \oplus X_0X_2$ $Y_2 = \overline{X_0}\,X_2 \oplus X_0X_1$	0	0	0	0	1	$Y_1 = \overline{X_0}\,X_1 \oplus X_0\overline{X_2}$ $Y_2 = \overline{X_0}\,X_2 \oplus X_0X_1$
1	0	0	0	0	$Y_1 = \overline{X_0}\,X_2 \oplus X_0X_1$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0X_2$	1	0	0	0	1	$Y_1 = \overline{X_0}\,\overline{X_2} \oplus X_0X_1$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0X_2$
0	1	0	0	0	$Y_1 = \overline{X_0}\,\overline{X_1} \oplus X_0X_2$ $Y_2 = \overline{X_0}\,X_2 \oplus X_0\overline{X_1}$	0	1	0	0	1	$Y_1 = \overline{X_0}\,\overline{X_1} \oplus X_0\overline{X_2}$ $Y_2 = \overline{X_0}\,X_2 \oplus X_0\overline{X_1}$
1	1	0	0	0	$Y_1 = \overline{X_0}\,X_2 \oplus X_0\overline{X_1}$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0X_2$	1	1	0	0	1	$Y_1 = \overline{X_0}\,\overline{X_2} \oplus X_0\overline{X_1}$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0X_2$
0	0	1	0	0	$Y_1 = \overline{X_0}\,X_1 \oplus X_0\overline{X_2}$ $Y_2 = \overline{X_0}\,\overline{X_2} \oplus X_0X_1$	0	0	1	0	1	$Y_1 = \overline{X_0}\,X_1 \oplus X_0X_2$ $Y_2 = \overline{X_0}\,\overline{X_2} \oplus X_0X_1$
1	0	1	0	0	$Y_1 = \overline{X_0}\,\overline{X_2} \oplus X_0X_1$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0\overline{X_2}$	1	0	1	0	1	$Y_1 = \overline{X_0}\,X_2 \oplus X_0X_1$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0\overline{X_2}$
0	1	1	0	0	$Y_1 = \overline{X_0}\,\overline{X_1} \oplus X_0\overline{X_2}$ $Y_2 = \overline{X_0}\,\overline{X_2} \oplus X_0\overline{X_1}$	0	1	1	0	1	$Y_1 = \overline{X_0}\,\overline{X_1} \oplus X_0X_2$ $Y_2 = \overline{X_0}\,\overline{X_2} \oplus X_0\overline{X_1}$
1	1	1	0	0	$Y_1 = \overline{X_0}\,\overline{X_2} \oplus X_0\overline{X_1}$ $Y_2 = \overline{X_0}\,\overline{X_1} \oplus X_0\overline{X_2}$	1	1	1	0	1	$Y_1 = \overline{X_0}\,X_2 \oplus X_0\overline{X_1}$ $Y_2 = \overline{X_0}\,\overline{X_1} \oplus X_0\overline{X_2}$
0	0	0	1	0	$Y_1 = \overline{X_0}\,X_1 \oplus X_0X_2$ $Y_2 = \overline{X_0}\,X_2 \oplus X_0\overline{X_1}$	0	0	0	1	1	$Y_1 = \overline{X_0}\,X_1 \oplus X_0\overline{X_2}$ $Y_2 = \overline{X_0}\,X_2 \oplus X_0\overline{X_1}$
1	0	0	1	0	$Y_1 = \overline{X_0}\,X_2 \oplus X_0X_1$ $Y_2 = \overline{X_0}\,\overline{X_1} \oplus X_0X_2$	1	0	0	1	1	$Y_1 = \overline{X_0}\,\overline{X_2} \oplus X_0X_1$ $Y_2 = \overline{X_0}\,\overline{X_1} \oplus X_0X_2$
0	1	0	1	0	$Y_1 = \overline{X_0}\,\overline{X_1} \oplus X_0X_2$ $Y_2 = \overline{X_0}\,X_2 \oplus X_0X_1$	0	1	0	1	1	$Y_1 = \overline{X_0}\,\overline{X_1} \oplus X_0\overline{X_2}$ $Y_2 = \overline{X_0}\,X_2 \oplus X_0X_1$
1	1	0	1	0	$Y_1 = \overline{X_0}\,X_2 \oplus X_0\overline{X_1}$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0X_2$	1	1	0	1	1	$Y_1 = \overline{X_0}\,\overline{X_2} \oplus X_0\overline{X_1}$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0X_2$
0	0	1	1	0	$Y_1 = \overline{X_0}\,X_1 \oplus X_0\overline{X_2}$ $Y_2 = \overline{X_0}\,\overline{X_2} \oplus X_0\overline{X_1}$	0	0	1	1	1	$Y_1 = \overline{X_0}\,X_1 \oplus X_0X_2$ $Y_2 = \overline{X_0}\,\overline{X_2} \oplus X_0\overline{X_1}$
1	0	1	1	0	$Y_1 = \overline{X_0}\,\overline{X_2} \oplus X_0X_1$ $Y_2 = \overline{X_0}\,\overline{X_1} \oplus X_0\overline{X_2}$	1	0	1	1	1	$Y_1 = \overline{X_0}\,X_2 \oplus X_0X_1$ $Y_2 = \overline{X_0}\,\overline{X_1} \oplus X_0\overline{X_2}$
0	1	1	1	0	$Y_1 = \overline{X_0}\,\overline{X_1} \oplus X_0\overline{X_2}$ $Y_2 = \overline{X_0}\,\overline{X_2} \oplus X_0X_1$	0	1	1	1	1	$Y_1 = \overline{X_0}\,\overline{X_1} \oplus X_0X_2$ $Y_2 = \overline{X_0}\,\overline{X_2} \oplus X_0X_1$
1	1	1	1	0	$Y_1 = \overline{X_0}\,\overline{X_2} \oplus X_0\overline{X_1}$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0\overline{X_2}$	1	1	1	1	1	$Y_1 = \overline{X_0}\,X_2 \oplus X_0\overline{X_1}$ $Y_2 = \overline{X_0}\,X_1 \oplus X_0\overline{X_2}$

На рисунку 3.1.2.1 наведено приклади двох варіантів синтезованої базової структури реконфігуровного блоку RRG. Наведені схеми наочно ілюструють принцип розподілу ліній на інформаційні X_i та конфігураційні K_i.

Завдяки маршрутизаційним властивостям вентилів Фредкіна, зміна логічних рівнів на конфігураційних входах дозволяє динамічно перемикати активні шляхи проходження сигналів. Таким чином, представлені структури здатні виконувати різні логічні функції без фізичної зміни своєї топології, що є концептуальною основою для побудови реконфігурованих обчислювальних середовищ.

Загалом простір станів для 3-бітного реконфігуровного вентиля Фредкіна в базисі NCT охоплює 32 унікальні топології. З них 8 конфігурацій є відомими базовими розширеннями [3], а 24 конфігурації були вперше синтезовані та верифіковані під час виконання роботи. У Таблиці 3.1.2.1 наведено повний перелік отриманих логічних функцій та відповідних їм топологій узагальненого вентиля Фредкіна. Наявність такого багатого алфавіту примітивів (32 функції замість 1) кардинально змінює підхід до проєктування вищого рівня. Традиційний синтез оперує однією таблицею істинності. Натомість, синтез RRG вимагає одночасного виконання кількох різних таблиць істинності в межах однієї фізичної структури, де вибір активної таблиці здійснюється станом керуючих входів.

Нехай необхідно синтезувати *N*-входовий компонент, який залежно від k керуючих сигналів $P = \{p_1, \dots, p_k\}$ реалізує одну з $M = 2^k$ функцій із заданого набору $F = \{f_0, f_1, \dots, f_{M-1}\}$. Наведена Таблиця. 3.1.2.2 наочно демонструє механізм обчислення цільового вектора на прикладі синтезу реконфігуровного елемента AND/OR. Загальний вектор станів алгоритму утворюється шляхом послідовної конкатенації двох незалежних таблиць істинності. Керуючий сигнал p у цьому випадку виступає старшим бітом вхідного слова, розділяючи загальний простір пошуку на дві підмножини. Верхня половина таблиці $p = 0$

кодує цільові значення AND, тоді як нижня половина $p = 1$ – OR. Отриманий результуючий вектор T_{rrg}, для даного прикладу – $[0,0,0,1,0,1,1,1]$ стає єдиним еталонним критерієм для обчислення фітнес-функції під час роботи генетичного алгоритму.

Таблиця 3.1.2.2. Формування цільового вектора для реконфігуровного елемента AND/OR

Керуючий сигнал	Вхід		Режим роботи	Цільовий вихід
p	x_1	x_2	AND/OR	Y
0	0	0	$x_1 \cdot x_2$	0
0	0	1	$x_1 \cdot x_2$	0
0	1	0	$x_1 \cdot x_2$	0
0	1	1	$x_1 \cdot x_2$	1
1	0	0	$x_1 \vee x_2$	0
1	0	1	$x_1 \vee x_2$	1
1	1	0	$x_1 \vee x_2$	1
1	1	1	$x_1 \vee x_2$	1

Цільовий вектор T_{rrg} формується як конкатенація векторів значень окремих функцій. Якщо вектори функцій f_i мають довжину L, то загальний вектор має довжину $M \times L$. Формально це описується виразом:

$$T_{rrg} = \bigoplus_{i=0}^{2^k-1} \big(V(f_i) \ll (i \cdot L)\big), \qquad (3.1.2.1)$$

де $V(f_i)$ — вектор значень i-ї функції, а операція зсуву ($\ll$) та суми символізують послідовне розташування сегментів у загальному векторі станів.

У ході досліджень за допомогою розробленого генетичного алгоритму було виділено та синтезовано три ключові класи реконфігурованих функцій, які мають найбільшу практичну цінність для побудови адаптивної логіки.

1. Логічно-комлементарні функції - це найпростіший клас блоків, де один керуючий біт інвертує логіку роботи або змінює тип операції на дуальну. Наприклад блок AND/OR. При $p = 0$ схема працює як логічне "І" ($x_1 \cdot x_2$), при $p = 1$ — як логічне "АБО" ($x_1 \vee x_2$). Це дозволяє динамічно змінювати архітектуру з кон'юнктивної нормальної форми на диз'юнктивну без перебудови схеми.

2. Арифметико-логічні функції – цей клас поєднує арифметичні операції з логічними, що є критично важливим для створення реконфігурованих арифметико-логічних пристроїв. Наприклад блок XOR/AND. При $p = 0$ виконується додавання за модулем 2 ($x_1 \oplus x_2$), при $p = 1$ — множення ($x_1 \cdot x_2$). Такий блок фактично є сплячим напівсуматором, який може бути активований або переведений у режим простої логіки.

3. Повні універсальні функції – найскладніший клас, метою якого є реалізація всіх 16 можливих двомісних булевих функцій в одному блоці. Математично функція такого блоку описується як:

$$Y = F(x_1, x_2, p_0, p_1, p_2, p_3), \qquad (3.1.2.2)$$

Для реалізації такого універсального модуля генетичний алгоритм синтезує структуру, яка діє як мультиплексор 4-в-1, що обирає значення з таблиці істинності, закодованої на входах конфігурації. Використання вентилів Фредкіна для цього класу задач показало значне зменшення кількості вентилів порівняно з класичними реалізаціями в базисі NCT, оскільки вентиль Фредкіна природно реалізує деревоподібну структуру мультиплексора.

Застосування розробленого методу та розширеного базису дозволило отримати нові схемні реалізації для зазначених класів функцій. Аналіз синтезованих структур демонструє:

1. Ефективність використання ліній – синтезовані блоки часто використовують надлишкові лінії як додатковий ресурс для маршрутизації, зменшуючи загальну ширину схеми.
2. Зменшення квантової вартості – для функції типу AND/OR вдалося знайти рішення з вартістю, що на 15-20% менша за теоретичні аналоги [43], побудовані з окремих блоків, значною мірою саме завдяки алгоритмічному підбору оптимальної конфігурації з 32 можливих варіантів вентиля Фредкіна.
3. Масштабованість – підхід дозволяє синтезувати не лише двомісні, а й тримісні реконфігуровні функції, наприклад Majority/Parity, лише змінюючи цільовий вектор Target.

3.1.3. Математичне моделювання керування логікою синтезованих вентилів

Після синтезу структури реконфігуровного елемента за допомогою генетичного алгоритму виникає необхідність формального опису його поведінки. Математичне моделювання керування дозволяє верифікувати коректність перемикання функцій та оцінити надійність синтезованого рішення.

У даній роботі запропоновано модель, що базується на декомпозиції Шеннона, адаптованій для зворотних логічних схем. Це дозволяє представити багатофункціональний вентиль як суперпозицію базових функцій, що активуються відповідними керуючими коефіцієнтами.

Нехай S — синтезована схема, що має набір інформаційних входів X та набір керуючих входів P. Вихідна функція схеми $Y(X, P)$ може бути представлена у вигляді розкладу за змінними керування. Для випадку з одним керуючим входом p, наприклад, для перемикання між AND/OR, функція описується рівнянням:

$$Y(X,p) = \bar{p} \cdot f_0(X) \oplus p \cdot f_1(X), \qquad (3.1.3.1)$$

де $f_0(X)$ — функція, що виконується при $p = 0$, а $f_1(X)$ — при $p = 1$. Вентиль Фредкіна, використаний як базис, фізично реалізує саме цю математичну залежність, де додавання за модулем 2 ($\oplus$) та множення ($\cdot$) відповідають логіці маршрутизації сигналу.

Узагальнена модель для k керуючих сигналів. Для складних універсальних блоків RRG, що керуються вектором $P = \{p_1, \dots, p_k\}$, математична модель узагальнюється через мінтерми керуючих змінних. Будь-яка реконфігуровна функція може бути записана як:

$$Y(X,P) = \bigoplus_{i=0}^{2^k-1} \big(m_i(P) \cdot f_i(X)\big), \qquad (3.1.3.2)$$

де $m_i(P)$ — і-й мінтерм (кон'юнкція) керуючих змінних, який дорівнює 1 тільки для однієї конкретної комбінації керуючих сигналів. $f_i(X)$ — цільова логічна функція для і-го режиму роботи.

Оскільки генетичний алгоритм працює з векторами перестановок, доцільно розглянути матричну модель керування. Синтезована схема представляється унітарною матрицею U розміром $2^{\mathrm{N+k}} \times 2^{\mathrm{N+k}}$. Вплив керування на логіку схеми можна описати як проекцію великої матриці U на підпростори, визначені станами керуючих ліній $|P\rangle$.

Якщо зафіксувати керуючі входи у стані $|k\rangle$, то ефективна матриця перетворення інформаційних сигналів $U_{eff}^{(k)}$ визначається як:

$$U_{eff}^{(k)} = \langle k|_p \cdot U \cdot |k\rangle_p$$

Цей вираз показує, що фізична схема U залишається незмінною, але з точки зору інформаційних потоків вона трансформується у зовсім інший логічний оператор U_{eff} в залежності від проєктора $|k\rangle$.

При математичному моделюванні синтезованих вентилів вводяться критерії оцінки якості керування, які перевіряються на етапі роботи фітнес-функції алгоритму:

1. *Ізольованість режимів* – зміна інформаційних сигналів X при фіксованому керуванні P_i не повинна призводити до появи артефактів, властивих функції P_j $(i \neq j)$. Математично це означає, що перехресні члени в розкладі повинні тотожно дорівнювати нулю.

2. *Повнота покриття* – для кожного можливого стану керуючого вектора P схема повинна реалізовувати визначену детерміновану функцію $f(X)$, уникаючи невизначених станів, коли вихід залежить від надлишкових входів.

3. *Збереження глобальної зворотності* – специфікою реконфігурованих елементів є те, що проекція схеми на конкретний керуючий стан може реалізовувати незворотну функцію, наприклад, логічне множення AND. Однак математична модель вимагає, щоб загальна матриця оператора U залишалася унітарною. Це досягається шляхом коректного врахування надлишкових виходів, які акумулюють інформацію, необхідну для забезпечення бієктивності відображення у всьому просторі станів $2^{\mathrm{N+k}}$.

Таким чином, запропонована математична модель дозволяє формалізувати задачу синтезу реконфігурованих схем як задачу пошуку унітарного оператора, що задовольняє системі матричних рівнянь для заданого набору проєкторів керування. Наведені аналітичні залежності слугують теоретичним фундаментом для побудови фітнес-функції, яка використовується в генетичному алгоритмі для оцінки коректності та якості синтезованих рішень. Детальний опис архітектури та програмної реалізації цього алгоритму наведено у наступному підрозділі.

3.2. Вдосконалений генетичний метод синтезу зворотних схем

Як було обґрунтовано в попередньому розділі, задача побудови реконфігурованих компонентів в базисі вентиля Фредкіна зводиться до пошуку оптимальної топології схеми, що задовольняє складним багатовимірним векторам істинності. Оскільки простір можливих комбінацій вентилів зростає експоненційно зі збільшенням кількості ліній та глибини схеми, задача синтезу мінімальної зворотної схеми належить до класу NP-складних задач комбінаторної оптимізації.

Застосування класичних детермінованих методів або повного перебору є неефективним для схем розмірністю більше 4-5 змінних. У зв’язку з цим у даній роботі розроблено та застосовано вдосконалений генетичний алгоритм, адаптований до специфіки зворотної логіки.

На відміну від стандартних реалізацій генетичних алгоритмів, запропонований метод, реалізований у програмному комплексі, інтегрує низку модифікацій, спрямованих на подолання проблеми передчасної збіжності та підвищення якості синтезу:

1. *Об'єктно-орієнтований підхід до кодування* – замість традиційного одновимірного бітового рядка використовується складна об'єктна модель, де хромосома представлена як екземпляр класу схеми з динамічним набором вентилів. Це дозволяє оперувати безпосередньо топологією пристрою, усуває необхідність етапу декодування генотипу у фенотип та дозволяє застосовувати матричне представлення для розміщення вентилів.
2. *Інтеграція Табу-пошуку* [46] – для уникнення, застрягання або стагнації, алгоритму в локальних екстремумах впроваджено механізм контролю різноманітності популяції. Алгоритм відстежує відсоток ідентичних хромосом і при перевищенні порогового значення (10–12%) обмежує додавання дублікатів або ініціює часткове оновлення популяції.

3. Багатокомпонентна мутація: Замість однієї операції зміни біта, алгоритм використовує набір ймовірнісних операторів – P_{ms}, P_{ma}, P_{mr} які дозволяють змінювати конфігурацію ліній, додавати нові вентилі у вільні слоти або видаляти зайві елементи для мінімізації вартості схеми.

Головною метою розробленого методу є автоматичний синтез схеми, яка:

- Реалізує задану цільову функцію, або набір функцій для реконфігурованих блоків.
- Мінімізує квантову вартість QC — сумарну складність використаних вентилів.
- Мінімізує кількість вентилів GC та глибину схеми.
- Ефективно використовує надлишкові виходи та допоміжні входи, необхідні для забезпечення зворотності.

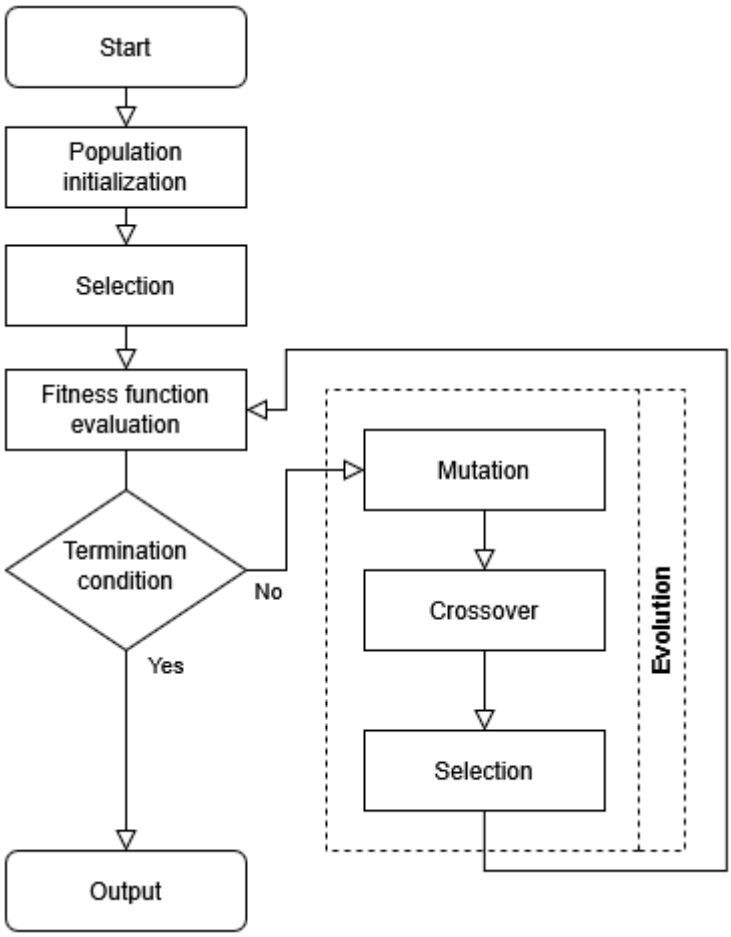


Рис 3.2.1 Блок схема генетичного алгоритму

Такий підхід дозволяє синтезувати як класичні структури в базисі NCT, так і запропоновані в даній роботі гібридні структури з використанням вентилів Фредкіна. Блок схему алгоритму представлено на рисунку 3.2.1

3.2.1. Об'єктно-орієнтована модель представлення хромосоми

Рисунок ілюструє процес відображення кожної лінії схеми на окремий програмний об'єкт.

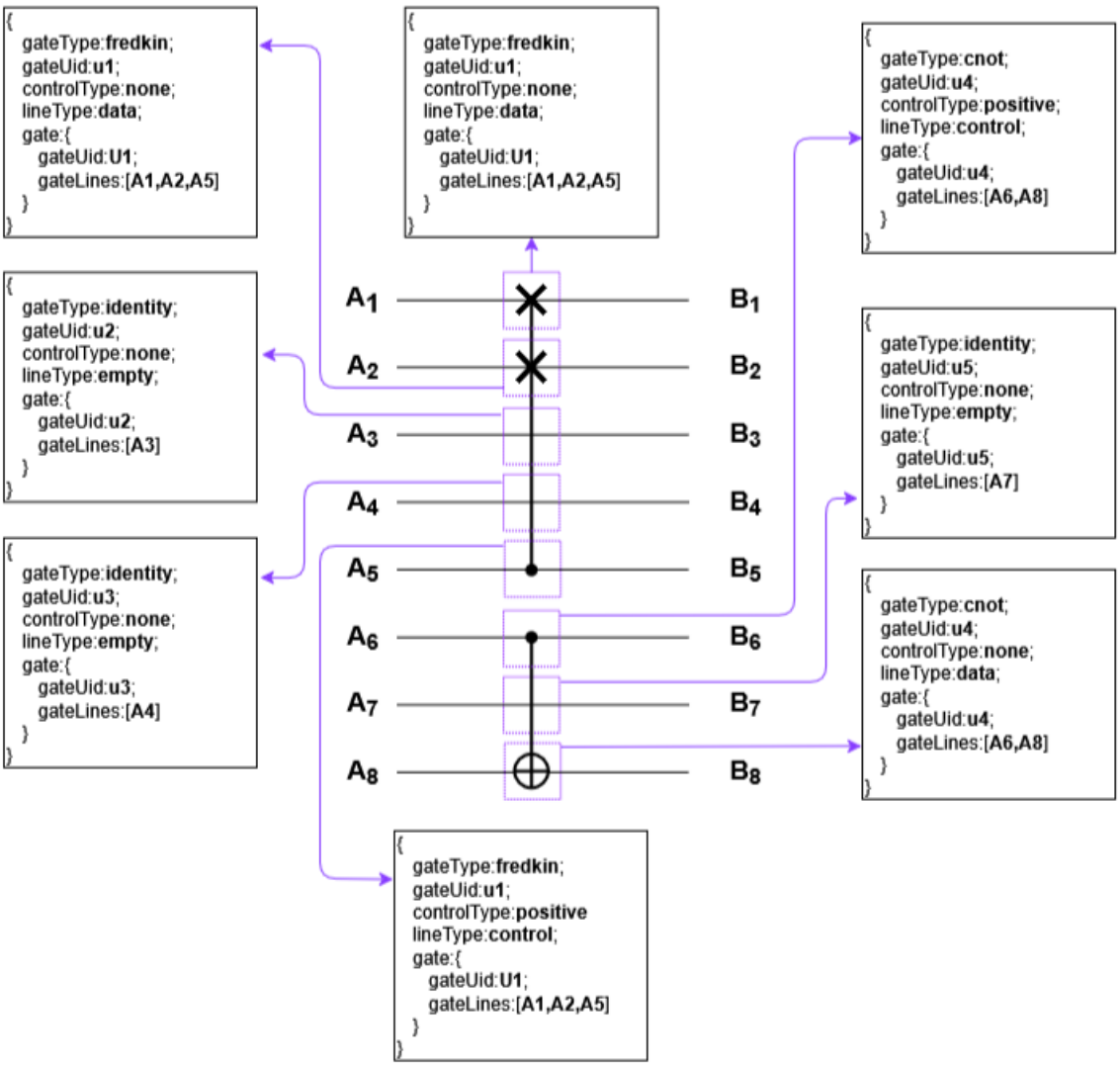


Рисунок 3.2.1.1 – відображення одного гена у вигляді об'єктно-орієнтованої програмної моделі.

Архітектура хромосоми. У розробленому генетичному алгоритмі хромосома не є пасивним набором даних, а реалізована як динамічний програмний об'єкт, що забезпечує пряме відображення – Direct Mapping генотипу у фенотип. Повна хромосома представляє собою впорядковану послідовність генів $\{G_1, G_2, \ldots, G_n\}$. Кожен окремий ген G відповідає рівно одному часовому такту роботи квантового алгоритму, під час якого може виконуватися набір незалежних та паралельних квантових операцій. Відповідно, вся хромосома має представлення у вигляді масиву таких послідовних генів.

Для розуміння архітектури розглянемо детальну будову одного гена. На рисунку 3.2.1.1 наведено схему трансформації фізичного представлення одного гена у відповідні структури даних програмної моделі. Ліва частина рисунка ілюструє паралельне виконання двох незалежних багатокубітних вентилів у межах одного такту: вентиля Фредкіна – на лініях A1, A2, A5 та вентиля CNOT – на лініях A6, A8. Лінії A3, A4 та A7 у поточному такті залишаються вільними.

Запропонована архітектура даних одного гена базується на таких принципах:

- Декомпозиція багатокубітних вентилів - кожен багатокубітний вентиль розбивається на набір об'єктів відповідно до кількості задіяних ліній. При цьому об'єкти чітко розділяються за ролями: керуючі лінії отримують властивість *lineType: control* з відповідним типом активації *controlType: positive*, а цільові – *lineType: data*.
- Зв'язність операцій – для збереження інформації про цілісність багатокубітного вентиля всі його складові об'єкти містять спільний унікальний ідентифікатор *gateUid*, наприклад, *u1* для вентиля Фредкіна, та посилання на об’єкти всіх задіяних ліній *gateLines: [A1, A2, A5]*. Це дозволяє програмному забезпеченню коректно відтворювати та оперувати вентилями при симуляції.
- Уніфікація пустот Identity – важливою особливістю моделі є те, що лінії без активних операцій *A3, A4, A7* не ігноруються, а явно формалізуються як

вентилі ідентичності *gateType: identity, lineType: empty*. Цей вентиль не має функції обчислення, і відповідне значення передається на вихід без змін. Це забезпечує структурну цілісність гена та уніфікує алгоритм обробки, оскільки система завжди працює зі сталим розміром масиву об'єктів, що дорівнює кількості кубітів у системі.

Оскільки повна хромосома є масивом генів, загальна модель кодування лінійно масштабується: додавання нових *gateState* роботи квантового кола просто розширює хромосому новими блоками ідентичної структури. Такий підхід до представлення дозволяє уникнути обчислювально витратних процедур декодування та ефективно виконувати генетичні оператори кросовер та мутацію на рівні окремих об'єктів або цілих генів без ризику порушення логіки квантових зв'язків між кубітами.

Програмний опис моделі. Кожен індивід популяції представляє собою композитний об'єкт, що складається з двох функціонально різних частин:

- *QuantumCircuit* – об'єкт, що описує топологію та логічну структуру схеми.
- *CircuitEvaluationResult* – об'єкт, що інкапсулює метрики ефективності даного рішення.

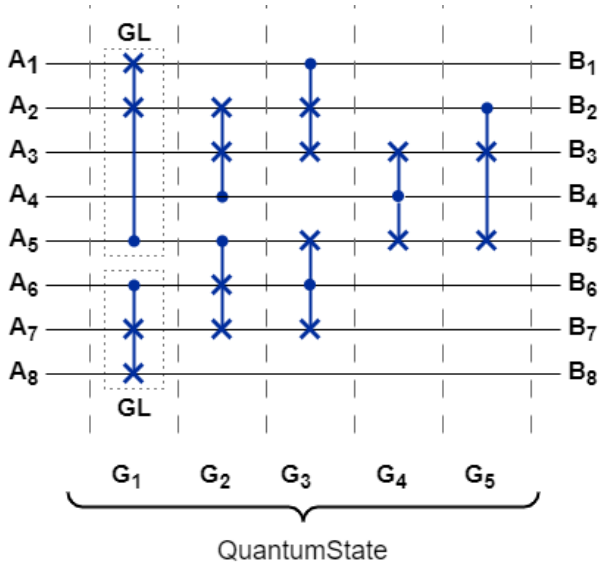


Рис. 3.2.1.1. Представлення хромосоми схеми.

Матрична організація топології. Структура *QuantumCircuit* побудована за принципом двовимірної матриці розмірністю $Q \times N$, де:

- Q (рядки) – кількість фізичних ліній – *qubit lines* у схемі.
- N (стовпці) – довжина хромосоми, яка визначає кількість послідовних часових шарів – генів.

Таке представлення дозволяє моделювати схему як дискретну послідовність перетворень квантового стану.

Ієрархія компонентів хромосоми. Внутрішня архітектура хромосоми представлена на рисунку 3.2.1.1 і базується на наступних рівнях абстракції:

1. Ген як часовий шар – *QuantumState* цей об’єкт є елементарною одиницею спадковості. Він відповідає одному стовпцю матриці схеми та містить набір операцій, що можуть виконуватися паралельно в один момент часу. На відміну від класичних ГА, де ген — це скалярне значення, тут ген є вектором, що описує стан усіх Q ліній на даному кроці.
2. Вектор комутації – *GateLines* кожен об'єкт *QuantumState* містить масив об'єктів *GateLine* фіксованої довжини Q.

   GateLine – це структурний елемент, прив'язаний до конкретного піна, тобто фізичної лінії. Він не містить логіки вентиля безпосередньо, а виконує роль контейнера-посилання.
3. *Програмна реалізація логічного елемента Gate.* Одним з ключових елементів архітектури є клас *Gate*, який інкапсулює обчислювальну логіку та фізичні характеристики квантового вентиля. Згідно з програмною реалізацією, цей клас містить:
   - *Uid* типу Guid – це унікальний ідентифікатор, що дозволяє групувати розрізнені *GateLines*, наприклад, керуючі та цільові лінії, в єдиний логічний елемент.

- *TransformationMatrix* - двовимірний масив, що зберігає матрицю унітарного перетворення.
- *QuantumCost* - ціле число – константа, що відображає вартість вентиля в елементарних операціях, наприклад для вентиля Фредкіна дорівнює 5.
- Особливістю реалізації є використання делегата функції *Measure*, який відповідає за обчислення логіки роботи конкретного вентиля. У розробленому алгоритмі підтримується дві стратегії обчислення поведінки вентиля:

  Стратегія 1: Матричні обчислення. Передбачає класичне моделювання квантових процесів шляхом перемноження матриць, тобто добуток Кронекера. Цей метод використовується для верифікації та роботи з комплексними амплітудами.

  Стратегія 2: Функціональна симуляція, поведінкове моделювання. Оскільки синтез реконфігурованих схем у даній роботі оперує класичною зворотною логікою, без введення суперпозиції на етапі синтезу функцій, реалізовано оптимізований підхід. Функція *Measure* симулює логічну дію вентиля через умовні оператори, маніпулюючи станами бітів безпосередньо. Це значно пришвидшує роботу фітнес-функції.

  Для реалізації різних типів логіки створено бібліотеку фабричних методів, які конфігурують об'єкт *Gate* та його функцію *Measure*:

  Наприклад для вентиль Фейнмана функція *Measure* ідентифікує індекси керуючої *PositiveControl* та цільової *Data* ліній. Якщо значення на керуючій лінії дорівнює 1, значення на цільовій лінії інвертується *0→1,1→0*. Відповідно, для вентиля Фредкіна функція визначає одну керуючу та дві лінії даних. Якщо керуючий сигнал активний – 1, відбувається обмін значень, тобто swap, між двома лініями даних. Ця операція реалізується через кортежний обмін *(qubits[i], qubits[j]) = (qubits[j], qubits[i])*.

Перед виконанням логіки кожен вентиль викликає метод *ValidateGateLines*, гарантуючи, що передані для обчислення лінії дійсно належать цьому екземпляру вентиля з перевіркою за *Uid*.

Об'єкт вентиля – *Gate* створюється в пам'яті в єдиному екземплярі. Відповідні об'єкти *GateLine* на різних лініях, керуючих та цільових, зберігають посилання “memory reference” на цей єдиний об'єкт *Gate*.

Для логічного групування ліній у межах одного гена використовується унікальний ідентифікатор *GateUid*. Причому одна лінія *GateLine* може містити одночасно, тобто в одному *GateState*, тільки однин *GateUid.*

Приклад: для вентиля Фредкіна, що займає лінії 0, 1 та 2, у поточному гені створюються три об'єкти *GateLine*. Всі вони мають однаковий *GateUid* та посилаються на один об'єкт *FredkinGate*. При цьому *GateLine* на лінії 0 позначається як *Control*, а на лініях 1 та 2 — як *Target*.

Така архітектура дозволяє операторам мутації "пересувати" вентиль по схемі, просто змінюючи індекси посилань, без необхідності перестворення об'єктів.

Об'єкт результатів оцінки – CircuitEvaluationResult. Для оптимізації обчислень хромосома містить відокремлений об'єкт *CircuitEvaluationResult*, який працює як кеш для фітнес-функції та інших властивостей синтезованої схеми. Він зберігає:

- Фітнес-значення F – оцінка якості або іншими словами пристосованості індивіда.
- Метрики складності – квантова вартість Q_c, та кількість вентилів G_c, обчислені шляхом обходу графа об'єктів *Gate*.
- Відстань Геммінга H_d – кількість помилок у таблиці істинності порівняно з еталоном.
- Маркери ліній: Індекси надлишкових виходів, що дозволяє алгоритму ігнорувати надлишкові значення при перевірці коректності.

- Таблицю обміну пінів *PinExchangeTable* – специфічний механізм, що дозволяє алгоритму віртуально перепризначати вхідні змінні на інші фізичні лінії без зміни структури вентилів, що значно розширює простір пошуку для топологічно чутливих функцій. Наприклад в очікуваній, тобто еталонній, таблиці істинності перший пін повинен мати значення 0110, але це значення було отримано на другому піні. І навпаки, очікуване значення 1001 з другого стовпця еталоної таблиці істинності отримано на першому. Ця інформація міститься в *PinExchangeTable* що допомагає після завершення синтезу додати необхідні SWAP вентилі щоб зберегти порядок стовпців еталонної таблиці істинності.

Запропонована архітектура хромосоми забезпечує високу гнучкість синтезу, дозволяючи оперувати складними реконфігурованими блоками як єдиними об'єктами, та створює фундамент для реалізації спеціалізованих генетичних операторів.

3.2.2. Механізми формування та керування популяцією

Ефективність генетичного алгоритму залежить не лише від структури окремої хромосоми, але й від стратегії, за якою відбувається оновлення генофонду [101]. У розробленому програмному комплексі реалізовано гібридну модель керування популяцією, яка поєднує стохастичну ініціалізацію, жорсткий елітизм для збереження кращих рішень та адаптивний механізм ін'єкції різноманітності для боротьби зі стагнацією.

Процес пошуку розпочинається зі створення нульового покоління, при чому було створено дві стратегії для ініціалізації:

1. Створення пустого об’єкту схеми з наступним заповненням його за допомогою операторів мутації, де параметри мутації вводились таким чином, щоб ймовірність додавання нового вентиля була більшою в

порівнянні з ймовірністю видалення. Таким чином схема вирощувалась наче кристал, поступовим заповненням вільних місць вентилями.

2. Заповнення популяції стохастично, де в подальному параметри мутації підбирались так, щоб ймовірність видалення вентиля була більшою, ніж ймовірність додавання, наче відсіканням надлишкових вентилів:

Параметри, якими описувалась популяція:

1. N – довжина хромосоми, кількість генів або тактових інтервалів;
2. Q – кількість ліній, ширина схеми;
3. C_{max} – розмір популяції, кількість індивідів.
4. *GateTypes* – масив можливих вентилів для синтезу схеми, реалізовувався бібліотекою імплементованих вентилів *QuantumLibrary*, Наприклад: Not, Toffoli, Feynman, Swap, Fredkin. Причому, запропонована реалізація дозволяє легко розширювати бібліотеку, імплементуючи новий вентиль і описуючи його поведінку. Так були реалізовані узагальнені вентилі Toffoli та Fredkin з кількостями керуючих ліній більше 2-х, а також зі змішаною полярністю, коли активація відбувається при значенні 0 на керуючих лініях.

При другій стратегії стохастичного заповнення популяції алгоритм випадковим чином обирає вентилі з заданого базису *GateTypes* та розміщує їх у матриці *QuantumCircuit*, формуючи початковий простір пошуку. Розміщення відбувається за наступним правилом – алгоритм обирає вентиль, розмірність – кількість ліній якого менша або рівна кількості доступних вільних ліній в поточному *GateState.* Процедура повторюється в циклі поки не будуть заповненні всі доступні місця для розміщення вентилів.

Елітизм. Для забезпечення стабільної збіжності алгоритму та запобігання втраті знайдених субоптимальних рішень застосовано принцип елітизму. Користувач задає параметр *EliteRate* – відсоток еліти, який визначає частку популяції E_{pop}, що гарантовано переходить у наступне покоління.

Алгоритм сортує всіх індивідів за значенням фітнес-функції. Найкращі E_{pop} хромосом копіюються в нову популяцію без змін, минаючи етапи кросоверу та мутації. Це гарантує, що найкраще знайдене рішення ніколи не погіршиться в процесі еволюціїі гарантує монотонне покращення якості.

Адаптивний контроль різноманітності (Tabu Search & Injection). Однією з критичних проблем синтезу квантових схем є наявність великої кількості локальних екстремумів, де класичний генетичний алгоритм схильний до передчасної збіжності або, іншими словами, стагнації. Коли популяція стає генетично однорідною, подальший пошук стає неефективним.

Для вирішення цієї проблеми розроблено дворівневий механізм контролю ентропії:

1. Фільтрація дублікатів (Tabu Filtering): На етапі формування нового покоління алгоритм перевіряє унікальність кожного кандидата. Якщо хромосома є точною копією вже існуючої в популяції, ідентичний фенотип, а значить і таблиця істинності, вона позначається як *Tabu*. Вводиться обмеження: кількість ідентичних хромосом не може перевищувати 10% від загального розміру популяції. Це запобігає миттєвому захопленню популяції одним домінантним, але не обов'язково оптимальним рішенням.

2. Вихід зі стагнації – алгоритм безперервно моніторить стан популяції. Якщо виявляється, що відсоток ідентичних за фітнесом індивідів перевищує критичний поріг який також задається параметром, зазвичай це значення 12%, це ідентифікується як стан стагнації. У цьому випадку активується процедура "ін'єкції":

1. Частина дублюючих особин позначених *Tabu* примусово видаляється з популяції.

2. На їх місце генеруються та додаються абсолютно нові, стохастичні індивіди.

Ця процедура штучно підвищує генетичну різноманітність, струшуючи популяцію та дозволяючи алгоритму вийти з локального оптимуму для

продовження пошуку глобального екстремуму в інших областях простору рішень.

3.2.3. Оператор багатокомпонентної мутації

Критичним недоліком класичних генетичних алгоритмів при синтезі квантових схем є швидка втрата генетичного різноманіття, що призводить до передчасної збіжності в локальних екстремумах [102]. Для вирішення цієї проблеми розроблено оператор багатокомпонентної мутації, який забезпечує баланс між дослідженням нових областей простору пошуку та оптимізацією знайдених рішень.

На відміну від стандартної бітової інверсії, запропонований оператор працює на рівні об'єктної моделі схеми і здатний змінювати як топологію з'єднань, так і кількісний склад вентилів.

Імовірнісна модель мутації. Мутація застосовується до індивіда з ймовірністю P_m. Оскільки хромосома має складну структуру, процес мутації розділено на три взаємовиключні стратегії. Вибір конкретного типу мутації для даного індивіда відбувається стохастично на основі нормованих вагових коефіцієнтів:

- P_{ms} – ймовірність зміни пінів (перекомутація) – mutation of shift;
- P_{ma} – ймовірність додавання нового вентиля – mutation of addition;
- P_{mr} – ймовірність видалення вентиля – mutation of removal.

Обов'язковою умовою функціонування моделі є умова строгої нормалізації простору подій:

$$P_{ms} + P_{ma} + P_{mr} = 1, \tag{3.2.3}$$

Це рівняння гарантує, що при активації оператора мутації завжди буде обрана одна і тільки одна з трьох стратегій.

Кожен з коефіцієнтів відповідає за певний напрямок еволюційного пошуку:

Структурна варіативність P_{ms} – ця компонента відповідає за горизонтальний пошук у просторі рішень. Зміна точок підключення вентиля до ліній квантової шини без зміни його типу чи кількості елементів у схемі. Квантова вартість Q_c в такому випадку залишається незмінною, змінюється лише логіка роботи. Застосування такого оператора мутації дозволяє знайти правильну конфігурацію входів та виходів для вже існуючого набору вентилів.

Експансія складності P_{ma} – ця компонента відповідає за розширення простору пошуку. Відбувається інтеграція нового вентиля з обраного базису у вільний слот *QuantumState* хромосоми. Відповідно цей оператор збільшує квантову вартість Q_c та глибину схеми. Застосовується, коли поточної кількості вентилів недостатньо для реалізації цільової функції (схема "недоукомплектована"). Це основний інструмент виходу з локальних екстремумів, де простіші схеми не дають нульової помилки.

Оптимізаційна редукція P_{mr} – ця компонента відповідає за спрощення рішення. Відбувається видалення випадкового вентиля зі схеми і відповідно зменшує квантову вартість. Оператор усуває надлишкові вентилі, які не впливають на результат (наприклад, дві послідовні інверсії на одній лінії) або спрощує схему після того, як основна логіка була знайдена.

Алгоритм стохастичного вибору. Програмна реалізація вибору стратегії базується на генерації псевдовипадкового числа $r \in [0,1]$. Визначення типу мутації відбувається за кумулятивним принципом:

1. Якщо $0 \leq r < P_{ms}$, виконується перекомутація пінів.
2. Якщо $P_{ms} \leq r < P_{ms} + P_{ma}$ виконується додавання вентиля.
3. Якщо $P_{ms} + P_{ma} \leq r \leq 1$, виконується видалення вентиля.

Така модель дозволяє гнучко налаштовувати поведінку алгоритму. Наприклад, встановивши високе значення P_{ma} на початку еволюції та високе P_{mr}

наприкінці, можна реалізувати стратегію при якій спочатку знаходиться рішення будь-якою ціною, а потім оптимізується його.

Кожна з трьох стратегій мутації реалізована як окрема процедура, що маніпулює об'єктною структурою хромосоми. Оскільки хромосома представлена матрицею посилань, оператори мутації гарантують збереження цілісності схеми (валідності з'єднань) на кожному кроці.

Нижче наведено формалізований опис алгоритмів для кожного типу мутації.

А. Оператор реконфігурації з'єднань – M_s. Цей оператор, представлений на рисунку 3.2.3.1, змінює топологію підключення існуючого вентиля без зміни його типу чи параметрів.

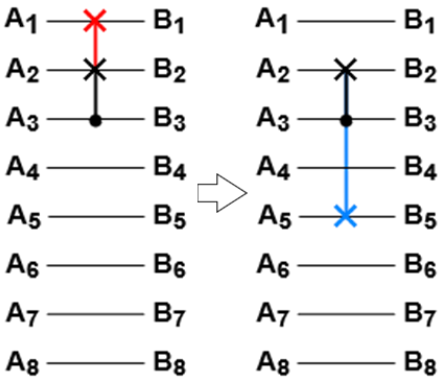


Рис. 3.2.3.1. Процес роботи оператора M_s

Він дозволяє алгоритму знаходити правильні вхідні лінії для реалізації логіки і працює наступним чином:

1. Вибір гена – зі списку генів хромосоми $G_1, \ldots, G_N$ випадковим чином обирається ген *QuantumState*, який містить хоча б один активний вентиль.

2. Вибір піна – у межах обраного гена випадковим чином обирається один об'єкт *GateLine* – gl_{old}, що посилається на вентиль.

3. Пошук цілі – у тому ж гені обирається інший пін - gl_{new}, який на даний момент є вільним –тобто не містить посилання на вентиль.

Після знайдених gl_{old} та gl_{new} відбувається перекомутація:

1. Індекс – номер лінії для gl_{old} замінюється на індекс gl_{new}.

2. Оновлюється структура вектора *GateLines* у поточному стані.

3. Валідація – перевіряється, чи не порушує нова конфігурація унікальність підключень, наприклад, чи не підключено дві керуючі лінії одного вентиля до однієї фізичної шини.

B. Оператор розширення схеми – M_a. Цей оператор відповідає за введення нових обчислювальних ресурсів у схему, візуальне представленя його роботи представлено на рисунку 3.2.3.2.

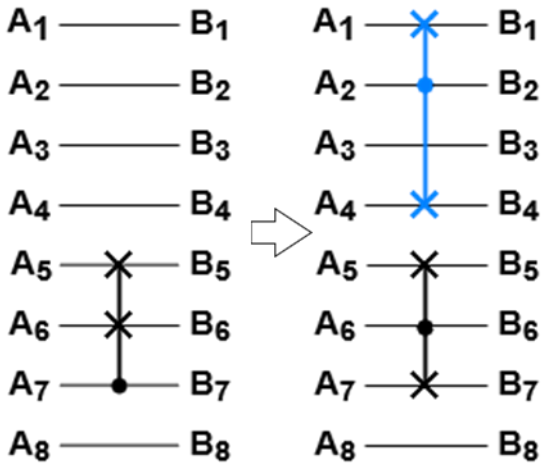


Рис. 3.2.3.2. Оператор розширення схеми – M_a

Його робота обмежується наявністю вільного простору в матриці хромосоми. Алгоритм виконання:

1. Вибір типу – з доступного базису *GateType*, наприклад, {Fredkin, Feynman, Toffoli} випадковим чином обирається тип нового вентиля $Gate_{new}$. Визначається необхідна для нього кількість ліній *K* (наприклад, для Fredkin *K=3*).

2. Сканування матриці – здійснюється обхід хромосоми для пошуку підходящого місця. Ген *QuantumState* вважається валідним кандидатом, якщо кількість вільних *GateLines* у ньому ≥*K*.

3. Локалізація – якщо знайдено декілька підходящих генів, алгоритм обирає перший доступний, залежно від налаштувань.

4. Інстанціювання – створюється новий об'єкт *Gate* з унікальним *Uid*. У обраному гені *QuantunState* знаходяться *K* вільних пінів. Ці піни ініціалізуються як об'єкти *GateLine* з посиланням на новий *Gate* та відповідними ролями Control чи *Target*.

В результаті схема отримує нову функціональність, а її квантова вартість.

C. Оператор редукції – M_r. Цей оператор(рисунок 3.2.3.3) діє як стохастичний оптимізатор, видаляючи елементи схеми.

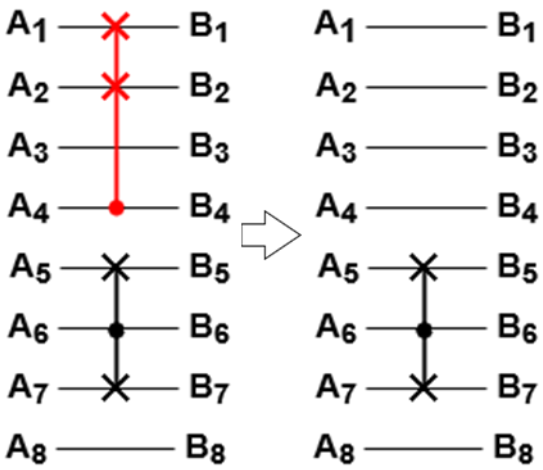


Рис 3.2.3.3 Процес редукції M_r

Це критично важливо для мінімізації вартості рішення та усунення надлишкових або неактивних вентилів, що накопичується в процесі еволюції. Алгоритм виконання:

1. Вибір жертви – зі списку випадковим чином обирається один вентиль $Gate_{target}$.

2. Очищення посилань – алгоритм сканує матрицю схеми. Знаходяться всі об'єкти *GateLine*, чий *GateUid* співпадає з $Gate_{target}$.

3. Ці об'єкти видаляються або замінюються на порожні стани *EmptyGate*, звільняючи фізичні лінії.

4. Видалення об'єкта – посилання на об'єкт $Gate_{target}$, знищується, і збирач сміття звільняє пам'ять.

Як результат квантова вартість схеми зменшується. Якщо видалений вентиль був зайвим, фітнес-функція зростає; якщо критичним – падає і така особина ймовірно не пройде селекцію.

Застосування цих трьох алгоритмів у комплексі дозволяє реалізувати стратегію пульсуючого пошуку. Спершу M_a насичує схему логікою, дозволяючи досягти нульової помилки за рахунок надлишкової складності. Потім M_s оптимізує з'єднання для кращого використання наявних ресурсів. Наостанок M_s відсікає зайве, залишаючи мінімально необхідну структуру, що відповідає критеріям оптимальності синтезу.

3.2.4. Оператори кросоверу та селекції

Процес репродукції в генетичному алгоритмі складається з двох послідовних етапів: селакція батьківських пар, тобто селекції, та безпосередньої рекомбінації їхнього генетичного матеріалу – кросоверу. У розробленому програмному комплексі реалізовано класичну схему, що поєднує метод пропорційної рулетки для селекції та двоточковий оператор кросоверу.

Стратегія селекції. Для формування батьківського пулу застосовано метод пропорційної селекції, відомий як метод рулетки. Його мета — надати перевагу більш пристосованим особинам, не виключаючи при цьому повністю шанс участі менш ефективних рішень, що важливо для збереження генетичного різноманіття. Ймовірність вибору конкретної особини для участі у формуванні наступного покоління обчислюється за методом пропорційної селекції. Математично цей процес описується наступним рівнянням:

$$P_i = \frac{F_i}{\sum_{\mathrm{j}=1}^{C_{max}} F_j}, \qquad (3.2.4.1)$$

де:

P_i – ймовірність вибору i-ї особини до батьківського пулу;

F_i – значення фітнес-функції i-ї особини;

C_{max} – загальний розмір популяції (кількість схем-кандидатів у поточному поколінні);

$\sum_{j=1}^{C_{max}} F_j$ – сумарна пристосованість усіх особин у поточній популяції.

Алгоритмічна реалізація виглядає наступним чином:

1. Обчислюється сумарна пристосованість популяції *fitnessSum*.
2. Генерується випадкове число *randomValue* в діапазоні [0, *fitnessSum*].
3. Здійснюється ітеративний обхід популяції з накопиченням поточної суми фітнесу *cumulativeFitness*.
4. Батьком обирається той індивід, на якому накопичена сума вперше перевищує *randomValue*.

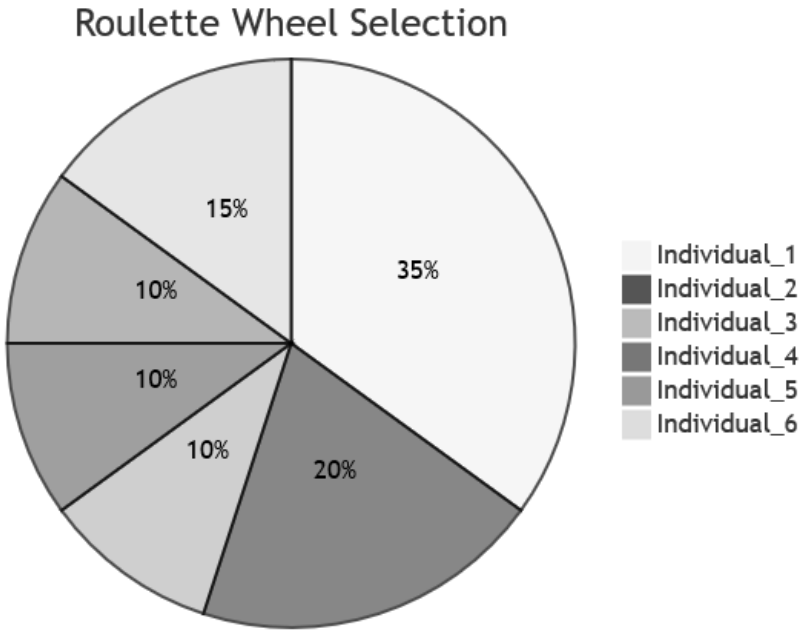


Рис 3.2.4.1 Представлення ймовірності вибору батьків методом рулетки

Згідно з представленим алгоритмом – кожна схема-кандидат отримує ймовірність відбору, прямо пропорційну її пристосованості відносно всієї популяції. Чим кращі показники має схема (менша відстань Геммінга та нижча квантова вартість, представлено як F_i), тим більшу "частку" загальної ймовірності вона займає. Водночас, навіть особини з низькими показниками зберігають ненульовий шанс бути обраними. Це є критично важливою властивістю методу, оскільки вона дозволяє підтримувати генетичне різноманіття в популяції та запобігає передчасному застряганню алгоритму в локальних оптимумах.

Ця процедура повторюється двічі для вибору пари батьків $Parent_1$ та $Parent_2$. Алгоритм містить перевірку, що запобігає кросоверу індивідам самим із собою.

Двохточковий оператор кросоверу. Враховуючи лінійну структуру хромосоми (послідовність генів-тактів), для рекомбінації обрано двоточковий кросовер. Цей метод є менш руйнівним для сформованих функціональних блоків схеми, ніж рівномірний кросовер, і забезпечує краще перемішування сегментів, ніж одноточковий.

Операція виконується з імовірністю P_c. Якщо умова ймовірності не виконується, нащадки є точними копіями батьків.

Алгоритм виконання:

1. Вибір точок розриву – генеруються дві випадкові точки k_1 та k_2 в межах довжини хромосоми *N*, де *N* = *dnaSize*, такі що $0 \leq k_1 \leq k_2 < N$.
   - k_1 – firstCrossPoint обирається випадково в першій частині хромосоми.
   - k_2 – secondCrossPoint обирається випадково між k_1 та кінцем хромосоми.
2. Формування зон обміну: Ці точки ділять хромосому на три сегменти:
   - Зона А (початок): індекси $[0 \dots k_1 - 1]$.

- Зона B (середина): індекси $[k_1 \dots k_2]$.
- Зона C (кінець): індекси $[k_2 + 1 \dots N - 1]$.

3. Рекомбінація генів – створюються два нащадки $Child_1$ та $Child_2$. Обмін генетичним матеріалом відбувається наступним чином:
   - Зона B – гени передаються прямо $Child_1 \leftarrow Parent_1$, $Child_2 \leftarrow Parent_2$.
   - Поза межами зони B, тобто зони A і C – відбувається перехресний обмін $Child_1 \leftarrow Parent_2$, $Child_2 \leftarrow Parent_1$.

Такий підхід, зображено на рисунку 3.2.4.2, дозволяє зберегти центральну частину схеми одного батька, оточивши її вхідними та вихідними каскадами іншого, що сприяє ефективному комбінуванню знайдених субоптимальних рішень.

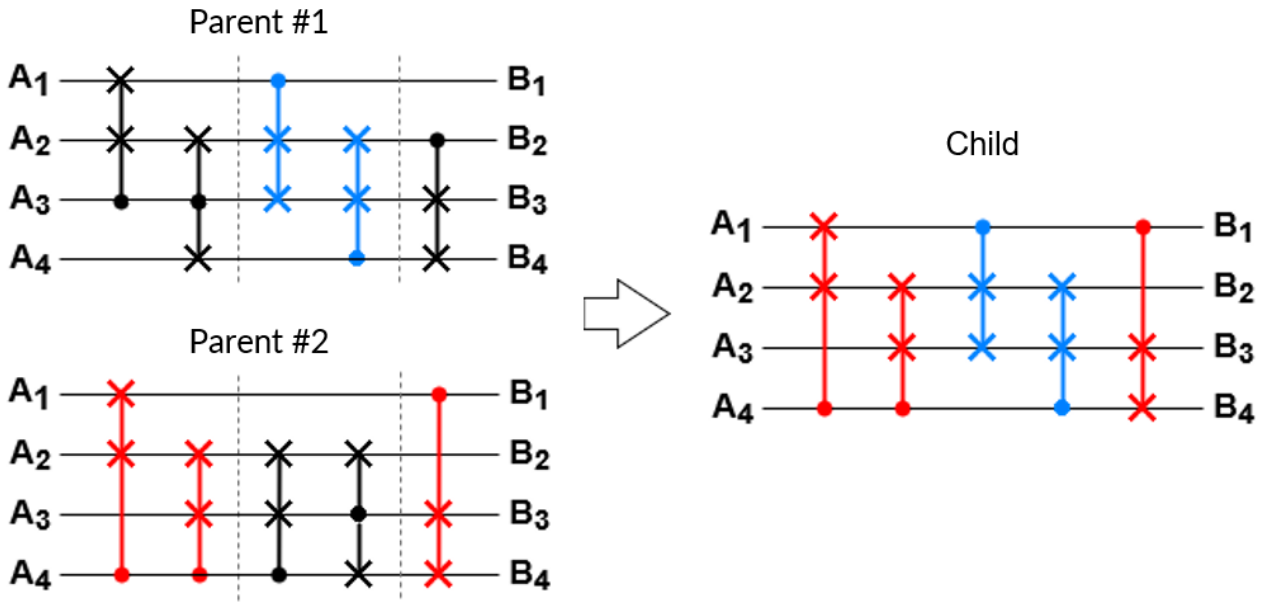


Рис 3.2.4.2 Процес кросоверу генів

3.2.5. Адаптація фітнес-функції для багатокритеріальної оптимізації

У розробленому методі задача синтезу квантової схеми формалізується як задача багатокритеріальної оптимізації. Метою генетичного алгоритму є пошук хромосоми, яка максимізує значення фітнес-функції.

Фітнес-функція F є композитною і враховує два ключові фактори:

1. Логічна коректність: наскільки точно схема відтворює задану таблицю істинності.
2. Апаратна ефективність: наскільки дешевою в контексті квантової вартості є схема.

У результаті оптимізаційного процесу обирається конфігурація схеми з найвищим рівнем фітнес-функції. Це означає, що найпристосованіший індивід повинен мати максимально можливе значення фітнесу, яке формується на основі мінімізації помилки та вартості.

Для реалізації цього підходу застосовано адитивну модель з оберненою пропорційністю. значення фітнесу для кожного індивіда обчислюється як сума двох зважених компонент:

$$F = F_{accuracy} + F_{efficiency} = \alpha \cdot \frac{1}{H_d + 1} + \beta \cdot \frac{1}{Q_c + 1}, \qquad (3.2.5.1)$$

де:

H_d – Hamming Distance — метрика логічної помилки, сумарна кількість бітових розбіжностей між таблицею істинності поточної схеми та цільової функції.

Q_c – Quantum Cost — метрика апаратної складності, сума квантових вартостей усіх активних вентилів схеми.

α – ErrorCoefficient — ваговий коефіцієнт, що визначає пріоритет коректності роботи.

β – CostCoefficient — ваговий коефіцієнт, що визначає пріоритет оптимізації розміру схеми.

Особливості моделі:

1. Математична стабільність: Додавання одиниці у знаменниках (+1) виконує роль згладжуючого фактора. Воно запобігає виникненню сингулярності – ділення на нуль, у випадках ідеально точного рішення $H_d = 0$ або порожньої схеми $Q_c = 0$.
2. Нормалізація внеску: Використання обернених величин як $\frac{1}{x+1}$ трансформує задачу мінімізації $x \to 0$ у задачу максимізації $F \to max$, обмежуючи внесок кожної компоненти її ваговим коефіцієнтом (α або β).
3. Ієрархія цілей: На практиці встановлюється співвідношення $\alpha \gg \beta$. Це створює сильний еволюційний тиск на початкових етапах для пошуку рішення з $H_d = 0$. Як тільки коректна схема знайдена, перша частина формули досягає свого максимуму α, і подальше зростання фітнесу можливе лише за рахунок зменшення Q_c, тобто збільшення значення другого доданку.

Така конструкція фітнес-функції дозволяє алгоритму автоматично перемикатися між фазами «пошуку рішення» та «оптимізації рішення» без необхідності зовнішнього втручання.

3.2.6 Алгоритм розрахунку логічної розбіжності та Swap-евристика

Суттєвою перепоною при синтезі квантових схем є проблема жорсткої прив'язки вихідних ліній. У класичних підходах вимагається, щоб k-й вихід цільової функції знаходився строго на k-й фізичній лінії схеми. Це змушує генетичний алгоритм витрачати значні ресурси на створення ланцюжків SWAP-вентилів для маршрутизації сигналів, замість того щоб фокусуватися на реалізації логіки.

Для вирішення цієї проблеми розроблено алгоритм динамічного пошуку найкращої перестановки виходів – Dynamic Output Permutation Search. Його суть полягає у знаходженні такого бієктивного відображення виходів схеми на

цільові змінні, при якому сумарна помилка, тобто відстань Геммінга є мінімальною.

Процедура розрахунку складається з трьох етапів:

1. Побудова матриці дистанцій На першому кроці алгоритм виконує перехресне порівняння всіх виходів поточної синтезованої схеми з усіма стовпцями цільової таблиці істинності. Нехай O — множина векторів виходів схеми, а T — множина векторів цільової функції. Для кожної пари i,k, де i — індекс фізичної лінії, а k — індекс цільової змінної, обчислюється локальна відстань Геммінга $d_{i,k}$:

$$d_{i,k} = \sum_{j=1}^{L} \left(O_{i,k} \oplus T_{t,k} \right), \tag{3.2.6.1}$$

де:

L — кількість рядків у таблиці істинності (2n);

$\oplus$ — операція XOR (сума за модулем 2);

$O_{i,k}$ — значення i-го виходу схеми на j-му вхідному наборі;

$T_{t,k}$ — значення k-го цільового виходу на j-му вхідному наборі.

2. Евристичний пошук відповідностей Після побудови множини всіх можливих дистанцій застосовується жадібний евристичний алгоритм для вибору оптимальної конфігурації. Метою є знаходження перестановки π, яка мінімізує інтегральну помилку H_d:

$$H_d = \min_{\pi} \sum_{k=1}^{Q} d_{\pi(k),k}, \tag{3.2.6.2}$$

У програмній реалізації алгоритм групує обчислені дистанції за цільовими індексами та ітеративно обирає пари з найменшим значенням помилки,

гарантуючи, що кожна фізична лінія використовується для відображення лише один раз.

3. Формування таблиці перестановок «Swap Table» Результатом роботи алгоритму є кортеж, що містить мінімально знайдену сумарну дистанцію H_d а список перестановок *SwapTable*. *SwapTable* визначає необхідні комутації ліній, які потрібно виконати в кінці схеми, щоб привести її виходи у відповідність до специфікації.

Використання цієї евристики дозволяє розділити задачу синтезу на дві підзадачі: синтез логічного ядра використовуючи GA та маршрутизація сигналів – вирішується аналітично через *SwapTable*, що суттєво прискорює збіжність алгоритму.

3.3. Архітектура програмного комплексу автоматизованого синтезу

Програмна реалізація системи виконана на платформі C#.NET з використанням принципів багатошарової «Clean Architecture». Такий підхід забезпечує незалежність бізнес-логіки від зовнішніх інтерфейсів та інфраструктури, що є критичним для наукоємних проєктів, які потребують постійної модифікації алгоритмів.

Структура рішення розділена на три основні рівні:

1. Domain Layer яким є проєкт Api.Domain – містить основні класи такі як QuantumCircuit, Gate, GateLine, об'єкти-значення та інтерфейси. Цей шар не має зовнішніх залежностей.
2. Application Layer реалізований проєктом Api.Application – описує сценарії використання, алгоритмічне ядро (генетичний алгоритм) та логіку симуляції схем. Тут застосовано патерн CQRS (Command Query Responsibility Segregation) для розділення операцій читання та модифікації.

3. Presentation/Infrastructure Layer (проєкт Api.WebApi): Забезпечує взаємодію з користувачем через REST API та інтеграцію з файловою системою.

3.3.1. Об'єктна модель квантової схеми

Діаграма класів доменної області представлена на рисунку 3.3.1.1.

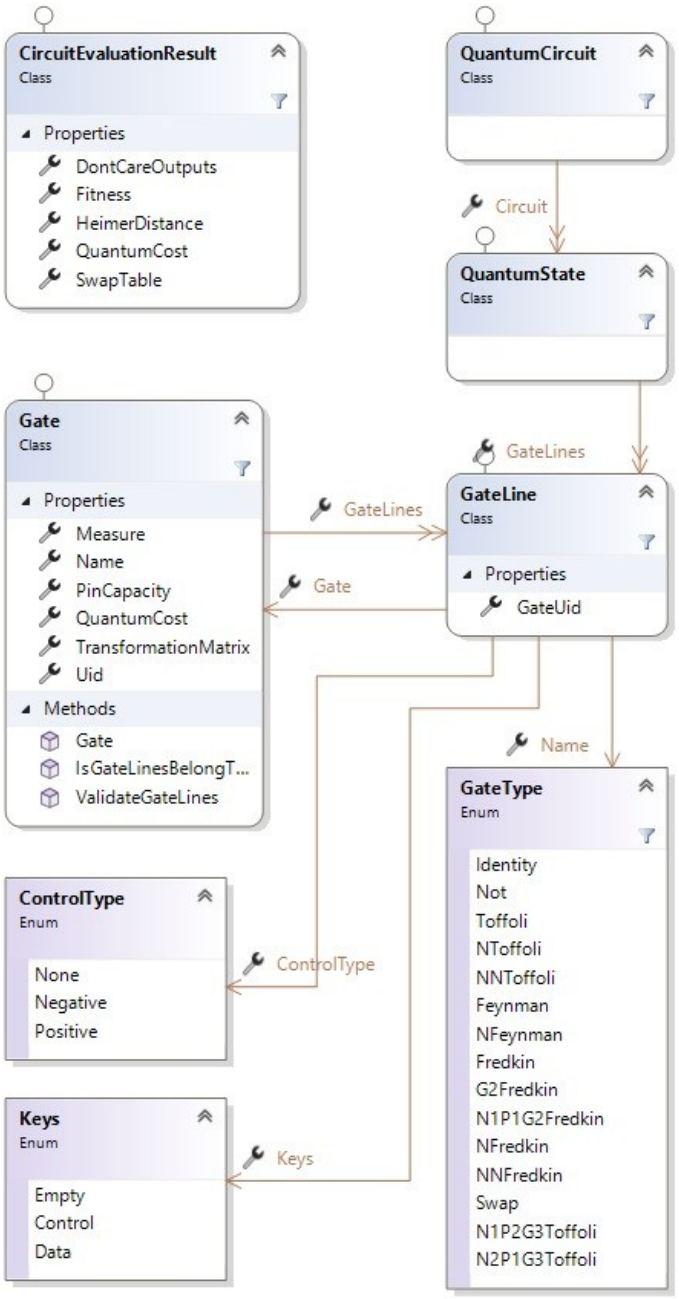


рис. 3.3.1.1. Діаграма базових класів об'єктної моделі квантової схеми

Основою програмного комплексу є шар домену (Api.Domain), який визначає структуру даних для представлення квантової схеми. Оскільки генетичний алгоритм вимагає виконання тисяч операцій мутації та кросоверу, модель даних спроєктована з урахуванням вимог до гнучкості, можливості серіалізації та глибокого клонування.

Ієрархія класів схеми. Структура схеми реалізує композиційний патерн і складається з чотирьох основних рівнів:

1. *QuantumCircuit* – цей клас представляє повну квантову програму. Він містить властивість *List<QuantumState> Circuit*, яка є впорядкованою послідовністю часових тактів. Цей клас відповідає за цілісність хромосоми.
2. *QuantumState* (тактовий зріз) – цей клас моделює один дискретний крок у часі, вертикальний стовпець схеми. Він містить колекцію *List<GateLine> GateLines*, розмір якої дорівнює кількості кубітів у схемі *N*. Важливо: *QuantumState* не зберігає самі об'єкти вентилів напряму, а зберігає "лінії", що на них посилаються. Це дозволяє одному такту містити кілька паралельних операцій, якщо вони діють на різні кубіти.
3. *GateLine* (точка підключення) – сутність, що зв'язує конкретний фізичний кубіт з логічним вентилем. Вона містить метадані про роль даного кубіта в операції:
   - *ControlType* – властивість яка визначає тип керування яке може бути трьох типів: *Positive* — керування одиницею, *Negative* — керування нулем, *None* — цільовий кубіт.
   - *Keys* – семантична роль лінії (дані або контроль).
   - GateUid - унікальний ідентифікатор логічного вентиля, до якого належить ця лінія.

4. *Gate* – клас, що інкапсулює поведінку квантового оператора. Він містить:
    - TransformationMatrix – матричне представлення оператора.
    - QuantumCost – вартість реалізації вентиля, використовується фітнес-функцією для підрахунку складності.
    - Measure - делегат *Func*<...>, який виконує безпосередню зміну станів бітів під час симуляції.

Однією з ключових особливостей реалізації є підтримка багатокубітних вентилів довільної складності, наприклад, *G4Toffoli* — вентиль Тоффолі з 4 контролюючими лініями.

Для цього використано механізм розподіленого посилання: Один об'єкт *Gate* існує в пам'яті в єдиному екземплярі, але на нього посилаються декілька об'єктів *GateLine*, розташованих на різних індексах у межах одного *QuantumState*. Зв'язок забезпечується через *Guid Uid*.

Приклад для вентиля *CNOT*:

1. На лінії A_1 розміщено *GateLine { Type: Control, Ref: GUID_A }*
2. На лінії A_2 розміщено *GateLine { Type: Target, Ref: GUID_A }*
3. Обидві лінії посилаються на спільний об'єкт *Gate (GUID_A)*.

Такий підхід дозволяє алгоритму мутації легко змінювати тип керування окремої лінії, наприклад, змінити позитивний контроль на негативний, не перестворюючи весь об'єкт вентиля.

Реалізація глибокого клонування deep copy.

У генетичних алгоритмах критично важливо, щоб зміни у нащадків не впливали на батьківські особини. Оскільки C# є об'єктно-орієнтованою мовою з посилальними типами, просте присвоєння змінних копіює лише посилання на об'єкт в пам'яті.

Для вирішення цієї проблеми всі доменні класи (*QuantumCircuit, QuantumState, Gate, GateLine*) реалізують інтерфейс *ICloneable*. Метод *Clone()* реалізує повну рекурсивну копію дерева об'єктів:

Система підтримує розширюваний набір типів вентилів, визначений у *GateType.cs*:

1. Базові: *Identity*, *Not*.
2. Контрольовані: *Toffoli*, *Feynman*, *Fredkin*.
3. Узагальнені (Generalized): *G3Toffoli...G10Toffoli* – багатоконтрольні версії, *G2Fredkin*.
4. Гібридні – вентилі зі змішаною полярністю активації: *NToffoli*, *N1P1G2Fredkin* (змішане керування).

3.3.2. Технологічний стек та оптимізація обчислювальних процесів

Задача синтезу квантових схем належить до класу NP-повних задач, що висуває підвищені вимоги до продуктивності програмного забезпечення. Для реалізації комплексу обрано платформу .NET 8 та мову програмування C#. Цей вибір обумовлений наявністю потужних інструментів для роботи з пам'яттю та вбудованої бібліотеки паралельних обчислень.

Найбільш ресурсоємною частиною алгоритму є обчислення фітнес-функції представленої методом *CalculateFitness*, оскільки вона вимагає повної симуляції роботи квантової схеми на всіх вхідних векторах. Враховуючи, що оцінка кожного індивіда в популяції є незалежною подією, було застосовано патерн паралелізму даних. У класі *GeneticAlgorithm* замість послідовного циклу *foreach* використано метод *Parallel.For*. Такий підхід дозволяє автоматично розподіляти навантаження між усіма доступними логічними ядрами процесора, що забезпечує лінійне прискорення роботи алгоритму пропорційно до кількості ядер.

Якість генетичного пошуку критично залежить від якості джерела ентропії. Стандартний генератор *System.Random* у .NET не є потокобезпечним і має обмежений період повторюваності. Для вирішення цієї проблеми у сервісі *QuantumCircuitService* використано генератор *Mcg31m1* із бібліотеки наукових

обчислень *MathNet.Numerics*. *Mcg31m1* — це мультиплікативний конгруентний генератор з довгим періодом, який забезпечує кращий статистичний розподіл псевдовипадкових чисел. Використання надійного генератора запобігає передчасній збіжності алгоритму до локальних екстремумів через "зациклення" мутацій.

Для зменшення навантаження на Garbage Collector реалізовано механізм кешування векторів істинності. Клас *TruthTableHelper* використовує статичний потокобезпечний словник для зберігання згенерованих вхідних комбінацій:

private static Dictionary<int, List<List<int>>> _truthTables;

При першому запиті для кубітів система генерує матрицю входів, наприклад, для 3 кубітів: 000, 001...111, і зберігає її в пам'яті. У всіх наступних поколіннях та для всіх індивідів використовуються вже існуючі посилання на масиви, що усуває мільйони зайвих операцій виділення пам'яті.

Оскільки об'єкти *QuantumCircuit* є посилальними типами, передача їх у методи мутації без клонування призвела б до побічних ефектів, коли зміна нащадка псує геном батька. Для запобігання цьому реалізовано механізм глибокого копіювання. Усі доменні сутності реалізують інтерфейс *ICloneable*. Метод *Clone()* рекурсивно створює дублікати всіх вкладених об'єктів *схема → такти → лінії*. Для складних об'єктів використано серіалізацію через *System.Text.Json* як універсальний метод клонування *DeepCopy<T>*.

Цей комплекс заходів дозволив досягти високої стабільності роботи системи навіть при популяціях у кілька десятків тисяч індивідів.

3.3.3. Інтерфейс взаємодії з зовнішніми середовищами: генерація QASM-коду для IBM Qiskit

Останнім етапом роботи алгоритму є перетворення внутрішнього об'єктного графа схеми у формат, придатний для виконання на квантових

процесорах або симуляторах. Стандартом де-факто в індустрії є мова OpenQASM 3.0 – мова квантового асемблера.

Реалізація цього етапу вирішує дві задачі:

1. Експорт – трансляція синтезованої схеми в інструкції QASM.
2. Верифікація: Незалежна перевірка коректності роботи схеми за допомогою фреймворку IBM Qiskit [103, 104, 105].

Трансляція внутрішнього представлення та декомпозиція керування.

Модуль експорту - *OpenOpenqasm20Converter* виконує функцію серіалізатора, перетворюючи об'єктний граф *QuantumCircuit* у лінійний скрипт мовою OpenQASM. Цей процес не є тривіальним перетворенням "один в один" через відмінності у наборах примітивів, що використовуються в генетичному алгоритмі та стандарті QASM.

Процес трансляції складається з трьох послідовних етапів для кожного такту схеми: ідентифікація операції, адаптація логіки керування та формування інструкції.

1. Словникове відображення операторів - для базових вентилів, що мають прямі аналоги в стандартній бібліотеці QASM (модулі *qelib1.inc* або *stdgates.inc*), використовується статичний словник відповідностей *Dictionary<GateType, string>*. Це дозволяє O(1) перетворення внутрішнього enum у рядковий ідентифікатор.

Таблиця 3.3. Відповідність внутрішніх типів вентилів інструкціям QASM

Внутрішній тип (GateType)	QASM інструкція	Опис
Not	x	Pauli-X (інверсія)
Feynman	cx	Controlled-NOT (CNOT)
Toffoli	ccx	Toffoli (CCNOT)
Fredkin	cswap	Controlled-SWAP
Swap	swap	Обмін станів

2. Алгоритм обробки полярності керування – найбільш складною частиною трансляції є підтримка ліній з негативним керуванням. Внутрішня модель *GateLine* дозволяє будь-якому керуючому кубіту мати стан *ControlType.Negative* – коли активація вентиля відбувається при значенні кубіта |0⟩. Проте стандартні вентилі QASM, наприклад ccx, працюють виключно з позитивним керуванням активація при |1⟩.

Для вирішення цієї проблеми реалізовано алгоритм динамічної інверсії, який автоматично огортає цільовий вентиль у оператори NOT. Математично операція з негативним керуванням U_{neg} еквівалентна операції з позитивним керуванням U_{pos}, виконаній у базисі, інвертованому по відношенню до керуючого кубіта q_c:

$$U_{\text{neg}}(q_c, q_t) = X(q_c) \cdot U_{\text{neg}}(q_c, q_t) \cdot X(q_c), \qquad (3.3.3.1)$$

У програмному коді це реалізовано наступним чином:

- Аналіз ліній – для кожного вентиля виділяються списки ліній з позитивним *positiveControlledLines* та негативним *negativeControlledLines* керуванням.
- Пре-інверсія – якщо список *negativeControlledLines* не порожній, генератор додає інструкції *x q[i]*; для кожного кубіта з цього списку.
- Виконання операції – генерується основна команда, наприклад ccx, яка сприймає всі лінії як позитивні.
- Пост-інверсія – повторно генеруються інструкції *x q[i]* для повернення керуючих кубітів у початковий стан.

3. Генерація блоку вимірювання – на завершальному етапі транслятор формує блок зчитування результатів. Важливою особливістю є підтримка часткового вимірювання. Система враховує словник circuitDontCareOutputs – виходи, значення яких не є важливими для таблиці істинності, тобто надлишкові лінії. Інструкції measure генеруються лише для значущих кубітів, що дозволяє

оптимізувати процес симуляції та зменшити кількість класичних регістрів у результуючому коді.

Такий підхід забезпечує повну семантичну еквівалентність між внутрішньою моделлю схеми, оптимізованою для еволюційного пошуку, та вихідним кодом, призначеним для виконання на фізичних пристроях.

Методика верифікації синтезованих схем у середовищі Qiskit. Критичним етапом підтвердження наукової достовірності отриманих результатів є незалежна перевірка синтезованих схем. Оскільки основний пошук рішень виконується на власному емуляторі реалізованому мовою C#, існує ризик виникнення системних похибок, пов’язаних з особливостями програмної реалізації. Для усунення цього ризику розроблено модуль перехресної валідації, який використовує індустріальний стандарт квантового моделювання — фреймворк IBM Qiskit.

Процес верифікації реалізовано у середовищі Jupyter Notebook використовуючи мову Python, що дозволяє поєднати обчислювальні експерименти з графічною інтерпретацією результатів.

На відміну від статичного запуску файлів, розроблена методика передбачає динамічну інтеграцію згенерованого QASM-коду в структуру квантового об'єкта. Використовується механізм композиції схем, що дозволяє автоматизувати тестування масивів синтезованих рішень без необхідності ручного втручання.

Програмно це реалізується через парсинг рядка QASM *QuantumCircuit.from_qasm_str* та його вбудовування в цільовий квантовий регістр за допомогою методу композиції *compose*. Такий підхід забезпечує гнучкість при проведенні серійних експериментів, дозволяючи динамічно змінювати параметри ініціалізації кубітів перед виконанням основної логіки схеми.

Важливим науково-технічним аспектом, виявленим у ході інтеграції C# та Python модулів, стала розбіжність у стандартах індексації бітів.

- Внутрішня модель у C# використовує прямий порядок індексації масивів Big-Endian для візуального представлення.
- Бібліотека Qiskit використовує стандарт Little-Endian, де кубіт q_0 відповідає молодшому біту 2^0 у класичному регістрі результатів.

Без врахування цієї особливості результати вимірювань інтерпретуються помилково. Для вирішення проблеми у скрипт верифікації впроваджено алгоритм реверсу бітового рядка вихідних даних. Це гарантує, що вектор стану $|q_n \dots q_0\rangle$, отриманий у Qiskit, коректно відображається на вектор $|x_0 \dots x_n\rangle$ таблиці істинності.

Оскільки синтезовані схеми реалізують зворотні булеві функції, їх поведінка є повністю детермінованою тому достатньо однієї симуляції – shots=1 на ідеальному симуляторі – AerSimulator.

Для аналізу топології отриманих рішень застосовується графічна візуалізація на базі бібліотеки Matplotlib. Це дозволяє візуально оцінити схему та коректність розміщення багатокубітних вентилів зокрема, узагальнених Тоффолі.

Таким чином, розроблений інструментарій забезпечує наскрізну верифікацію результатів дисертаційного дослідження, підтверджуючи їх фізичну реалізовність та логічну коректність.

Висновки до розділу 3

У третьому розділі вирішено задачу розробки спеціалізованого методу та програмних засобів для автоматизованого синтезу зворотних схем. Основні наукові та практичні результати полягають у наступному:

1. Вдосконалення методу синтезу. Запропоновано модифікацію генетичного алгоритму на основі об'єктно-орієнтованої моделі хромосом. Перехід від числових масивів до структурних об'єктів (з параметрами типу вентиля та

ліній) дозволив формалізувати правила комутації та суттєво скоротити простір пошуку рішень.

2. Оператор багатокомпонентної мутації. Розроблено еволюційний оператор, що поєднує структурні зміни (додавання, видалення, модифікація вентилів) та оптимізацію перестановки виходів. Імовірнісне керування цими складовими забезпечує баланс між складністю та коректністю схеми, а динамічна зміна маршрутизації вихідних каналів дозволяє ефективно виходити з локальних екстремумів, зменшуючи квантову вартість.
3. Розширений базис реконфігурованих елементів. Сформовано бібліотеку логічних компонентів на основі модифікованих вентилів Фредкіна. Їх ключовою властивістю є здатність змінювати булеву функцію під дією керуючих сигналів при незмінній топології зв'язків, що створює апаратну основу для синтезу адаптивних шифраторів.
4. Програмна реалізація. Розроблено програмний комплекс мовою C# для синтезу зворотних схем. Реалізовано алгоритм трансляції внутрішнього представлення хромосоми у код стандарту OpenQASM, що забезпечило пряму сумісність із середовищем IBM Qiskit та коректність експорту даних.
5. Верифікація результатів. Реалізовано модуль інтеграції з IBM Qiskit для проведення вичерпного тестування. Симуляція повного перебору вхідних векторів та порівняння вихідних станів з еталонною таблицею істинності математично доводить функціональну коректність синтезованих схем та підтверджує можливість їх фізичної реалізації.

РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ, ВЕРИФІКАЦІЯ ТА ПРАКТИЧНЕ ЗАСТОСУВАННЯ

У попередніх розділах було теоретично обґрунтовано та описано програмну реалізацію автоматизованого синтезу квантових та зворотних схем на основі модифікованого генетичного алгоритму. Ключовою особливістю розробленого підходу є використання спеціалізованих генетичних операторів та фітнес-функції спрямованої на мінімізацію квантової вартості при забезпеченні повної функціональної коректності схеми.

Метою четвертого розділу є комплексна експериментальна верифікація запропонованого методу, оцінка його ефективності порівняно з існуючими аналогами, а також демонстрація практичного застосування синтезованих рішень у задачах криптографічного захисту інформації.

Програма експериментальних досліджень побудована за принципом «від теоретичних моделей до апаратної реалізації» та включає наступні етапи:

1. Синтез еталонних benchmark схем - перевірка збіжності алгоритму та якості отриманих рішень на стандартному наборі тестових функцій, зокрема арифметичних вузлів в базисі вентилів Фредкіна та Тоффолі.
2. Моделювання криптографічних пристроїв – застосування розробленого методу для синтезу зворотних потокових шифраторів та їх верифікація у незалежному середовищі квантового моделювання IBM Qiskit для підтвердження коректності логіки шифрування та дешифрування.
3. Апаратна реалізація – розробка фізичного прототипу пристрою шифрування на базі програмованої логічної інтегральної схеми – FPGA, що підтверджує можливість інтеграції синтезованих схем у реальні обчислювальні системи.

Експерименти проводилися з використанням розробленого програмного комплексу на базі C# .NET, скриптів верифікації за допомогою – Python/Qiskit та засобів апаратного синтезу Verilog HDL.

4.1. Дослідження ефективності генетичного методу на еталонних схемах

Першочерговим завданням експериментальної частини роботи є верифікація розробленого генетичного алгоритму та підтвердження його здатності синтезувати зворотні схеми з мінімальною квантовою вартістю. Для об'єктивної оцінки якості отриманих рішень було обрано методику порівняльного аналізу, що базується на синтезі стандартного набору логічних функцій, загальноприйнятих у науковій спільноті для тестування методів автоматизованого проєктування.

В якості еталонної бази даних використано бібліотеку Reversible Logic Synthesis Benchmarks [63], яка містить специфікації (таблиці істинності) для широкого спектру зворотних функцій: від базових арифметичних операцій до складних функцій керування та кодування. Вибір саме цього набору бенчмарків дозволяє провести пряме зіставлення результатів роботи запропонованого алгоритму з існуючими методами синтезу (Monte Carlo Tree Search, класичні генетичні алгоритми, алгоритми на основі декомпозиції), результати яких опубліковані у провідних фахових виданнях.

Експериментальні дослідження проводилися з використанням розробленого програмного комплексу. Для кожного тестового завдання алгоритм виконував пошук оптимальної топології схеми в заданому базисі вентилів: generalized Toffoli, Fredkin, Feynman, намагаючись мінімізувати фітнес-функцію F – як було описано в розділі 3, що враховує як функціональну помилку, так і структурну складність схеми.

Ключовим критерієм ефективності синтезу визначено квантову вартість. Розрахунок вартості здійснювався за стандартною шкалою, де вартість вентиля NOT дорівнює 1, CNOT — 1, а вартість вентилів Тоффолі та Фредкіна зростає експоненційно або поліноміально залежно від кількості керуючих ліній, що відповідає фізичним витратам на їх реалізацію в реальних квантових обчислювачах [63].

Окрім квантової вартості, аналізувалися такі показники:

1. Кількість вентилів – загальне число логічних операцій у схемі.
2. Кількість надлишкових виходів – кількість додаткових кубітів, необхідних для забезпечення зворотності, стан яких не є частиною корисного результату.
3. Час синтезу та збіжність – кількість поколінь, необхідних алгоритму для знаходження першого валідного рішення та подальшої його оптимізації.

Такий підхід дозволив комплексно оцінити не лише кінцевий результат – топологію схеми, але й динамічні характеристики самого процесу пошуку, зокрема ефективність запропонованих операторів мутації та стратегії селекції.

4.1.1. Методика тестування та набір бенчмарків Reversible Logic Synthesis Benchmarks

Для проведення експериментальних досліджень було обрано набір тестових завдань з відкритого репозиторію Reversible Logic Synthesis Benchmarks Page [63], який є де-факто стандартом для оцінки ефективності методів синтезу зворотних схем. Використання цього набору дозволяє провести пряме порівняння результатів роботи розробленого генетичного алгоритму з альтернативними підходами, такими як алгоритми на основі діаграм рішень, методи трансформації та інші евристичні алгоритми.

До переліку тестових функцій увійшли схеми різної розмірності – від 3 до 6 кубітів та функціонального призначення, що дозволяє оцінити масштабованість алгоритму. Характеристика обраних бенчмарків:

1. Арифметичні схеми – реалізують операції додавання та обчислення за модулем, наприклад rd32 і mod5adder. Ці функції характеризуються жорсткою логічною структурою.
2. Симетричні функції: Схеми, вихід яких залежить лише від ваги Гемінга вхідного вектора (наприклад, sym6, hwb4 — Hidden Weighted Bit). Вони є складними для синтезу через високу заплутаність станів.

3. Логічні функції загального призначення – випадкові перестановки та функції керування – rand_3_1, rand_3_9, що використовуються для перевірки здатності алгоритму знаходити нестандартні топологічні рішення.

Параметри конфігурації генетичного алгоритму Для забезпечення відтворюваності результатів експерименти проводилися з фіксованим набором гіперпараметрів, визначених емпіричним шляхом на етапі попереднього налаштування системи. Основні параметри наведено в Таблиці 4.1.1.1.

Оцінка якості синтезованих схем здійснювалася за моделлю вартості NCT, розширеною для вентилів Фредкіна. Ця метрика відображає кількість елементарних квантових операцій (1-кубітних та 2-кубітних), необхідних для емуляції складного вентиля на фізичному квантовому процесорі. Використані значення вартості [66]:

1. NOT / X: 1
2. CNOT (Feynman): 1
3. Swap: 3 (реалізується через 3 CNOT)
4. Toffoli (CCNOT): 5
5. Fredkin (CSWAP): 5

Для вентилів з кількістю керуючих ліній $n > 2$ (Generalized Toffoli) вартість розраховувалася за квадратичною залежністю, що відповідає стандартним методам декомпозиції Баренка [106].

Кожен експеримент повторювався 20 разів для нівелювання впливу стохастичної природи генетичного алгоритму, після чого обчислювалися середні та найкращі показники.

Таблиця 4.1.1.1. Приклад конфігурації параметрів генетичного алгоритму для серії експериментів

Параметр	Значення/діапазон	Опис
Розмір популяції	100-200	Залежить від кількості входів схеми
Максимальна кількість поколінь	1500	Критерій зупинки
Кількість вхідних ліній	3-7	Варіюється від очікуваної схеми
Ваговий коефіцієнт фітнес функції помилки	$\alpha = 0.9$	Пріоритет функціональної коректності
Ваговий коефіцієнт фітнес функції вартості	$\beta = 0.1$	Пріоритет вартості
Коефіцієнт елітизму	5-7%	Збереження найкращих хромосом без змін
Базова ймовірність мутації	$P_m = 0.05 - 0.15$	Ймовірність мутації гена індивіда
Ймовірність мутації вентиля	$P_{ms} = 0.4$	Зміна лінії комутації
Ймовірність мутації додавання	$P_{ma} = 0.1$	Ймовірність додавання вентиля до топології схеми
Ймовірність мутації видалення	$P_{mr} = 0.5$	Ймовірність видалення вентиля з топології схеми
Ймовірність кросинговеру	$P_c = 0.7$	Використано двохточковий кросинговер
Функція	rand_3_1/rand_3_9 /full_adder	

4.1.2. Синтез арифметичних вузлів: повний суматор в базисі вентилів Фредкіна

Одним із найбільш показових тестів для оцінки ефективності алгоритмів синтезу є реалізація арифметичних операцій, зокрема повного однорозрядного суматора – Full Adder. Цей вузол є фундаментальним елементом будь-якого процесора, і його ефективна реалізація у зворотній логіці має критичне значення для побудови енергоефективних обчислювальних систем.

Повний суматор приймає на вхід три біти A, B, C_{in} та обчислює суму S і біт переносу C_{out} згідно з формулами:

$$S = A \oplus B \oplus C_{in}, \quad (4.1.2.1)$$

$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)), \quad (4.1.2.2)$$

Специфіка синтезу даної схеми у зворотній логіці, особливо при використанні базису Фредкіна (CSWAP), полягає у двох аспектах:

1. Забезпечення зворотності: Оскільки класичний суматор перетворює 3 входи у 2 виходи, втрачається інформація. Для зворотності необхідно додати надлишкові виходи, щоб розмірність схеми становила 3×3 або 5×5 (при використанні константного входу).
2. Збереження парності - вентиль Фредкіна зберігає вагу Гемінга – кількість одиниць на вході дорівнює кількості на виході. Це накладає додаткові обмеження на простір пошуку рішень, роблячи задачу значно складнішою для генетичного алгоритму порівняно з синтезом у базисі Тоффолі.

Для проведення експерименту було налаштовано генетичний алгоритм на використання розширеного набору мутацій:

- $GateType = \{Fredkin\}$. – тип вентилю;
- N = 5 – максимальна довжина хромосоми;
- Q = 5 – максимальна кількість ліній;

- Кількість поколінь G = 1500;
- C_{max} = 100 – максимальний розмір популяції;
- α = 0.9 – коефіцієнт ваги похибки у фітнес-функції;
- β = 0.1 – коефіцієнт ваги вартості (складності) схеми;
- E = 8% – кількість елітних індивідів;
- P_m = 0.1 – імовірність мутації хромосоми;
- P_{ms} = 0.4 – імовірність мутації зміни комутації вентиля;
- P_{ma} = 0.1 – імовірність мутації додавання нового вентиля;
- P_{mr} = 0.5 – імовірність мутації видалення вентиля;
- P_c = 0.7 – імовірність кросоверу;

Фітнес-функція F була налаштована на пріоритетну мінімізацію квантової вартості Q_c після досягнення нульової помилки за відстанню Гемінга.

У ході серії запусків алгоритм успішно синтезував валідну схему суматора. Отримана схема складається з оптимальної комбінації вентилів, що дозволило мінімізувати кількість надлишкових виходів.

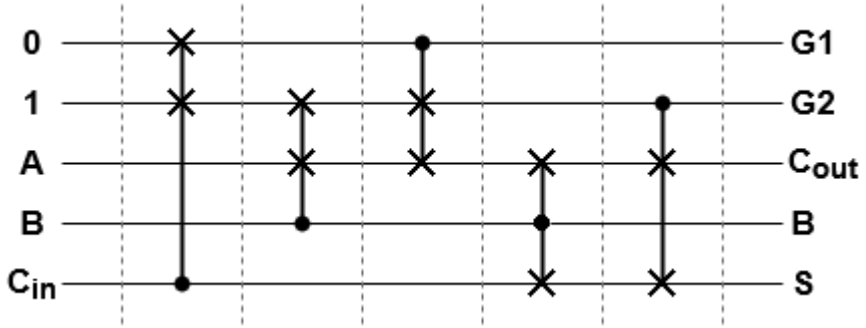


Рис. 4.1.2.1. Варіант схеми повного однорозрядного суматора синтезованого розробленим GA

Експеримент проводився більше десяти разів, причому кожен раз синтезував валідну, схему одна з топологій якої представлено на рисунку 4.1.2.1. Як видно з діаграми, алгоритм автоматично визначив оптимальне розміщення

керуючих ліній для вентилів Фредкіна, що дозволило реалізувати логіку Majority Vote для C_{out} та Parity S з мінімальною глибиною схеми.

Квантова вартість отриманого рішення склала 25 одиниць де $Cost(Fredkin) = 5$. В отриманій схемі вхідні сигнали 0 та 1 є постійними, тоді як вхідні сигнали A, B та C_{in} відповідають вхідним сигналам, згідно з таблицею істинності повного суматора зображеної в таблиці 4.1.2.1. Вихід схеми S представляє суму, C_{out} – вихід, який відповідає переносу, а $G1, G2$ є надлишковими виходами. Причому варто зазначити що $G1$ повністю повторює C_{in}, що додатково допомагає ідентифікувати помилку. Стовпець $G2$ демонструє побічні дані, необхідні для збереження зворотності, які ігноруються при зчитуванні результату.

Таблиця 4.1.2.1. Таблиця істинності синтезованого повного однорозрядного суматора

Вхід					Вихід				
0	1	A	B	C_{in}	G1	G2	C_{out}	B	S
0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	1	0	0	0	1
0	1	0	1	0	0	0	0	1	1
0	1	0	1	1	1	0	1	1	0
0	1	1	0	0	0	1	0	0	1
0	1	1	0	1	1	1	1	0	0
0	1	1	1	0	0	1	1	1	0
0	1	1	1	1	1	0	1	1	1

Ефективність еволюційних операторів досліджено шляхом аналізу зміни найкращого значення фітнес-функції в популяції протягом поколінь. Графік збіжності для задачі синтезу суматора наведено на рисунку 4.1.2.2,

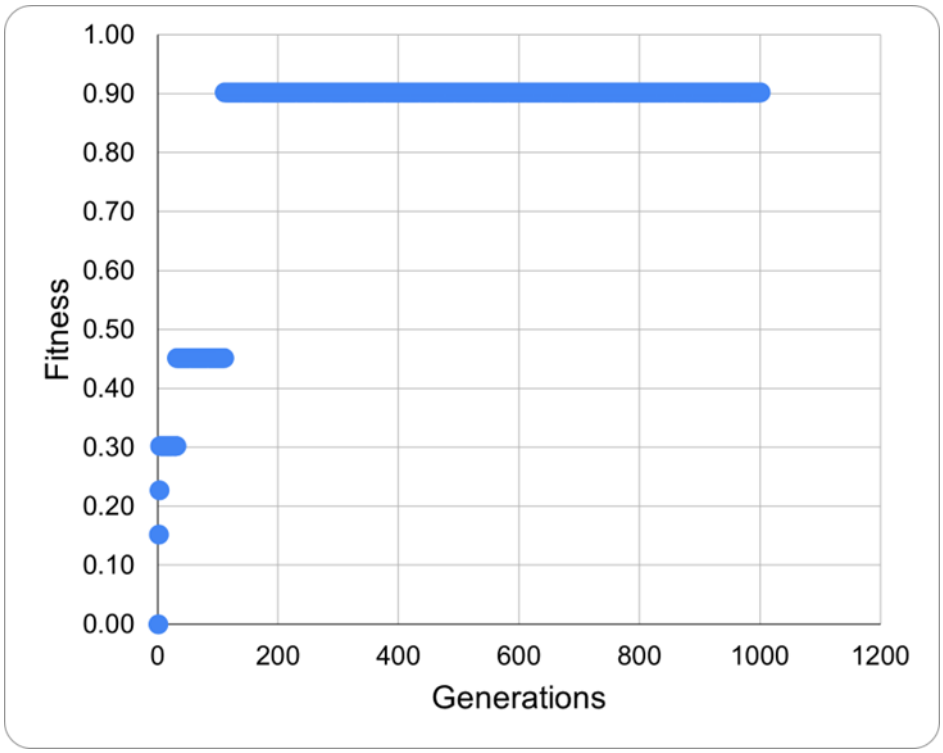


Рис. 4.1.2.2. Динаміка зростання фітнес-функції під час синтезу повного суматора

Аналіз кривої збіжності дозволяє виділити три характерні фази роботи алгоритму:

1. Фаза розвідки (покоління 0–20) – спостерігається стрімке зростання пристосованості. На цьому етапі алгоритм швидко знаходить грубі наближення до рішення, наприклад схеми, що правильно обчислюють суму, але помиляються у переносі.
2. Фаза структурної адаптації (покоління 20–45): Графік демонструє "сходинки" (плато), що чергуються з різкими стрибками. Це свідчить про подолання локальних екстремумів. Стрибки корелюють із успішним застосуванням мутації M_s – зміна лінії комутації, яка дозволяє виправити логіку без суттєвого збільшення вартості.
3. Фаза оптимізації вартості (покоління 100+): Після досягнення $H_d=0$ – коли схема вже працює вірно, фітнес-функція продовжує незначно

зростати. Це результат роботи очищувальних мутацій M_r, які видаляють надлишкові вентилі, зменшуючи Q_c при збереженні функціональності.

Такий характер графіку підтверджує, що впроваджена стратегія адаптивної мутації дозволяє алгоритму не застрягати в локальних екстремумах, забезпечуючи стабільну збіжність до глобального оптимуму за прийнятний час менше 100 поколінь для даного класу задач.

4.1.3. Порівняльний аналіз синтезованих схем з існуючими аналогами

Для верифікації розробленого генетичного алгоритму було проведено двоетапне порівняння. На першому етапі аналізується якість синтезу конкретного арифметичного вузла — повного суматора, з акцентом на архітектурні особливості та відмовостійкість. На другому етапі розглядається ефективність методу на широкому класі еталонних логічних бенчмарк функцій.

Аналіз ефективності синтезу арифметичних вузлів на прикладі повного однорозрядного суматора. Як об'єкт дослідження було обрано схему повного однорозрядного суматора, яка є фундаментальним будівельним блоком для будь-яких арифметико-логічних пристроїв. Порівняння проводилося з результатами двох знакових робіт у галузі зворотної логіки:

1. Дослідження [96] демонструє можливості класичних генетичних алгоритмів для синтезу у базисі вентилів Фредкіна.
2. Робота [107] представляє алгоритмічний підхід до мінімізації схем у базисі NCT.

Результати порівняльного аналізу за критеріями апаратної складності, квантової вартості та швидкості збіжності наведено в Таблиці 4.1.3.1.

Таблиця 4.1.3.1. Порівняння характеристик синтезованих схем повного суматора

Метод синтезу/ джерело	Базис вентилів	К-сть вентилів	Квантова вартість	Швидкість збіжності	Збереження парності
Запропонований GA	Fredkin	5	25	~100	Так
Deibuk & Grytsku [96]	Fredkin	5	25	> 120	Так
Gupta et al. [107]	NCT	4	~12	N/A	Ні

Запропонований у дисертаційній роботі метод успішно відтворив топологію схеми суматора, отриману в роботі Deibuk & Grytsku [96], що складається з 5 вентилів Фредкіна. Це рішення є глобальним оптимумом для даного логічного базису при фіксованій кількості ліній. Ключова перевага розробленого методу полягає у підвищенні обчислювальної ефективності:

- Завдяки впровадженню матричного кодування хромосом, що дозволяє паралельне опрацювання вентилів, та використанню Tabu-списку, який блокує повторний розгляд тупикових гілок еволюції, алгоритм демонструє швидшу збіжність.
- Оптимальне рішення було знайдено вже на 100-му поколінні, що свідчить про високу результативність розроблених операторів мутації. Це дозволяє стверджувати, що запропонований підхід досягає аналогічної якості рішень за менших обчислювальних витрат.

В той самий час алгоритм [107] дозволив синтезувати суматор із 4 вентилів, використовуючи комбінації Toffoli та CNOT. З точки зору формальної мінімізації кількості елементів, цей результат перевершує наше рішення (4 проти 5). Проте, таке пряме порівняння не враховує критично важливу характеристику — відмовостійкість (Fault Tolerance):

- Схема Гупти реалізована у базисі NCT. Цей базис не зберігає парність, що унеможливлює виявлення помилок шляхом простої перевірки парності.

- Схема, синтезована нашим методом, побудована виключно на вентилях Фредкіна. Вона забезпечує суворе збереження парності $P_{in} = P_{out}$ для будь-якої комбінації вхідних сигналів.

У контексті квантових обчислень, де проблема декогеренції та виникнення помилок є ключовим викликом [108], властивість збереження парності виправдовує використання одного додаткового вентиля та підвищення квантової вартості. Таким чином, розроблений алгоритм орієнтований на синтез не просто мінімальних, а надійних схем, придатних для фізичної реалізації на відмовостійких архітектурах.

Оцінка універсальності алгоритму на наборі перестановочних функцій. Якщо на попередньому етапі досліджувалася здатність алгоритму синтезувати структурно-визначені арифметичні вузли, то метою поточного етапу стала верифікація ефективності методу при роботі з довільними логічними функціями, заданими векторами перестановок. Цей клас задач характеризується високою ентропією та відсутністю очевидних закономірностей у таблицях істинності, що висуває підвищені вимоги до пошукових здібностей алгоритму.

Для об'єктивізації отриманих результатів було проведено порівняльний аналіз із двома альтернативними підходами, що репрезентують різні парадигми синтезу:

- Детермінований підхід: Метод MOSAIC Moving Forward Algorithm, запропонований M. Saeedi, M. S. Zamani та M. Sedighi (2008) [118]. Даний метод базується на матричній декомпозиції без використання пошуку і вважається еталоном швидкодії для синтезу в базисі CNOT.
- Еволюційний підхід – адаптивний генетичний алгоритм, представлений T. N. Sasamal et al. (2015) [61]. Ця робота є прямим аналогом, оскільки також використовує еволюційні принципи, проте обмежується стандартним базисом NCT та класичними операторами рекомбінації.
- Еволюційний підхід методом мурашиних колоній – вдосконалений алгоритм мурашиних колоній ACO, розроблений та досліджений [92, 2].

Алгоритм використовує матрицю феромонів для багатокритеріальної навігації у просторі рішень.

Результати експериментального моделювання. Дослідження проводилося на тестовому наборі з 8 функцій перестановок розмірністю 3 біти, які охоплюють різні типи перетворень: від локальних інверсій до циклічних зсувів. Синтез здійснювався у змішаному базисі – NCT та Fredkin.

Порівняльні характеристики апаратної складності отриманих схем наведено в Таблиці 4.1.3.2.

Таблиця 4.1.3.2. Порівняльний аналіз ефективності синтезу на наборі 3-бітних перестановок

№	Вектор перестановки функції	Кількість вентилів				Результат
		Saeedi et al. [109]	Sasamal et al. [61]	ACO [2]	**Запропонований GA**	
1	[0,1,2,3,4,6,5,7]	3	3	1	**3**	-
2	[0,1,2,4,3,5,6,7]	4	5	3	**4**	Паритет з [109]
3	[1,0,3,2,5,7,4,6]	4	4	3	**2**	**Найкращий**
4	[1,2,3,4,5,6,7,0]	4	3	3	**3**	Паритет з [61]
5	[1,2,7,5,6,3,0,4]	6	7	16	**9**	-
6	[3,6,2,5,7,1,0,4]	6	7	12	**8**	-
7	[4,3,0,2,7,5,6,1]	5	6	9	**6**	Паритет з [61]
8	[7,0,1,2,3,4,5,6]	3	3	5	**3**	Паритет

* - відповідні схеми наведені в додатку А

Аналізуючи результати Таблиці 4.1.3.2, можна виділити чітку залежність між складністю функції та ефективністю різних еволюційних підходів. По-перше, для функцій з низькою ентропією (№1 та №2) ACO продемонстрував абсолютну перевагу, знайшовши рішення всього за 1 та 3 вентилі відповідно, що перевершує як існуючі аналоги, так і розроблений GA. Це пояснюється тим, що феромонна модель ACO дозволяє надзвичайно швидко знаходити оптимальні

короткі шляхи у невеликих просторах пошуку. По-друге, із зростанням складності перетворень ситуація кардинально змінюється. Найбільш показовим є результат для функції №3 [1,0,3,2,5,7,4,6], яка представляє собою попарний обмін сусідніх значень.

- Альтернативні методи (Saeedi, Sasamal), обмежені базисом NCT/CNOT, реалізують цю функцію за 4 вентилі.
- Запропонований алгоритм синтезував схему всього за 2 вентилі. Це підтверджує, що використання змішаного базису дозволяє генетичному алгоритму знаходити структурні скорочення, недоступні для методів, що працюють у фіксованому базисі. В даному випадку, вентилі Фредкіна виявилися топологічно більш відповідними для структури перестановки, ніж каскади CNOT.

Для векторів, що описують функції зсувів №1, №4, №8 представлених в таблиці 4.1.3.2, розроблений метод досяг результатів, ідентичних найкращим показникам аналогів в 3 вентилі. Це свідчить про те, що алгоритм успішно знаходить глобальний оптимум для функцій з низькою ентропією, не поступаючись спеціалізованим детермінованим методам.

Обмеження стохастичного пошуку на складних перестановках. Для функцій №5 та №6 з таблиці 4.1.3.2 спостерігається незначне зростання апаратної складності рішень в 8-9 вентилів порівняно з детермінованим методом Saeedi в 6 вентилів. Це пояснюється природою алгоритмів: метод MOSAIC [109] є конструктивним і математично гарантує певний верхній поріг складності. Генетичний алгоритм є пошуковим; при відсутності очевидної симетрії у функції простір пошуку стає складним для навігації, що може призводити до знаходження субоптимальних, хоча й коректних рішень. Водночас, порівняно з прямим конкурентом — адаптивним GA Sasamal [61], розроблений метод демонструє близькі результати – різниця в 1-2 вентилі, що є прийнятним для класу NP-складних задач.

Наведені експерименти підтверджують гіпотезу дисертації:

- Розширення базису синтезу NCT + Fredkin у поєднанні з еволюційним пошуком дозволяє скоротити складність схем до 50% для певних класів функцій (симетричні перестановки, попарні обміни).
- Порівняння методів штучного інтелекту підтвердило, що генетичний алгоритм з об'єктним кодуванням є більш масштабованим та ефективним для синтезу складних квантових функцій, тоді як алгоритм мурашиних колоній доцільно застосовувати для швидкого синтезу локальних вузлів з мінімальною глибиною.
- Алгоритм стабільно знаходить оптимальні рішення для стандартних вузлів таких як суматори.
- Запропонований підхід є життєздатною альтернативою існуючим методам, забезпечуючи баланс між мінімізацією вартості та адаптивністю до різних логічних базисів.

4.2. Розробка та верифікація зворотних шифраторів у середовищі IBM Qiskit

Отримання оптимальної топології схеми за допомогою генетичного алгоритму є необхідним, але недостатнім етапом для створення повноцінного криптографічного пристрою. Критично важливою умовою є перевірка придатності синтезованих структур для реалізації на архітектурі квантових обчислювачів, а також підтвердження їх логічної коректності та криптографічних властивостей в умовах, наближених до фізичних.

У даному підрозділі описано розроблений програмний комплекс, призначений для автоматизації процесу імплементації синтезованих рішень. Комплекс забезпечує трансляцію результатів роботи генетичного алгоритму у формат квантових інструкцій, верифікацію таблиць істинності та моделювання роботи потокового шифратора на реальних даних.

В якості базового інструментального середовища для моделювання обрано IBM Qiskit SDK (Quantum Information Science Kit) [110]. Вибір даного фреймворку зумовлений наступними факторами:

1. Підтримка низькорівневого моделювання – Qiskit надає доступ до симулятора AerSimulator, який дозволяє аналізувати унітарні матриці перетворень та вектори станів *statevectors*, що є критичним для перевірки зворотності схем.
2. Індустріальний стандарт – використання стандартних бібліотек та формату OpenQASM [110] забезпечує переносимість розроблених рішень та можливість їх майбутнього запуску на реальних квантових процесорах.

Загальна схема проведених експериментальних досліджень включає чотири етапи: трансляцію топології у квантовий код, валідацію логіки на векторах станів, побудову архітектури шифратора та аналіз його стійкості до колізій у просторі ключів.

4.2.1 Трансформація хромосом генетичного алгоритму у формат квантових інструкцій OpenQASM

Ключовим етапом переходу від теоретичного синтезу до практичного моделювання є автоматизована трансляція абстрактного представлення схеми у виконуваний код. Генетичний алгоритм оперує хромосомами — числовими векторами або матрицями, які кодують топологію пристрою, проте не є зрозумілими для квантових симуляторів.

Для розв'язання цієї задачі розроблено модуль-транслятор – Mapper, який забезпечує конвертацію результатів оптимізації у формат OpenQASM – Open Quantum Assembly Language. Вибір QASM як цільового формату зумовлений його універсальністю: він дозволяє описувати квантові схеми незалежно від фізичної реалізації кубітів та підтримується більшістю сучасних фреймворків, включаючи IBM Qiskit, Google Cirq та Rigetti Forest.

Процес трансформації базується на детермінованому відображенні функціональних примітивів генетичного алгоритму на набір стандартних інструкцій бібліотеки qelib1.inc для версії OpenQASM 2.0 або stdgates.inc для OpenQASM 3.0.

1. Синтезована хромосома – схема Ch – впорядкований масив генів, де кожен ген g_i описує один логічний вентиль. Ген містить ідентифікатор типу вентиля та індекси кубітів, на які він діє.
2. Розрядність схеми – N – кількість ліній, тобто кубітів, необхідних для реалізації функції.

Формалізований алгоритм трансляції представлено нижче.

1. Ініціалізація:
    - Сформувати заголовок файлу QASM: OPENQASM 3.0; include "stdgates.inc";.
    - Оголосити квантовий регістр qreg q[N] та класичний регістр creg c[N] для запису результатів вимірювань.
2. Аналіз необхідності додавання узагальнених вентилів, не представлених в існуючій бібліотеці OPENQASM 3.0:
    - Перевірка наявності у схемі складних узагальнених вентилів, наприклад, подвійно контрольованого вентиля Фредкіна - G2Fredkin або потрійного Тоффолі - G3Toffoli.
    - Якщо такі елементи присутні, генерується блок визначення гейту (gate definition) з використанням модифікатора ctrl @, наприклад: *gate c2swap a, b, c, d { ctrl @ cswap a, b, c, d; }*.
3. Ітеративна генерація інструкцій:
    - Для кожного такту схеми групуються вентилі за їх унікальним ідентифікатором – *Uid*.
    - Визначається тип вентиля та його відповідник у QASM згідно словника маппингу – *GateTypeMappings*.

4. Обробка полярності керування – Control Logic:
    - Розділення керуючих ліній на позитивні - *Positive* та негативні *Negative*.
    - Пре-інверсія – якщо присутні негативні керуючі лінії, для них генеруються інструкції *NOT* перед виконанням основного вентиля.
    - Виклик вентиля – формування інструкції виклику, наприклад: ccx, cswap з передачею списку позитивних, негативних (вже інвертованих) та цільових ліній.
    - Пост-інверсія – повторна генерація інструкцій *NOT* для негативних ліній для відновлення початкового стану – зворотне перетворення.
5. Фіналізація – Measurement:
    - Генерація інструкцій вимірювання measure для всіх кубітів, які не позначені як "DontCare", із записом результату в класичний регістр.

Логіка конвертера реалізована у середовищі .NET. На рисунку 4.2.1.1 наведено ключовий фрагмент методу *ConvertToOpenQasm*, що демонструє обробку негативних керуючих зв'язків.

Такий підхід дозволяє коректно реалізувати логіку вентилів зі змішаним керуванням у базисі стандартних вентилів QASM, забезпечуючи точну відповідність синтезованої схеми її програмній моделі. Згенерований файл .qasm надалі передається до Python-скриптів для симуляції у Qiskit.

```
foreach (var quantumState in quantumCircuit.Circuit) {
    var gates = quantumState.GateLines.GroupBy(t => t.Gate.Uid);

    foreach (var groupedGate in gates) {
        var gate = groupedGate.FirstOrDefault()?.Gate;
        if (gate == null || gate.Name == GateType.Identity)
            continue;

        var gateName = gateTypeMappings[gate.Name];
        var positiveControlledLines = groupedGate.Where(t => t.ControlType == ControlType.Positive)
            .Select(t => $"q[{quantumState.GateLines.IndexOf(t)}]");
        var negativeControlledLines = groupedGate.Where(t => t.ControlType == ControlType.Negative)
            .Select(t => $"q[{quantumState.GateLines.IndexOf(t)}]");
        var dataLines = groupedGate.Where(t => t.ControlType == ControlType.None)
            .Select(t => $"q[{quantumState.GateLines.IndexOf(t)}]");

        if (negativeControlledLines.Any()) {
            foreach (var controlledLine in negativeControlledLines) {
                sb.Append("x "); sb.Append(controlledLine); sb.AppendLine(";");
            }
        }

        sb.Append($"{gateName} ");

        if (positiveControlledLines.Any())
        {
            sb.AppendJoin(", ",positiveControlledLines); sb.Append(", ");
        }

        if (negativeControlledLines.Any())
        {
            sb.AppendJoin(", ",negativeControlledLines); sb.Append(", ");
        }

        sb.AppendJoin(", ",dataLines); sb.AppendLine(";");

        if (negativeControlledLines.Any()) {
            foreach (var controlledLine in negativeControlledLines) {
                sb.Append("x "); sb.Append(controlledLine); sb.AppendLine(";");
            }
        }
    }
}
```

Рис. 4.2.1.1. C# фрагмент логіки конвертації гена в QASM представлення схеми

4.2.2. Автоматизоване тестування логічної коректності схем методом симуляції векторів станів

Після трансляції топології схеми у формат OpenQASM наступним критичним етапом є верифікація її функціональної відповідності заданій таблиці істинності. Оскільки синтезовані зворотні схеми призначені для роботи як з квантовими, так і з класичними даними, первинна перевірка виконується на множині базисних станів обчислювального базису ($|0\ldots0\rangle \ldots |1\ldots1\rangle$).

Для реалізації цього етапу розроблено модуль *CircuitEvaluation* програмний код наведений в додатку В, який функціонує як автоматизований

випробувальний стенд – Test Bench у середовищі Python з використанням бібліотеки Qiskit. Фрагмент програмного коду модуля представлено на рисунку 4.2.2.1.

Процедура перевірки базується на властивості детермінованості зворотних схем при роботі з класичними вхідними векторами. Це дозволяє уникнути необхідності набору великої кількості статистики для кожного входу, зводячи перевірку до одного тестового запуску схеми для кожної можливої комбінації вхідних значень.

Алгоритм тестування реалізує повний перебір простору вхідних даних і складається з наступних кроків:

1. Завантаження схеми – зчитування згенерованого файлу .qasm або об'єкта QuantumCircuit.
2. Генерація вхідного простору – формування повного набору бінарних рядків довжиною *N*, де *N* — розрядність схеми, від 0 до $2^N - 1$.
3. Ітеративна симуляція - для кожного вхідного вектора V_{in}:
    - Створення нового екземпляра квантової схеми.
    - Ініціалізація кубітів у стан, що відповідає V_{in}, за допомогою інструкції *initialize* (або серії вентилів *x* для встановлення одиниць).
    - Приєднання основної логіки тестованої схеми.
    - Додавання вимірювань – measure для всіх вихідних ліній.
    - Виконання симуляції на бекенді *AerSimulator* з параметром *shots=1*.
4. Аналіз результатів:
    - Отримання вихідного бітового рядка V_{out} з результатів вимірювання.
    - Порівняння пари V_{in} та V_{out} з еталонним значенням з таблиці істинності.
5. Візуалізація – побудова гістограми розподілу вихідних станів для підтвердження відсутності побічних ефектів (шуму або помилкових переходів).

```
def evaluate_circuit(qc_base, num_qubits, simulator):
    # Перебір усіх можливих вхідних станів (від 00..0 до 11..1)
    for i in range(2**num_qubits):
        # Формування бінарного рядка (input state)
        input_state_str = format(i, f'0{num_qubits}b')

        # Створення нової схеми для поточного тесту
        qc_test = QuantumCircuit(num_qubits, num_qubits)

        # Ініціалізація вхідного стану
        qc_test.initialize(input_state_str, qc_test.qubits)

        # Додавання логіки синтезованої схеми
        qc_test.compose(qc_base, inplace=True)

        # Вимірювання
        qc_test.measure_all()

        # Симуляція (shots=1 достатньо для детермінованої схеми)
        job = simulator.run(qc_test, shots=1)
        result = job.result()
        counts = result.get_counts()

        # Отримання вихідного стану (ключ з найбільшою кількістю відліків)
        output_state = next(iter(counts.keys()))

        print(f"Input: {input_state_str} | Output: {output_state}")
```

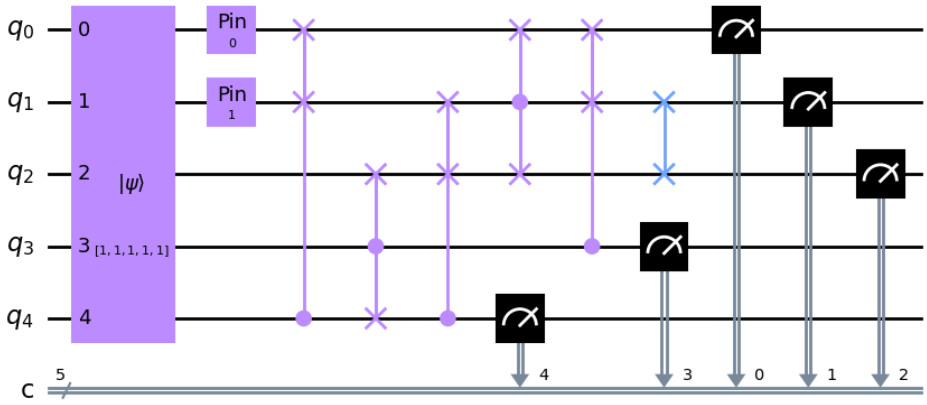
Рис. 4.2.2.1. Python фрагмент коду модуля *CircuitEvaluation*

Реалізація даного алгоритму тестування виконана з використанням бібліотеки Qiskit фрагмент якого представлений на рисунку 4.2.2.1 Особливістю реалізації є використання методу *initialize*, який автоматично переводить кубіти у потрібний стан перед початком обчислень.

За результатами роботи модуля *CircuitEvaluation* формується протокол відповідності. Для синтезованих у розділі 4.1 схем бенчмарків та арифметичних вузлів було отримано повне співпадіння вихідних векторів з очікуваними значеннями.

На Рис. 4.2.2.2 наведено приклад схеми, згенерованої модулем *circuit_drawer* для одного з тестових випадків. Отримані результати вхідних та вихідних векторів підтверджують, що синтезована схема працює як

детермінована булева функція без виникнення суперпозиційних артефактів у класичному режимі роботи.




```
Input: 00000 | Output: 00000
Input: 00001 | Output: 00001
Input: 00010 | Output: 00010
...
Input: 11101 | Output: 11101
Input: 11110 | Output: 01111
Input: 11111 | Output: 11111
```

Рис 4.2.2.1 Результат роботи скрипта аналізатора

Таким чином, розроблений інструментарій дозволяє автоматично верифікувати коректність трансляції та логіки роботи схеми перед її використанням у більш складних криптографічних конструкціях.

4.2.3. Архітектура потокового шифрування текстових даних на основі каскадування зворотних вузлів

Отримані за допомогою генетичного алгоритму зворотні схеми мають обмежену розрядність (зазвичай 3–6 кубітів), що зумовлено експоненціальним зростанням простору пошуку. Однак для практичного застосування у системах захисту інформації необхідно оперувати даними значно більшої розмірності (наприклад, символами стандарту ASCII — 8 біт, або блоками Unicode — 16 біт).

Для вирішення проблеми масштабування було запропоновано та реалізовано архітектуру каскадного включення зворотних модулів. Даний підхід, а також результати синтезу базових криптографічних вузлів, були представлені автором на міжнародній конференції AIMEE2024 [4].

Суть запропонованого методу полягає у декомпозиції задачі шифрування: замість синтезу однієї складної схеми великої розмірності використовується композиція оптимізованих малорозрядних вузлів. Ключові особливості архітектури:

1. Усі входи квантової схеми поділяються на два класи:
    - Інформаційні лінії (Data Lines, D) – несуть бінарний код повідомлення, що підлягає шифруванню.
    - Керуючі лінії (Key Lines, K) - несуть значення секретного ключа, який визначає конкретну перестановку інформаційних бітів.
2. Каскадування - для шифрування 8-бітного символу ASCII використовується паралельне з'єднання синтезованих блоків. Наприклад, схема може бути побудована з двох 4-бітних зворотних модулів, кожен з яких обробляє частину інформаційного слова під керуванням своєї частини ключа.
3. Властивість "Lossless": Завдяки використанню виключно зворотних вентилів (Toffoli, Fredkin, CNOT), система гарантує відсутність втрати інформації, що є критичним для криптографії.

Для експериментальної перевірки розробленої архітектури було створено програмний модуль *Key_Analyser* наведений в додатку В, який реалізує повний цикл захищеної передачі текстових даних. Процес обробки складається з наступних етапів:

Етап 1. Попередня обробка - вхідна секретна фраза *SecretPhrase*, задана користувачем у вигляді рядка символів, конвертується у числовий формат. Кожен символ рядка перетворюється у відповідний 8-бітний код згідно таблиці ASCII.

$$S = \{char_1, char_2, \dots, char_n\} \xrightarrow{ASCII} \{d_1, d_2, \dots, d_n\}, d_i \in [0{,}255] \quad (4.2.3.1)$$

Етап 2. Квантове шифрування – для кожного символу d_i формується квантовий ланцюг. На вхідні регістри подається:

- Значення даних d_i – через ініціалізацію кубітів у стани |0⟩ та |1⟩.
- Значення ключа K – фіксований набір бітів для поточної сесії.

Виконується пряме перетворення U_{enc}:

$$|\, C_i\rangle = U_{enc}(|\, di\rangle \otimes |\, K\rangle) \quad (4.2.3.2)$$

Результат вимірювання стану $|C_i\rangle$ формує зашифрований символ. У контексті симулятора це відповідає отриманню бітового рядка, що є результатом перестановки.

Етап 3. Реверсне дешифрування – отриманий шифротекст подається на вхід інверсної схеми. У зворотній логіці операція дешифрування реалізується шляхом виконання тієї ж послідовності вентилів, що і при шифруванні, але у зворотному порядку, за умови що схема не є інволютивна, (для унітарних операторів (AB)†=B†A†).

$$|\, d_i'\rangle = U_{dec}(|\, C_i\rangle \otimes |\, K\rangle), \text{де } U_{dec} = U_{enc}^{\dagger} \quad (4.2.3.3)$$

Етап 4. Пост-обробка та верифікація – отримані значення d_i' конвертуються з бінарного коду назад у символи ASCII, формуючи розшифровану фразу – *DecodedPhrase*.

Фінальним кроком є автоматична перевірка цілісності даних. Програмний алгоритм порівнює вихідний рядок з початковою секретною фразою:

$$IsSuccess = (OriginalPhrase == DecodedPhrase) \quad (4.2.3.4)$$

Під час проведення експериментів у середовищі Qiskit для всіх синтезованих схем (за умови використання коректних ключів) було досягнуто повної ідентичності вхідної та вихідної фраз. Це підтверджує, по-перше, правильність синтезованої топології, а по-друге — ефективність запропонованого методу каскадування для обробки реальних текстових масивів.

4.2.4. Проєктування та структурний синтез реконфігурованих зворотних шифраторів на базі елементів RRG2 та RRG3

Окремим напрямком досліджень у даній роботі стала розробка спеціалізованих криптографічних вузлів, які поєднують у собі властивості зворотності та реконфігурованості.

Використання розширеного базису узагальнених вентилів Фредкіна, теоретично обґрунтованого у третьому розділі, є фундаментальною основою для побудови цих пристроїв. Головна проблема класичних підходів до апаратного шифрування полягає у жорсткій фіксації логіки перетворень та високих енерговитратах на незворотні операції. Суть запропонованого методу шифрування полягає у використанні реконфігураційної здатності вентилівяк основного криптографічного механізму [3]:

1. Керуючі сигнали як ключ - конфігураційні входи підключаються до генератора ключового потоку, і поточне значення ключа безпосередньо обирає одну з 32 можливих функцій вентиля для поточного такту.
2. Динамічне перемішування - оскільки логіка перетворень змінюється динамічно, проходження блоку даних через каскад таких RRG-елементів забезпечує глибоке нелінійне перемішування.
3. Апаратна зворотність - процес розшифрування здійснюється шляхом пропускання шифротексту через ту саму схему з відповідним ключем керування.

4. Захист від атак – властивість консервативності вентилів Фредкіна гарантує, що будь-яка індукована помилка миттєво порушує парність сигналів, дозволяючи детектувати апаратне втручання.

У цьому підрозділі описано схемотехнічну реалізацію запропонованих базових елементів RRG2 та RRG3 – Reconfigurable Reversible Gates, принцип побудови масштабованих шифраторів на їх основі, а також результати часового моделювання.

Як базовий будівельний блок було обрано вентиль Фредкіна та його розширені модифікації. На відміну від стандартного базису NCT, вентиль Фредкіна забезпечує збереження паритету, що є критичним для побудови відмовостійких схем. На Рис. 4.2.4.1 наведено порівняння топології відомого аналога та запропонованих рішень:

1. RRG1 – відомий аналог, побудований в базисі NCT. Використовує 3 допоміжні лінії та має високу глибину схеми [89].
2. RRG2 – синтезована схема в базисі реконфігурованих вентилів Фредкіна. Схема містить вхідні інвертори (b0–b2), контрольовані інвертори полярності (b3, b4) та основний комутаційний елемент (b5). Ця конфігурація дозволила відмовитись від постійних входів, суттєво зменшивши квантову вартість.
3. RRG3 – модифікація RRG2 з функцією відмовостійкості. Додавання елементів c1-c6 та використання допоміжних ліній дозволило забезпечити властивість збереження парності вхідних даних – $P_{in} = P_{out}$, що дозволяє детектувати помилки. Перевагою RRG3 є можливість використовувати асинхронні ключі шифрування $K_{enc} \neq K_{dec}$ де ключі шифрування K_{enc} відрізняються від ключів дешифрування K_{dec}. На сьогодні є першою відомою реалізацією асиметричного шифрування на базі зворотних схем.

Для шифрування даних довільної розрядності розроблено каскадну архітектуру RE – Reversible Encryptor. На рисунку 4.2.4.1 зображено структуру 4-бітного модуля шифрування, що складається з паралельних гілок.

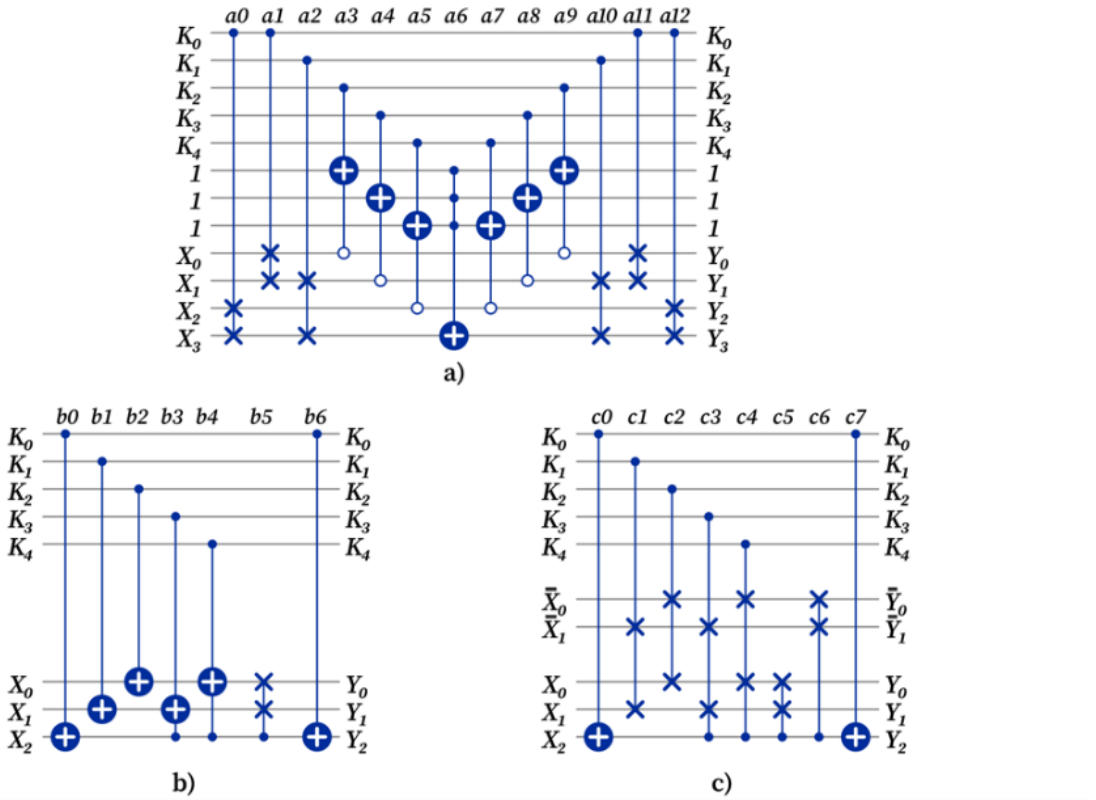


Рис. 4.2.4.1 Зворотні реконфігуровані вентилі: a) RRG1 [89], b) шифратор з мінімізованою квантовою вартістю RRG2 [4]; c) новий RRG3 зі збереженням парності [4];

Принцип дії полягає у наступному:

1. Вхідне повідомлення розбивається на сегменти по 4 біти.
2. Кожен сегмент обробляється відповідним модулем RRG, конфігурація якого задається частиною секретного ключа.
3. Каскадне з'єднання, зображене на рисунку 4.2.4.2, дозволяє масштабувати схему для обробки ключів довжиною 40, 80 і більше біт без втрати швидкодії, оскільки операції в гілках виконуються паралельно.

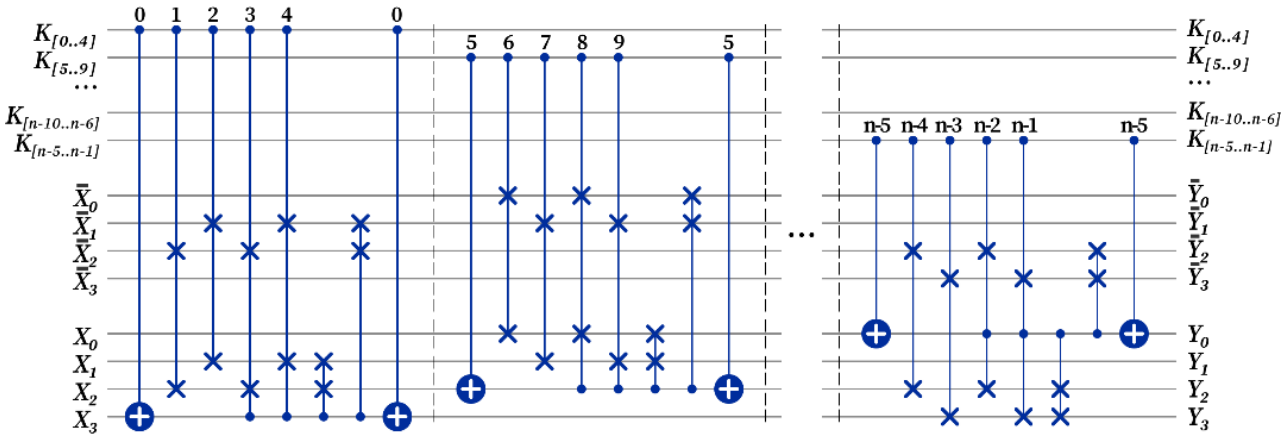


Рис. 4.2.4.2 Реконфігурований 4-бітний шифратор на основі реконфігурованих вентилів Фредкіна зі збереженням парності (RE3) для n бітів ключа

Таблиця 4.2.4.1. Порівняльні характеристики зворотних шифраторів (для ключа n=40 біт)

Схема	**RRG1 [89]**	**RRG2 [4]**	**RRG3 [4]**	**RE1 [89]**	**RE2**	**RE3**
Біти ключів	5	5	5	80	80	80
Біти данних	4	3	3	8	8	8
Додаткові входи	3	0	2	3	0	8
Кількість RRG	1	1	1	16	16	16
Реалізує функцій	32	32	32	2^{80}	2^{80}	2^{80}
Квантова вартість	79	19	48	1264	304	768
Глибина схеми	9	5	6	72	33	34
Збереження парності	Ні	Ні	Так	Ні	Ні	Так

Аналіз метрик синтезованих схем RE2 та RE3 у порівнянні з аналогом RE1 підтвердив значну перевагу запропонованого підходу результати якого продемонстровано в таблиці 4.2.4.1.

Шифратор RE2 демонструє зниження квантової вартості на 76% та зменшення затримки на 54% порівняно з RE1 на базі RRG1 [89], що робить його оптимальним для застосування в умовах обмежених обчислювальних ресурсів.

Для підтвердження коректності логіки роботи шифратора було проведено симуляцію у середовищі Active-HDL [111]. На Рис. 4.2.4.3 наведено часові діаграми повного циклу "шифрування-дешифрування" для схеми на базі RRG3. На діаграмі позначено:

1. X – вхідний вектор даних.
2. A – вхідний вектор постійних входів.
3. K_{enc}, K_{dec} – ключі шифрування та дешифрування.
4. A_{dec}, A_{enc} – постійні входи після шифрування та дешифрування
5. X_{enc}, X_{decr} – зашифроване та відновлене повідомлення.

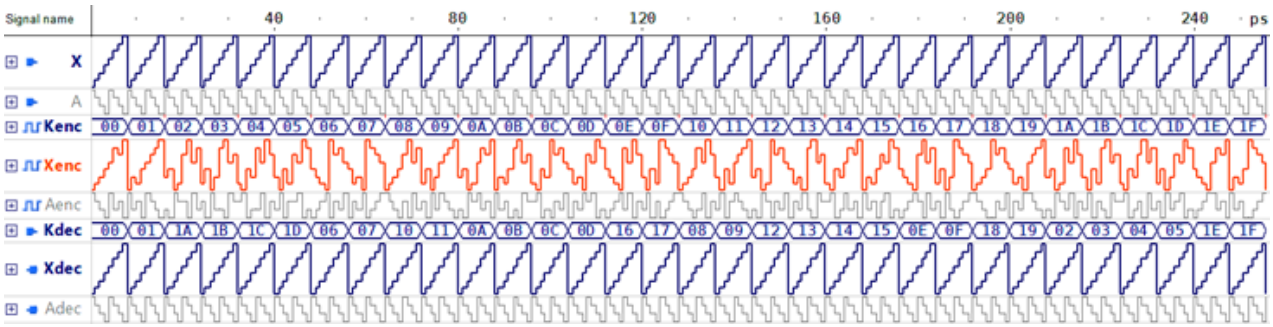


Рисунок 4.2.4.3. Часова діаграми одного цикл (256 пс) моделювання для реконфігурованого розширеного вентиля Фредкіна зі збереженням парності.

Як видно з діаграми, при подачі коректної пари ключів вихідний сигнал повністю співпадає з вхідним – $X = X_{decr}$, а допоміжні лінії повертаються у початковий стан, що підтверджує зворотність схеми та відсутність втрат інформації. Також діаграма візуально демонструє наявність як симетричних – $K_{enc} = K_{dec}$, так і асиметричних режимів роботи.

4.2.5. Автоматизований аналіз простору ключів та колізій методами квантової симуляції

Після верифікації топології окремих вентилів та структурного синтезу шифратора критично важливим етапом є перевірка його функціонування на

реальних потоках даних. Для цього було розроблено програмний комплекс Key_Analyser на базі Python та Qiskit, який емулює роботу шифратора в режимі "чорної скриньки", аналізуючи відповідність вхідних та вихідних даних. Основні завдання цього етапу:

1. Перевірка цілісності даних при проходженні повного циклу шифрування-дешифрування.
2. Дослідження простору ключів на наявність слабких ключів або колізій.
3. Класифікація ключів за типом симетрії.

Розроблений алгоритм працює за ітеративним принципом, перебираючи можливі комбінації ключів для заданого вхідного повідомлення *SecretPhrase*. Логіку роботи програмного засобу представлено у вигляді блок-схеми на Рис. 4.2.5.1, а опис виглядає наступним чином:

1. Ініціалізація даних:
   - Користувач задає секретну фразу (наприклад, "TEST").
   - Програма конвертує кожен символ рядка у відповідний 8-бітний код згідно таблиці ASCII:

$$Char \xrightarrow{ASCII} \{b7, b6, \dots, b0\} \qquad (4.2.5.1)$$

2. Генерація ключів:
   - Запускається цикл перебору ключів K_i з допустимого простору, для тестування використовується підмножина ключів або повний перебір для малорозрядних версій.
3. Квантова емуляція:
   - Крок шифрування – формується квантова схема, де кубіти ініціалізуються значеннями X та ключа K_i. Виконується симуляція. Результат шифрування зберігається як X_{enc}.

- Крок дешифрування – створюється нова схема. Вхідні кубіти ініціалізуються значенням X_{enc} та ключем K_{enc}. Виконується операція дешифрування. Результат зберігається в X_{decr}.

4. Валідація і логування:
   - Виконується побітове порівняння: $X == X_{decr}$.
   - Якщо рівність виконується, ключ K_i маркується як "Valid".
   - Результати записуються у лог-файли: Working_keys.log та Redundant_keys.log).

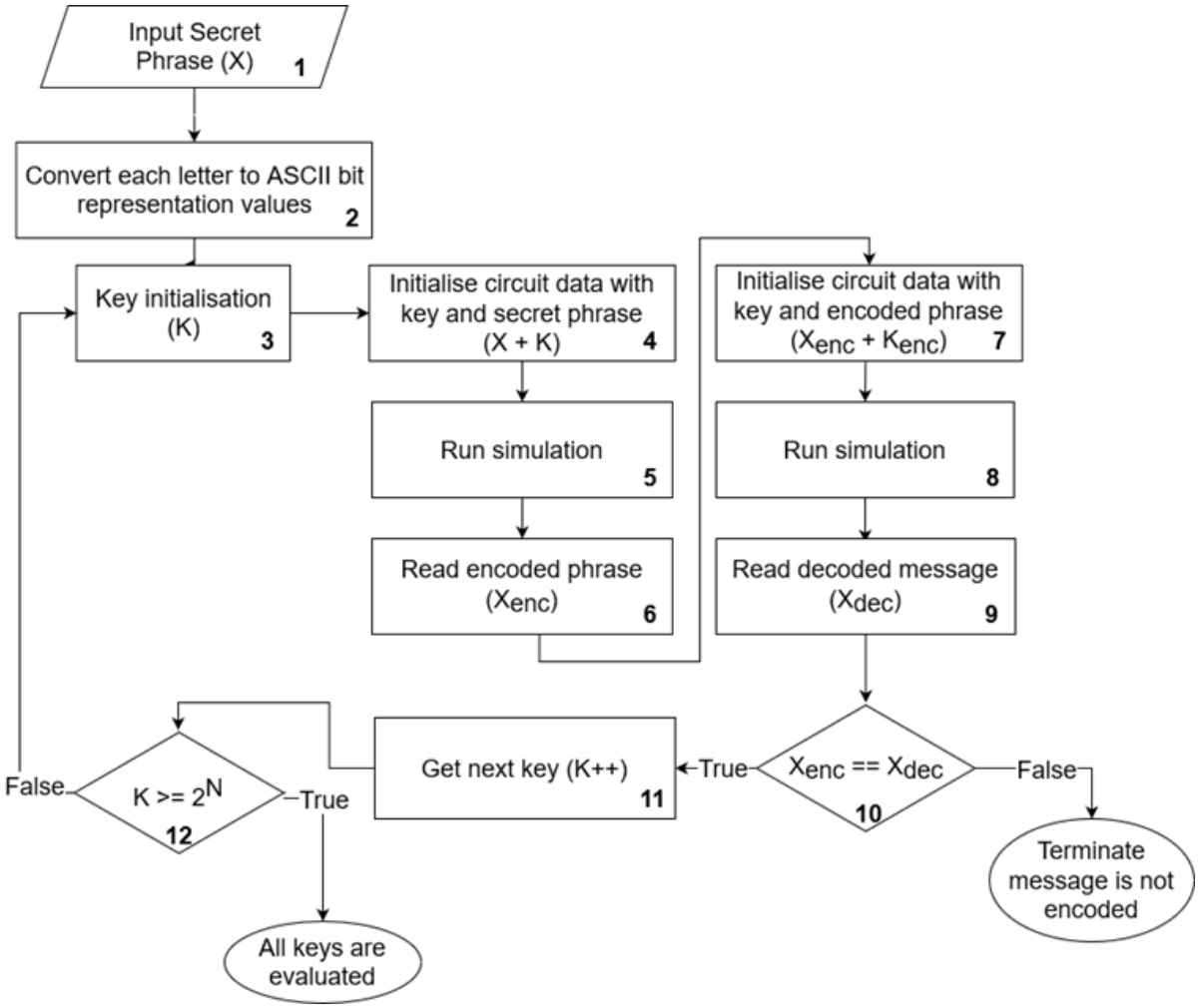


Рис 4.2.5.1. Алгоритм роботи модуля Key_Analyser

Аналіз лог-файлів, отриманих після повного перебору простору ключів для шифратора RE2 та RE3, дозволив виявити важливу закономірність,

візуалізовану на рисунку. 4.2.4.3. Встановлено, що множина валідних ключів чітко розділяється на дві підмножини рівної потужності:

1. Симетричні ключі:

$$Enc(X, K) \to X_{enc} \Longrightarrow Dec(X_{enc}, K) \to X \qquad (4.2.5.2)$$

У цьому режимі один і той самий ключ використовується для обох операцій. Це дозволяє використовувати розроблений пристрій як класичний блоковий шифр.

2. Асиметричні ключі:

$$Enc(X, K) \to X_{enc} \Longrightarrow Dec(X_{enc}, K) \neq X \qquad (4.2.5.3)$$

Для успішного дешифрування необхідно застосувати інверсний (комплементарний) ключ K_{enc}:

$$Dec(X_{enc}, K_{enc}) \to X, \text{де } K_{enc} \neq K \qquad (4.2.5.4)$$

Ця властивість є унікальною для запропонованої архітектури RRG і відкриває можливості для побудови схем з розділеним доступом.

4.3 Апаратна реалізація шифратора RRG3 на базі FPGA

Попередні етапи дослідження, описані у розділах 4.1 та 4.2, фокусувалися на алгоритмічному синтезі та програмному моделюванні зворотних схем. Однак, більшість сучасних досліджень у цій галузі залишаються теоретичними, обмежуючись симуляціями [112] без підтвердження їх працездатності на фізичному рівні.

Водночас, актуальність впровадження зворотної логіки зумовлена наближенням класичної CMOS-технології до фундаментальних термодинамічних меж. Для подолання розриву між теоретичними моделями та практичним застосуванням, у даній роботі здійснено апаратну імплементацію розробленого відмовостійкого шифратора RRG3.

В якості цільової платформи для прототипування обрано технологію FPGA. Використання налагоджувальної плати на базі чіпа Altera Cyclone IV версії EP4CE6E22C8N дозволило:

1. Реалізувати схему на рівні конфігурованих логічних блоків, що максимально наближено до створення спеціалізованої інтегральної схеми.
2. Забезпечити роботу пристрою в реальному часі з апаратною підтримкою паралелізму, властивого запропонованій каскадній архітектурі.

Розроблений апаратний комплекс не обмежується реалізацією лише криптографічного ядра. Це повноцінна інтерактивна система, що інтегрує інтерфейси введення – PS/2 клавіатуру та візуалізації за допомогою монітора через VGA інтерфейс. Такий підхід дозволяє провести побудувати пристрій і провести верифікацію роботи шифратора, демонструючи його придатність для обробки реальних потоків даних та стійкість до фізичних несправностей.

4.3.1 Структурна організація та Verilog-опис ядра (RRG)

Програмний опис апаратної частини виконано мовою опису апаратури Verilog HDL. Архітектура проєкту побудована за модульним ієрархічним принципом. Це дозволяє абстрагувати логіку квантових вентилів від високорівневої логіки обробки даних, забезпечуючи гнучкість та можливість масштабування.

Найнижчим рівнем ієрархії є бібліотека базових зворотних елементів. Оскільки FPGA базується на класичній CMOS-логіці, яка є незворотною за своєю природою, поведінка квантових вентилів емулюється через відповідні

булеві функції. Кожен вентиль описується як окремий Verilog-модуль, що реалізує таблицю істинності відповідного квантового оператора.

На рисунках 4.3.1.1 - 4.3.1.4 наведено фрагменти реалізації основних будівельних блоків:

1. Вентиль *X* (Pauli-X / NOT Gate) – є квантовим аналогом класичного інвертора. Цей елемент виконує безумовну зміну стану кубіта на протилежний $|0\rangle \rightarrow |1\rangle$, $|1\rangle \rightarrow |0\rangle$, у Verilog це реалізується оператором побітового заперечення ~. З точки зору апаратної реалізації на FPGA, це найпростіша операція, яка описується оператором побітового заперечення. Важливою властивістю є те, що подвійне застосування вентиля відновлює початкове значення *NOT(NOT(x))=x*.

```
module X (
    input wire in,
    output wire out
);
    assign out = ~in;
endmodule
```

Рис 4.3.1.1. Реалізація вентиля NOT мовою Verilog.

2. Вентиль CNOT – реалізується модулем *cx.v*. Логіка роботи описується операцією XOR: керуючий кубіт залишається без змін, а цільовий інвертується, якщо керуючий дорівнює 1 $out = in \oplus ctrl$.

```
module CX (
    input wire ctrl,
    input wire in,
    output wire out
);
    assign out = ctrl
        ? ~in
        : in;
endmodule
```

Рис 4.3.1.2. Реалізація вентиля CNOT мовою Verilog.

3. Вентиль Tofolli – реалізується модулем *c2x.v* та має дві керуючі лінії. Інверсія цільового біта відбувається при умові логічного "1" на обох керуючих входах $out = in \oplus (\text{ctrl0} \cdot \text{ctrl1})$.

```
module C2X (
    input wire ctrl0,
    input wire ctrl1,
    input wire in,
    output wire out
);
    assign out = (ctrl0 & ctrl1)
        ? ~in
        : in;
endmodule
```

Рис 4.3.1.3. Реалізація вентиля Tofolli мовою Verilog.

4. Вентиль C3X - це розширена, узагальнена, версія вентиля Тоффолі з трьома керуючими лініями. Реалізація описана модулем *c3x.v*. Цільовий біт інвертується лише тоді, коли всі три керуючі біти знаходяться в стані 1.

```
module C3X (
    input wire ctrl0,
    input wire ctrl1,
    input wire ctrl2,
    input wire in,
    output wire out
);
    assign out = (ctrl0 & ctrl1 & ctrl2)
        ? ~in
        : in;
endmodule
```

Рис 4.3.1.4. Verilog реалізація узагальненого вентиля Tofolli з трьома керуючими лініями.

5. Вентиль Фредкіна – здійснює контрольовану перестановку *Swap* двох каналів даних. Реалізується через логіку мультиплексування: якщо керуючий сигнал активний, входи міняються місцями.

```
module CSWAP (
    input wire ctrl,
    input wire in1,
    input wire in2,
    output wire out1,
    output wire out2
);
    wire ctrl_out;
    assign ctrl_out = ctrl;
    assign out1 = ctrl ? in2 : in1;
    assign out2 = ctrl ? in1 : in2;
endmodule
```

Рис 4.3.1.5. Реалізація вентиля Фредкіна мовою Verilog.

Імплементація RRG2.v. Центральним елементом архітектури є модуль *RRG2.v*, який виконує функцію криптографічного ядра. Його завдання — трансформувати вхідний 3-бітний вектор даних *X* у вихідний вектор *Y* згідно з алгоритмом, визначеним топологією синтезованої схеми RRG2.

Внутрішня структура модуля реалізована як комбінаційна схема без використання тактових сигналів, що дозволяє сигналу проходити крізь весь ланцюжок вентилів за час, що дорівнює сумарній затримці перемикання транзисторів.

Організація внутрішніх зв'язків: ключовим аспектом Verilog-опису є організація передачі даних між каскадами. Оскільки у квантовій схемотехніці кубіти існують безперервно, а у цифровій логіці вентилі є дискретними елементами, було застосовано масив провідників *wire_array*.

- *wire [2:0] wire_array [6:0]*; — цей масив є цифровим аналогом часової еволюції квантової системи. Індекс масиву відповідає номеру етапу обчислень $b_0 \ldots b_6$.
- wire_array[0] — стан системи на вході.
- wire_array[6] — фінальний стан системи на виході.

Логіка каскадування: Реалізація кожного етапу $b_0 \ldots b_6$ включає два типи операцій:

1. Обчислення відповідного вентиля, *CX, C2X, CSWAP*, який змінює цільові біти.
2. Транзит – явна передача бітів, які не задіяні в поточній операції, на наступний рівень як *assign wire_array[1][0] = wire_array[0][0];*). Це гарантує цілісність вектора даних і полегшує читання схеми.

```
module RRG2(
    input  wire [4:0] K,
    input  wire [2:0] X,
    output wire [2:0] Y
);
    wire [2:0] wire_array [6:0];

    //b0
    assign wire_array[0][0] = X[0];
    assign wire_array[0][1] = X[1];
    CX b0 (.ctrl(K[0]), .in(X[2]), .out(wire_array[0][2]));

    //b1
    assign wire_array[1][0] = wire_array[0][0];
    CX b1 (.ctrl(K[1]), .in(wire_array[0][1]), .out(wire_array[1][1]));
    assign wire_array[1][2] = wire_array[0][2];

    //b2
    CX b2 (.ctrl(K[2]), .in(wire_array[1][0]), .out(wire_array[2][0]));
    assign wire_array[2][1] = wire_array[1][1];
    assign wire_array[2][2] = wire_array[1][2];

    //b3
    assign wire_array[3][0] = wire_array[2][0];
    C2X b3 (.ctrl0(K[3]), .ctrl1(wire_array[2][2]), .in(wire_array[2][1]), .out(wire_array[3][1]));
    assign wire_array[3][2] = wire_array[2][2];

    //b4
    C2X b4 (.ctrl0(K[4]), .ctrl1(wire_array[3][2]), .in(wire_array[3][0]), .out(wire_array[4][0]));
    assign wire_array[4][1] = wire_array[3][1];
    assign wire_array[4][2] = wire_array[3][2];

    //b5
    CSWAP b5 (
        .ctrl(wire_array[4][2]),
        .in1(wire_array[4][0]), .in2(wire_array[4][1]),
        .out1(wire_array[5][0]), .out2(wire_array[5][1]));
    assign wire_array[5][2] = wire_array[4][2];

    //b6
    assign wire_array[6][0] = wire_array[5][0];
    assign wire_array[6][1] = wire_array[5][1];
    CX b6 (.ctrl(K[0]), .in(wire_array[5][2]), .out(wire_array[6][2]));

    //output
    assign Y[0] = wire_array[6][0];
    assign Y[1] = wire_array[6][1];
    assign Y[2] = wire_array[6][2];
endmodule
```

Рис 4.3.1.6. Реалізація RRG2 мовою Verilog.

На рисунку 4.3.1.6. представлено реалізацію модуля *RRG2*, де видно, як вентилі Тоффолі та Фредкіна інтегровані в єдиний конвеєр. Варто зазначити, що

керування вентилями здійснюється як зовнішнім ключем *K*, так і самими даними, внутрішніми лініями *wire_array*.

```
module RRG2_Inverse(
    input  wire [4:0] K,
    input  wire [2:0] X,
    output wire [2:0] Y
);
    wire [2:0] wire_array [6:0];

    //b6
    assign wire_array[0][0] = X[0];
    assign wire_array[0][1] = X[1];
    CX b6 (.ctrl(K[0]), .in(X[2]), .out(wire_array[0][2]));

    //b5
    CSWAP b5 (
        .ctrl(wire_array[0][2]),
        .in1(wire_array[0][0]), .in2(wire_array[0][1]),
        .out1(wire_array[1][0]), .out2(wire_array[1][1]));
    assign wire_array[1][2] = wire_array[0][2];

    //b4
    C2X b4 (.ctrl0(K[4]), .ctrl1(wire_array[1][2]), .in(wire_array[1][0]), .out(wire_array[2][0]));
    assign wire_array[2][1] = wire_array[1][1];
    assign wire_array[2][2] = wire_array[1][2];

    //b3
    assign wire_array[3][0] = wire_array[2][0];
    C2X b3 (.ctrl0(K[3]), .ctrl1(wire_array[2][2]), .in(wire_array[2][1]), .out(wire_array[3][1]));
    assign wire_array[3][2] = wire_array[2][2];

    //b2
    CX b2 (.ctrl(K[2]), .in(wire_array[3][0]), .out(wire_array[4][0]));
    assign wire_array[4][1] = wire_array[3][1];
    assign wire_array[4][2] = wire_array[3][2];

    //b1
    assign wire_array[5][0] = wire_array[4][0];
    CX b1 (.ctrl(K[1]), .in(wire_array[4][1]), .out(wire_array[5][1]));
    assign wire_array[5][2] = wire_array[4][2];

    //b0
    assign wire_array[6][0] = wire_array[5][0];
    assign wire_array[6][1] = wire_array[5][1];
    CX b0 (.ctrl(K[0]), .in(wire_array[5][2]), .out(wire_array[6][2]));

    //output
    assign Y[0] = wire_array[6][0];
    assign Y[1] = wire_array[6][1];
    assign Y[2] = wire_array[6][2];

endmodule
```

Рис 4.3.1.7. Реалізація RRG2_Inverse мовою Verilog.

Важливою особливістю запропонованого алгоритму *RRG2* є те, що отримана схема не є інволютивною. Це означає, що повторна подача шифротексту на вхід того ж самого модуля *RRG2* не відновить початкове повідомлення. Для забезпечення двосторонньої передачі даних було розроблено модуль *RRG2_Inverse.v* реалізація якого представлена на рисунку 4.3.1.7. Його структура базується на фундаментальній властивості зворотних схем – обернена

схема будується шляхом розміщення тих самих вентилів у зворотному порядку від виходу до виходу. У модулі *RRG2_Inverse* потік даних проходить шлях від етапу b_6 до b_0. При цьому самі вентилі залишаються незмінними, змінюється лише топологія їх підключення до масиву *wire_array*.

Такий підхід гарантує точне математичне обернення перетворення. Якщо на вхід *RRG2* подати текст *X*, отримати шифротекст X_{enc}, а потім подати X_{enc} на вхід *RRG2_Inverse* з тим самим ключем *K*, на виході гарантовано отримаємо *X*.

```
`timescale 1ns / 1ps

// ASCII Character Encoder/Decoder
// Кодер/Декодер для 7-бітних ASCII символів з перемикачем
module ascii_codec(
    input  wire        encode_mode, // 1 = encode, 0 = decode
    input  wire [8:0]  K_flat,      // Ключ: 9 біт (3 блоки по 3 біти)
    input  wire [6:0]  char_in,     // Вхідний ASCII символ (7 біт)
    output wire [6:0]  char_out     // Вихідний символ (7 біт)
);
    // Розпаковуємо ключ на окремі блоки
    wire [2:0] K [2:0];

    assign K[0] = K_flat[2:0];
    assign K[1] = K_flat[5:3];
    assign K[2] = K_flat[8:6];

    // Розбиваємо 7-бітний вхід на групи по 3 біти + 1 біт
    wire [2:0] X_block [2:0];
    wire [2:0] Y_enc [2:0];  // Результат encode
    wire [2:0] Y_dec [2:0];  // Результат decode

    assign X_block[0] = char_in[2:0];      // Біти 0-2
    assign X_block[1] = char_in[5:3];      // Біти 3-5
    assign X_block[2] = {2'b00, char_in[6]}; // Біт 6

    // Encode path (forward)
    RRG2 enc_block0 (.K(K[0]), .X(X_block[0]), .Y(Y_enc[0]));
    RRG2 enc_block1 (.K(K[1]), .X(X_block[1]), .Y(Y_enc[1]));
    RRG2 enc_block2 (.K(K[2]), .X(X_block[2]), .Y(Y_enc[2]));

    // Decode path (inverse)
    RRG2_Inverse dec_block0 (.K(K[0]), .X(X_block[0]), .Y(Y_dec[0]));
    RRG2_Inverse dec_block1 (.K(K[1]), .X(X_block[1]), .Y(Y_dec[1]));
    RRG2_Inverse dec_block2 (.K(K[2]), .X(X_block[2]), .Y(Y_dec[2]));

    // Вибір encode/decode
    wire [6:0] encoded_out = {Y_enc[2][0], Y_enc[1], Y_enc[0]};
    wire [6:0] decoded_out = {Y_dec[2][0], Y_dec[1], Y_dec[0]};

    assign char_out = encode_mode ? encoded_out : decoded_out;

endmodule
```

Рис. 4.3.1.8. Verilog-реалізація кодека ASCII з паралельними каналами

Масштабування архітектури - модуль ascii_codec.v. Для практичної реалізації шифрування текстової інформації розроблено модуль верхнього рівня *ascii_codec.v*. Оскільки розроблений базовий блок *RRG2* оперує 3-бітними векторами, а стандартний символ ASCII кодується 7 бітами, необхідна схема адаптації розрядності. На рисунку 4.3.1.8. наведено Verilog-опис цього модуля. Його архітектура базується на трьох ключових принципах:

1. Сегментація даних – вхідний 7-бітний вектор *char_in* розділяється на три незалежні сегменти для подачі на входи трьох паралельних блоків шифрування:
    - Молодші біти [2:0] подаються на X_block0.
    - Середні біти [5:3] подаються на X_block1.
    - Старший біт [6] доповнюється двома нулями (2'b00) для формування повноцінного 3-бітного входу і подається на X_block2.
2. Архітектура подвійного шляху – для забезпечення миттєвого перемикання між режимами шифрування та дешифрування реалізовано апаратний паралелізм. В FPGA одночасно інстанціюються два незалежні ланцюжки:
    - *Encode path* – три модулі *RRG2* для прямого перетворення.
    - *Decode path* – три модулі *RRG2_Inverse* для зворотного перетворення.

    Обидва ланцюжки працюють постійно, обробляючи вхідні дані паралельно.
3. Вихідне мультиплексування – вибір фінального результату здійснюється за допомогою мультиплексора, керованого сигналом *encode_mode*. Це дозволяє змінювати функцію пристрою без переконфігурації ПЛІС, просто змінюючи логічний рівень на керуючому вході.

Така організація модуля *ascii_codec* дозволяє абстрагуватися від складності внутрішніх перетворень на вищих рівнях системи. Для зовнішнього контролера цей модуль виглядає як "чорна скринька", що приймає символ і ключ, та видає результат за один такт, приховуючи всередині складну логіку квантових вентилів та маршрутизації.

4.3.2. Розробка контролерів периферії (PS/2, VGA)

Для перетворення розробленого криптографічного ядра на повноцінну інтерактивну систему необхідно реалізувати інтерфейс взаємодії з користувачем. Система використовує клавіатуру стандарту PS/2 для введення даних, VGA-монітор для візуалізації результатів та семисегментний дисплей для індикації станів. Нижче описано архітектуру кожного контролера.

Підсистема введення: Контролер клавіатури PS/2. Для забезпечення інтерфейсу користувача було обрано стандарт PS/2. На відміну від USB, який вимагає складного стека протоколів та наявності процесора, PS/2 є ідеальним для реалізації на "чистій" логіці FPGA. Фізичний рівень складається лише з двох сигнальних ліній: *PS2_CLK* – тактування та *PS2_DATA* – дані, які працюють за принципом "відкритий колектор". Реалізація контролера *ps2_keyboard.v* вирішує три інженерні задачі: синхронізацію доменів тактових частот, десеріалізацію пакетів та декодування скан-кодів.

```
21      always @(posedge clk or negedge rst_n) begin
22          if (!rst_n) begin
23              ps2_clk_sync1 <= 1;
24              ps2_clk_sync2 <= 1;
25              ps2_clk_sync3 <= 1;
26              ps2_data_sync <= 1;
27          end else begin
28              ps2_clk_sync1 <= ps2_clk;
29              ps2_clk_sync2 <= ps2_clk_sync1;
30              ps2_clk_sync3 <= ps2_clk_sync2;
31              ps2_data_sync <= ps2_data;
32          end
33      end
34
35      wire ps2_clk_falling = ps2_clk_sync3 && !ps2_clk_sync2;
```

Рис 4.3.2.1 Синхронізація PS/2 сигналів (3-стадійна).

1. Проблема перетину тактових доменів Клавіатура генерує власний тактовий сигнал *PS2_CLK* з частотою в діапазоні 10–16.7 кГц. Цей сигнал є асинхронним відносно системного тактового генератора FPGA (50 МГц). Пряме використання *PS2_CLK* як тактового сигналу для тригерів всередині

FPGA, наприклад: *always @(negedge ps2_clk)* є небезпечним через ризик виникнення метастабільності та шумів на лінії. Для вирішення цієї проблеми у модулі реалізовано триступеневий синхронізатор:

Такий підхід дозволяє обробляти дані клавіатури повністю в синхронному домені 50 МГц, використовуючи *ps2_clk_falling* як сигнал дозволу.

2. Десеріалізація протоколу – дані передаються фреймами по 11 біт. Контролер реалізує простий скінченний автомат на базі лічильника *bit_count*, який відстежує структуру пакету:
   - Біт 1: Старт-біт – завжди 0.
   - Біти 2-9: 8 біт даних – (Least Significant Bit first).
   - Біт 10: Біт парності (Odd Parity) — у даній реалізації ігнорується для спрощення, оскільки відстань передачі мінімальна.
   - Біт 11: Стоп-біт – завжди 1.

   Логіка накопичує біти даних у регістрі *data_reg* лише в моменти, коли детектовано *ps2_clk_falling*.
3. Фільтрація та декодування відбувається перетворення Scan Code в ASCII. Клавіатура надсилає не символи, а скан-коди позицій клавіш в поточній клавіатурі використовувались Scan Code Set 2. Натискання клавіші генерує один байт (наприклад, 0x1C для 'A'), а відпускання — послідовність з двох байтів: префікс 0xF0 і код клавіші (0xF0, 0x1C).

Контролер виконує фільтрацію потоку даних:

- Ігнорує префікс відпускання 0xF0 та розширений код 0xE0.
- Реагує лише на завершення прийому повного байта коли стоп біт отримано.
- Перевіряє дублікати *(data_reg != last_key)*, щоб уникнути багаторазового введення символу при утриманні клавіші.

Фінальним етапом є трансляція чистого скан-коду в ASCII за допомогою таблиці істинності, реалізованої через конструкцію case. Це дозволяє передавати в

систему стандартні коди символів (наприклад, 8'h41 для 'A'), які безпосередньо розуміє модуль шифрування та драйвер шрифтів.

Підсистема відеовиводу: VGA-контролер. Для візуалізації даних обрано інтерфейс VGA – Video Graphics Array. Незважаючи на свій вік, він залишається стандартом де-факто для вбудованих систем завдяки простоті реалізації на рівні регістрових передач та відсутності потреби у ліцензуванні чи складних драйверах, на відміну від HDMI.

Розроблений модуль *vga_controller.v* реалізує промисловий стандарт відеорежиму 640×480 @ 60 Гц.

1. Формування піксельної частоти. Згідно зі специфікацією VESA, для даного режиму ідеальна частота піксельного тактування становить 25.175 МГц. Системний генератор плати працює на частоті 50 МГц. Для спрощення схеми використано дільник частоти на 2, реалізований Т-тригером у модулі верхнього рівня, що дає рівно 25 МГц. Відхилення у 0.7% є допустимим — сучасні LCD-монітори автоматично підлаштовують фазу дискретизації, тому зображення залишається стабільним.

2. Архітектура двовимірної розгортки Формування зображення базується на емуляції руху електронного променя ЕПТ-монітора. Це реалізовано за допомогою двох каскадних лічильників:

- Горизонтальний лічильник – *h_count* – працює на частоті 25 МГц. Він відраховує пікселі в межах одного рядка. Повний цикл складає 800 тактів від 0 до 799, з яких перші 640 — це видима зона.
- Вертикальний лічильник – *v_count* – інкрементується щоразу, коли горизонтальний лічильник завершує рядок, тобто переповнюється. Він відраховує рядки кадру. Повний цикл складає 525 рядків (0…524), з яких видимі — 480.

3. Генерація синхросигналів – коректне відображення вимагає суворого дотримання часових інтервалів гасіння променя. Контролер формує сигнали *hsync* (рядкова синхронізація) та *vsync* (кадрова синхронізація) за комбінаційною

логікою, порівнюючи значення лічильників з константами стандарту представленими в таблиці 4.3.2.1

Таблиця 4.3.2.1 Константи стандарту розгортки монітора 640 на 480

Інтервал	Горизонталь (пікселі)	Вертикаль (рядки)	Опис
Visible Area	640	480	Активна зона відображення даних
Front Porch	16	10	Захисний інтервал перед імпульсом
Sync Pulse	96	2	Імпульс синхронізації (активний рівень — 0)
Back Porch	48	33	Захисний інтервал після імпульсу
Total	800	525	Повний цикл розгортки

4. Координатна сітка та сигнал *video_on* який є критично важливим. Він приймає значення 1 тільки тоді, коли промінь знаходиться в активній зоні $x<640$ та $y<480$. Цей сигнал використовується наступними модулями для примусового "зачорнення" RGB-каналів під час інтервалів гасіння. Якщо цього не зробити, на моніторі будуть видимі артефакти зворотного ходу променя.

Вихідні шини *h_count* та *v_count* експортуються з модуля і слугують глобальними координатами X,Y для підсистеми відображення тексту, дозволяючи точно визначати, який піксель якого символу малювати в даний момент часу.

Модуль текстової візуалізації та знакогенератор. Відображення інформації реалізовано за "тайловим" принципом – Tile-based rendering, який був стандартом для ранніх комп'ютерних терміналів. Замість того, щоб зберігати колір кожного з 307200 пікселів екрана що вимагало б сотні кілобайт пам'яті, система зберігає лише коди символів, які потрібно відобразити. Це дозволяє скоротити використання пам'яті FPGA до мізерних 80 байт (для одного рядка

тексту). Реалізація складається з двох ключових компонентів: знакогенератора та логіки рендерингу.

1. Апаратний знакогенератор або шрифти – описаний модулем font_rom_8x16 виконує роль постійного запам'ятовувального пристрою, що зберігає растрові маски символів. Кожен символ ASCII описується матрицею розміром 8×16 пікселів. У термінах апаратної логіки, це велика таблиця відповідності, яка має два входи:

- char_code [7:0]: ASCII-код символу, наприклад: 0x41 для 'A'.
- row [3:0]: номер рядка всередині символьної матриці (від 0 до 15).

```
// Letter A
8'h41: begin
    case (row)
        4'h0: font_data = 8'b00000000;
        4'h1: font_data = 8'b00000000;
        4'h2: font_data = 8'b00111100;
        4'h3: font_data = 8'b01100110;
        4'h4: font_data = 8'b01100110;
        4'h5: font_data = 8'b01111110;
        4'h6: font_data = 8'b01100110;
        4'h7: font_data = 8'b01100110;
        4'h8: font_data = 8'b01100110;
        4'h9: font_data = 8'b00000000;
        4'hA: font_data = 8'b00000000;
        4'hB: font_data = 8'b00000000;
        4'hC: font_data = 8'b00000000;
        4'hD: font_data = 8'b00000000;
        4'hE: font_data = 8'b00000000;
        4'hF: font_data = 8'b00000000;
    endcase
end
```

Рис 4.3.2.3. Символ ‘A’ в матриці шрифтів.

Виходом є 8-бітний вектор *font_data*, де кожен біт відповідає одному пікселю по горизонталі. Логічна одиниця означає, що піксель має світитися, нуль — фон. Наприклад, для літери 'A' (код 0x41) у рядку 4 вектор буде 01101110, що формує перекладину літери що і зображено на рисунку 4.2.3.2.

2. Конвеєр рендерингу перставлений модулем *simple_keyboard_display.v,* він синхронно з розгорткою VGA обчислює колір поточного пікселя. Процес

перетворення координат X, Y у піксель на екрані відбувається в кілька етапів за один такт:

1. Адресація символу – поточні координати сканування *h_count* та *v_count* трансформуються в координати текстової сітки:

- Індекс символу в рядку: *Index=h_count/8* – реалізовано як відкидання 3 молодших бітів: *h_count[9:3]*.
- Рядок всередині символу: *Row=v_count(mod16)* – взяття 4 молодших бітів: *v_count[3:0]*.

2. Вибірка даних – за обчисленим індексом з буфера *text_line* витягується ASCII-код. Цей код разом з номером рядка *Row* подається на вхід *Font ROM*.

3. Серіалізація пікселів - отриманий байт *font_data* проходить через мультиплексор, який вибирає один біт на основі молодших бітів горизонтального лічильника.

$$Pixel = font\_data[7 - (h\_count(mod8))] \qquad (4.3.2.1)$$

3. Текстовий буфер та курсор – система використовує буфер *text_line* розміром 80 байт для зберігання одного рядка тексту. Логіка модуля обробляє вхідні сигнали від клавіатури, реалізуючи функції найпростішого текстового редактора:

- При надходженні друкованого символу (ASCII 0x20–0x7E) він записується в поточну позицію курсора, а курсор зміщується вправо.
- При натисканні Backspace (ASCII 0x08) курсор зміщується вліво, а символ видаляється (записується пробіл).

Для покращення ергономіки реалізовано апаратний курсор. Він являє собою інверсію кольору або спеціальний символ, який миготить з частотою близько 1 Гц. Це досягається використанням 24-бітного лічильника, який перемикає сигнал *cursor_blink* при переповненні.

Блок індикації стану та керування UI/UX. Оскільки розроблений пристрій є автономним, критично важливо забезпечити зручне керування режимами роботи та візуальний зворотний зв'язок без необхідності підключення налагоджувального обладнання. Модуль *seven_seg_display.v* виконує роль панелі керування, вирішуючи завдання обробки нестабільних вхідних сигналів та динамічної індикації.

1. Цифрова фільтрація сигналів – механічні кнопки, встановлені на платі FPGA, при натисканні генерують серію паразитних імпульсів ("брязкіт контактів" або Bouncing), що триває кілька мілісекунд. Без фільтрації це призвело б до хаотичного перемикання режимів.

У проєкті реалізовано програмно-апаратний дебаунсер. Для кожної кнопки *key1, key2, key3* працює незалежний 16-бітний лічильник. Логіка роботи наступна:

- Вхідний сигнал безперервно зчитується.
- Якщо сигнал стабільний, лічильник інкрементується.
- Якщо сигнал змінюється (шум), лічильник скидається в нуль.
- Зміна внутрішнього стану реєструється лише тоді, коли лічильник досягає переповнення (16'hFFFF), що гарантує стабільність сигналу протягом ≈1.3 мс (при 50 МГц).

2. Після фільтрації сигнали надходять на скінченний автомат, який керує глобальним режимом роботи шифратора:

- KEY1 – активує режим шифрування *mode = 00*.
- KEY2 – активує режим дешифрування *mode = 01*.
- KEY3 – активує режим збереження ключів *mode = 10*.

3. Динамічна індикація та знаковий декодер. Для економії виводів FPGA чотири розряди семисегментного дисплея підключені за схемою зі спільним анодом/катодом. Керування здійснюється методом часового мультиплексування. Лічильник *refresh* ділить вхідну частоту, і його старші біти

циклично перемикають активний розряд *dig*. Частота оновлення підібрана так, щоб людське око сприймало світіння як безперервне.

Особливістю реалізації є кастомний знакогенератор. Замість стандартного декодера 0-9, розроблено таблицю істинності, яка формує мнемонічні слова, зрозумілі користувачеві:

- Відображає "EnCr" – encryption.
- Відображає "dECr" – decryption.
- Відображає "KEYS".

Це значно покращує ергономіку порівняно з використанням умовних числових кодів.

Таким чином, розроблений набір периферійних модулів формує повноцінну інфраструктуру введення-виведення, яка функціонує повністю на апаратному рівні. Відмова від використання готових софт-процесорів на користь спеціалізованих контролерів дозволила досягти миттєвої реакції інтерфейсу та абсолютної передбачуваності часових характеристик а також зменшити кількість витрачених ресурсів. Реалізована система візуалізації та керування перетворює криптографічне ядро з абстрактного математичного алгоритму на завершений автономний пристрій для кінцевого користувача.

4.3.3. Загальна архітектура системи та організація потоків даних

Структурна організація розробленого апаратно-програмного комплексу базується на принципах модульної декомпозиції та конвеєрної обробки даних. Інтеграція функціональних блоків у єдину систему виконана на рівні верхньої ієрархії, що дозволило реалізувати наскрізний потік даних від інтерфейсів введення до підсистеми візуалізації з мінімальними часовими затримками.

Узагальнена структурна схема, що відображає архітектуру системи та маршрутизацію сигналів між функціональними доменами, наведена на рисунку 4.3.3.1.

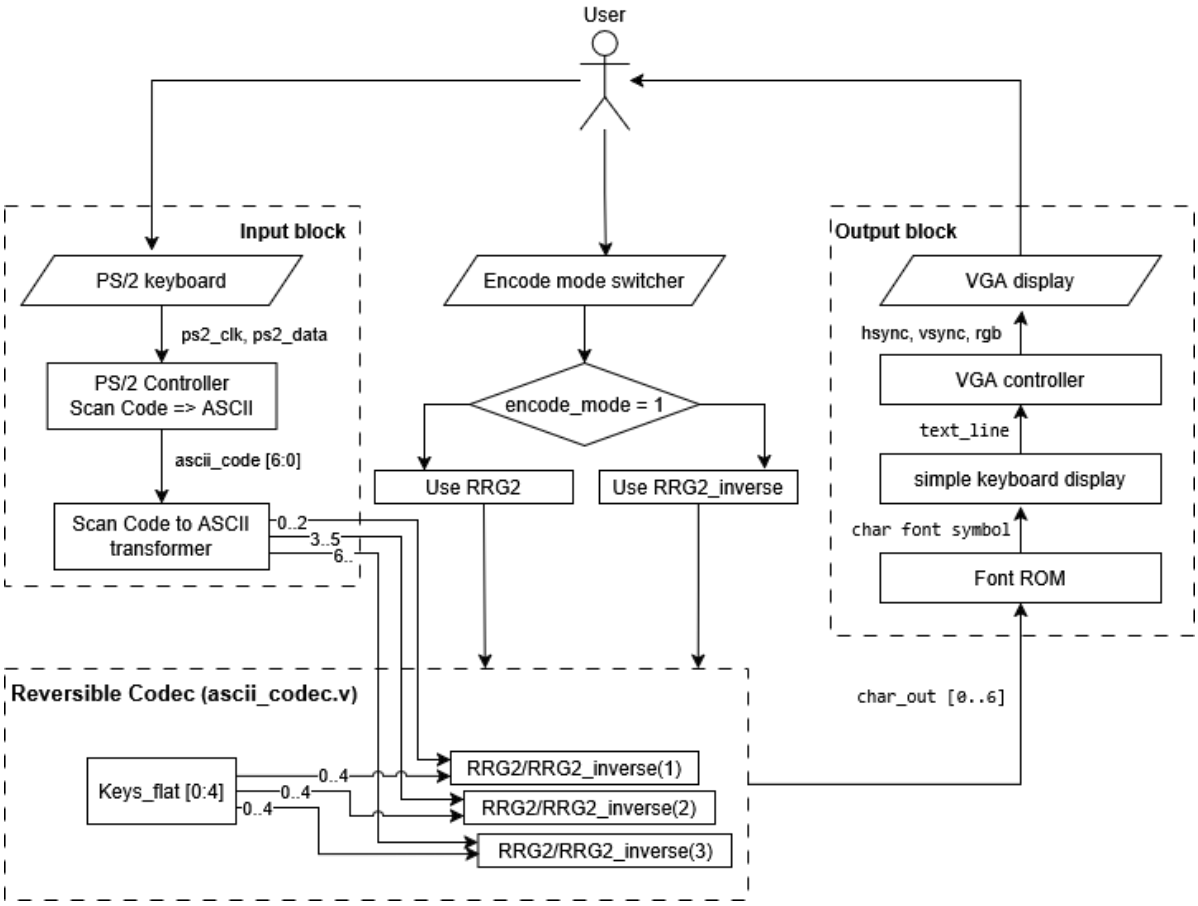


Рис 4.3.3.1. Системна блок-схема апаратного прототипу шифрування.

Архітектура системи є гетерогенною та функціонує у двох незалежних доменах синхронізації, що обумовлено специфікою периферійних інтерфейсів:

1. Домен обробки та керування $F_{sys} = 50$ МГц – забезпечує функціонування контролерів введення, скінченних автоматів керування та безпосередньо криптографічного ядра.
2. Домен візуалізації $F_{pix} = 25$ МГц – синхронізований з вимогами стандарту VESA для формування відеосигналу.

Обмін даними між доменами реалізовано через буферизовані регістрові структури, що гарантує цілісність сигналу та стійкість до метастабільних станів.

Логіку функціонування системи можна представити як послідовність перетворень інформаційного вектора. Відповідно до схеми представленої на рисунку 4.3.3.1, виділяються три етапи трансформації даних:

1. Формування та адаптація вхідного вектора, відповідний блок Input Block – це підсистема введення виконує функцію перетворення асинхронних натискання клавіш у синхронний цифровий потік. Ключовим архітектурним елементом на цьому етапі є блок адаптації даних – Scan Code to ASCII transformer. На відміну від класичних схем, де дані передаються повною шиною, у розробленій архітектурі застосовано метод фрагментації розрядної сітки. Вхідний 7-бітний ASCII-вектор розбивається на незалежні підгрупи: біти 0..2, 3..5, 6.. . Таке рішення обумовлене топологією зворотних елементів RRG2 та дозволяє реалізувати просторово-паралельну обробку даних, де кожна група бітів обробляється окремим логічним каскадом.

2. Реконфігуровне ядро зворотного перетворення – Reversible Codec – центральний обчислювальний вузол побудований за схемою комбінаційної мережі. Його особливістю є відсутність тактування у ланцюзі обробки даних, що забезпечує режим нульової затримки — результат перетворення формується протягом одного такту системної частоти. Архітектура ядра передбачає динамічну реконфігурацію маршрутів проходження даних залежно від керуючого сигналу *encode_mode*:

- Сигнал керування ініціює перемикання мультиплексорів, обираючи між гілкою прямого перетворення – Use RRG2 та гілкою зворотного перетворення – Use RRG2_inverse.
- Вектор ключів шифрування *Keys_flat* розподіляється між паралельними обчислювальними гілками, забезпечуючи унікальність перетворення для кожної підгрупи бітів.

3. Підсистема візуалізації результатів – Output Block – це вихідний каскад який реалізує функцію людино-машинного інтерфейсу. Архітектурно він відділений від обчислювального ядра, отримуючи на вхід вже модифікований вектор даних *char_out*. Застосування знакогенератора на основі ПЗП – шрифтів які розміщенів в модулі Font ROM, дозволяє абстрагуватися від графічного представлення на етапах обробки. Система візуалізації виконує зворотне

перетворення цифрового коду в растровий образ символу синхронно з розгорткою відеоконтролера.

4.3.4. Аналіз апаратних витрат та експериментальна верифікація

Оцінка ефективності запропонованої архітектури виконувалася за двома критеріями: обсяг використаних логічних ресурсів FPGA та часові характеристики обробки даних. Синтез та імплементація проєкту здійснювалися в середовищі Intel Quartus Prime Lite Edition 24.1 для цільового кристала серії Altera Cyclone IV версії EP4CE6E22C8N.

Зведений аналіз отриманих показників(таблиця 4.3.4.1):

1. Логічна ємність – загальне використання складає 1 424 LE, що становить 23% від доступного обсягу кристала. Це свідчить про те, що навіть після імплементації повної системи введення-виведення, у пристрої залишається значний запас ресурсів, понад 75%, для потенційного масштабування алгоритму, наприклад, для збільшення розрядності ключа або додавання нових криптографічних примітивів.
2. Організація пам'яті – важливою особливістю синтезованого проєкту є показник *Загальна кількість бітів пам'яті = 0*. Це вказує на те, що модуль знакогенератора *font_rom.v*, а також буфери даних, були реалізовані синтезатором безпосередньо на базі розподіленої логіки LUTs, а не з використанням вбудованих блоків пам'яті *M9K*. Таке рішення є виправданим для малих обсягів даних і дозволяє економити спеціалізовані ресурси пам'яті FPGA для більш складних задач.
3. Регістри – використання 750 регістрів пов'язане переважно з організацією конвеєра відеоконтролера, лічильників синхронізації та буферів для усунення брязкоту контактів. Сама схема зворотного кодування *ascii_codec.v* є комбінаційною і мінімально впливає на цей показник.

Таблиця 4.3.4.1. Використання ресурсів FPGA Cyclone IV EP4CE6E22C8.

Ресурс	Використано	Загалом	Відсоток (%)
Всього логічних елементів	1,424	6,272	23%
-- *Комбінаційні логічні функції*	*1,350*	—	—
-- *Виділені логічні регістри*	750	—	—
Загальна кількість виводів	29	92	32%
Загальна кількість бітів пам'яті	0	276,480	0%
Вбудований множник 9-бітний	0	30	0%
Загальна кількість PLL*	0	2	0%

*PLL – система фазового автопідлаштування частоти (використовується для генерації тактових сигналів)

Часовий аналіз, виконаний інструментом Timing Analyzer, підтвердив надійність системи. Результати розрахунку максимальної робочої частоти F_{max} для двох тактових доменів наведені нижче:

1. Системний домен *clk_50mhz*:
   - Необхідна частота – 50 МГц.
   - Максимальна частота F_{max} – 310.08 МГц.
   - Запас продуктивності становить понад 600%. Це підтверджує, що критичний шлях схеми є надзвичайно коротким, а логіка криптографічного перетворення не вносить суттєвих затримок.
2. Відеодомен *clk_div*:
   - Необхідна частота ~25 МГц – стандарт VGA 640x480.
   - Максимальна частота F_{max} – 79.4 МГц.
   - Схема гарантує стабільну генерацію відеосигналу без джитеру та зривів синхронізації.

Випробування проводилися на базі налагоджувального стенда. Перевірка підтвердила коректність функціонування всіх підсистем:

1. Інтерфейс користувача – введення тексту з клавіатури відбувається без затримок та пропусків символів завдяки апаратному модулю *ps2_keyboard*.
2. Візуалізація – зображення на моніторі стабільне, артефакти перемикання режимів відсутні.

3. Криптографічна стійкість – у режимі шифрування символи на екрані замінюються псевдовипадковими значеннями згідно з таблицею істинності RRG2, а при перемиканні в режим дешифрування початковий текст відновлюється без помилок, що доводить властивість зворотності реалізованої схеми.

Висновки до розділу 4

У четвертому розділі проведено комплексну експериментальну верифікацію розробленого програмного забезпечення та алгоритмічних методів. За результатами досліджень зроблено наступні висновки:

1. На наборі еталонних бенчмарків експериментально підтверджено здатність генетичного методу мінімізувати квантову вартість схем. Програмний комплекс забезпечує 100% логічну коректність генерації як класичних арифметичних, так і специфічних зворотних функцій.
2. Засобами розробленого ПЗ синтезовано структуру зворотного шифратора. Підтверджено архітектурну універсальність згенерованого рішення: завдяки властивості зворотності одна й та сама логічна схема виконує функції шифрування та дешифрування без зміни конфігурації.
3. Фізична імплементація шифратора на ПЛІС Altera Cyclone IV підтвердила, що логічна структура, згенерована алгоритмом, є коректною та придатною для апаратної реалізації. Тестування довело повну відповідність фізичної поведінки схеми її програмній моделі.
4. Часовий аналіз підтвердив ефективність мінімізації критичного шляху: досягнуто максимальну частоту $F_{max} \approx 310$ МГц. Запас продуктивності у 600% свідчить про здатність методу генерувати високошвидкісні рішення, придатні для обробки даних у реальному часі.
5. Натурні випробування підтвердили можливість інтеграції синтезованих модулів у складні системи на кристалі. Стабільна робота шифратора в

комплексі з периферійними контролерами (PS/2, VGA) демонструє високу завадостійкість та надійність отриманих рішень.

ВИСНОВКИ

У дисертаційній роботі вирішено актуальну науково-прикладну задачу підвищення ефективності автоматизованого проєктування зворотних та квантових логічних схем шляхом розробки спеціалізованого генетичного методу, алгоритмічного та програмного забезпечення.

Основні наукові та практичні результати роботи:

1. Встановлено, що існуючі методи синтезу, які базуються на детермінованих алгоритмах або класичних еволюційних стратегіях, мають експоненційну обчислювальну складність і схильні до застрягання в локальних екстремумах. Це унеможливлює їх ефективне використання для мінімізації квантової вартості схем великої розмірності, що обумовило необхідність розробки нових евристичних підходів.
2. Розроблено математичну модель структурного синтезу зворотних схем, яка, на відміну від існуючих, розглядає процес проєктування як багатокритеріальну оптимізаційну задачу. Сформульовано фітнес-функцію, яка комплексно враховує не лише логічну коректність, але й апаратні витрати як квантову вартість і кількість вентилів, що дозволило спрямувати пошук на отримання найбільш економічних рішень.
3. Удосконалено генетичний алгоритм шляхом впровадження об’єктно-орієнтованої моделі хромосоми. Перехід від бітових масивів до оперування функціональними об'єктами (вентилями) дозволив формалізувати правила комутації та суттєво зменшити простір пошуку рішень, підвищивши швидкість збіжності алгоритму майже в два рази порівняно з іншими підходами.
4. Розроблено адаптивні оператори багатокомпонентної мутації та динамічної перестановки виходів. Механізм імовірнісного керування структурними змінами у поєднанні з динамічною маршрутизацією

каналів забезпечує ефективний вихід із локальних екстремумів, дозволяючи синтезувати оптимальні топології навіть для функцій зі складною логікою комутації.

5. Запропоновано та обґрунтовано використання розширеного базису на основі реконфігурованих вентилів Фредкіна, що вперше дозволило синтезувати відмовостійкі системи зі збереженням парності. Доведено, що застосування елементів, здатних змінювати логічну функцію під дією керуючих сигналів без зміни фізичної архітектури, створює основу для побудови адаптивних криптографічних систем із можливістю переналаштування в режимі реального часу.
6. Створено автоматизовану систему синтезу мовою C#, інтегровану з середовищем IBM Qiskit. Реалізований наскрізний маршрут проєктування, що включає трансляцію схем у формат OpenQASM та їх вичерпну верифікацію, що гарантує повну функціональну сумісність синтезованих рішень із сучасними квантовими процесорами.
7. Ефективність запропонованих методів підтверджено синтезом зворотного потокового шифратора та його успішною імплементацією на FPGA Altera Cyclone IV. Отримані результати свідчать про те, що його параметри порівняно з відомими світовими аналогами (квантова вартість, глибина схеми, апаратна складність) покращуються більше як в три рази. а розроблене алгоритмічне забезпечення дозволяє генерувати високошвидкісні та надійні схемотехнічні рішення, придатні для використання в системах реального часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kyryliuk T., Deibuk V., Kyryliuk S. Synthesis of reversible circuits by genetic algorithm with multicomponent mutation. *Інформаційні технології та комп'ютерне моделювання* : матеріали міжнар. наук.-практ. конф. (м. Івано-Франківськ, 6–8 лип. 2023 р.). Івано-Франківськ, 2023. С. 150–151.
2. Kyryliuk T., Palahuta M., Deibuk V. Using artificial intelligence methods for the optimal synthesis of reversible networks. *Radioelectronic and Computer Systems*. 2024. Vol. 2024, no. 4. P. 112–122. URL:https://doi.org/10.32620/reks.2024.4.10
3. Deibuk V., Dovhaniuk O., Kyryliuk T. The Extended Fredkin Gates with Reconfiguration in NCT Basis. *Lecture Notes on Data Engineering and Communications Technologies*. 2023. Vol. 181 : Advances in Computer Science for Engineering and Education VI : ICCSEEA 2023 (Warsaw, Poland, 17–19 March 2023). Warsaw, P. 95–105. URL: https://doi.org/10.1007/978-3-031-36118-0_9.
4. Dovhaniuk O., Kyryliuk T., Deibuk V. Reversible Fault-Tolerant Encryption Using Extended Fredkin Gate with Reconfiguration. *Advances in Transdisciplinary Engineering*. 2025. Vol. 65 : Artificial Intelligence, Medical Engineering and Education : AIMEE 2024 (Huangshi, China, 26–27 Oct. 2024). Huangshi, P. 69–76. URL: https://doi.org/10.3233/atde250113.
5. Burgholzer L., Wille R. Exploiting Reversible Computing for Verification: Potential, Possible Paths, and Consequences. *Proceedings of the 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023. P. 429–435. DOI: https://doi.org/10.1145/3566097.3567914
6. Han J., Zhang X., Wang X. Application Research of Evolutionary Algorithm in Synthesis of Reversible Logic Circuits. *Journal of Physics*. 2019. Vol. 1237, № 2. 022083. P. 5. DOI: https://doi.org/10.1088/1742-6596/1237/2/022083
7. Dennard R. H., Gaensslen F. H., Yu H.-N., Rideout V. L., Bassous E., LeBlanc A. R. Design of ion-implanted MOSFET's with very small physical dimensions.

IEEE Journal of Solid-State Circuits. 1974. Vol. 9, № 5. P. 256–268. DOI: https://doi.org/10.1109/JSSC.1974.1050511

8. Moore G. E. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, April 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Society Newsletter*. 2006. Vol. 11, № 3. P. 33–35. DOI: https://doi.org/10.1109/n-ssc.2006.4785860
9. Deibuk V. G., Yuriychuk I. M., Lemberski I. Fidelity of noisy multiple-control reversible gates. *Semiconductor Physics, Quantum Electronics & Optoelectronics*. 2020. Vol. 23, № 4. P. 385–392. DOI: https://doi.org/10.15407/spqeo23.04.385
10. Shauly E. N. CMOS leakage and power reduction in transistors and circuits: process and layout considerations. *Journal of Low Power Electronics and Applications*. 2012. Vol. 2, № 1. P. 1–29. DOI: https://doi.org/10.3390/jlpea2010001
11. Weste N., Harris D. CMOS VLSI design: a circuits and systems perspective. 4th ed. Boston: Pearson Education, Limited, 2013. 743 p.
12. Akshata S., Arpitha E., Deepashree V. A., Pushpa L. N. D. Low power comparator design using reversible logic gates – adiabatic circuits. *International Journal of Engineering Research & Technology*. 2018. Vol. 6, № 13. URL: https://www.ijert.org/research/low-power-comparator-design-using-reversible-low-power-comparator-design-using-reversible-IJERTCONV6IS13214.pdf (дата звернення: 23.05.2025).
13. Beloglazov A. et al. A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. *Advances in Computers*. 2011. Vol. 82. P. 47–111. DOI: https://doi.org/10.1016/b978-0-12-385512-1.00003-7
14. Shafi A., Bahar A. N. Fredkin circuit in nanoscale: a multilayer approach. *International Journal of Information Technology and Computer Science (IJITCS)*. 2018. Vol. 10, № 10. P. 38–43. DOI: https://doi.org/10.5815/ijitcs.2018.10.05

15. Sourabh. T, Sanyal W., Seethur R. Design of a Reversible Full Adder Using Quantum Cellular Automata. *The Eurasia Proceedings of Science Technology Engineering and Mathematics*. 2022. Vol. 21. P. 500–505. DOI: https://doi.org/10.55549/epstem.1227552
16. Ying Z., Feng C., Zhao Z., Soref R., Pan D., Chen R. T. Integrated multi-operand electro-optic logic gates for optical computing. *Applied Physics Letters*. 2019. Vol. 115, № 17. 171104. DOI: https://doi.org/10.1063/1.5126517
17. Peres A., Mayer M. E. Quantum Theory: Concepts and Methods. *Physics Today*. 1994. Vol. 47, № 12. P. 65–66. DOI: https://doi.org/10.1063/1.2808757
18. Landauer R. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*. 1961. Vol. 5, № 3. P. 183–191. DOI: https://doi.org/10.1147/rd.53.0183
19. Bérut A., Petrosyan A., Ciliberto S. Information and thermodynamics: experimental verification of Landauer's erasure principle. *Journal of Statistical Mechanics: Theory and Experiment*. 2015. Vol. 2015, № 6. P06015. DOI: https://doi.org/10.1088/1742-5468/2015/06/P06015
20. Bennett C. H. Notes on Landauer's principle, reversible computation, and Maxwell's Demon. *Studies in History and Philosophy of Science Part B: Studies in History and Philosophy of Modern Physics*. 2003. Vol. 34, № 3. P. 501–510. DOI: https://doi.org/10.1016/s1355-2198(03)00039-x
21. Jun Y., Gavrilov M., Bechhoefer J. High-precision test of Landauer's principle in a feedback trap. *Physical Review Letters*. 2014. Vol. 113, № 19. 190601. DOI: https://doi.org/10.1103/PhysRevLett.113.190601
22. Dovhaniuk O., Deibuk V. CMOS simulation of mixed-polarity generalized Fredkin gates. *2022 12th International Conference on Advanced Computer Information Technologies (ACIT)*. (Ruzomberok, Slovakia, 2022). IEEE. Ruzomberok, 2022. P. 388–391. DOI: https://doi.org/10.1109/ACIT54803.2022.9913119

23. Fredkin E., Toffoli T. Conservative logic. *International Journal of Theoretical Physics*. 1982. Vol. 21, № 3–4. P. 219–253. DOI: https://doi.org/10.1007/BF01857727
24. Zhang M., Zhao S., Wang X. Automatic synthesis of reversible logic circuit based on genetic algorithm. *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS 2009)*. (Shanghai, China, 20–22 Nov. 2009). IEEE. Shanghai, 2009. Vol. 1. P. 605–609. DOI: https://doi.org/10.1109/ICICISYS.2009.5358132
25. Bennett C. H. Logical reversibility of computation. *IBM Journal of Research and Development*. 1973. Vol. 17, № 6. P. 525–532. DOI: https://doi.org/10.1147/rd.176.0525
26. Jamuna S., Dinesha P., Shashikala K. P., Kishore Kumar K. Design and implementation of reliable encryption algorithms through soft error mitigation. *International Journal of Computer Network and Information Security (IJCNIS)*. 2020. Vol. 12, № 4. P. 41–50. DOI: https://doi.org/10.5815/ijcnis.2020.04.04
27. Liao H., Han J., Xiao Y. et al. Anti-freezing dough for renewable and reconfigurable flexible strain sensors. *Journal of Electronic Materials*. 2024. Vol. 53, № 5. P. 2524–2532. DOI: https://doi.org/10.1007/s11664-024-10981-6
28. Thakur G., Keshri S., Chaudhari K. Design of Energy Efficient 16-Bit Reversible ALU for Low Power IoT Applications. *2025 12th International Conference on Computing for Sustainable Global Development (INDIACom)*. (Delhi, India, 2025). IEEE. Delhi, 2025. P. 1–6. DOI: https://doi.org/10.23919/INDIACom66777.2025.11115804
29. Chauhan A., Das S. Design of Parity Preserving Reversible Design-Based Fault-Tolerant ALU on 32 nm CNTFET. *Lecture Notes in Networks and Systems*. Singapore: Springer Nature Singapore, 2025. P. 371–387. DOI: https://doi.org/10.1007/978-981-96-8901-9_33
30. Chandrika V. O., Kumar S. M. Design and analysis of SRAM cell using reversible logic gates towards smart computing. *The Journal of*

Supercomputing. 2022. Vol. 78, № 2. P. 2287–2306. DOI: https://doi.org/10.1007/s11227-021-03851-z

31. Pawlowski M., Szyprowski Z. Implementation of reversible gates in FPGA structure. *Proceedings of SPIE*. 2017. Vol. 10445. 1044517. DOI: https://doi.org/10.1117/12.2280690
32. Mehic M., Michalek L., Dervisevic E. et al. Quantum cryptography in 5G networks: a comprehensive overview. *IEEE Communications Surveys & Tutorials*. 2024. Vol. 26, № 1. P. 302–346. DOI: https://doi.org/10.1109/COMST.2023.3309051
33. Murali Krishna B., Sri Kavya K. S., Sai Kumar P. V. S., Karthik K., Siva N. Y. FPGA implementation of image cryptology using reversible logic gates. *International Journal of Advanced Trends in Computer Science and Engineering*. 2020. Vol. 9, № 3. P. 2522–2526. DOI: https://doi.org/10.30534/ijatcse/2020/06932020
34. Rajesh K., Umamaheswara R. G. FPGA implementation of encryption and decryption of a message using optimized reconfigurable reversible gate. *International Journal of Recent Technology and Engineering (IJRTE)*. 2019. Vol. 8, № 2. P. 1654–1658. DOI: https://doi.org/10.35940/ijrte.B2396.078219
35. Biswas S., Goswami R. S., Reddy K. H. K. Advancing quantum steganography: a secure IoT communication with reversible decoding and customized encryption technique for smart cities. *Cluster Computing*. 2024. Vol. 27. P. 9395–9414. DOI: https://doi.org/10.1007/s10586-024-04429-z
36. Shor P. W. Algorithms for quantum computation: discrete logarithms and factoring. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, Santa Fe, NM, USA, IEEE. Santa Fe, 1994. P. 124–134. DOI: https://doi.org/10.1109/sfcs.1994.365700
37. Kumaresan G., Gopalan N. P., Vetriselvi T. An Efficient Image Block Encryption for Key Generation using Non-Uniform Cellular Automata.

International Journal of Computer Network and Information Security (IJCNIS). 2019. Vol. 11, № 2. P. 28–35. DOI: https://doi.org/10.5815/ijcnis.2019.02.04

38. Saranya K., Vijeyakumar K. A. Low Area FPGA Implementation of Reversible Gate Encryption with Heterogeneous Key Generation. *Circuits, Systems, and Signal Processing*. 2021. Vol. 40. P. 3836–3865. DOI: https://doi.org/10.1007/s00034-021-01649-1
39. Szymański Z., Pawłowski M. Symmetric block encoder based on reversible circuits. *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018* / ed. by R. S. Romaniuk, M. Linczuk. SPIE, 2018. P. 568–574. DOI: https://doi.org/10.1117/12.2501438
40. Chouhan V., Aldarwbi M., Sadeghi S., Ghorbani A., Chow A., Burko R. Assessing the quantum readiness of cryptographic standards: Recommendations toward quantum-era compliance. *Computer Standards & Interfaces*. 2025. Vol. 101. P. 104114. DOI: https://doi.org/10.2139/ssrn.5256004
41. Mehrnahad Z., Latif A. A Novel Image Encryption Scheme Based on Reversible Cellular Automata and Chaos. *International Journal of Information Technology and Computer Science (IJITCS)*. 2019. Vol. 11, № 11. P. 15–23. DOI: https://doi.org/10.5815/ijitcs.2019.11.02
42. Dovhaniuk O., Deibuk V. Synthesis and implementation of reconfigurable reversible generalized Fredkin gate. *Proceedings of the 2021 IEEE 12th International Conference on Electronics and Information Technologies (ELIT)*, Ukraine, IEEE, 2021. P. 165–169. DOI: https://doi.org/10.1109/ELIT53502.2021.9501129
43. Moraga C., Hajam F. Z. The Fredkin gate in reversible and quantum environments. *Facta Universitatis, Series: Electronics and Energetics*. 2023. Vol. 36, № 2. P. 253–266. DOI: https://doi.org/10.2298/fuee2302253m
44. Singh S., Choudhary A., Kumar Jain M. An Optimized Approach towards Reversible Adder/Subtractor Design on QCA. *International Journal of Modern*

Education and Computer Science (IJMECS). 2019. Vol. 11, № 10. P. 47–53. DOI: https://doi.org/10.5815/ijmecs.2019.10.06

45. McCluskey E. J. Minimization of Boolean Functions. *Bell System Technical Journal.* 1956. Vol. 35, № 6. P. 1417–1444. DOI: https://doi.org/10.1002/j.1538-7305.1956.tb03835.x
46. Gendreau M., Potvin J.-Y. Tabu search. *Handbook of metaheuristics* / ed. by M. Gendreau, J.-Y. Potvin. 3rd ed. Cham : Springer International Publishing, 2018. P. 37–55. DOI: https://doi.org/10.1007/978-3-319-91086-4_2
47. Automatic Test Generation for Combinational Circuits. *Logic Synthesis and Verification Algorithms*. Boston : Springer US, 2002. P. 475–503. DOI: https://doi.org/10.1007/0-306-47592-8_12
48. Abdessaied N., Drechsler R. *Reversible and Quantum Circuits. Optimization and Complexity Analysis*. Cham : Springer, 2016. 238 p. DOI: https://doi.org/10.1007/978-3-319-31937-7
49. Miller D. M., Wille R., Drechsler R. Reducing Reversible Circuit Cost by Adding Lines. *Proceedings of the 2010 40th IEEE International Symposium on Multiple-Valued Logic*, Barcelona, Spain, IEEE. Barcelona, 2010. P. 217–222. DOI: https://doi.org/10.1109/ISMVL.2010.48
50. Hadjam F. Z., Moraga C. RIMEP2. *ACM Journal on Emerging Technologies in Computing Systems*. 2014. Vol. 11, № 3. P. 1–23. DOI: https://doi.org/10.1145/2629534
51. Synthesis of Reversible Circuits Using Heuristic Search Method / K. Datta et al. *Proceedings of the 2012 25th International Conference on VLSI Design*, Hyderabad, India, IEEE. Hyderabad, 2012. P. 139–144. DOI: https://doi.org/10.1109/vlsid.2012.92
52. Efficient adder circuits based on a conservative reversible logic gate / J. W. Bruce et al. *Proceedings of the IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design (ISVLSI 2002)*, Pittsburgh, PA,

USA, IEEE. Pittsburgh, 2002. P. 74–79. DOI: https://doi.org/10.1109/isvlsi.2002.1016879

53. Held M., Karp R. M. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*. 1971. Vol. 1, № 1. P. 6–25. DOI: https://doi.org/10.1007/bf01584070

54. Sekanina L. Evolutionary hardware design. *Proceedings of SPIE 8067, VLSI Circuits and Systems V* / ed. by T. Riesgo, E. de la Torre-Arnanz. Prague, Czech Republic : SPIE. Prague, 2011. P. 806702. DOI: https://doi.org/10.1117/12.886269

55. Volpe D., Orlandi G., Turvani G. Improving the Solving of Optimization Problems: A Comprehensive Review of Quantum Approaches. *Quantum Reports*. 2025. Vol. 7, № 1. P. 3. DOI: https://doi.org/10.3390/quantum7010003

56. Wille R., Drechsler R. BDD-based synthesis of reversible logic. *International Journal of Applied Metaheuristic Computing (IJAMC)*. 2010. Vol. 1, № 4. P. 25–41. DOI: https://doi.org/10.4018/jamc.2010100102

57. Shahidi S. M., Borujeni E. S. A new method for reversible circuit synthesis using a simulated annealing algorithm and don't-cares. *Computing and Electronics in Agriculture*. 2021. Vol. 20, № 1. P. 718–734. DOI: https://doi.org/10.1007/s10825-020-01620-4

58. Deibuk V., Biloshytskyi A. Design of a ternary reversible/quantum adder using genetic algorithm. *International Journal of Information Technology and Computer Science (IJITCS)*. 2015. Vol. 7, № 9. P. 38–45. DOI: https://doi.org/10.5815/ijitcs.2015.09.06

59. Ghosh M., Dey N., Mitra D., Chakrabarti A. A novel quantum algorithm for ant colony optimization. *IET Quantum Communication*. 2022. Vol. 3, № 1. P. 13–29. DOI: https://doi.org/10.1049/qtc2.12023

60. Khilkova L. O. Cross-Selection Based Evolution Strategies. *International Journal of Computing*. 2023. Vol. 22, № 1. P. 69–77. DOI: https://doi.org/10.47839/ijc.22.1.2881

61. Sasamal T. N., Singh A. K., Mohan A. Reversible Logic Circuit Synthesis and Optimization using Adaptive Genetic Algorithm. *Procedia Computer Science*. 2015. Vol. 70. P. 407–413. DOI: https://doi.org/10.1016/j.procs.2015.10.054
62. Wang X. et al. Hybrid Encoding Evolutionary Algorithm for Reversible Logic Synthesis. *Journal of Physics*. 2023. Vol. 2504, № 1. P. 012040. DOI: https://doi.org/10.1088/1742-6596/2504/1/012040
63. Wille R., Große D., Teuber L., Dueck G. W., Drechsler R. RevLib: An Online Resource for Reversible Functions and Reversible Circuits. *Proceedings of the 2008 38th International Symposium on Multiple-Valued Logic (ISMVL)*, Dallas, TX, USA, IEEE. Dallas, 2008. P. 220–225. DOI: https://doi.org/10.1109/ismvl.2008.43
64. Moraga C. OR-Toffoli and OR-peres reversible gates. *Reversible Computation (RC 2021)* / ed. by S. Yamashita, T. Yokoyama. Lecture Notes in Computer Science, vol. 12805. Springer, Cham, 2021. P. 235–241. DOI: https://doi.org/10.1007/978-3-030-79837-6_17
65. Rozhdov O. I., Yuriychuk I. M., Deibuk V. G. Building a Generalized Peres Gate with Multiple Control Signals. *Advances in Intelligent Systems and Computing*. Cham : Springer, 2018. Vol. 754. P. 155–164. DOI: https://doi.org/10.1007/978-3-319-91008-6_16
66. Maslov D., Dueck G. W. Comparison of the Cost Metrics for Reversible and Quantum Logic Synthesis. *arXiv preprint quant-ph/0511008*. 2005. DOI: https://doi.org/10.48550/arXiv.quant-ph/0511008
67. Dalcumune E. et al. A reversible circuit synthesis algorithm with progressive increase of controls in generalized Toffoli gates. *JUCS — Journal of Universal Computer Science*. 2021. Vol. 27, № 6. P. 544–563. DOI: https://doi.org/10.3897/jucs.69617
68. Naz S. F., Shah A. P. Reversible Gates: A Paradigm Shift in Computing. *IEEE Open Journal of Circuits and Systems*. 2023. Vol. 4. P. 192–215. DOI: https://doi.org/10.1109/OJCAS.2023.3305557

69. Feynman R. P. Quantum mechanical computers. *Foundations of Physics*. 1986. Vol. 16, № 6. P. 507–531. DOI: https://doi.org/10.1007/bf01886518
70. Sasamal T. N., Gaur H. M., Singh A. K., Mohan A. Reversible circuit synthesis using evolutionary algorithms. *Lecture Notes in Electrical Engineering*. 2020. Vol. 577. P. 115–128. DOI: https://doi.org/10.1007/978-981-13-8821-7_7
71. Gorskyi G. P., Deibuk V. G. Four spins model of universal quantum Fredkin gate. *Information Technologies and Computer Engineering.* 2011. Vol. 21, № 2. P. 56–63.
72. Wootters W. K., Zurek W. H. A single quantum cannot be cloned. *Nature*. 1982. Vol. 299, № 5886. P. 802–803. DOI: https://doi.org/10.1038/299802a0
73. Deb A. et al. An Efficient Reduction of Common Control Lines for Reversible Circuit Optimization. *Proceedings of the 2015 IEEE 45th International Symposium on Multiple-Valued Logic (ISMVL)*, Waterloo, ON, Canada, IEEE. Waterloo, 2015. P. 120–125. DOI: https://doi.org/10.1109/ismvl.2015.26
74. Lee J. et al. Toffoli-depth reduction method preserving in-place quantum circuits and its application to SHA3-256. *Quantum Information Processing*. 2024. Vol. 23, № 4. DOI: https://doi.org/10.1007/s11128-024-04365-2.
75. Bennett C. H. Logical Reversibility of Computation. *IBM Journal of Research and Development*. 1973. Vol. 17, № 6. P. 525–532. DOI: https://doi.org/10.1147/rd.176.0525
76. Kheirandish D., Haghparast M., Reshadi M., Hosseinzadeh M. Efficient techniques for fault detection and location of FPGA implementation of a fault-tolerant encryptor based on reconfigurable Fredkin gates. *Quantum Information Processing*. 2021. Vol. 20. Art. № 370. DOI: https://doi.org/10.1007/s11128-021-03292-w
77. Gaur H. M., Singh A. K., Ghanekar U. Synthesis of Fault-tolerant Reversible Circuits using Parity Preserving Gates. *Defence Science Journal*. 2018. Vol. 68, № 4. P. 370–376. DOI: https://doi.org/10.14429/dsj.68.11328

78. Patel A., Markov I. L., Hayes J. P. Fault testing for reversible circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2004. Vol. 23, № 8. P. 1220–1230. DOI: https://doi.org/10.1109/TCAD.2004.831576
79. Handique M., Deka J. K., Biswas S. Detection of Stuck-at and Bridging Fault in Reversible Circuits using an Augmented Circuit. *Proceedings of the 2021 IEEE 30th Asian Test Symposium (ATS)*, Matsuyama, Ehime, Japan, IEEE. Matsuyama, Ehime, 2021. P. 55–60. DOI: https://doi.org/10.1109/ats52891.2021.00022
80. Haghparast M., Shoaei S. Design of a New Parity Preserving Reversible Full Adder. *Journal of Circuits, Systems and Computers*. 2014. Vol. 24, № 01. Art. № 1550006. DOI: https://doi.org/10.1142/S0218126615500061
81. Battistel F. et al. Real-time decoding for fault-tolerant quantum computing: progress, challenges and outlook. *Nano Futures*. 2023. Vol. 7, № 3. Art. № 032003. DOI: https://doi.org/10.1088/2399-1984/aceba6
82. Lima R. B. F., Kowada L. A. B., Santos F. G. d. Improving Qiskit strategies for circuit synthesis using the Reed-Muller spectrum. *Anais do Seminário Integrado de Software e Hardware (SEMISH)*, Brasil. 2025. P. 573–584. DOI: https://doi.org/10.5753/semish.2025.9256
83. Shende V. V., Prasad A. K., Markov I. L., Hayes J. P. Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2003. Vol. 22, № 6. P. 710–722. DOI: https://doi.org/10.1109/tcad.2003.811448
84. Miller D. M., Maslov D., Dueck G. W. A transformation based algorithm for reversible logic synthesis. *Proceedings of the 40th annual Design Automation Conference (DAC '03)*. Anaheim, CA, USA. Anaheim, 2003. P. 318–323. DOI: https://doi.org/10.1145/775832.775915

85. Maslov D., Dueck G. W., Miller D. M. “Synthesis of Fredkin-Toffoli reversible networks”. *IEEE Trans. Very Large Scale Integration (VLSI) Syst.* 2005. Vol. 13, No. 6, P. 765–769
86. Miller J. F. Introduction to Evolutionary Computation and Genetic Programming. *Cartesian Genetic Programming*. Berlin, Heidelberg: Springer, 2011. P. 1–16. DOI: https://doi.org/10.1007/978-3-642-17310-3_1
87. Ravi L.S, Naveen K.B, Dankan Gowda V, Nagesh R. Simulation and Analysis of Reversible Fault-Tolerant Gate-Based Configurable Logic Blocks for Robust FPGA Architecture and Computational Efficiency. *International Journal of Electronics and Communication Engineering (IJECE)*. 2025. Vol. 12, № 3. P. 68–78. DOI: https://doi.org/10.14445/23488549/ijece-v12i3p106.
88. Rajesh K. et al. Encryption and Decryption using optimized Reconfigurable Reversible Gate. *Proceedings of the 2023 3rd International Conference on Artificial Intelligence and Signal Processing (AISP)*, Vijayawada, India, IEEE. Vijayawada, 2023. P. 1–5. DOI: https://doi.org/10.1109/AISP57993.2023.10134930
89. Bryk M. et al. Encryption using reconfigurable reversible logic gate and its simulation in FPGAs. *Proceedings of the 2016 23rd International Conference Mixed Design of Integrated Circuits and Systems (MIXDES)*, Lodz, Poland, IEEE. Lodz, 2016. P. 203–206. DOI: https://doi.org/10.1109/MIXDES.2016.7529732
90. Thapliyal H., Srinivas M. B. Novel Reversible ‘TSG’ Gate and Its Application for Designing Components of Primitive Reversible/Quantum ALU. *Proceedings of the 2005 5th International Conference on Information Communications & Signal Processing*, Bangkok, Thailand, IEEE. Bangkok, 2005. P. 1425–1429. DOI: https://doi.org/10.1109/icics.2005.1689293.
91. Sarif B. A. B., Abd-El-Barr M., Sait S. M., Al-Saiari U. Fuzzified ant colony optimization algorithm for efficient combinational circuits synthesis. *Proceedings of the 2004 Congress on Evolutionary Computation*. Portland, OR,

USA, IEEE. Portland, 2004. Vol. 2. P. 1317–1324. DOI: https://doi.org/10.1109/cec.2004.1331049

92. Podlaski K. Ant colony optimization implementation for reversible synthesis in Walsh-Hadamard domain. *Intelligent Information Technologies (IIS)*. Lecture Notes in Computer Science, vol. 12141. Springer, Cham, 2020. P. 230–243. DOI: https://doi.org/10.1007/978-3-030-50426-7_18

93. Korb O., Stützle T., Exner T. E. An ant colony optimization approach to flexible protein–ligand docking. *Swarm Intelligence*. 2007. Vol. 1, № 2. P. 115–134. DOI: https://doi.org/10.1007/s11721-007-0006-9

94. Li M., Zheng Y., Hsiao M. S., Huang C. Reversible logic synthesis through ant colony optimization. *Proceedings of the 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Dresden, Germany, IEEE. Dresden, 2010. P. 441–446. DOI: https://doi.org/10.1109/date.2010.5457190

95. Hu Z., Deibuk V. Design of ternary reversible/quantum sequential elements. *Journal of Thermoelectricity*. 2018. № 1. P. 5–17.

96. Deibuk V., Grytsku I. Optimal synthesis of reversible quantum summators using genetic algorithm. *Journal of Computing*. 2013. Vol. 12, № 1. P. 32–41.

97. Smith T., Husbands P., Layzell P., O'Shea M. Fitness Landscapes and Evolvability. *Evolutionary Computation*. 2002. Vol. 10, № 1. P. 1–34. DOI: https://doi.org/10.1162/106365602317301754

98. Maslov D., Dueck G. W., Miller D. M. Techniques for the synthesis of reversible Toffoli networks. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*. 2007. Vol. 12, № 4. Art. № 42. DOI: https://doi.org/10.1145/1278349.1278355

99. Erdoğan G., Laporte G., Rodríguez-Chía A. M. Exact and heuristic algorithms for the Hamiltonian -median problem. *European Journal of Operational Research*. 2016. Vol. 253, № 2. P. 280–289. DOI: https://doi.org/10.1016/j.ejor.2016.02.012

100. Скобцов Ю. О. Основи еволюційних обчислень : навч. посіб. Київ : Видавничий дім «Слово», 2011. 248 с.

101. Almasaoodi M. R. et al. New Quantum Genetic Algorithm Based on Constrained Quantum Optimization. *Karbala International Journal of Modern Science*. 2023. Vol. 9, № 4. P. 505–516. DOI: https://doi.org/10.33640/2405-609x.3325

102. Ye F. et al. What Performance Indicators to Use for Self-Adaptation in Multi-Objective Evolutionary Algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '24)*. Melbourne, VIC, Australia, ACM. Melbourne, 2024. P. 1004–1012. DOI: https://doi.org/10.1145/3638529.3654073.

103. Wille R., Van Meter R., Naveh Y. IBM's Qiskit tool chain: working with and developing for real quantum computers. *Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Florence, Italy, IEEE. Florence, 2019. P. 1234–1240. DOI: https://doi.org/10.23919/date.2019.8715261

104. Singh R. et al. Design and Analysis of Quantum Binary Adder-Subtractor Using IBM Qiskit. *Proceedings of the 2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS)*. Bhubaneswar, India, IEEE. Bhubaneswar, 2023. P. 1–5. DOI: https://doi.org/10.1109/CCPIS59145.2023.10292104

105. Newbold T., Moraglio A. Evolving Quantum Logic Gate Circuits in Qiskit. *Companion Proceedings of the 2024 Genetic and Evolutionary Computation Conference (GECCO '24 Companion)*. Melbourne, VIC, Australia, ACM. Melbourne, 2024. P. 631–634. DOI: https://doi.org/10.1145/3638530.3664108

106. Barenco A. et al. Elementary gates for quantum computation. *Physical Review A*. 1995. Vol. 52, № 5. P. 3457–3467. DOI: https://doi.org/10.1103/physreva.52.3457

107. Gupta P., Agrawal A., Jha N. K. An Algorithm for Synthesis of Reversible Logic Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 2006. Vol. 25, № 11. P. 2317–2330. DOI: https://doi.org/10.1109/tcad.2006.871622

108. Battistel F. et al. Real-time decoding for fault-tolerant quantum computing: Progress, challenges and outlook. *Nano Futures*. 2023. Vol. 7, № 3. Art. № 032003. DOI: https://doi.org/10.48550/arXiv.2303.00054

109. Saeedi M., Zamani M. S., Sedighi M. Moving forward: A non-search based synthesis method toward efficient CNOT-based quantum circuit synthesis algorithms. *Proceedings of the 2008 Asia and South Pacific Design Automation Conference (ASP-DAC)*. Seoul, South Korea, IEEE. Seoul, 2008. P. 448–453. DOI: https://doi.org/10.1109/aspdac.2008.4484066

110. IBM Quantum. Qiskit [Електронний ресурс]. URL: https://www.ibm.com/quantum/qiskit (дата звернення: 20.03.2025).

111. Pedroni V. A. *Circuit Design with VHDL*. 3rd Edition. Cambridge, MA: The MIT Press, 2020. 600 p.

112. Cheng Y. L. et al. FPGA Implementation of a Real-Time Quantum Image Encryption System Based on Reversible Quantum Gate Computing. *Proceedings of the 2024 10th International Conference on Applied System Innovation (ICASI)*. Kyoto, Japan, IEEE. Kyoto, 2024. P. 7–9. DOI: https://doi.org/10.1109/ICASI60819.2024.10547832

113. De Vos A., De Baerdemacker S., Van Rentergem Y. *Synthesis of Quantum Circuits vs. Synthesis of Classical Reversible Circuits*. Cham: Springer International Publishing, 2018. DOI: https://doi.org/10.1007/978-3-031-79895-5

114. Wang B., Xie Y., Zhou S., Zhou C., Zheng X. Reversible data hiding based on DNA computing. *Computational Intelligence and Neuroscience*. 2017. Vol. 2017. P. 1–9. DOI: https://doi.org/10.1155/2017/7276084

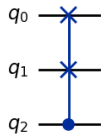
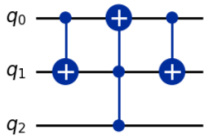
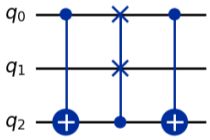
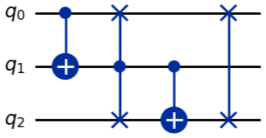
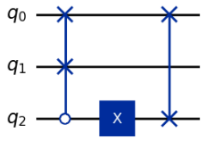
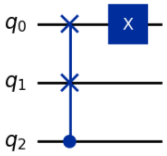
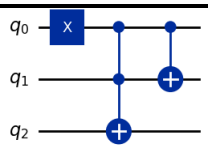
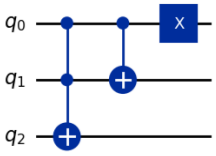
115. Miller D. M., Thornton M. A. QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits. *Proceedings of the 36th International Symposium on Multiple-Valued Logic (ISMVL'06)*. Singapore, IEEE. Singapore, 2006. P. 30–30. DOI: https://doi.org/10.1109/ismvl.2006.35

ДОДАТКИ

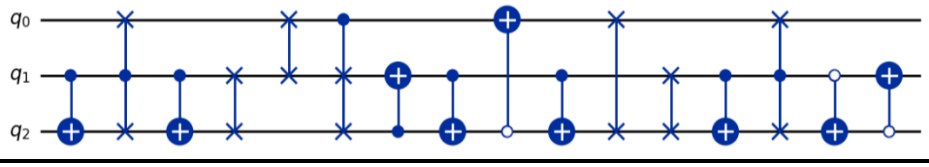
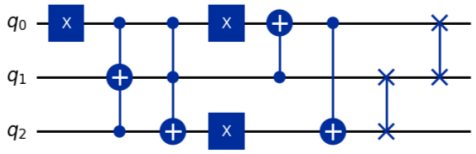
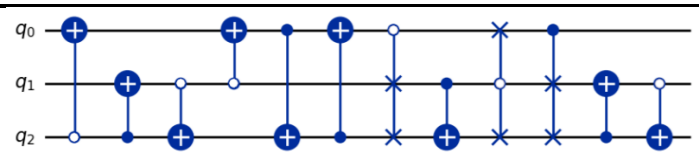
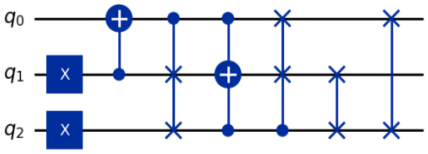
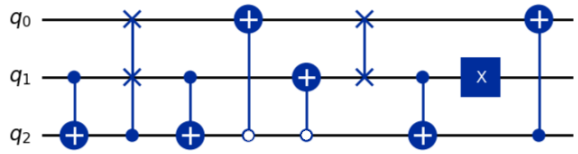
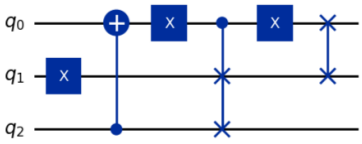
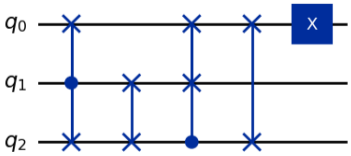
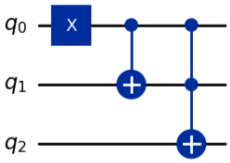
Додаток А

Синтезовані схеми в процесі дослідження

Таблиця Синтезовані бенчмарк схеми

Func-tion	Algorithm	Qc	Delay	Circuits
[0,1,2,3,4,6,5,7]	ACO	5	1	
	GA	7	3	
[0,1,2,4,3,5,6,7]	ACO	7	3	
	GA	8	4	
[1,0,3,2,5,7,4,6]	ACO	8	3	
	GA	6	2	
[1,2,3,4,5,6,7,0]	ACO	7	3	
	GA	7	3	

Продовження таблиці синтезовані бенчмарк схеми

[1,2,7,5,6,3,0,4]	ACO	31	16	
	GA	17	8	
[3,6,2,5,7,1,0,4]	ACO	30	12	
	GA	20	7	
[4,3,0,2,7,5,6,1]	ACO	15	9	
	GA	10	5	
[7,0,1,2,3,4,5,6]	ACO	13	5	
	GA	7	3	

Додаток Б

Лістинг основних частин коду програми C#

синтезу зворотних схем

Лістинг конвертера синтезованої хромосоми в QASM представлення.

```
using System.Text;
using QuantumCircuitGenerator.Api.Domain.Dto;
using QuantumCircuitGenerator.Api.Domain.Enum;

namespace QuantumCircuitGenerator.Api.Application.Helper;

public static class OpenOpenqasm20Converter
{
   private static Dictionary<GateType, string> gateTypeMappings = new()
   {
      { GateType.Identity, "id" },
      { GateType.Not, "x" },
      { GateType.Toffoli, "ccx" },
      { GateType.G3Toffoli, "c3x" },
      { GateType.G4Toffoli, "c4x" },
      { GateType.NToffoli, "ccx" },   //TODO Create a new custom field or find better solution see
https://qiskit.org/documentation/stubs/qiskit.circuit.ControlledGate.html
      { GateType.NNToffoli, "ccx" },  //TODO Create a new custom field or find better solution see
https://qiskit.org/documentation/stubs/qiskit.circuit.ControlledGate.html
      { GateType.Feynman, "cx" },
      { GateType.NFeynman, "cx" },    //TODO Create a new custom field or find better solution see
https://qiskit.org/documentation/stubs/qiskit.circuit.ControlledGate.html
      { GateType.Fredkin, "cswap" },
      { GateType.G2Fredkin, "c2swap" },
      { GateType.N1P1G2Fredkin, "c2swap" }, //TODO Temporary use c2swap with additional inversion before
transforming to QASM
      { GateType.NFredkin, "cswap" }, //TODO Create a new custom field or find better solution see
https://qiskit.org/documentation/stubs/qiskit.circuit.ControlledGate.html
      { GateType.NNFredkin, "c2swap" }, //TODO Temporary use c2swap with additional inversion before
transforming to QASM
      { GateType.N1P2G3Toffoli, "c3x" }, //TODO Temporary use c2swap with additional inversion before
transforming to QASM
      { GateType.N2P1G3Toffoli, "c3x" }, //TODO Temporary use c2swap with additional inversion before
transforming to QASM
      { GateType.Swap, "swap" },
   };

   public static string ConvertToOpenQasm(this Domain.Dto.Circuit.QuantumCircuit quantumCircuit, Dictionary<int,
int> circuitDontCareOutputs)
   {
      var sb = new StringBuilder();

      sb.AppendLine("OPENQASM 3.0;");
      sb.AppendLine("include \"stdgates.inc\";");
      sb.AppendLine();

      var numQubits = GetNumQubits(quantumCircuit);
      sb.AppendLine($"qreg q[{numQubits}];");
      sb.AppendLine($"creg c[{numQubits}];");
```

```
sb.AppendLine();

if (quantumCircuit.Circuit != null && quantumCircuit.Circuit.Exists(t =>
        t.GateLines.Exists(x => x.Name == GateType.G2Fredkin)
        || t.GateLines.Exists(x => x.Name == GateType.N1P1G2Fredkin)
        || t.GateLines.Exists(x => x.Name == GateType.NNFredkin) ))
{
    sb.AppendLine("gate c2swap a, b, c, d { ctrl @ cswap a, b, c, d; }");
}

if (quantumCircuit.Circuit != null && quantumCircuit.Circuit.Exists(t =>
    t.GateLines.Exists(x => x.Name == GateType.N1P2G3Toffoli)
    || t.GateLines.Exists(x => x.Name == GateType.N2P1G3Toffoli)
    || t.GateLines.Exists(x => x.Name == GateType.G3Toffoli)))
{
    sb.AppendLine("gate c3x a, b, c, d { ctrl @ ccx a, b, c, d; }");
}

// NFeynman
// gate N1cx() q1, q2 {
//     negctrl @ x q1, q2;
// }

if (circuitDontCareOutputs != null && circuitDontCareOutputs.Any())
{
    sb.AppendLine("gate state_preparation(param0) q0 { x q0; }");
    sb.AppendLine("gate pin(param0) q0 { }");

    foreach (var dontCareOutput in circuitDontCareOutputs)
    {
        sb.AppendLine($"pin({ dontCareOutput.Value }) q[{ dontCareOutput.Key }];");
    }
}

if (quantumCircuit.Circuit == null)
{
    return sb.ToString();
}

foreach (var quantumState in quantumCircuit.Circuit) {
    var gates = quantumState.GateLines.GroupBy(t => t.Gate.Uid);

    foreach (var groupedGate in gates) {
        var gate = groupedGate.FirstOrDefault()?.Gate;
        if (gate == null || gate.Name == GateType.Identity)
            continue;

        var gateName = gateTypeMappings[gate.Name];
        var positiveControlledLines = groupedGate.Where(t => t.ControlType == ControlType.Positive)
            .Select(t => $"q[{quantumState.GateLines.IndexOf(t)}]");
        var negativeControlledLines = groupedGate.Where(t => t.ControlType == ControlType.Negative)
            .Select(t => $"q[{quantumState.GateLines.IndexOf(t)}]");
        var dataLines = groupedGate.Where(t => t.ControlType == ControlType.None)
            .Select(t => $"q[{quantumState.GateLines.IndexOf(t)}]");

        if (negativeControlledLines.Any()) {
            foreach (var controlledLine in negativeControlledLines) {
                sb.Append("x "); sb.Append(controlledLine); sb.AppendLine(";");
            }
        }
```

```
            sb.Append($"{gateName} ");

            if (positiveControlledLines.Any())
            {
               sb.AppendJoin(", ",positiveControlledLines); sb.Append(", ");
            }

            if (negativeControlledLines.Any())
            {
               sb.AppendJoin(", ",negativeControlledLines); sb.Append(", ");
            }

            sb.AppendJoin(", ",dataLines); sb.AppendLine(";");

            if (negativeControlledLines.Any()) {
               foreach (var controlledLine in negativeControlledLines) {
                  sb.Append("x "); sb.Append(controlledLine); sb.AppendLine(";");
               }
            }
         }
      }


      for (var i = 0; i < quantumCircuit.Circuit[0].GateLines.Count; i++)
      {
         if (circuitDontCareOutputs != null ||
            circuitDontCareOutputs.ContainsKey(i))
         {
            sb.AppendLine($"measure q[{i}] -> c[{i}];");
         }
      }

      return sb.ToString();
   }

   private static int GetNumQubits(Domain.Dto.Circuit.QuantumCircuit quantumCircuit) =>
      quantumCircuit.Circuit.FirstOrDefault()?.GateLines.Count ?? 0;
}
```

Лістинг сервісу обрахунку таблиці істинності синтезованої схеми-хромосоми

```
public class ManualCircuitMeasurementService : ICircuitMeasurementService
{
   public int[][] CalculateCircuitTruthTable(Domain.Dto.Circuit.QuantumCircuit circuit, Dictionary<int, int>
persistentInputs = null)
   {
      var qubitCount = circuit.Circuit.First().GateLines.Count;
      var outputTruthTable = new List<int[]>();

      var inputTruthTable = TruthTableHelper.InitInputTruthTable(qubitCount);

      var filteredInputTruthTable = TruthTableHelper.FilterPersistInput(persistentInputs, inputTruthTable);

      foreach (var truthTableVariant in filteredInputTruthTable)
      {
         foreach (var quantumState in circuit.Circuit)
         {
            // extracting gates from current quantum State, each gateLine contain corresponding gate Uid
            var gateLinesList = quantumState.GateLines.GroupBy(t => t.Gate);
```

```
            foreach (var gateLines in gateLinesList)
            {
                if (gateLines.Key.Name == GateType.Identity)
                {
                    continue;
                }

                var gateLineIndexes = gateLines
                    .Select(t => quantumState.GateLines.IndexOf(t))
                    .ToList();

                var qubitsEffectedByGate = gateLineIndexes
                    .Select(t => truthTableVariant[t])
                    .ToArray();

                var gate = gateLines.Key;
                gate.Measure(qubitsEffectedByGate, gateLines.ToList());

                var qubitEffectedByGateEnumerator = qubitsEffectedByGate.GetEnumerator();
                qubitEffectedByGateEnumerator.MoveNext();
                foreach (var effectedGateLineIndex in gateLineIndexes)
                {
                    truthTableVariant[effectedGateLineIndex] = (int)qubitEffectedByGateEnumerator.Current;
                    qubitEffectedByGateEnumerator.MoveNext();
                }
            }
        }

        outputTruthTable.Add(truthTableVariant);
    }

    return outputTruthTable.ToArray();
  }
}
```

Основний сервіс запуску синтезу

```
namespace QuantumCircuitGenerator.Api.Application.Services;

public class QuantumCircuitService : IQuantumCircuitService
{
  private readonly ICircuitMeasurementService _circuitMeasurementService;

  // private static readonly Random _random = new Random();
  private GeneticAlgorithm GeneticAlgorithm;
  private Random _random;

  public QuantumCircuitService(ICircuitMeasurementService circuitMeasurementService)
  {
    _random = new Mcg31m1();
    _circuitMeasurementService = circuitMeasurementService;
  }

  public (Domain.Dto.Circuit.QuantumCircuit BestCircuit, float BestFitness, int QuantumCost, int HeimerDistance,
    Dictionary<int, int> DontCareOutputs, List<float> FitnessProgression, IList<SwapGate> SwapTable)
    InitCircuitViaGeneticAlgorithm(CircuitParameters circuitParameters,
      GeneticAlgorithmParameters geneticAlgorithmParameters)
  {
    var dnaSize = circuitParameters.QuantumDelay;
    var populationSize = geneticAlgorithmParameters.PopulationSize;
```

```
        var errorCoefficient = geneticAlgorithmParameters.ErrorCoefficient;
        var costCoefficient = geneticAlgorithmParameters.CostCoefficient;
        var elitism = geneticAlgorithmParameters.Elitism;
        var mutationRate = geneticAlgorithmParameters.MutationRate;
        var mutationTypeProbability = geneticAlgorithmParameters.MutationTypeProbability;
        var persistentInputs = circuitParameters.PersistentInputs;

        var timer = new Stopwatch();
        timer.Start();

        GeneticAlgorithm = new GeneticAlgorithm(
            populationSize,
            dnaSize,
            _random,
            InitRandomQuantumStatesWrapper,
            CalculateFitnessFunctionWrapper,
            MutateWrapper,
            elitism,
            mutationRate,
            geneticAlgorithmParameters.CrossoverRate,
            populationSize / 10
        );

        CircuitEvaluationResult CalculateFitnessFunctionWrapper(Dna dna) =>
            CalculateFitnessFunction(circuitParameters.OutTruthTable, dna, errorCoefficient, costCoefficient,
persistentInputs);

        QuantumState InitRandomQuantumStatesWrapper(bool isInitialPopulation = false)
        {
            return isInitialPopulation
                ? InitRandomQuantumStates(circuitParameters.QubitCount, new List<GateType> { GateType.Identity })
                : InitRandomQuantumStates(circuitParameters.QubitCount, circuitParameters.GateTypes);
        }

        (bool isMutated, QuantumState gene) MutateWrapper(QuantumState gene) =>
            Mutate(circuitParameters.QubitCount, circuitParameters.GateTypes, mutationTypeProbability, gene);

        //TODO Move to input params
        var exitStagnationValue = 0.10f;
        var stagnationCounter = 0;
        var maxStagnationCounter = 10;
        var mutationRateStagnationValue = 0.01f;
        var crossoverRateStagnationValue = 0.01f;

        var generationCounter = 0;
        var fitnessProgression = new List<float>();
        var lastBestFitness = 0f;
        var generationToFindCorrectAnswer = 0;

        while (GeneticAlgorithm.BestResult.Fitness < (errorCoefficient + costCoefficient) && generationCounter <
geneticAlgorithmParameters.MaxGenerationCount)
        {
            GeneticAlgorithm.NewGeneration();
            generationCounter++;
            fitnessProgression.Add(GeneticAlgorithm.BestIndividual.CircuitEvaluationResult.Fitness);


            if (timer.IsRunning && GeneticAlgorithm.BestResult.HeimerDistance == 0)
            {
                generationToFindCorrectAnswer = generationCounter;
```

```
            timer.Stop();
        }

    }

    Log.Debug("Time taken: {TimerElapsed}; Generation = {GenerationToFindCorrectAnswer}", timer.Elapsed,
generationToFindCorrectAnswer);

    var bestCircuit = AdjustWithSwaps(GeneticAlgorithm.BestGenes, GeneticAlgorithm.BestResult.SwapTable,
circuitParameters.QubitCount);

    return (
        BestCircuit: bestCircuit,
        BestFitness: GeneticAlgorithm.BestResult.Fitness,
        QuantumCost: GeneticAlgorithm.BestResult.QuantumCost,
        HeimerDistance: GeneticAlgorithm.BestResult.HeimerDistance,
        DontCareOutputs: GeneticAlgorithm.BestResult.DontCareOutputs,
        FitnessProgression: fitnessProgression,
        SwapTable: GeneticAlgorithm.BestResult.SwapTable
    );
}

private Domain.Dto.Circuit.QuantumCircuit AdjustWithSwaps(
    List<QuantumState> genes,
    IList<SwapGate> swapTable,
    int qubitCount)
{
    var swapAdjustmentCircuit = new List<QuantumState>();

    foreach (var swapItem in swapTable)
    {
        var gateList = new List<Gate>();
        for (var i = 0; i < qubitCount; i++)
        {
            gateList.Add(QuantumLibrary.GetEmptyGate());
        }

        var gateLines = gateList
            .SelectMany(t => t.GateLines)
            .ToList();
        var swap = QuantumLibrary.GetSwapGate();

        //Replace empty gates by line of switch gate
        gateLines[swapItem.InputIndex] = swap.GateLines.First();
        gateLines[swapItem.OutputIndex] = swap.GateLines.Last();

        swapAdjustmentCircuit.Add(new QuantumState
        {
            GateLines = gateLines
        });
    }

    genes.AddRange(swapAdjustmentCircuit);
    return new Domain.Dto.Circuit.QuantumCircuit
    {
        Circuit = genes
    };
}

private CircuitEvaluationResult CalculateFitnessFunction(
```

```
        int[][] specimenTruthTable,
        Dna dna,
        float errorCoefficient,
        float costCoefficient,
        Dictionary<int, int> persistentInputs = null)
    {
        var measuredTruthTable = _circuitMeasurementService.CalculateCircuitTruthTable(new
Domain.Dto.Circuit.QuantumCircuit
        {
            Circuit = dna.Genes.ToList()
        }, persistentInputs);

        var heimerDistance = CalculateTruthTableHeimerDistance(measuredTruthTable, specimenTruthTable);
        var costCount = dna.Genes
            .SelectMany(t =>
                t.GateLines
                    .Where(x => x.Gate.Name != GateType.Identity)
                    .Select(x => (x.GateUid, x.Gate))
                )
            .DistinctBy(x => x.GateUid)
            .Sum(x => x.Gate.QuantumCost);

        var errorFitnessComponent = errorCoefficient * (1.0f / (heimerDistance.Distance + 1));
        var costFitnessComponent = costCoefficient * (1.0f / (costCount + 1));

        return new CircuitEvaluationResult{
            Fitness = errorFitnessComponent + costFitnessComponent,
            HeimerDistance = heimerDistance.Distance,
            QuantumCost = costCount,
            SwapTable = heimerDistance.SwapTable
                .Select(t => new SwapGate
                {
                    InputIndex = t.InputIndex,
                    OutputIndex = t.OutputIndex,
                })
                .ToList(),
            DontCareOutputs = persistentInputs
        };
    }

    private (bool isMutated, QuantumState gene) Mutate(
        int qubitCount,
        List<GateType> gateTypes,
        (float switchLineProbability, float addGateProbability, float removeGateProbability) mutationTypeProbability,
        QuantumState gene)
    {
        var linesCount = qubitCount;
        var mutateVariator = _random.NextDouble();

        //switch line
        if (mutateVariator <= mutationTypeProbability.switchLineProbability && qubitCount > 1)
        {
            var firstLineIndex = _random.Next(linesCount);
            int secondLineIndex;
            do
            {
                secondLineIndex = _random.Next(linesCount);
            } while (secondLineIndex == firstLineIndex);

            (gene.GateLines[firstLineIndex], gene.GateLines[secondLineIndex]) =
```

```
            (gene.GateLines[secondLineIndex], gene.GateLines[firstLineIndex]);

        return (true, gene);
    }

    //add gate
    if (mutateVariator <= mutationTypeProbability.addGateProbability)
    {
        var qubitStateAvailable = gene.GateLines.Count(t => t.Gate.Name == GateType.Identity);
        var newGateFunc = GetGateFunc(gateTypes, qubitStateAvailable);
        var newGate = newGateFunc();
        //return same gate, as there is no free space for adding new gate
        if (newGate.Name == GateType.Identity)
        {
            return (true, gene);
        }

        //Adding the new gate to the quantumState
        gene.GateLines.AddRange(newGate.GateLines);

        //remove empty overflow gateLines
        var overflowCount = gene.GateLines.Count - qubitCount;
        var count = 0;
        gene.GateLines.RemoveAll(t =>
        {
            if (count < overflowCount && t.Name == GateType.Identity)
            {
                count++;
                return true;
            }

            return false;
        });
        return (true, gene);
    }

    //remove random gate from gene
    if (mutateVariator <= mutationTypeProbability.removeGateProbability)
    {
        var gateToRemove = gene.GateLines.FirstOrDefault(t => t.Gate.Name != GateType.Identity)?.Gate;
        if (gateToRemove == null)
        {
            return (true, gene);
        }

        for (var i = 0; i < gene.GateLines.Count; i++)
        {
            if (gene.GateLines[i].GateUid == gateToRemove.Uid)
            {
                var newEmptyGate = QuantumLibrary.GetEmptyGate();
                gene.GateLines[i] = newEmptyGate.GateLines.Single();
            }
        }
    }

    return (true, gene);
}

private (int Distance, List<(int InputIndex, int OutputIndex)> SwapTable)
    CalculateTruthTableHeimerDistance(int[][] measuredTruthTable, int[][] specimenTruthTable)
```

```
{
   var distances = new List<(int InputIndex, int OutputIndex, int distance)>();
   for (int k = 0; k < specimenTruthTable[0].Length; k++)
   {
      for (int i = 0; i < measuredTruthTable[0].Length; i++)
      {
         var counter = 0;
         for (int j = 0; j < measuredTruthTable.Length; j++)
         {
            if (measuredTruthTable[j][i] != specimenTruthTable[j][k])
            {
               counter++;
            }
         }

         distances.Add((i, k, counter));
      }
   }
   // TODO distances remove some indexes so 1,0 will have same distance as 0,1

   var lowestDistances = new List<(int InputIndex, int OutputIndex, int distance)>();

   foreach (var kv in distances.GroupBy(t => t.OutputIndex))
   {
       var item = kv
         .OrderBy(t => t.distance)
         .FirstOrDefault(t =>
            !lowestDistances
               .Select(ld => ld.InputIndex)
               .Contains(t.InputIndex)
         );

      if (item != default)
      {
         lowestDistances.Add(item);
      }
   }

   var distanceSum = lowestDistances.Sum(t => t.distance);
   var switchTable = lowestDistances
      .Select(t => (t.InputIndex, t.OutputIndex))
      .Where(t => t.InputIndex != t.OutputIndex)
      .ToList();

   //adjust swapTable if it contain 2 swap item in a row like 0 -> 1, 1 -> 0
   /*if (switchTable.Count > 1){
      var adjustedSwitchTable = switchTable;
      for (var index = 0; index < switchTable.Count-1; index++)
      {
         var current = switchTable[index];
         var next = switchTable[index + 1];

         if (current.InputIndex == next.OutputIndex && current.OutputIndex == next.InputIndex)
         {
            Log.Information("Adjusting swap table");
            adjustedSwitchTable.Remove(current);
         }
      }

      switchTable = adjustedSwitchTable;
```

```
        }*/

        return (
            Distance: distanceSum,
            SwapTable: switchTable
        );
    }

    private static QuantumState InitRandomQuantumStates(int qubitCount, List<GateType> gateTypes)
    {
        var filledQubitStateSpace = 0;
        var gateList = new List<Gate>();
        while (filledQubitStateSpace < qubitCount)
        {
            var qubitStateSpaceLeft = qubitCount - gateList.SelectMany(t => t.GateLines).Count();
            gateList.Add(GenerateGate(gateTypes, qubitStateSpaceLeft));
            filledQubitStateSpace = gateList.SelectMany(t => t.GateLines).Count();
        }

        var quantumState = new QuantumState
        {
            GateLines = gateList
                .SelectMany(t => t.GateLines)
                .ToList()
                .Shuffle()
                .ToList()
        };

        return quantumState;
    }

    private static Gate GenerateGate(List<GateType> gateTypes, int qubitStateSpaceLeft, bool
shouldBeOnlyEmptyGate = false, bool shouldIncludeEmptyGate = true)
    {
        var getGateMethods = new List<Func<Gate>>
        {
            GetGateFunc(gateTypes, qubitStateSpaceLeft)
        };

        if (shouldIncludeEmptyGate)
        {
            getGateMethods.Add(QuantumLibrary.GetEmptyGate);
        }

        //shuffled list of functions for generating gates and take top one
        var shuffledGateMethods = getGateMethods
            .Shuffle()[0];

        var gate = shuffledGateMethods();
        return gate;
    }

    private static Func<Gate> GetGateFunc(List<GateType> gateTypes, int qubitStateSpaceLeft)
    {
        //TODO Tage only gate which can fit to current gene
        //TODO Take gates for random which have capacity <= qubitStateSpaceLeft
        gateTypes ??= Enum.GetValues<GateType>()
            .ToList();

        var random = new Random();
```

```
        var gateType = gateTypes[random.Next(gateTypes.Count)];

        return QuantumLibrary.GetGateByTypeAndCapacity(qubitStateSpaceLeft, gateType);
    }
}
```

Код генетичного алгоритму

```
using QuantumCircuitGenerator.Api.Application.Helper;
using QuantumCircuitGenerator.Api.Domain.Dto;
using QuantumCircuitGenerator.Api.Domain.Dto.Circuit;
using Serilog;

namespace QuantumCircuitGenerator.Api.Application.GeneticStrategy;

public class GeneticAlgorithm
{
    public List<Dna> Population { get; private set; }
    public int Generation { get; private set; }

    public CircuitEvaluationResult BestResult { get; private set; } = new CircuitEvaluationResult();
    public List<QuantumState> BestGenes { get; private set; }
    public Dna BestIndividual { get; private set; }

    public int Elitism { get; set; }
    public float MutationRate { get; set; }
    public float CrossoverRate { get; set; }

    private List<Dna> _newPopulation;
    private readonly Random _random;
    private float _fitnessSum;
    private readonly int _dnaSize;
    private readonly Func<bool, QuantumState> _getRandomGene;
    private readonly Func<Dna, CircuitEvaluationResult> _fitnessFunction;
    private readonly Func<QuantumState, (bool isMutated, QuantumState gene)> _mutationFunc;
    private readonly int _populationSize;
    private readonly int _minimumUniquenessCount;

    public GeneticAlgorithm(
        int populationSize,
        int dnaSize,
        Random random,
        Func<bool, QuantumState> getRandomGene,
        //TODO Add Specific type for that func
        Func<Dna, CircuitEvaluationResult> fitnessFunction,
        Func<QuantumState, (bool isMutated, QuantumState gene)> mutationFunc,
        int elitism,
        float mutationRate = 0.01f,
        float crossoverRate = 1f,
        int minimumUniquenessCount = 10)
    {
        Generation = 1;
        Elitism = elitism;
        MutationRate = mutationRate;
        CrossoverRate = crossoverRate;
        Population = new List<Dna>(populationSize);

        _newPopulation = new List<Dna>(populationSize);
        _random = random;
        _dnaSize = dnaSize;
        _getRandomGene = getRandomGene;
```

```
        _fitnessFunction = fitnessFunction;
        _mutationFunc = mutationFunc;
        _minimumUniquenessCount = minimumUniquenessCount;
        _populationSize = populationSize;

        BestGenes = new List<QuantumState>();

        // Parallel.For(0, populationSize, (i, state) =>
        for (int i = 0; i < populationSize; i++)
        {
            Population.Add(new Dna(dnaSize, random, getRandomGene, fitnessFunction, mutationFunc, shouldInitGenes:
true, initialPopulation: true));
        }

        //TODO Add evaluation of the generation
        CalculateFitness();

        // );
    }

    public void NewGeneration(int numNewDna = 0, bool shouldEvolve = false)
    {
        try
        {
            var finalCount = _populationSize + numNewDna;

            if (finalCount <= 0) {
                return;
            }

            // if (Generation <= 1 && Population.Any()) {
            //     CalculateFitness();
            // }

            var newPopulation = new List<Dna>(_populationSize);

            var elites = SelectElites(Elitism);
            newPopulation.AddRange(elites);

            while (newPopulation.Count < _populationSize)
            {
                var parent1 =
                    //TournamentSelection(50);
                    RouletteWheelSelection();
                var parent2 =
                    //TournamentSelection(50);
                    RouletteWheelSelection();

                while (parent1 == null || parent2 == null || parent2 == parent1)
                {
                    parent2 = RouletteWheelSelection();
                }

                parent1 = parent1.Clone() as Dna;
                parent2 = parent2.Clone() as Dna;

                var children = Crossover(parent1, parent2);

                children.child1.Mutate(MutationRate);
                children.child2.Mutate(MutationRate);
```

```
            EnsureUnique(ref children.child1);
            EnsureUnique(ref children.child2);

            newPopulation.Add(children.child1);
            if (newPopulation.Count < _populationSize) //if was selected some unpair value for population size or elits
            {
                newPopulation.Add(children.child2);
            }
        }

        Population = new List<Dna>(newPopulation);
        CalculateFitness();
        // ReUniquePopulation();

        // Log.Debug(string.Join("   " ,Population.Select(t => $"|{t.CircuitEvaluationResult.Fitness,10}|")));
        // Log.Debug("Uniqueness Counter: {Count}", Population.DistinctBy(t =>
t.CircuitEvaluationResult.Fitness).Count());

        elites = Population
            .Take(Elitism)
            .ToList();

        var elitesForLog = string.Join(";", elites.Select(t => $"|{t.CircuitEvaluationResult.Fitness,10}|"));
        Log.Debug("Generation: {Generation}; Top fitness: {ElitesForLog}", Generation, elitesForLog);
        // Log.Debug(elitesForLog);

        Generation++;
    }
    catch (Exception e)
    {
        Log.Error(e, "Exception occured");
    }
}

private (Dna child1, Dna child2) Crossover(Dna parent1, Dna parent2)
{
    var child1 = new Dna(_dnaSize, _random, _getRandomGene, _fitnessFunction, _mutationFunc, shouldInitGenes:
false);
    var child2 = new Dna(_dnaSize, _random, _getRandomGene, _fitnessFunction, _mutationFunc, shouldInitGenes:
false);

    if (_random.NextDouble() < CrossoverRate && _dnaSize >= 2)
    {
        //TODO recheck cross points if all possible variants included
        var firstCrossPoint = SelectFirstCrossPoint();
        var secondCrossPoint = SelectSecondCrossPoint(firstCrossPoint);

        for (var i = 0; i < _dnaSize; i++)
        {
            if (i >= firstCrossPoint && i <= secondCrossPoint)//INCLUSIVE BETWEEN
            {
                child1.Genes[i] = parent1.Genes[i];
                child2.Genes[i] = parent2.Genes[i];
            }
            else
            {
                child1.Genes[i] = parent2.Genes[i];
                child2.Genes[i] = parent1.Genes[i];
            }
```

```
            }
        }
        else
        {
            child1.Genes = parent1.Genes;
            child2.Genes = parent2.Genes;
        }

        return (child1, child2);
    }

    private int SelectSecondCrossPoint(int firstCrossPoint)
    {
        int secondCrossPoint; //SELECT the very last point;
        var maxSecondCrossPointIndex = _dnaSize - 1;
        if (maxSecondCrossPointIndex <= 0)
        {
            secondCrossPoint = 0;
        }
        else
        {
            do
            {
                secondCrossPoint = _random.Next(0, maxSecondCrossPointIndex);
            } while (secondCrossPoint < firstCrossPoint);
        }

        return secondCrossPoint;
    }

    private int SelectFirstCrossPoint()
    {
        var maxFirstCrossPointIndex = _dnaSize - 2;
        var firstCrossPoint = maxFirstCrossPointIndex <= 0 //SELECT before the last point to prevent selecting the very
last cross point;
                ? 0
                : _random.Next(0, _dnaSize - 2);
        return firstCrossPoint;
    }

    private List<Dna> SelectElites(int numElites)
    {
        var elites = new List<Dna>(numElites);
        for (int i = 0; i < numElites; i++)
        {
            elites.Add(Population[i].Clone() as Dna);
        }
        return elites;
    }

    private int CompareDna(Dna a, Dna b)
    {
        if (a.CircuitEvaluationResult.Fitness < b.CircuitEvaluationResult.Fitness) {
            return 1;
        }

        return a.CircuitEvaluationResult.Fitness > b.CircuitEvaluationResult.Fitness
            ? -1
            : 0;
    }
```

```
private void CalculateFitness()
{
    _fitnessSum = 0;
    // for (var i = 0; i < _populationSize; i++)
    // {
    //     if (Population[i].CircuitEvaluationResult.Fitness == 0)
    //     {
    //         Population[i].CalculateFitness();
    //     }
    // }

    Parallel.For(0, _populationSize, i =>
    {
        if (Population[i].CircuitEvaluationResult.Fitness == 0)
        {
            Population[i].CalculateFitness();
        }
    });
    _fitnessSum += Population.Sum(t => t.CircuitEvaluationResult.Fitness);

    Population.Sort(CompareDna);

    BestIndividual = Population[0];
    BestResult = new CircuitEvaluationResult
    {
        Fitness = BestIndividual.CircuitEvaluationResult.Fitness,
        QuantumCost = BestIndividual.CircuitEvaluationResult.QuantumCost,
        HeimerDistance = BestIndividual.CircuitEvaluationResult.HeimerDistance,
        SwapTable = BestIndividual.CircuitEvaluationResult.SwapTable.DeepCopy(),
        DontCareOutputs = BestIndividual.CircuitEvaluationResult.DontCareOutputs
    };
    if (BestIndividual.Genes.Any(t => t.GateLines.Exists(x => x.Name == GateType.N1P1G2Fredkin)))
    {
        ;
    }

    BestGenes = new List<QuantumState>(BestIndividual.Genes);
}

private Dna TournamentSelection(int tournamentSize)
{
    // Create a tournament pool
    var tournamentPool = new List<Dna>();

    // Randomly select individuals for the tournament pool
    for (var i = 0; i < tournamentSize; i++)
    {
        var randomIndex = _random.Next(0, Population.Count);
        tournamentPool.Add(Population[randomIndex]);
    }

    // Find the fittest individual in the tournament pool
    var fittestIndividual = tournamentPool[0];
    for (var i = 1; i < tournamentPool.Count; i++)
    {
        if (tournamentPool[i].CircuitEvaluationResult.Fitness > fittestIndividual.CircuitEvaluationResult.Fitness)
        {
            fittestIndividual = tournamentPool[i];
        }
```

```
        }

        return fittestIndividual;
    }

    private Dna RouletteWheelSelection()
    {
        //Roulette wheel
        var randomValue = _random.NextDouble() * _fitnessSum;
        double cumulativeFitness = 0;

        foreach (var individual in Population)
        {
            cumulativeFitness += individual.CircuitEvaluationResult.Fitness;
            if (cumulativeFitness >= randomValue)
            {
                return individual;
            }
        }

        // Fallback in case of numerical precision issues
        return Population[_random.Next(0, Population.Count)];
    }

    private void EnsureUnique(ref Dna chromosome)
    {
        //Add chromosome only if item is unique
        if (Population.DistinctBy(t => t.CircuitEvaluationResult.Fitness).Count() > _minimumUniquenessCount)
        {
            return;
        }

        chromosome = new Dna(_dnaSize, _random, _getRandomGene, _fitnessFunction, _mutationFunc, true);
        // chromosome.CalculateFitness();
    }
}
```

Додаток в

Лістинг основних частин коду програми Phyton з використанням Qiskit бібліотеки

Python скрипт з використанням бібліотеки Qiskit для верифікації роботи шифратора.

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
from qiskit import transpile
from qiskit.visualization import plot_state_qsphere, plot_histogram
import qiskit.qasm3

def dec_to_bit(data, size):
    return format(data, '0' + str(size) + 'b')[::-1]

class KeyData:
    def __init__(self, keyId, key, size):
        self.keyId = keyId
        self.keys = key
        self.size = size

def get_key_input(prompt, upper_bound):
    while True:
        try:
            user_input = int(input(prompt))
            if 0 <= user_input < upper_bound:
                return user_input
            else:
                print("Please enter a valid key")
        except ValueError:
            print("Please enter a valid integer.")

def get_ascii_to_encode(phrase_to_encode, data_size, shouldLog):
    bit_ascii_to_encode = []
    for char in phrase_to_encode:
        ascii_code = ord(char)
        ascii_code_binary = dec_to_bit(ascii_code, str(data_size))
        bit_ascii_to_encode.append(ascii_code_binary)
        if (shouldLog):
            print(f'Letter to encode:|{char}|, ASCII:|{ascii_code}|, Binnary ASCII:|{ascii_code_binary}|')

    return bit_ascii_to_encode

def get_keys_registers(integer_key, keys_size, quantum_registers, default_key_register_name, shouldLog):
    key_registers = {}
    bit_key = dec_to_bit(integer_key, keys_size)
    j = 0
    for register in quantum_registers:
        if default_key_register_name.lower() not in register.name.lower():
            continue
        key_registers[register.name] = bit_key[j:j + register.size]
        j += register.size
        if (shouldLog):
            print(f'Key part integer:|{int(key_registers[register.name], 2)}|, Binnary:|{key_registers[register.name]}|')
```

```
    if (shouldLog):
        print(f"Key integer:|{integer_key}|, Binnary:|{bit_key}|")

    return key_registers

def Encode(bit_ascii_to_encode, quantum_registers, default_register_name, keys_registers, quantum_circuit, shouldLog):
    encoded_phrase = ""
    for bit_ascii in bit_ascii_to_encode:
        j = 0
        initialization_layer = QuantumCircuit(quantum_circuit.num_qubits)
        for register in quantum_registers:
            if default_register_name.lower() in register.name.lower():
                for i in range(register.size):
                    if keys_registers[register.name][i] == '1':
                        initialization_layer.initialize(1, j)
                    j += 1
            else:#init data
                for i in range(register.size):
                    if bit_ascii[i] == '1':
                        initialization_layer.initialize(1, j)
                    j += 1

        quantum_circuit_initialized = quantum_circuit.compose(initialization_layer, front=True)
        new_circuit = transpile(quantum_circuit_initialized, simulator)
        job = simulator.run(quantum_circuit_initialized)

        # Get the measurement outcome
        result = job.result()
        counts = result.get_counts()

        # Get the output state
        output_state = next(iter(counts.keys()))
        ascii_decimal = int(output_state, 2)
        encoded_character = chr(ascii_decimal)
        encoded_phrase += encoded_character

        if (shouldLog):
            print(f"Encoded decimal:|{ascii_decimal}|, Binnary: |{output_state}|, Char:|{encoded_character}|")

        #quantum_circuit_image = quantum_circuit_initialized.draw(output='mpl')
        #quantum_circuit_image.savefig(f"quantum_circuit_initialized_{bit_ascii[::-1]}.png")
    return encoded_phrase

def string_to_bool(s):
    return s.lower() in ['true', 't', 'yes', 'y', '1']


qasm_code = """
OPENQASM 3.0;
include "stdgates.inc";

qreg key1[6];
qreg key2[6];
qreg key3[6];
qreg key4[6];
qreg data[8];
creg mdata[8];
```

```
cx key1[0], data[0];
cx key1[1], data[1];
cx key1[2], key1[5];
ccx key1[3], key1[5], data[0];
ccx key1[4], key1[5], data[1];
cx data[1], data[0];
ccx key1[5], data[0], data[1];
cx key1[2], key1[5]; // restore control
// end 1st ERRG
cx data[1], data[0];
barrier key1[0], key1[1], key1[2], key1[3], key1[4], key1[5], key2[0], key2[1], key2[2], key2[3], key2[4], key2[5], key3[0], key3[1], key3[2], key3[3], key3[4], key3[5], key4[0], key4[1], key4[2], key4[3], key4[4], key4[5], data[0], data[1], data[2], data[3], data[4], data[5], data[6], data[7];
cx key2[0], data[2];
cx key2[1], data[3];
cx key2[2], key2[5];
ccx key2[3], key2[5], data[2];
ccx key2[4], key2[5], data[3];
cx data[3], data[2];
ccx key2[5], data[2], data[3];
cx key2[2], key2[5]; // restore control
// end 1st ERRG
cx data[3], data[2];
barrier key1[0], key1[1], key1[2], key1[3], key1[4], key1[5], key2[0], key2[1], key2[2], key2[3], key2[4], key2[5], key3[0], key3[1], key3[2], key3[3], key3[4], key3[5], key4[0], key4[1], key4[2], key4[3], key4[4], key4[5], data[0], data[1], data[2], data[3], data[4], data[5], data[6], data[7];
cx key3[0], data[4];
cx key3[1], data[5];
cx key3[2], key3[5];
ccx key3[3], key3[5], data[4];
ccx key3[4], key3[5], data[5];
cx data[5], data[4];
ccx key3[5], data[4], data[5];
cx key3[2], key3[5];
cx data[5], data[4];
barrier key1[0], key1[1], key1[2], key1[3], key1[4], key1[5], key2[0], key2[1], key2[2], key2[3], key2[4], key2[5], key3[0], key3[1], key3[2], key3[3], key3[4], key3[5], key4[0], key4[1], key4[2], key4[3], key4[4], key4[5], data[0], data[1], data[2], data[3], data[4], data[5], data[6], data[7];
cx key4[0], data[6];
cx key4[1], data[7];
cx key4[2], key4[5];
ccx key4[3], key4[5], data[6];
ccx key4[4], key4[5], data[7];
cx data[7], data[6];
ccx key4[5], data[6], data[7];
cx key4[2], key4[5];
cx data[7], data[6];
barrier key1[0], key1[1], key1[2], key1[3], key1[4], key1[5], key2[0], key2[1], key2[2], key2[3], key2[4], key2[5], key3[0], key3[1], key3[2], key3[3], key3[4], key3[5], key4[0], key4[1], key4[2], key4[3], key4[4], key4[5], data[0], data[1], data[2], data[3], data[4], data[5], data[6], data[7];
measure data -> mdata;
"""

quantum_circuit = qiskit.qasm3.loads(qasm_code)
simulator = AerSimulator(shots = 1, max_memory_mb = 0, method='matrix_product_state')

default_register_name = "key"
bit_ascii_to_encode = []
keys_registers = {}
keys_size = 0
```

```
data_size = 0

# Get the quantum registers from the circuit
quantum_registers = quantum_circuit.qregs
for register in quantum_registers:
    if default_register_name.lower() not in register.name.lower():
        continue
    keys_size += register.size
data_size = quantum_circuit.num_qubits - keys_size

phrase_to_encode = input("Enter word or phrase to encode:")
showLog = string_to_bool(input("Enter 1 if need to display logs or 0 if not needed:"))
bit_ascii_to_encode = get_ascii_to_encode(phrase_to_encode, data_size, showLog)


start_keys_range = int(input("Enter start key range:"))
end_keys_range = int(input("Enter end key range:"))

for key_cursor in range(start_keys_range, end_keys_range + 1):
    keys_registers = get_keys_registers(key_cursor, keys_size, quantum_registers, default_register_name, showLog)
    encoded_phrase = Encode(bit_ascii_to_encode, quantum_registers, default_register_name, keys_registers,
quantum_circuit, showLog)
    if (showLog):
        print(f"Encoded phrase is:{encoded_phrase}:")

    bit_ascii_to_decode = get_ascii_to_encode(encoded_phrase, data_size, showLog)
    decoded_phrase = Encode(bit_ascii_to_decode, quantum_registers, default_register_name, keys_registers,
quantum_circuit, showLog)
    if (showLog):
        print(f"Encoded phrase is:{decoded_phrase}:")

    if (encoded_phrase == decoded_phrase):
        print(f"Encoded phrase |{encoded_phrase}| is equal to decoded |{decoded_phrase}|, Key |{key_cursor}| not
working:{decoded_phrase}:")
        with open("Redundant_keys.txt", "a") as redundant_keys_file:
            redundant_keys_file.write(str(key_cursor) + "\n")
    else:
        with open("Working_keys.txt", "a") as working_keys_file:
            working_keys_file.write(str(key_cursor) + "\n")
```

Додаток Г

Лістинг основних частин коду опису апаратури на мові Verilog

Енкодер для 7-бітних ASCII символів

```
`timescale 1ns / 1ps
// ASCII Character Encoder
module ascii_encoder(
   input  wire [14:0] K_flat,  // Ключ: {K4, K3, K2, K1, K0} кожен по 3 біти (останній K4 використовує тільки 1 біт)
   input  wire [6:0]  X,       // Вхідний ASCII символ (7 біт)
   output wire [6:0]  Y        // Вихідний зашифрований символ (7 біт)
);
   // Розпаковуємо ключ на окремі блоки (тільки перші 3 біти з кожного)
   wire [2:0] K [2:0];  // 3 блоки по 3 біти

   assign K[0] = K_flat[2:0];    // Перші 3 біти ключа
   assign K[1] = K_flat[5:3];    // Наступні 3 біти
   assign K[2] = K_flat[8:6];    // Останні 3 біти (використаємо тільки 1)

   // Розбиваємо 7-бітний вхід на групи по 3 біти
   wire [2:0] X_block [2:0];
   wire [2:0] Y_block [2:0];

   assign X_block[0] = X[2:0];    // Біти 0-2
   assign X_block[1] = X[5:3];    // Біти 3-5
   assign X_block[2] = {2'b00, X[6]}; // Біт 6 (доповнений нулями)

   // Застосовуємо RRG2 блоки
   RRG2 block0 (.K(K[0]), .X(X_block[0]), .Y(Y_block[0]));
   RRG2 block1 (.K(K[1]), .X(X_block[1]), .Y(Y_block[1]));
   RRG2 block2 (.K(K[2]), .X(X_block[2]), .Y(Y_block[2]));
   // Збираємо вихід назад в 7 біт
   assign Y = {Y_block[2][0], Y_block[1], Y_block[0]};

endmodule
```

ПЗП шрифтів 8x16 ASCII - На основі стандартного шрифту VGA 8x16

```
// 8x16 ASCII Font ROM
// Based on VGA standard 8x16 font
module font_rom_8x16(
   input wire [7:0] char_code,
   input wire [3:0] row,
   output reg [7:0] font_data
);

always @(*) begin
   case (char_code)
      // Space
      8'h20: font_data = 8'h00;
```

```
// Digit 0
8'h30: begin
   case (row)
      4'h0: font_data = 8'b00000000;
      4'h1: font_data = 8'b00000000;
      4'h2: font_data = 8'b00111100;
      4'h3: font_data = 8'b01100110;
      4'h4: font_data = 8'b01101110;
      4'h5: font_data = 8'b01110110;
      4'h6: font_data = 8'b01100110;
      4'h7: font_data = 8'b01100110;
      4'h8: font_data = 8'b00111100;
      4'h9: font_data = 8'b00000000;
      4'hA: font_data = 8'b00000000;
      4'hB: font_data = 8'b00000000;
      4'hC: font_data = 8'b00000000;
      4'hD: font_data = 8'b00000000;
      4'hE: font_data = 8'b00000000;
      4'hF: font_data = 8'b00000000;
   endcase
end


//repeating for all ascii main symbols

// Letter X
8'h58: begin
   case (row)
      4'h0: font_data = 8'b00000000;
      4'h1: font_data = 8'b00000000;
      4'h2: font_data = 8'b01100110;
      4'h3: font_data = 8'b01100110;
      4'h4: font_data = 8'b00111100;
      4'h5: font_data = 8'b00011000;
      4'h6: font_data = 8'b00111100;
      4'h7: font_data = 8'b01100110;
      4'h8: font_data = 8'b01100110;
      4'h9: font_data = 8'b00000000;
      4'hA: font_data = 8'b00000000;
      4'hB: font_data = 8'b00000000;
      4'hC: font_data = 8'b00000000;
      4'hD: font_data = 8'b00000000;
      4'hE: font_data = 8'b00000000;
      4'hF: font_data = 8'b00000000;
   endcase
end

// Letter Z
8'h5A: begin
   case (row)
      4'h0: font_data = 8'b00000000;
      4'h1: font_data = 8'b00000000;
      4'h2: font_data = 8'b01111110;
      4'h3: font_data = 8'b00000110;
      4'h4: font_data = 8'b00001100;
```

```
            4'h5: font_data = 8'b00011000;
            4'h6: font_data = 8'b00110000;
            4'h7: font_data = 8'b01100000;
            4'h8: font_data = 8'b01111110;
            4'h9: font_data = 8'b00000000;
            4'hA: font_data = 8'b00000000;
            4'hB: font_data = 8'b00000000;
            4'hC: font_data = 8'b00000000;
            4'hD: font_data = 8'b00000000;
            4'hE: font_data = 8'b00000000;
            4'hF: font_data = 8'b00000000;
         endcase
      end

      // Colon :
      8'h3A: begin
         case (row)
            4'h0: font_data = 8'b00000000;
            4'h1: font_data = 8'b00000000;
            4'h2: font_data = 8'b00000000;
            4'h3: font_data = 8'b00011000;
            4'h4: font_data = 8'b00011000;
            4'h5: font_data = 8'b00000000;
            4'h6: font_data = 8'b00011000;
            4'h7: font_data = 8'b00011000;
            4'h8: font_data = 8'b00000000;
            4'h9: font_data = 8'b00000000;
            4'hA: font_data = 8'b00000000;
            4'hB: font_data = 8'b00000000;
            4'hC: font_data = 8'b00000000;
            4'hD: font_data = 8'b00000000;
            4'hE: font_data = 8'b00000000;
            4'hF: font_data = 8'b00000000;
         endcase
      end

      default: font_data = 8'h00;
   endcase
end

endmodule
```

// Спрощений VGA-контролер для 640x480 @ 60Hz
// Оптимізовано для EP4CE6E22C8

```
module vga_controller(
   input wire clk,          // 25MHz clock for 640x480 @ 60Hz
   input wire rst_n,        // Reset (active low)
   output wire hsync,       // Horizontal sync
   output wire vsync,       // Vertical sync
   output wire [9:0] h_count, // Horizontal counter
   output wire [9:0] v_count, // Vertical counter
   output wire video_on     // Video enable signal
```

```
);

// VGA timing parameters for 640x480 @ 60Hz
parameter H_DISPLAY = 640;
parameter H_FRONT_PORCH = 16;
parameter H_SYNC_PULSE = 96;
parameter H_BACK_PORCH = 48;
parameter H_TOTAL = H_DISPLAY + H_FRONT_PORCH + H_SYNC_PULSE + H_BACK_PORCH;

parameter V_DISPLAY = 480;
parameter V_FRONT_PORCH = 10;
parameter V_SYNC_PULSE = 2;
parameter V_BACK_PORCH = 33;
parameter V_TOTAL = V_DISPLAY + V_FRONT_PORCH + V_SYNC_PULSE + V_BACK_PORCH;

// Internal registers
reg [9:0] h_count_reg, v_count_reg;

// Horizontal counter
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
      h_count_reg <= 0;
   end else begin
      if (h_count_reg == H_TOTAL - 1) begin
         h_count_reg <= 0;
      end else begin
         h_count_reg <= h_count_reg + 1;
      end
   end
end

// Vertical counter
always @(posedge clk or negedge rst_n) begin
   if (!rst_n) begin
      v_count_reg <= 0;
   end else begin
      if (h_count_reg == H_TOTAL - 1) begin
         if (v_count_reg == V_TOTAL - 1) begin
            v_count_reg <= 0;
         end else begin
            v_count_reg <= v_count_reg + 1;
         end
      end
   end
end

// Output assignments
assign h_count = h_count_reg;
assign v_count = v_count_reg;

// Sync signals (combinational)
assign hsync = (h_count_reg >= H_DISPLAY + H_FRONT_PORCH &&
            h_count_reg < H_DISPLAY + H_FRONT_PORCH + H_SYNC_PULSE) ? 0 : 1;

assign vsync = (v_count_reg >= V_DISPLAY + V_FRONT_PORCH &&
```

```
            v_count_reg < V_DISPLAY + V_FRONT_PORCH + V_SYNC_PULSE) ? 0 : 1;

// Video enable signal
assign video_on = (h_count_reg < H_DISPLAY) && (v_count_reg < V_DISPLAY);

endmodule

// PS/2 Keyboard Interface
// Інтерфейс PS/2 клавіатури з підтримкою A4Tech KL-23MU
module ps2_keyboard(
    input wire clk,
    input wire rst_n,
    input wire ps2_clk,
    input wire ps2_data,
    output reg [7:0] ascii_code,
    output reg key_pressed
);

    // Простий детектор натискання клавіш
    reg [3:0] bit_count;
    reg [7:0] data_reg;
    reg [7:0] last_key;

    // Синхронізація PS/2 сигналів (3-стадійна)
    reg ps2_clk_sync1, ps2_clk_sync2, ps2_clk_sync3;
    reg ps2_data_sync;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            ps2_clk_sync1 <= 1;
            ps2_clk_sync2 <= 1;
            ps2_clk_sync3 <= 1;
            ps2_data_sync <= 1;
        end else begin
            ps2_clk_sync1 <= ps2_clk;
            ps2_clk_sync2 <= ps2_clk_sync1;
            ps2_clk_sync3 <= ps2_clk_sync2;
            ps2_data_sync <= ps2_data;
        end
    end

    wire ps2_clk_falling = ps2_clk_sync3 && !ps2_clk_sync2;

    // Проста таблиця для основних клавіш (великі літери)
    // Стандартний PS/2 Scan Code Set 2
    always @(*) begin
        case (data_reg)
            8'h1C: ascii_code = 8'h41; // 'A'
            8'h32: ascii_code = 8'h42; // 'B'
            8'h21: ascii_code = 8'h43; // 'C'
            8'h23: ascii_code = 8'h44; // 'D'
            8'h24: ascii_code = 8'h45; // 'E'
            8'h2B: ascii_code = 8'h46; // 'F'
            8'h34: ascii_code = 8'h47; // 'G'
```

```
        8'h33: ascii_code = 8'h48; // 'H'
        8'h43: ascii_code = 8'h49; // 'I'
        8'h3B: ascii_code = 8'h4A; // 'J'
        8'h42: ascii_code = 8'h4B; // 'K'
        8'h4B: ascii_code = 8'h4C; // 'L'
        8'h3A: ascii_code = 8'h4D; // 'M'
        8'h31: ascii_code = 8'h4E; // 'N'
        8'h44: ascii_code = 8'h4F; // 'O'
        8'h4D: ascii_code = 8'h50; // 'P'
        8'h15: ascii_code = 8'h51; // 'Q'
        8'h2D: ascii_code = 8'h52; // 'R'
        8'h1B: ascii_code = 8'h53; // 'S'
        8'h2C: ascii_code = 8'h54; // 'T'
        8'h3C: ascii_code = 8'h55; // 'U'
        8'h2A: ascii_code = 8'h56; // 'V'
        8'h1D: ascii_code = 8'h57; // 'W'
        8'h22: ascii_code = 8'h58; // 'X'
        8'h35: ascii_code = 8'h59; // 'Y'
        8'h1A: ascii_code = 8'h5A; // 'Z'

        // Цифри 0-9
        8'h45: ascii_code = 8'h30; // '0'
        8'h16: ascii_code = 8'h31; // '1'
        8'h1E: ascii_code = 8'h32; // '2'
        8'h26: ascii_code = 8'h33; // '3'
        8'h25: ascii_code = 8'h34; // '4'
        8'h2E: ascii_code = 8'h35; // '5'
        8'h36: ascii_code = 8'h36; // '6'
        8'h3D: ascii_code = 8'h37; // '7'
        8'h3E: ascii_code = 8'h38; // '8'
        8'h46: ascii_code = 8'h39; // '9'

        8'h29: ascii_code = 8'h20; // space
        8'h5A: ascii_code = 8'h0D; // enter
        8'h66: ascii_code = 8'h08; // backspace
        default: ascii_code = 8'h00;
    endcase
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        bit_count <= 0;
        data_reg <= 0;
        key_pressed <= 0;
        last_key <= 0;
    end else begin
        key_pressed <= 0;

        if (ps2_clk_falling) begin
            if (bit_count == 0 && !ps2_data_sync) begin
                // Start bit
                bit_count <= 1;
            end else if (bit_count > 0 && bit_count < 9) begin
                // Data bits - БЕЗ реверсування (LSB first - правильний порядок)
                data_reg[bit_count-1] <= ps2_data_sync;
```

```
                bit_count <= bit_count + 1;
            end else if (bit_count == 9) begin
                // Stop bit
                // Фільтруємо E0, F0 та дублікати
                if (data_reg != 8'hE0 && data_reg != 8'hF0 &&
                    data_reg != last_key && data_reg != 8'h00) begin
                    key_pressed <= 1;
                    last_key <= data_reg;
                end
                bit_count <= 0;
            end
        end
    end
end

endmodule

// Keyboard Text Displayer with Quantum Encoding
// Система введення тексту з клавіатури на VGA монітор + квантове шифрування
module keyboard_text(
    input wire clk_50mhz,     // 50MHz input clock
    input wire rst_n,         // Reset button
    input wire encode_switch,  // Switch for encode/decode mode (PIN_25)
    input wire key1,          // Button for ENCR mode (PIN_88)
    input wire key2,          // Button for DECR mode (PIN_89)
    input wire key3,          // Button for KEYS mode (PIN_90)
    input wire ps2_clk,       // PS/2 clock (PIN_119)
    input wire ps2_data,      // PS/2 data (PIN_120)
    output wire hsync,        // VGA horizontal sync
    output wire vsync,        // VGA vertical sync
    output reg red,           // VGA red
    output reg green,         // VGA green
    output reg blue,          // VGA blue
    output reg [7:0] seg,     // 7-segment display segments
    output reg [3:0] dig,     // 7-segment display digits
    output reg [3:0] led      // Status LEDs
);

    // Internal signals
    wire clk_25mhz;
    wire [9:0] h_count, v_count;
    wire video_on;

    // Keyboard signals
    wire [7:0] ascii_code;
    wire key_pressed;
    wire [7:0] key_code;

    // Quantum encoder signals
    parameter [8:0] QUANTUM_KEY = 9'b101010101; // Фіксований ключ
    wire [7:0] encoded_ascii;
    reg encode_mode; // 0 = decode, 1 = encode

    // 7-segment display signals
    wire [7:0] seg_sig;
```

```
wire [3:0] dig_sig;
wire [3:0] led_sig;

// Button toggle for encode/decode
reg encode_switch_prev;
wire encode_switch_pressed = encode_switch_prev && !encode_switch;

always @(posedge clk_50mhz or negedge rst_n) begin
    if (!rst_n) begin
        encode_mode <= 0;
        encode_switch_prev <= 1;
    end else begin
        encode_switch_prev <= encode_switch;
        if (encode_switch_pressed) begin
            encode_mode <= ~encode_mode; // Toggle
        end
    end
end

// Clock divider
reg clk_div;
always @(posedge clk_50mhz or negedge rst_n) begin
    if (!rst_n) begin
        clk_div <= 0;
    end else begin
        clk_div <= !clk_div;
    end
end
assign clk_25mhz = clk_div;

// VGA Controller
vga_controller vga_ctrl (
    .clk(clk_25mhz),
    .rst_n(rst_n),
    .hsync(hsync),
    .vsync(vsync),
    .h_count(h_count),
    .v_count(v_count),
    .video_on(video_on)
);

// PS/2 Keyboard
ps2_keyboard keyboard (
    .clk(clk_25mhz),
    .rst_n(rst_n),
    .ps2_clk(ps2_clk),
    .ps2_data(ps2_data),
    .ascii_code(ascii_code),
    .key_pressed(key_pressed)
);

// Quantum Encoder/Decoder
ascii_codec encoder (
    .encode_mode(encode_mode),
    .K_flat(QUANTUM_KEY),
```

```
        .char_in(ascii_code[6:0]),   // ASCII тільки 7 біт (0-127)
        .char_out(encoded_ascii[6:0])
    );
    assign encoded_ascii[7] = 1'b0; // Старший біт завжди 0 для ASCII

    // Internal color signals
    wire red_sig, green_sig, blue_sig;

    // Simple text display (відображає зашифрований текст)
    simple_keyboard_display text_disp (
        .clk(clk_25mhz),
        .rst_n(rst_n),
        .h_count(h_count),
        .v_count(v_count),
        .video_on(video_on),
        .ascii_code(encoded_ascii),   // Зашифрований або дешифрований символ
        .key_pressed(key_pressed),
        .red(red_sig),
        .green(green_sig),
        .blue(blue_sig)
    );

    // 7-segment display
    seven_seg_display seg_display (
        .clk_50mhz(clk_50mhz),
        .key1(key1),
        .key2(key2),
        .key3(key3),
        .ascii_code(ascii_code),
        .key_pressed(key_pressed),
        .seg(seg_sig),
        .dig(dig_sig),
        .led(led_sig)
    );

    // Assign internal signals to outputs
    always @(posedge clk_25mhz or negedge rst_n) begin
        if (!rst_n) begin
            red <= 0;
            green <= 0;
            blue <= 0;
            seg <= 8'b11111111;
            dig <= 4'b1111;
            led <= 4'b0000;
        end else begin
            red <= red_sig;
            green <= green_sig;
            blue <= blue_sig;
            seg <= seg_sig;
            dig <= dig_sig;
            led <= led_sig;
        end
    end

endmodule
```

```
`timescale 1ns / 1ps

// ASCII Character Encoder/Decoder
// Кодер/Декодер для 7-бітних ASCII символів з перемикачем
module ascii_codec(
    input  wire       encode_mode, // 1 = encode, 0 = decode
    input  wire [8:0]  K_flat,      // Ключ: 9 біт (3 блоки по 3 біти)
    input  wire [6:0]  char_in,     // Вхідний ASCII символ (7 біт)
    output wire [6:0]  char_out     // Вихідний символ (7 біт)
);
    // Розпаковуємо ключ на окремі блоки
    wire [2:0] K [2:0];

    assign K[0] = K_flat[2:0];
    assign K[1] = K_flat[5:3];
    assign K[2] = K_flat[8:6];

    // Розбиваємо 7-бітний вхід на групи по 3 біти + 1 біт
    wire [2:0] X_block [2:0];
    wire [2:0] Y_enc [2:0];  // Результат encode
    wire [2:0] Y_dec [2:0];  // Результат decode

    assign X_block[0] = char_in[2:0];      // Біти 0-2
    assign X_block[1] = char_in[5:3];      // Біти 3-5
    assign X_block[2] = {2'b00, char_in[6]}; // Біт 6

    // Encode path (forward)
    RRG2 enc_block0 (.K(K[0]), .X(X_block[0]), .Y(Y_enc[0]));
    RRG2 enc_block1 (.K(K[1]), .X(X_block[1]), .Y(Y_enc[1]));
    RRG2 enc_block2 (.K(K[2]), .X(X_block[2]), .Y(Y_enc[2]));

    // Decode path (inverse)
    RRG2_Inverse dec_block0 (.K(K[0]), .X(X_block[0]), .Y(Y_dec[0]));
    RRG2_Inverse dec_block1 (.K(K[1]), .X(X_block[1]), .Y(Y_dec[1]));
    RRG2_Inverse dec_block2 (.K(K[2]), .X(X_block[2]), .Y(Y_dec[2]));

    // Вибір encode/decode
    wire [6:0] encoded_out = {Y_enc[2][0], Y_enc[1], Y_enc[0]};
    wire [6:0] decoded_out = {Y_dec[2][0], Y_dec[1], Y_dec[0]};

    assign char_out = encode_mode ? encoded_out : decoded_out;

endmodule
```

Додаток Ґ

Акти впровадження результатів дисертаційної роботи

«Затверджую»
Директор Навчально-наукового
інституту фізико-технічних та
комп'ютерних наук Чернівецького
національного університету
імені Юрія Федьковича
кандидат технічних наук, доцент

_____ П. М. Шпатар

Акт
впровадження результатів дисертаційної роботи
Кирилюка Тараса Петровича
«Синтез відмовостійких зворотних логічних пристроїв методами штучного інтелекту» до захисту на здобуття наукового ступеня доктора філософії з галузі знань 12 – «Інформаційні технології», спеціальністю 121 – «Інженерія програмного забезпечення» в навчальному процесі

Комісія Чернівецького Національного університету імені Юрія Федьковича в складі:

заступника директора ННІФТКН з наукової роботи, доктора ф.-м.н., професора Дуболазова О. В.;

завідувача кафедри комп'ютерних систем та мереж, кандидата ф.-м. н., доцента Воробця Г. І.;

завідувача кафедри програмного забезпечення комп'ютерних систем, доктора філософії, доцента Газдюк К. П.,

склали цей акт про те, що результати дисертаційної роботи здобувача наукового ступеня доктора філософії за спеціальністю «Інженерія програмного забезпечення» Кирилюк Т. П. впроваджені в навчальний процес кафедр ПЗКС та КСМ, зокрема в дисциплінах: «Квантовий комп'ютинг» та «Машинне навчання на .NET».

При викладанні цих дисциплін використовуються наступні матеріали досліджень, які отримані в дисертаційній роботі:

- Методи та алгоритми синтезу зворотних логічних схем у квантових базисах NCT та вентилів Фредкіна, що дозволяють студентам

опанувати принципи побудови енергоефективних та оборотних обчислювальних структур;

- Розроблена модель реконфігуровних вентилів Фредкіна та методи забезпечення відмовостійкості на основі збереження парності, що використовуються при вивченні архітектурних особливостей сучасних квантових процесорів;
- Програмний модуль генерації QASM-коду для інтеграції з платформою IBM Qiskit, що застосовується під час виконання практичних та лабораторних робіт для верифікації синтезованих схем на реальних квантових симуляторах;
- Вдосконалений генетичний метод з оператором багатокомпонентної мутації як приклад застосування еволюційних стратегій штучного інтелекту для розв'язання задач комбінаторної оптимізації високої складності;
- Об'єктно-орієнтована модель представлення хромосом та реалізація алгоритмічного ядра мовою C#, що використовуються як приклад при вивченні розробки інтелектуальних систем у середовищі .NET;
- Підходи до багатокритеріальної оптимізації фітнес-функцій для пошуку оптимальних рішень у просторі станів, що демонструють студентам можливості адаптації стандартних алгоритмів ML під специфічні інженерні задачі.

Використання зазначених результатів дозволило підвищити якість навчального процесу із згаданих дисциплін.

Заступник директора ННІФТКН
з наукової роботи, д. ф.-м.н., професор _____ О. В. Дуболазов

Завідувач кафедри КСМ
кандидат ф.-м. н., доцент _____ Г. І. Воробець

Завідувач кафедри ПЗКС
доктор філософії, доцент _____ К. П. Газдюк

additiv

Additiv AG
Talstrasse 65
8001 Zurich
Switzerland

To whom it may concern:

18.03.2026

Act of implementation of the results of the dissertation research by Taras Kyryliuk on the topic: "Synthesis of fault-tolerant reversible logic devices using artificial intelligence methods" submitted for the degree of Doctor of Philosophy in the field of 12 – "Information Technologies" specialty 121 – "Software Engineering"

This act confirms that we have reviewed and evaluated the findings presented in the dissertation research of Taras Kyryliuk, a PhD candidate at the Yuriy Fedkovych Chernivtsi National University. Based on the technical assessment, we state the following:

- Within the company's R&D framework, the proposed functional module for investment portfolio optimization has been analyzed. Its potential to enhance computational efficiency for risk modeling and asset management within the WealthTech sector has been recognized.
- The research findings are considered valuable for the company's future technological roadmap, particularly in ensuring compatibility with modern quantum computing platforms and advancing initiatives in the field of Quantum Finance.

We acknowledge the practical significance of these results and may consider them for potential integration into future software development cycles.

Adrian Weiss
Head Corporate Services
Member of the Group Executive Board, additiv

Additiv AG
Talstrasse 65
8001 Zurich www.additiv.com

Акт
впровадження результатів дисертаційної роботи
Кирилюка Тараса Петровича
на тему: «Синтез відмовостійких зворотних логічних пристроїв методами штучного інтелекту», представленої до захисту на здобуття наукового ступеня доктора філософії в галузі знань 12 – «Інформаційні технології» за спеціальністю 121 – «Інженерія програмного забезпечення»

Цей акт підтверджує, що ТОВ «ШАРПМАЙНДЗ ЮЕЙ» використало та впровадило результати, представлені в дисертаційному дослідженні здобувача ступеня доктора філософії Чернівецького національного університету імені Юрія Федьковича Кирилюка Тараса Петровича, а саме:

- Впроваджено методологію інтелектуального синтезу логічних структур для оцінки перспектив розробки високонадійного програмного забезпечення у проєктах з високими вимогами до цілісності даних;
- Використано розроблений генетичний алгоритм як інструмент прототипування для пошуку оптимальних архітектурних рішень у новітніх обчислювальних парадигмах а саме: квантових та зворотних обчисленнях;
- Застосовано результати дослідження для розширення технологічного стеку R&D-підрозділу в напрямку енергоефективних обчислень та автоматизованого тестування логічних модулів на стійкість до збоїв.

Результати роботи дозволяють компанії підвищити рівень наукоємності своїх розробок та пропонувати клієнтам інноваційні рішення у сфері проєктування відмовостійких систем на базі квантових обчислень.

12.10.2025

Головний
операційний директор
Гордієнко А. Ю.

Акт
впровадження результатів дисертаційної роботи
Кирилюка Тараса Петровича
на тему: «Синтез відмовостійких зворотних логічних пристроїв методами штучного інтелекту», представленої до захисту на здобуття наукового ступеня доктора філософії з галузі знань 12 – «Інформаційні технології» за спеціальністю 121 – «Інженерія програмного забезпечення»

Цей акт підтверджує, що «Yukon Software Ltd» використала та впровадила результати, представлені в дисертаційному дослідженні здобувача ступеня доктора філософії Чернівецького національного університету імені Юрія Федьковича Кирилюка Тараса Петровича, а саме:

- У межах науково-дослідної діяльності компанії впроваджено програмний комплекс автоматизованого синтезу логічних структур, побудований на базі вдосконаленого генетичного методу;
- Застосовано розроблені методи побудови енергоефективних та відмовостійких архітектур.

Використання об'єктно-орієнтованої моделі та операторів багатокомпонентної мутації дозволило оптимізувати процеси проєктування складних обчислювальних графів, а також мінімізувати вплив технічних помилок при виконанні високонавантажених обчислень

10.10.2025 Директор «Yukon Software Ltd» Шкурей М. Р.

Додаток Д

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці у виданнях, включених до переліку наукових фахових видань України та проіндексованих у наукометричних базах даних Web of Science Core Collection та/або Scopus:

1. Kyryliuk T., Palahuta M., Deibuk V. Using artificial intelligence methods for the optimal synthesis of reversible networks. *Radioelectronic and Computer Systems*. 2024. Vol. 2024, no. 4. P. 112–122. URL:https://doi.org/10.32620/reks.2024.4.10 (Scopus, Q3)

   Внесок авторів: **Кирилюк Т.** – розроблення алгоритму, проведення експериментів, підготовка основного рукопису; Палагута М. – розроблення алгоритму, проведення експериментів, підготовка основного рукопису; Дейбук В. – рецензування та редагування рукопису.

Наукові праці у виданнях, проіндексованих у наукометричній базі даних Scopus:

2. Deibuk V., Dovhaniuk O., Kyryliuk T. The Extended Fredkin Gates with Reconfiguration in NCT Basis. *Lecture Notes on Data Engineering and Communications Technologies*. 2023. Vol. 181 : Advances in Computer Science for Engineering and Education VI : ICCSEEA 2023 (Warsaw, Poland, 17–19 March 2023). Warsaw, P. 95–105. URL: https://doi.org/10.1007/978-3-031-36118-0_9. (Scopus)

   Внесок авторів Дейбук В. – наукове керівництво, концептуальна перевірка, рецензування та редагування рукопису; Довганюк О. – написання рукопису, проведення експериментів та аналіз результатів; **Кирилюк Т.** – проведення експериментів, перевірка результатів та доопрацювання рукопису
3. Dovhaniuk O., Kyryliuk T., Deibuk V. Reversible Fault-Tolerant Encryption Using Extended Fredkin Gate with Reconfiguration. *Advances in*

Transdisciplinary Engineering. 2025. Vol. 65 : Artificial Intelligence, Medical Engineering and Education : AIMEE 2024 (Huangshi, China, 26–27 Oct. 2024). Huangshi, P. 69–76. URL: https://doi.org/10.3233/atde250113. (Scopus).

Внесок авторів: Довганюк О. – проведення експериментів; **Кирилюк Т.** – основне написання рукопису, верифікація експериментів, підготовка презентації; Дейбук В. – рецензування рукопису та наукове керівництво.

Наукові праці, які засвідчують апробацію матеріалів дисертації;

4. Kyryliuk T., Deibuk V., Kyryliuk S. Synthesis of reversible circuits by genetic algorithm with multicomponent mutation. *Інформаційні технології та комп'ютерне моделювання* : матеріали міжнар. наук.-практ. конф. (м. Івано-Франківськ, 6–8 лип. 2023 р.). Івано-Франківськ, 2023. С. 150–151.

   Внесок авторів: **Кирилюк Т.** – проведення експериментів, розроблення алгоритму, підготовка основного рукопису; Дейбук В. – наукове керівництво, доопрацювання рукопису; Кирилюк С. – проведення експериментів, фіксація експериментальних результатів

Додаток Е

Відомості про апробацію результатів дисертації

1. The 6th International Conference on Computer Science, Engineering and Education Applications (ICCSEEA2023) – Warsaw, Poland, 17–19 March, 2023, форма участі – дистанційна, за результатами конференції опублікована стаття у серії «Lecture Notes on Data Engineering and Communications Technologies», що індексується у наукометричній базі Scopus.
2. Міжнародна науково-практична конференція «Інформаційні технології та комп'ютерне моделювання» (ІТСМ 2023) – м. Івано-Франківськ, 6-8 липня 2023, форма участі – дистанційна з публікацією тез
3. The 8th International Conference of Artificial Intelligence, Medical Engineering, Education (AIMEE2024) – Huangshi, China, 26 – 27 October, 2024, форма участі – дистанційна, за результатами конференції опублікована стаття у серії «Advances in Transdisciplinary Engineering», що індексується у наукометричній базі Scopus.