Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка та тестування програмного забезпечення застосунку перевірки ризиків в стохастичних системах

Виконав: студент IV курсу, групи СП-42 спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Момотюк В. С.
(підпис) (прізвище та ініціали)

Керівник Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Нормоконтроль Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Момотюку Владиславу Святославовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та тестування програмного забезпечення застосунку перевірки ризиків в стохастичних системах

Керівник роботи к.т.н., доц. Стоянов Ю.М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 2026 року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області. Розділ 2. Проєктування
Розділ 3. Безпека життєдіяльності та основи охорони праці. Висновок.
Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми, знімки екрану з проробленою роботою, ілюстративні зображення, інформативні зображення для доповнення тексту.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи	6.04 – 12.04	
2	Підбір джерел по темі кваліфікаційної роботи	13.04 – 26.04	
3	Опрацювання джерел інформації по темі кваліфікаційної роботи	13.04 – 26.04	
4	Розробка архітектури програмного рішення	27.04 – 03.05	
5	Ознайомлення з новими технологіями	04.05 – 17.05	
6	Розробка на основі опрацьованої інформації	18.05 – 24.05	
7	Виконання звіту по проробленій роботі	18.05 – 24.05	
8	Виконання завдання до підрозділу «Безпека життєдіяльності»	18.05 – 24.05	
9	Виконання завдання до підрозділу «Основи охорони праці»	18.05 – 24.05	
10	Оформлення кваліфікаційної роботи	25.05 – 7.06	
11	Нормоконтроль	8.06 – 14.06	
12	Перевірка на плагіат	8.06 – 14.06	
13	Попередній захист кваліфікаційної роботи	15.06 – 21.06	
14	Захист кваліфікаційної роботи		

Студент _____ Момотюк В. С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка та тестування програмного забезпечення застосунку перевірки ризиків в стохастичних системах // Кваліфікаційна робота освітнього рівня «Бакалавр» // Момотюк Владислав Святославович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42 // Тернопіль, 2026 // Ст. – _103_, рис. – 18, табл. – 6, додат. – 1, бібліогр. – 83.

Ключові слова: Програмна платформа, моніторинг ризиків, стохастичні системи, штучні нейронні мережі, стохастичні обчислення, глибоке навчання.

Головною метою цієї кваліфікаційної роботи є розробка програмної платформи для моніторингу ризиків у стохастичних системах із застосуванням нейромережевих модулів та методів стохастичних обчислень.

Перший розділ присвячено аналізу предметної області, зокрема проблемам моніторингу та оцінювання ризиків у системах із випадковими факторами. Досліджено особливості застосування штучних нейронних мереж для розв'язання цих задач. Виконано огляд наявних підходів, включно з апаратними та програмними реалізаціями на базі стохастичних обчислень, а також обґрунтовано вибір архітектурних рішень для створення платформи.

У другому розділі розглянуто архітектуру розробленого програмного рішення. Описано математичну модель стохастичних процесів, структуру нейромережевих модулів, а також реалізацію логіки аналізу даних і формування звітів щодо виявлених ризиків. Крім того, наведено результати тестування платформи на модельних наборах даних, виконано оцінювання точності та продуктивності нейромережевих алгоритмів. Окреслено перспективи масштабування системи та її інтеграції з іншими аналітичними сервісами.

У третьому розділі подано аналіз умов праці розробника програмного забезпечення в контексті охорони праці та безпеки життєдіяльності. Розглянуто вимоги щодо безпечної експлуатації обчислювальної техніки та організації ергономічного робочого місця інженера-програміста відповідно до чинних

нормативних документів. Також досліджено вплив динамічних природних явищ і визначено принципи організації робіт в аварійних умовах.

ABSTRACT

Development and testing of software for risk checking application in stochastic systems // Qualification Thesis for the Bachelor's Degree // Vladyslav Sviatoslavovych Momotiuk // Ternopil Ivan Puluj National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, Group SP-42 // Ternopil, 2026 // Pages – 103, Figures – 18, Tables – 6, Appendices – 1, References – 83.

Keywords: Software platform, risk monitoring, stochastic systems, artificial neural networks, stochastic computing, deep learning.

The main goal of this thesis is to develop an effective software platform for risk monitoring in stochastic systems using neural network modules and stochastic computing methods.

The first chapter is dedicated to the analysis of the subject area, the problems of monitoring and assessing risks in systems with unpredictable (random) factors, the specific features of applying artificial neural networks for these tasks, the review of existing approaches (in particular, hardware and software implementations based on stochastic computing), and the rationale for choosing architectural solutions for the platform creation.

In the second chapter, the architecture of the developed software solution is considered; the mathematical model of stochastic processes, the structure of neural network modules, the implementation of data analysis logic, and the generation of reports on identified risks are described. It also presents the results of platform testing on model datasets, the evaluation of the accuracy and performance of neural network algorithms, as well as the prospects for scaling the system and its integration with other analytical services.

The third chapter the working conditions of a software developer from an occupational health and safety perspective, discussing requirements for the safe operation of computing equipment and the organization of an ergonomic workstation for a software engineer in accordance with occupational health and safety regulations. Dynamic

phenomena on the earth's surface are considered, as well as the organization of work in emergency situations.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ANN – Artificial Neural Network (штучна нейронна мережа)

DNN – Deep Neural Network (глибока нейронна мережа)

MLP – Multi-layer Perceptron (багатошаровий перцептрон)

SC – Stochastic Computing (стохастичні обчислення)

PE – Probability Estimator (оцінювач ймовірності)

FPGA – Field Programmable Gate Array (програмована користувачем вентильна матриця)

API – Application Programming Interface (програмний інтерфейс прикладного програмування)

UI – User Interface (інтерфейс користувача)

UX – User Experience (досвід користувача)

DB – Database (база даних)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

CI/CD – Continuous Integration / Continuous Delivery (безперервна інтеграція / безперервне постачання)

IDE – Integrated Development Environment (інтегроване середовище розробки)

CSV – Comma-Separated Values (значення, розділені комами)

ВСТУП

На сучасному етапі штучні нейронні мережі (ШНМ) та алгоритми глибокого навчання є основою для розв'язання складних задач класифікації, розпізнавання образів та аналітики в системах штучного інтелекту. Проте розмір сучасних глибоких нейронних мереж часто перевищує тисячі вузлів. Це робить виключно програмні реалізації на базі стандартних процесорів недостатньо продуктивними для застосунків, які вимагають тривалого навчання та миттєвого використання моделей. Серйозним викликом для систем реального часу є апаратна реалізація багатошарових нейронних мереж, зокрема багатошарових перцептронів. Традиційні обчислювальні архітектури потребують використання десятків тисяч складних апаратних множників. Це призводить до значного енергоспоживання та надмірних витрат ресурсів процесора, що суттєво обмежує можливість використання таких архітектур у портативних пристроях.

Серед розмаїття методів оптимізації особливу увагу дослідників привертають обчислювальні архітектури на основі стохастичних обчислень. Зазначений підхід дає змогу замінити складні арифметичні операції простими логічними елементами (наприклад, використання XNOR-елементів замість багаторозрядних множників). Це дозволяє суттєво мінімізувати апаратні витрати та значно підвищити енергоефективність системи.

Разом з тим, функціонуванню систем на основі стохастичних обчислень притаманні суттєві технічні обмеження. Критичним недоліком є низька швидкість збіжності: для отримання результату із заданим рівнем точності системі та традиційним блокам оцінювання ймовірності необхідна значна кількість тактових циклів, що може сягати кількох мільйонів. Крім того, виникають труднощі з масштабуванням у бік зменшення, а також спостерігається висока ймовірність виникнення флуктуаційних похибок під час виконання арифметичних операцій.

Оптимальною платформою для розгортання таких нейронних мереж є програмовані логічні інтегральні схеми (ПЛІС) завдяки їхній реконфігурованості та можливості виконання високошвидкісних паралельних обчислень. Проте

обмежена кількість виводів введення-виведення на ПЛІС створює обмежувальний фактор («вузьке місце») під час передачі великих масивів даних. Для мінімізації обчислювальних ризиків, зокрема часових затримок та втрати точності, необхідно впроваджувати складні конвеєрні архітектури та механізми послідовного введення даних. У зв'язку з цим, через високу вартість розгортання стохастичних систем на фізичному апаратному забезпеченні, особливої актуальності набуває розробка програмних інструментів для попереднього тестування та верифікації, орієнтованої на дані.

Актуальність теми зумовлена необхідністю створення ефективної програмної платформи для моделювання функціонування нейромережевих модулів на основі стохастичних обчислень. Розробка такого інструментарію дозволить здійснювати моніторинг супутніх обчислювальних ризиків — зокрема флуктуаційних похибок та затримок збіжності — і забезпечить можливість оптимізації архітектурних рішень ще до етапу їхньої безпосередньої апаратної реалізації у вбудованих системах.

Метою роботи є розробка програмної платформи для моніторингу ризиків у стохастичних системах із використанням нейромережевих модулів. Створене рішення забезпечує моделювання стохастичних обчислювальних процесів, оцінювання їхньої точності та верифікацію надійності алгоритмів перед їхньою імплементацією в системах реального часу.

Об’єктом дослідження є процеси функціонування та оцінювання надійності стохастичних динамічних систем на базі нейромережевих модулів в умовах жорстких обчислювальних обмежень. Предметом дослідження є методи, алгоритми та програмні засоби моніторингу ризиків (зокрема флуктуаційних похибок та часу збіжності) у штучних нейронних мережах, побудованих на принципах стохастичних обчислень. Для досягнення поставленої мети було сформульовано та розв’язано такі основні завдання:

- Провести аналіз існуючих методів реалізації глибоких нейронних мереж (ГНМ) та виявити ключові ризики і проблеми використання стохастичних обчислень (SC) в системах реального часу.

- Розробити математичні та алгоритмічні моделі нейромережевих модулів, які функціонують на принципах стохастичних бітових потоків, з урахуванням новітніх методів прискорення збіжності.
- Спроєктувати архітектуру програмної платформи, здатну симулювати послідовний ввід даних та конвеєрну обробку, характерну для апаратних обмежень ПЛІС.
- Здійснити практичну розробку програмного рішення для дата-орієнтованої верифікації та моніторингу обчислювальних ризиків.
- Провести тестування розробленої платформи на модельних наборах даних (зокрема для задач обробки зображень) та порівняльний аналіз із сучасними аналогами.

Практичне значення отриманих результатів. В роботі розроблено програмну платформу, яка дозволяє комплексно моніторити ризики стохастичних нейромереж. Її практичне значення підтверджується тим, що платформа дозволяє верифікувати та тестувати архітектурні рішення, які на етапі апаратної реалізації здатні забезпечити скорочення використання ресурсів більш ніж на 82% та мінімальне енергоспоживання. Крім того, завдяки точному моніторингу та моделюванню синхронізації, розроблений інструментарій допомагає знизити середній рівень помилки змодельованої нейронної мережі на 2%, що робить систему ідеальною для підготовки надійних Edge-AI рішень для обробки зображень.

1. АНАЛІЗ СТАНУ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ РОБОТИ

У розділі здійснено детальний огляд предметної області, зосереджений на проблемах апаратної реалізації глибоких штучних нейронних мереж та перевагах переходу до імовірнісного представлення даних. Крім того, проаналізовано наявні програмні інструменти для стохастичних систем і обґрунтовано вибір архітектурних рішень для створення нової платформи.

1.1 Аналіз предметної області та теоретичні основи роботи

Штучні нейронні мережі були вперше введені в науковий обіг ще у 1950-х роках як концептуальна спроба забезпечити обчислювальну модель, здатну імітувати складні внутрішні процеси людського мозку (згідно з [1]). З розвитком обчислювальної техніки ці моделі трансформувалися в потужні інструменти аналізу даних. Мережева архітектура ШНМ зазвичай поділяється на дві великі категорії: мережі прямого поширення [2] та рекурентні мережі [3].

Окрему увагу в сучасних дослідженнях приділено мережам прямого поширення, зокрема багатошаровим перцептронам (MLP). Типова архітектура MLP містить щонайменше три типи шарів: вхідний, приховані та вихідний. Ці шари складаються з повнозв’язних обробних елементів — штучних нейронів. За винятком нейронів вхідного шару, кожен вузол використовує нелінійну функцію активації. Це дає змогу системі розв'язувати задачі, що не піддаються лінійному розділенню. Саме наявність прихованих шарів у поєднанні з нелінійними функціями активації суттєво розширює можливості MLP щодо моделювання складних нелінійних залежностей та процесів [4]. Протягом кількох десятиліть архітектура MLP залишається одним із найбільш затребуваних інструментів для моделювання комплексних систем, класифікації даних та апроксимації функцій. З огляду на це, для розробки програмної платформи моніторингу ризиків як базову

архітектуру було обрано саме MLP, хоча запропоновані принципи можуть бути адаптовані і для інших типів глибоких нейронних мереж (ГНМ).

Попри доведену ефективність, більшість попередніх досліджень була зосереджена виключно на програмній реалізації MLP. Такий підхід виявляється недостатньо ефективним для систем, що функціонують у режимі реального часу. Оскільки в сучасних задачах машинного навчання розмір ГНМ часто налічує тисячі вузлів, використання виключно програмних рішень є недоцільним через тривалий час навчання та виконання моделей. Крім того, паралельні програмні реалізації на багатоядерних процесорних архітектурах часто демонструють обмежену продуктивність через архітектурні особливості послідовного доступу до пам’яті [5].

Через значну кількість нейронів, залучених до розв’язання задач класифікації, апаратна імплементація ГНМ є значно складнішою порівняно з традиційними алгоритмами [6-8]. Виділяють два основні підходи до апаратної реалізації ГНМ. Перший підхід базується на використанні класичного проектування цифрових схем (зокрема готових IP-ядер) для виконання арифметичних операцій (додавання, віднімання, множення, ділення), а також для обчислення функцій активації. Попри можливість оптимізації продуктивності, витрати апаратних ресурсів за такого підходу залишаються надмірними. Для ілюстрації масштабу проблеми розглянемо приклад: у разі реалізації нейромережі на 1000 нейронів, де кожен нейрон потребує 10 операцій множення вагових коефіцієнтів на вхідні сигнали, пряма апаратна імплементація такої ГНМ вимагатиме 10 000 множників. Спроба мінімізації кількості обчислювальних блоків шляхом використання одного множника на кожен нейрон (через часове мультиплексування) призведе до десятикратного зростання часової складності обробки даних.

Другий, значно перспективніший підхід, спрямований на розв’язання проблеми надмірного використання апаратних ресурсів, ґрунтується на концепції стохастичних обчислень. Основи цього підходу були закладені Гейнсом ще у 1960 році [9] як напрям наближених обчислень, що базується на імовірнісному

представленні даних [10]. Останнім часом методи на основі стохастичних обчислень привертають значну увагу дослідників та інженерів завдяки таким перевагам як низька апаратна складність: можливість заміни складних арифметичних блоків простими логічними елементами, а також мінімальне енергоспоживання: суттєве зниження витрат енергії, що критично для вбудованих систем. Стійкість до відмов: властива підходу здатність зберігати працездатність при виникненні поодиноких збоїв у бітових потоках.

У стохастичних схемах дійсні значення масштабуються до діапазону [0, 1] для уніполярного подання або [-1, 1] для біполярного подання. Використання методів стохастичних обчислень дозволяє суттєво зменшити габарити та площу арифметичних вузлів, зокрема суматорів, блоків віднімання [11, 12] та множників [13, 14]. На сьогодні цей підхід успішно застосовується у цифровій обробці сигналів [15–18], декодуванні кодів із низькою щільністю перевірок на парність [19], проектуванні цифрових фільтрів [20], системах реального часу [21], векторному квантуванні [22], обробці зображень [23, 24] та під час розробки систем штучного інтелекту [25–29].

Ключовою перевагою стохастичних обчислень є можливість реалізації складних математичних операцій за допомогою найпростіших логічних елементів. Операції додавання та віднімання можуть бути виконані з використанням лише одного логічного елемента OR (для уніполярного подання даних) або мультиплексора (для біполярного подання). Для реалізації операції множення застосовуються стандартні логічні елементи AND та XNOR. Таким чином, 10 000 складних цифрових множників, згаданих у попередньому прикладі та кожен з яких містить десятки логічних компонентів, можуть бути замінені на 10 000 елементарних вузлів XNOR. Застосування стохастичних обчислень при побудові глибоких нейронних мереж забезпечує суттєву економію площі кристала та зниження енергоспоживання, що є основним мотивом розробки ефективної методології на базі цієї технології.

Попри очевидні переваги, впровадження стохастичних систем супроводжується специфічними обчислювальними ризиками, які необхідно

враховувати та моніторити під час розробки програмних платформ. Оскільки числа в стохастичних обчисленнях подаються у вигляді випадкових бітових потоків, для перетворення дійсних значень у стохастичну форму та зворотної операції потрібні спеціалізовані модулі: генератор випадкових чисел, цифровий перетворювач ймовірностей та блок оцінювання ймовірностей.

Традиційна структура та принципи функціонування блоку оцінювання ймовірностей передбачають, що для отримання вихідного дійсного значення він має збігатися до шуканого числа через контур зворотного зв'язку. Такий процес збіжності є критичним недоліком, оскільки для досягнення прийнятної точності він вимагає мільйонів тактових циклів. Це призводить до суттєвого зростання часових затримок, знижує загальну продуктивність системи та фактично нівелює переваги від економії процесорного часу і енергоспоживання.

Ще однією суттєвою проблемою використання мультиплексора (MUX) як біполярного суматора в стохастичних обчисленнях є наявність селектора у k-входовому мультиплексорі. Це призводить до того, що результуюче значення завжди масштабується вниз шляхом ділення на k. Унаслідок цього виникають додаткові значні витрати апаратних ресурсів, енергоспоживання та часової затримки на відновлення коректного масштабу результату [16, 30, 31]. Сучасні архітектурні рішення, зокрема нові типи процесорних елементів (PE) без петлі зворотного зв’язку на базі скінченних автоматів (FSM) та реверсивних лічильників, спрямовані на повне усунення зазначених недоліків — проблем повільної збіжності та масштабування результату вниз.

Найбільш доцільним середовищем для фізичного розгортання таких оптимізованих стохастичних нейромереж є програмовані логічні інтегральні схеми (ПЛІС, або FPGA). Вибір на користь чипів FPGA зумовлений їхньою здатністю до реконфігурації [32], що дозволяє надзвичайно легко змінювати топологію реалізованої ГНМ (кількість шарів, нейронів тощо). Протягом останніх років спостерігається стрімке зростання пропускної здатності вводу-виводу (I/O), швидкості, продуктивності ПЛІС, а також значне зниження їх вартості, енергоспоживання та ризиків розробки у порівнянні з розробкою спеціалізованих

інтегральних схем (ASIC) [33]. Пристрої FPGA є за своєю природою паралельними обчислювальними машинами, тому реалізації ГНМ на їхній базі можуть бути глибоко налаштовані та швидко переконфігуровані для підтримки різних завдань [34]. Більше того, порівняно з центральними (CPU) та графічними процесорами (GPU), реалізації на базі FPGA здаються набагато більш придатними завдяки швидкому часу впровадження, гнучкості та низькому енергоспоживанню [35].

Попри значні можливості ПЛІС, пряма паралельна реалізація складних мереж стикається з апаратними обмеженнями. Основним ризиком є обмежена кількість фізичних виводів на кристалі ПЛІС. Для подолання цього бар'єра сучасні архітектурні підходи передбачають застосування послідовного введення даних. За такого методу дані (наприклад, пікселі зображення) надходять у ПЛІС послідовно та обробляються базовими стохастичними модулями нейронів шар за шаром, а проміжні результати зберігаються в незалежних регістрах пам'яті. Такий підхід забезпечує колосальне скорочення використання площі та ресурсів кристала, не створюючи при цьому додаткових затримок завдяки конвеєрній природі обчислювального процесу.

Саме тому, розробка програмної платформи для моніторингу та оцінки ризиків функціонування таких стохастичних систем є критично важливим етапом перед їх фізичним розгортанням на базі FPGA. Платформа має дозволяти перевіряти гіпотези щодо збіжності, оцінювати похибки втрати точності (scale-down) та верифікувати апаратні оптимізації.

Структура роботи логічно вибудована наступним чином: спочатку подано детальний аналіз суміжних наукових досліджень та стислий огляд загальної структури нейронних мереж типу MLP відповідно. Далі глибоко розглядаються фундаментальні принципи стохастичних обчислень (SC) та детально описуються їхні базові обчислювальні елементи. В третьому розділі подано вичерпний опис нашої унікальної запропонованої архітектури PE, розкрито та проаналізовано апаратний дизайн запропонованої реалізації системи безпосередньо на базі FPGA. Результати нашого комп'ютерного моделювання та оцінка загальних апаратних

витрат розробленої системи всебічно порівнюються з найкращими найсучаснішими існуючими дослідженнями у цій галузі. Загальні підсумки та висновки до роботи наведено в кінці роботи.

1.2 Інструментарій та технології розробки програмних платформ для стохастичних систем

Апаратна реалізація глибоких нейронних мереж (ГНМ) останнім часом привертає значну увагу дослідницької та інженерної спільноти. Це зумовлено тим, що багато сучасних застосунків вимагають виконання обчислень з високою швидкістю, що можливо забезпечити лише завдяки спеціалізованому апаратному забезпеченню. Для формування об’єктивного уявлення про проблему та обґрунтування вибору архітектури розроблюваної програмної платформи доцільно розглянути як класичні (двійкові), так і стохастичні підходи до реалізації нейронних мереж.

Для глибшого порівняльного розуміння переваг стохастичних систем автори багатьох робіт проводили оцінку продуктивності нестохастичних реалізацій штучних нейронних мереж (ШНМ) на базі програмованих логічних інтегральних схем (ПЛІС, або FPGA). Зокрема, Valencia та ін. [43] представили реалізацію ШНМ на кристалі Xilinx Artix-7, використовуючи спеціалізовану архітектуру набору команд. Отримані результати продемонстрували суттєве зменшення як площі, що займає мережа на кристалі, так і загального енергоспоживання системи.

Іншим поширеним підходом є використання арифметики з фіксованою комою. Зокрема, Park and Sung [44] запропонували FPGA-реалізацію глибоких нейронних мереж (ГНМ) з фіксованою комою, у якій більшість вагових коефіцієнтів зберігалася безпосередньо у внутрішній пам’яті кристала. Такий метод дозволив повністю уникнути звернення до зовнішньої пам’яті, яка часто є основним джерелом затримок, та значно прискорити обчислення вихідних значень нейромережі. Проте через обмежену ємність вбудованої пам’яті ПЛІС авторам довелося застосувати жорстку оптимізацію — використовувати лише 3-бітні ваги.

Це рішення, хоча й знизило енергоспоживання, водночас негативно вплинуло на пропускну здатність і точність мережі.

Серед інших помітних рішень варто виділити архітектури без використання множників. У одній із робіт [45] була представлена нейронна мережа з одним прихованим шаром, реалізована на платформі Zynq-7000 ZC706. Завдяки квантуванню вхідних даних і функцій активації до значень, кратних степеням двійки, розробники замінили складні апаратні множники на прості блоки зсуву. Це дозволило максимально паралелізувати обчислення та реалізувати велику кількість нейронів, уникнувши високих затримок і надмірного використання ресурсів ПЛІС.

Крім того, у роботі [46] було запропоновано простий та ефективний спосіб реалізації ГНМ шляхом створення IP-ядра на платі Xilinx Zybo XC7Z010CLG400-1, де автори досягли точності 99.38%, використавши лише 27.9% загальної площі кристала:

- Створення спеціалізованих IP-ядер (Intellectual Property cores) на платах типу Xilinx Zybo, що дозволило досягти точності розпізнавання на рівні 99.38%, займаючи лише 27.9% загальної площі кристала [47, 48].
- Швидкі реалізації попередньо навчених мереж із застосуванням глибокої конвеєризації (pipelining) та біт-серійних або паралельних методів обробки даних [49].
- Реалізації на основі мереж Хопфілда для параметричної ідентифікації динамічних систем [50].
- Оптимізації за рахунок значного спрощення обчислень та дискретизації функцій активації (що, однак, призводить до критично низької точності) [51].
- Триетапні підходи до реалізації реконфігурованих нейромереж реального часу (RRANN) [52].
- Використання імпульсної арифметики, де сигнал, що передається між нейронами, представляється у вигляді дельта-кодованих двійкових послідовностей (корисно для мереж середнього розміру, але масштабування до великих структур є неможливим) [54-56].

Відредагована версія: Незважаючи на значний прогрес нестохастичних методів, саме стохастичні обчислення забезпечують найбільш суттєве заощадження апаратних ресурсів. У роботі [31] запропоновано ефективну реалізацію глибоких нейронних мереж на основі стохастичних обчислень. Розроблена архітектура, реалізована на ПЛІС Virtex-7, забезпечує в середньому скорочення площі на 45 % та зменшення затримок на 62 % порівняно з найкращими існуючими аналогами. Синтез цих схем за 65-нм технологією CMOS продемонстрував зниження енергоспоживання на 21 % порівняно з двійковою реалізацією при тій самій швидкості класифікації. З урахуванням ризиків виникнення похибок, притаманних стохастичним архітектурам, також розглядалася квазісинхронна реалізація, яка дозволила додатково зменшити енергоспоживання на 33 % без втрати продуктивності.

B. Li та ін. [36] застосували підхід стохастичних бітових потоків для повної реалізації обмеженої машини Больцмана (RBM), призначеної для розпізнавання рукописних цифр на ПЛІС. Це рішення дозволило суттєво заощадити апаратні ресурси. Для згорткових нейронних мереж (CNN) було запропоновано новаторський стохастичний метод із використанням суматорів на базі T-тригерів, який підвищив швидкість і точність обчислень, проте супроводжувався збільшенням енергоспоживання [40]. Крім того, у системі динамічних стохастичних обчислень (DSC) [41] заміна статичних чисел на стохастично кодовані змінні сигнали дозволила значно підвищити точність обчислень.

Генератори стохастичних послідовностей найчастіше базуються на лінійних регістрах зсуву з лінійним зворотним зв’язком (LFSR), які можуть займати понад 80 % площі кристала та характеризуються високим енергоспоживанням [37]. У зв’язку з цим проєктування ефективних генераторів бітових потоків є критичним завданням. Для вирішення цієї проблеми в роботах [37, 38] запропоновано метод спільного використання переставлених виходів одного LFSR кількома генераторами, що дозволяє суттєво зменшити апаратні витрати та знизити кореляцію між потоками. Крім того, ефективний метод обміну провідниками для підвищення точності обчислень було представлено в роботі [39].

Для зменшення обчислювальних ризиків, зокрема похибок флуктуації, Yidong Liu та ін. [42] розробили оцінювач ймовірностей (PE) та дільник на основі структури потрійного модульного резервування (TMR). У запропонованій архітектурі застосовано розширену стохастичну логіку (ESL) для реалізації як прямого, так і зворотного поширення сигналів у багатошаровому перцептроні. Крім того, блок активації на базі стохастичних обчислень (SCAU) забезпечує реалізацію різних функцій активації, зокрема tanh і ReLU. У цьому блоці класичні суматори та віднімачі замінено на регістри зсуву та компаратори. Для підвищення точності роботи операцію перцептрона чітко розділено на дві фази: фазу обчислень та фазу стабілізації.

1.3 Огляд існуючих програмних рішень та суміжних систем

Апаратна реалізація штучних нейронних мереж (ШНМ) на базі програмованих логічних інтегральних схем (ПЛІС/FPGA) є складним інженерним завданням. Перехід від абстрактної математичної моделі, розробленої в середовищі програмування, до фізичної реалізації у вигляді логічних вентилів на кристалі потребує застосування багаторівневих програмних інструментів і засобів автоматизації проєктування. Оскільки метою даного дослідження є розробка програмної платформи для моніторингу ризиків стохастичних систем з нейромережевими модулями, критично важливим є детальний аналіз існуючих програмних екосистем, призначених для розгортання штучного інтелекту на апаратному рівні. Такий аналіз дозволить виявити недоліки сучасних інструментів у контексті стохастичних обчислень (SC) та обґрунтувати архітектуру запропонованого власного рішення.

Більшість сучасних проєктів у сфері глибокого навчання (Deep Learning) розпочинаються з використання стандартизованих високорівневих фреймворків, таких як TensorFlow (розробка Google) або PyTorch (розробка Meta). Ці платформи надають розробникам потужні та зручні інтерфейси програмування додатків (API) для конструювання, тренування та тестування топологій багатошарових

перцептронів (MLP), згорткових (CNN) та рекурентних (RNN) нейромереж. Проте, архітектура TensorFlow та PyTorch історично оптимізована під виконання математичних операцій на графічних (GPU) або тензорних (TPU) процесорах з використанням арифметики з рухомою комою (формати FP32 або FP16). Вони не містять вбудованих механізмів для безпосередньої трансляції моделей у конфігураційні файли (бітстріми) для спеціалізованих кристалів ПЛІС і, тим більше, не підтримують парадигму стохастичних обчислень, де інформація кодується не позиційними числами, а ймовірнісними бітовими потоками.

Для подолання розриву між програмним забезпеченням та апаратним рівнем використовується проміжне програмне забезпечення (middleware) та формати обміну, зокрема ONNX (Open Neural Network Exchange). Цей стандарт дозволяє експортувати навчену модель з будь-якого популярного фреймворку в уніфікований граф обчислень, який згодом може бути оброблений та оптимізований спеціалізованими компіляторами апаратних платформ.

Процес розгортання традиційної нейронної мережі на апаратному забезпеченні зазвичай включає етапи квантування (переходу від 32-бітної точності до 8-бітної або нижчої), прунінгу (видалення надлишкових нульових зв’язків) та трансляції математичного графа моделі в апаратні інструкції. Однак класичні методи квантування є повністю несумісними з генераторами випадкових чисел (зокрема, LFSR), які становлять основу стохастичних архітектур. Це створює нагальну потребу в розробці альтернативних програмних інструментів, орієнтованих на роботу зі стохастичними обчисленнями.

На ринку інструментів для апаратного прискорення нейронних мереж на базі ПЛІС домінують кілька великих компаній, насамперед AMD (Xilinx) та Intel, які пропонують власні закриті й напіввідкриті програмні стеки. Платформа AMD Xilinx Vitis AI є одним із провідних індустріальних стандартів для оптимізації та розгортання нейронних мереж на пристроях сімейств Zynq, Alveo та Versal. У її основі лежить використання DPU (Deep Learning Processor Unit) — попередньо синтезованого та оптимізованого апаратного оверлея (soft-core процесора), який завантажується безпосередньо на кристал ПЛІС. Компілятор Vitis AI приймає граф

моделі, виконує жорстке квантування до цілочисельного формату (INT8) і генерує мікрокод для виконання на DPU. Незважаючи на високу ефективність і зручність для комерційного застосування, Vitis AI повністю абстрагує розробника від фізичного рівня логічних вентилів. Це робить неможливим втручання в архітектуру арифметичних блоків, зокрема заміну стандартних двійкових множників на стохастичні елементи (наприклад, вентилі XNOR).

Окремої уваги заслуговує науково-дослідницький фреймворк FINN, розроблений лабораторією Xilinx Research. На відміну від оверлейного підходу Vitis AI, FINN орієнтований на генерацію просторових dataflow-архітектур — конвеєрних потоків даних, у яких кожен шар нейронної мережі розгортається безпосередньо на кристалі ПЛІС у вигляді виділеного апаратного блоку. Фреймворк FINN спеціалізується на екстремальному квантуванні (до 1–2 бітів), тобто на бінаризованих нейронних мережах (BNN). Хоча BNN мають певну подібність зі стохастичними обчисленнями через використання одно- та двобітових представлень, вони ґрунтуються на принципах детермінованої логічної алгебри. Інструменти FINN не підтримують роботу з ймовірнісними послідовностями, а отже, не містять модулів моніторингу для оцінки ризиків флуктуаційних похибок та проблем збіжності, які є критичними для стохастичних обчислень (SC).

Програмний стек OpenVINO компанії Intel призначений для крос-платформної оптимізації інференсу штучних нейронних мереж (ШНМ). Платформа підтримує розгортання моделей на центральних процесорах, інтегрованій графіці, VPU та ПЛІС (зокрема, серій Intel Agilex і Stratix). OpenVINO перетворює моделі у власний проміжний формат (Intermediate Representation, IR) і пропонує потужні інструменти профілювання продуктивності, зокрема Model Analyzer. Подібно до рішень AMD Xilinx, OpenVINO повністю орієнтований на детерміновані обчислення з фіксованою комою. У контексті аналізу продуктивності платформи Intel надають детальні метрики затримок (latency) та пропускної здатності (throughput). Однак вони не містять інструментів для верифікації проблеми масштабування вниз та оцінки похибок наближення, які

виникають через кореляцію паралельних бітових потоків у стохастичних нейромережах.

Оскільки провідні промислові компанії орієнтуються виключно на двійкову арифметику, розробка й дослідження нейронних мереж на базі стохастичних обчислень змушені спиратися на спеціалізовані академічні симулятори або інструменти низькорівневого автоматизованого проєктування електроніки (Electronic Design Automation, EDA). Серед EDA-інструментів базовим рішенням залишається використання мов опису апаратури (VHDL, Verilog, SystemVerilog) у середовищах Xilinx Vivado або Siemens ModelSim. Такий підхід вимагає ручного написання RTL-коду (Register-Transfer Level) для кожного генератора випадкових чисел, стохастичного нейрона та оцінювача ймовірностей (PE). Хоча RTL-симуляція забезпечує математично точне моделювання та дозволяє аналізувати ризики функціонування системи на рівні кожного тактового імпульсу, вона є надзвичайно ресурсо- та часозатратною. Симуляція роботи навіть базової глибокої нейронної мережі (наприклад, класифікація одного тестового зображення розміром 28×28 пікселів) на рівні логічних вентилів може тривати кілька годин. Це робить практично неможливим збір статистично достовірних даних для оцінки ризиків збіжності при обробці великих наборів даних, що містять тисячі зображень.

З огляду на ці проблеми, у світовому науковому середовищі виникла потреба в програмних симуляторах високого рівня (High-Level SC Simulators). Такі платформи, що створюються переважно мовами C++ або Python, імітують поведінку стохастичних бітових потоків математичним, а не фізичним шляхом. Вони надають змогу:

- Швидко генерувати матриці псевдовипадкових чисел необхідного розподілу.
- Моделювати логічні операції (наприклад, AND, MUX, XNOR) як математичні операції над векторами ймовірностей.
- Оцінювати дисперсію похибки системи в залежності від довжини стохастичного бітового потоку (N).
- Досліджувати вплив взаємної кореляції між потоками на остаточну точність класифікації.

Таблиця 1.3 – Переваги Google AppSheet у розробці системи обліку запасів

Характеристика / Платформа	AMD/Xilinx Vitis AI	Xilinx FINN	Intel OpenVINO	Системи EDA (Vivado, ModelSim)	Академічні SC-симулятори	Розроблювана програмна платформа
Основне призначення	Комерційне розгортання моделей на ПЛІС	Генерація конвеєрних BNN-архітектур	Оптимізація інференсу на пристроях екосистеми Intel	Низькорівневий апаратний синтез та перевірка таймінгів	Підтвердження гіпотез щодо стохастичної логіки	Комплексне моделювання та моніторинг ризиків стохастичних ГНМ
Базова арифметика	Фіксована кома (переважно INT8/INT4)	Детермінована бінарна (1-2 біти)	Фіксована кома / FP16	Будь-яка (встановлюється на рівні RTL-коду)	Стохастичні бітові потоки (SC)	Стохастичні бітові потоки (SC)
Рівень абстракції	Дуже високий (використання DPU оверлеїв)	Середній (синтез через HLS)	Високий (оптимізація графів обчислень)	Низький (рівень фізичних логічних вентилів)	Середній (математичні моделі)	Високий (математичне та структурне моделювання систем)
Моніторинг похибок збіжності та флуктуації	Не застосовується (системи є повністю детермінованими)	Не застосовується	Не застосовується	Можливий, але не масштабується на великі обсяги даних	Присутній, але обмежений можливостями ручного аналізу	Вбудовані аналітичні модулі для автоматичної оцінки та візуалізації ризиків
Швидкість моделювання великих датасетів	Дуже висока (апаратне прискорення)	Висока	Висока	Критично низька (через потактову симуляцію)	Від низької до середньої	Висока (завдяки дата-орієнтованій паралельній архітектурі)
Наявність інтегрованого UI та UX рішень	Так	Ні	Так	Так	Зазвичай відсутній	Так

Головним недоліком наявних дослідницьких SC-симуляторів є їх розрізненість, відсутність стандартизації та вузька спеціалізація. Більшість із них створюються дослідниками як одноразові консольні скрипти для підтвердження результатів конкретної наукової публікації. Їм бракує графічного інтерфейсу користувача (UI), сучасного API, механізмів безперервної інтеграції (CI/CD) та комплексних модулів для систематичного збору аналітики і моніторингу ризиків у зручному для інженера вигляді.

Для об'єктивної систематизації інформації щодо можливостей сучасних програмних інструментів у контексті розробки апаратно-орієнтованих нейромережевих модулів розроблено зведену порівняльну характеристику (Таблиця 1.3).

Детальний аналіз даних, консолідованих у таблиці, яскраво демонструє наявність так званого «інструментального вакууму» на перетині технологій штучного інтелекту та стохастичних обчислень. З одного боку індустрії існують потужні корпоративні платформи (Vitis AI, OpenVINO), які забезпечують феноменальну швидкість роботи і мають зручні інтерфейси, але вони фундаментально не підтримують імовірнісну природу SC-систем. З іншого боку, наявні низькорівневі системи автоматизованого проєктування (Vivado), які здатні реалізувати будь-яку архітектуру логіки, проте не дозволяють швидко симулювати роботу всієї нейромережі для моніторингу статистичних похибок на великих масивах вхідних даних (наприклад, тестових колекціях зображень). Розробка власної програмної платформи для моніторингу ризиків стохастичних систем покликана заповнити цю технологічну прогалину.

Проєктована платформа поєднуватиме високорівневий підхід до дата-орієнтованого моделювання нейромереж із глибокою математичною імітацією стохастичних процесів (генерації бітових потоків, виникнення похибок кореляції, аналізу повільної збіжності та втрати масштабу при мультиплексуванні). Такий програмний інструмент надасть інженерам можливість проводити достовірну верифікацію архітектури та оцінювати обчислювальні ризики ще до етапу трудомісткої генерації VHDL/Verilog коду та його фізичного розгортання на ПЛІС.

Це, своєю чергою, дозволить суттєво заощадити час розробки, знизити фінансові витрати та гарантувати високу надійність нейромережевих контролерів при роботі в складних умовах реального часу.

1.4 Обґрунтування вибору програмного середовища розробки та застосовуваних підходів

Вибір технологічного стеку для розробки програмної платформи моніторингу ризиків стохастичних систем є критичним етапом, оскільки він визначає не лише швидкість розробки, а й гнучкість аналітичних інструментів, можливість інтеграції з існуючими нейромережевими фреймворками та масштабованість системи при імітації складних обчислювальних процесів. Основним інструментарієм для реалізації програмної частини було обрано мову програмування Python. Нижче наведено детальний опис та аргументацію вибору компонентів середовища розробки.

Незважаючи на те, що кінцеві стохастичні архітектури найчастіше розгортаються на низькорівневих апаратних мовах (Verilog, VHDL), Python виступає найкращим кандидатом для створення «високорівневого цифрового двійника» таких систем.

- Екосистема наукових обчислень: Завдяки бібліотеці NumPy, платформа отримує можливість виконувати високоефективні операції над великими масивами даних. У контексті стохастичних обчислень це дозволяє моделювати мільйони бітових потоків паралельно, використовуючи векторні операції, що значно швидше за класичні цикли.
- Аналіз ризиків та візуалізація: Оскільки ключовою функцією платформи є моніторинг ризиків (зокрема, флуктуаційної похибки та затримок збіжності), необхідні потужні засоби статистичного аналізу. Використання бібліотек SciPy та Pandas забезпечує можливість швидкого обчислення дисперсії, середньоквадратичного відхилення та кореляційних матриць між стохастичними сигналами. Для візуалізації процесів у реальному часі обрано

Matplotlib та Seaborn, що дозволяють будувати динамічні графіки збіжності, як це описано в базовому дослідженні.

- Інтеграція з ШНМ: Використання фреймворку PyTorch дозволяє легко імпортувати вже навчені «класичні» моделі нейромереж, отримувати їхні вагові коефіцієнти та автоматично трансформувати їх у параметри для стохастичного моделювання.

Таблиця 1.4 – Порівняння технологічних стоків для розробки платформи моніторингу

Критерій порівняння	C++ / Qt	MATLAB	Python (Обрано)
Швидкість розробки	Низька	Висока	Дуже висока
Бібліотеки ШІ	Обмежені	Власні (Toolboxes)	Найширший вибір (PyTorch/TF)
Аналіз даних (Statistics)	Потребує сторонніх SDK	Відмінно	Відмінно (SciPy/Pandas)
Вартість ліцензії	Free (Open Source)	Висока (Proprietary)	Free (Open Source)
Гнучкість інтерфейсу	Висока (складна)	Середня	Висока (швидка)

При виборі технології розглядалися варіанти використання мов C++ та середовища MATLAB. Хоча C++ забезпечує вищу швидкість виконання, він вимагає значно більших часових витрат на розробку інтерфейсів моніторингу та управління пам'яттю. MATLAB, своєю чергою, є потужним інструментом, проте він є закритим пропрієтарним середовищем, що обмежує можливості безкоштовного розгортання платформи та її інтеграції з сучасними хмарними сервісами.

Для написання коду обрано інтегроване середовище розробки Visual Studio Code (VS Code). Вибір обґрунтований наявністю розширень для роботи з Jupyter Notebooks, що дозволяє проводити ітеративне тестування окремих модулів стохастичної логіки (наприклад, генераторів LFSR) та одразу бачити результат

візуалізації ризиків без перекомпіляції всього проєкту. Враховуючи необхідність контролю версій та колективної роботи над проєктом, використовується система Git. Це дозволяє відстежувати зміни в алгоритмах оцінки ймовірностей (Probability Estimators) та забезпечує надійність збереження коду платформи.

Для забезпечення відтворюваності результатів моделювання, управління залежностями здійснюється через середовище Anaconda або віртуальні оточення venv. Це гарантує, що всі бібліотеки для аналізу ризиків будуть мати ідентичні версії на різних робочих станціях.

Таким чином, використання мови Python у поєднанні з бібліотеками NumPy, PyTorch та Matplotlib є найбільш обґрунтованим рішенням для розробки платформи моніторингу ризиків. Цей стек технологій забезпечує ідеальний баланс між продуктивністю обчислень, швидкістю розробки аналітичного функціоналу та легкістю інтеграції з сучасними нейромережевими архітектурами. Обране середовище дозволяє не лише моделювати стохастичну природу обчислень, описану вище, але й створити зручний інструментарій для інженерного аналізу та візуалізації помилок, що є критичним для систем реального часу.

1.5 Ресурсозатратна ефективність реалізації програмної платформи

Будь-який інженерний проєкт, окрім наукової новизни, повинен мати чітке економічне та ресурсне обґрунтування. Розробка програмної платформи для моніторингу ризиків стохастичних систем демонструє високу вартісну ефективність на всіх етапах життєвого циклу — від початкового проєктування до етапу тестування та верифікації моделей.

Економічна привабливість проєкту першочергово зумовлена обраним технологічним стеком. Використання мови програмування Python та екосистеми відкритих бібліотек (NumPy, PyTorch, Pandas, Matplotlib) зводить прямі витрати на придбання ліцензій до нуля. На відміну від використання комерційних математичних пакетів (наприклад, MATLAB) або корпоративних систем автоматизованого проєктування електроніки (EDA інструментів), досліднику чи

інженеру не потрібно сплачувати дорогі щорічні підписки для моделювання процесів.

Пряма фізична реалізація та тестування стохастичних архітектур на базі ПЛІС вимагає закупівлі дорогих відлагоджувальних плат (наприклад, на базі чипів Xilinx Virtex-7 або аналогів, вартість яких може сягати тисяч доларів). Розроблена програмна платформа виконує роль цифрового двійника системи. Вона дозволяє моделювати генерування мільйонів стохастичних бітових потоків та перевіряти архітектурні гіпотези на звичайних персональних комп'ютерах або доступних хмарних серверах, повністю усуваючи необхідність передчасної закупівлі спеціалізованого обладнання.

Робочий час інженера є найдорожчим ресурсом у розробці апаратного забезпечення. Процес синтезу та трасування (synthesis and routing) RTL-коду для великих нейромереж у середовищах ПЛІС може займати від кількох годин до доби. Натомість векторне математичне моделювання стохастичних процесів у середовищі Python дозволяє отримувати результати збіжності, оцінки похибок та аналітику ризиків за лічені хвилини. Це кардинально прискорює цикл тестування та дозволяє проводити набагато більше ітерацій для покращення моделі.

Головний економічний ефект розроблюваної платформи полягає у запобіганні «пізнім помилкам». Стохастичні обчислення схильні до специфічних ризиків: флуктуаційних похибок, проблем масштабування даних та повільної збіжності. Виявлення цих дефектів вже на етапі фізичного розгортання контролерів призводить до необхідності повного перепроєктування апаратної частини, що тягне за собою величезні фінансові збитки. Програмний моніторинг ризиків дозволяє виявити та математично нівелювати ці загрози ще до написання низькорівневого коду апаратури.

Створення спеціалізованої програмної платформи є економічно та ресурсно доцільним кроком. Вона мінімізує фінансові витрати завдяки Open-Source інструментам, економить сотні годин інженерного часу та запобігає дорогим помилкам при майбутньому апаратному впровадженні складних стохастичних нейромереж.

1.6 Проблеми безпеки програмної платформи, режими доступу та можливості вдосконалення системи

Розробка сучасного інженерного та наукомісткого програмного забезпечення вимагає не лише забезпечення точності математичних обчислень, але й закладення надійної архітектури надійності, безпеки даних та можливостей для подальшого росту системи. Оскільки розроблювана платформа призначена для моніторингу ризиків та верифікації моделей глибоких нейронних мереж (ГНМ), вона оперуватиме значними масивами тестових даних, конфігураціями архітектур та результатами аналітики, що потребує відповідних механізмів захисту.

Хоча на початковому етапі розробки платформа може функціонувати як локальне середовище моделювання, її архітектура повинна підтримувати базові принципи інформаційної безпеки. Ключовим аспектом є впровадження рольової моделі контролю доступу (Role-Based Access Control, RBAC). Це передбачає логічне розділення прав користувачів:

- Інженери-дослідники (Developers): мають повний доступ до завантаження нових топологій нейромереж, зміни параметрів генераторів стохастичних послідовностей (наприклад, LFSR) та налаштування метрик оцінки ризиків.
- Аналітики (Viewers): мають доступ у режимі лише для читання (Read-Only) до згенерованих звітів, графіків збіжності та статистичних даних щодо флуктуаційних похибок без можливості втручання в конфігурацію системи.

Окремо варто відзначити можливості масштабування AppSheet-рішення.

Для забезпечення цілісності даних результати тривалого моделювання, вагові коефіцієнти ШНМ та логи виявлених обчислювальних ризиків повинні зберігатися у структурованому вигляді (наприклад, локальних базах даних SQLite або форматах JSON/CSV) із захистом від несанкціонованої модифікації. При подальшому перенесенні платформи у вебсередовище обов’язковим є використання протоколів шифрування (HTTPS/TLS) для захисту даних під час передачі між клієнтом та сервером.

Особливістю стохастичних обчислень є висока ресурсомісткість процесу моделювання: для отримання статистично достовірного результату збіжності необхідно згенерувати мільйони тактових імпульсів для кожного нейрона. Зі збільшенням розміру нейромережі (переходу від простих багатошарових перцептронів до глибоких згорткових мереж) обчислювальне навантаження зростатиме експоненційно. Тому архітектура платформи має бути готовою до масштабування за кількома напрямками:

- Контейнеризація та мікросервісна архітектура: Переведення модулів генерації бітових потоків, оцінки ймовірностей та візуалізації у незалежні мікросервіси з використанням технології Docker. Це дозволить розгортати платформу не лише на локальних ПК, але й у хмарних середовищах (AWS, Google Cloud Platform, Azure), легко нарощуючи потужності (горизонтальне масштабування). Розподілені обчислення та апаратне прискорення: Інтеграція платформи з технологіями паралельних обчислень (наприклад, Apache Spark або Dask) та перенесення векторних матричних розрахунків бітових потоків з центрального процесора (CPU) на графічні прискорювачі (GPU) за допомогою технології CUDA, підтримуваної фреймворком PyTorch. Це кардинально зменшить час симуляції ризиків для великих датасетів. API-інтеграція: Створення повноцінного RESTful API дозволить інтегрувати розроблену платформу з промисловими системами автоматизованого проєктування (EDA), такими як Xilinx Vivado. Це відкриє перспективи для створення замкненого циклу (CI/CD): від програмної оцінки ризиків до автоматичної генерації Verilog-коду на основі верифікованої стохастичної моделі.

Закладення базових механізмів безпеки та орієнтація на майбутнє масштабування (через контейнеризацію та хмарні технології) забезпечують високу життєздатність розроблюваної платформи. Це дозволить системі еволюціонувати від дослідницького інструменту до потужного корпоративного рішення для проєктування безпечних Edge-AI пристроїв на базі стохастичних обчислень.

2. ПРОЄКТУВАННЯ І РОЗРОБКА ПРОГРАМНОЇ ПЛАТФОМИ ТА ЇЇ КОМПОНЕНТІВ

В розділі описано математичні моделі та архітектуру розробленого рішення, включаючи вдосконалений оцінювач ймовірності (PE) без петлі зворотного зв’язку для прискорення обчислень. Також наведено результати практичного тестування платформи на програмованих логічних інтегральних схемах (FPGA), які підтверджують значне скорочення використання апаратних ресурсів та підвищення енергоефективності системи.

2.1 Формальні характеристики програмної платформи. Стохастичні обчислення

В підрозділі описано та деталізовано структуру та характеристики програмної платформи

2.1.1 Функціональні характеристики та властивості програмної системи

Людський мозок, який є одним із найбільших і найскладніших органів людського тіла, складається зі 100 мільярдів нервових клітин, з'єднаних тисячами трильйонів зв'язків, що називаються синапсами. Штучна нейронна мережа (ШНМ) — це обчислювальна система, розроблена для моделювання функцій, натхненних біологічними нейронними мережами мозку людини [51]. Глибока нейронна мережа (ГНМ) містить тисячі нейронів, які називаються блоками обробки (processing units). Вхідні блоки отримують різноманітні дані, після чого вихідний сигнал генерується на основі навчених вагових коефіцієнтів та значень зміщень (biases). Цілі нейронних мереж поділяються на два класи: розпізнавання образів (класифікація) та апроксимація функцій. Загалом архітектури ГНМ фундаментально поділяються на три класи: одношарові мережі прямого поширення, багатошарові мережі прямого поширення та рекурентні мережі. У даній роботі ми зосереджуємося на

багатошаровому перцептроні (MLP) як типі нейронної мережі прямого поширення через його широке застосування в задачах класифікації та апроксимації. Метод керованого навчання зі зворотним поширенням помилки (backpropagation) використовується для навчання мереж MLP для таких застосувань, як машинне навчання, цифро-аналогові перетворювачі [57] та обробка зображень [58].

MLP складається з вузлів вхідного шару, одного або кількох обчислювальних вузлів, які називаються прихованими шарами, та вихідного шару. Повнозв'язний MLP зображений на рис. 2.1, де кожен вузол у кожному шарі з'єднаний з усіма вузлами наступного шару. Нехай L — кількість шарів, а w_{ij}^{l+1} позначає вагу ребра, що з'єднує i-й нейрон l-го шару з j-м нейроном шару $l+1$. Якщо в l-му шарі існує n^l нейронів, то вхідний вектор для j-го шару визначається згідно з рівнянням:

$$X = (x_0^l, x_1^l, x_2^l, \ldots, x_{n^l}^l)^T. \tag{2.1}$$

Вихідний сигнал j-го нейрона шару $l+1$ обчислюється як:

$$y_j^{l+1} = \phi(v_j^{l+1}), \tag{2.2}$$

де ϕ — функція активації, а v_j^{l+1} — функція прямого поширення (feed-forward), що визначається як:

$$v_j^{l+1} = \sum_{i=1}^{n^l} (w_{ij}^{l+1} \times x_i^l). \tag{2.3}$$

Ми обираємо ReLU (Rectified Linear Unit) як функцію активації через її простоту, здатність забезпечувати розрідженість представлень та лінійну поведінку [59, 60]:

$$\phi(v_j^{l+1}) = \min\left(1, \max(0, v_j^{l+1})\right)$$

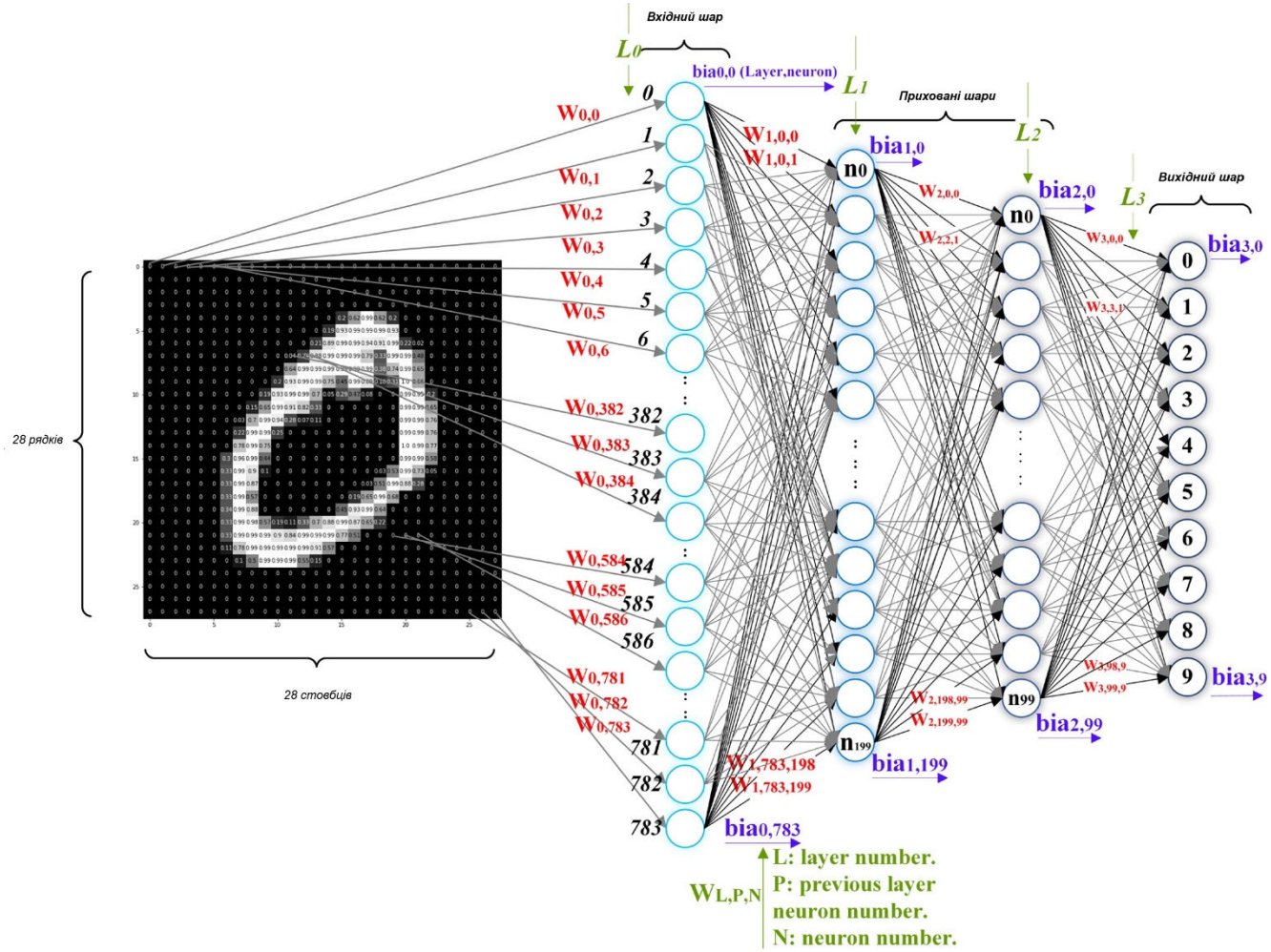


Рисунок 2.1 – Структура багатошарового перцептрона (MLP)

2.1.2 Підходи до прямого стохастичного програмування

У стохастичних обчисленнях (Stochastic Computing, SC) кожне число представляється послідовністю випадкових бітів, що дозволяє реалізувати апаратно-ефективні схеми, використовуючи дуже малу кількість фундаментальних логічних вентилів [61]. Зі збільшенням довжини цієї послідовності точність подання стохастичного числа (Stochastic Number, SN) поступово підвищується. Стохастичні числа зазвичай представляються у двох формах: уніполярній (unipolar) та біполярній (bipolar). Діапазон відповідних дійсних чисел для цих форм становить [0,1] та [–1,1] відповідно. Якщо випадковий двійковий бітовий потік довжиною N містить p_a одиниць (1s), то він репрезентує дійсні значення за

такими формулами. Для уніполярного представлення: $\frac{p_a}{N}$ Для біполярного представлення: $\frac{2p_a - N}{N}$ Для кращого розуміння розглянемо приклад бітового потоку $X = 00100010$. У цьому випадку $N = 8$, а кількість одиниць $p_a = 2$. Цей потік представляє уніполярне значення $\frac{2}{8}$ та біполярне значення $-\frac{4}{8}$. Стохастичні числа (SN) зазвичай генеруються модулями генераторів випадкових чисел (Random Number Generator, RNG), такими як лінійний регістр зсуву зі зворотним зв'язком (LFSR), у комбінації з компаратором.

2.1.3 Операція додавання на основі стохастичного програмування

Спочатку розгляне традиційний модуль оцінювання ймовірності (Probability Estimator, PE) та проаналізуємо його основні недоліки. Після цього детально опишемо запропоновану вдосконалена апаратну архітектуру PE, яка розроблена для подолання існуючих обмежень.

Точний традиційний оцінювач ймовірності (PE) являє собою простий лічильник, який підраховує кількість логічних одиниць у бітовому потоці стохастичних обчислень для обчислення відповідного дійсного значення [65, 66]. Незважаючи на те, що такий підхід забезпечує високу точність, для апроксимації кінцевої ймовірності йому потрібно 2^n тактів синхронізації (де n — розрядність лічильника). Це призводить до значних часових затримок (latency) та суттєвих накладних витрат енергії в обчислювальних системах, що є критичним недоліком при реалізації ресурсомістких алгоритмів, таких як глибокі нейронні мережі. Одним із широко використовуваних оцінювачів, спрямованих на зменшення цих часових затримок, є наближений паралельний лічильник (Approximate Parallel Counter, APC). Структура APC складається з послідовних обчислювальних блоків, побудованих на основі кількох повних суматорів (Full Adders, FA) та напівсуматорів (Half Adders, HA) [14, 62-64]. Оскільки сигнал перенесення (carry)

враховується лише для послідовних компонентів, ця архітектура функціонує як наближений лічильник, що неминуче призводить до зниження загальної точності обчислень. Крім того, такий наближений модуль вимагає залучення більшої кількості апаратних ресурсів порівняно з простими послідовними схемами. Важливо зазначити, що основним джерелом похибки в таких наближених модулях є саме фундаментальне ігнорування сигналів перенесення під час паралельного підсумовування бітів.

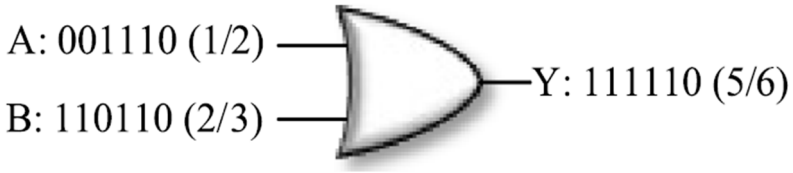


Рисунок 2.2 – Стохастичне уніполярне додавання з використанням логічного елемента АБО

Стохастичне додавання може бути реалізоване за допомогою логічного вентиля OR (АБО) для уніполярного формату, за допомогою мультиплексора (MUX) для біполярного формату, або ж, як еквівалентна універсальна альтернатива, за допомогою акумулюючого паралельного лічильника (Accumulative Parallel Counter, APC), який підходить для обох форм представлення [14, 62-64]. За умови наявності двох незалежних стохастичних чисел A та B, операція додавання через вентиль OR виконується так, як показано на рис. 2.2. З точки зору теорії ймовірностей, ця операція OR математично описується рівнянням (2.5), яке повністю відповідає результатам, наведеним на рис. 2.2. Такий підхід базується на обчисленні ймовірності об’єднання двох незалежних подій у статистиці.

$$p_y = p_a + p_b - (p_a \cdot p_b) == \frac{1}{2} + \frac{2}{3} - \left(\frac{1}{2} \times \frac{2}{3}\right) = \frac{5}{6}. \quad (2.5)$$

Для виконання біполярного підсумовування двох стохастичних чисел використовується мультиплексор (MUX), відповідно до рис. 2.3. Оскільки мультиплексор почергово пропускає на вихід один із двох вхідних сигналів

залежно від стану стохастичного керуючого сигналу (селектора), відповідні ймовірнісні розрахунки виконуються наступним чином:

$$p_y = p_a \cdot (p_{sel}) + p_b \cdot (1 - p_{sel}) = \left(\frac{6}{8} \times \frac{1}{2}\right) + \left(-\frac{2}{8} \times \frac{1}{2}\right) = \frac{2}{8}. \quad (2.6)$$

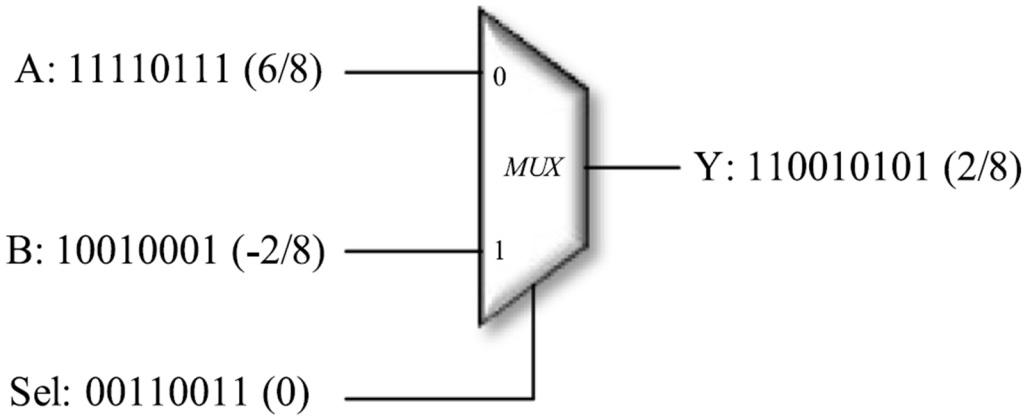


Рисунок 2.3 – Реалізація біполярного стохастичного додавання

У цьому рівнянні використання селектора із заданою ймовірністю $p_{sel} = 1/2$ забезпечує так зване масштабоване додавання. Це дозволяє об'єднати два біполярні сигнали, зважуючи кожен із них рівно наполовину, що гарантує уникнення переповнення діапазону [–1,1].

В нашому випадку p_a, p_b та p_{sel} — це ймовірності стохастичних чисел A, B та керівного сигналу (селектора) sel відповідно. Важливо зазначити, що для коректної роботи мультиплексор (MUX) повинен рівномірно (з однаковою ймовірністю) пропускати сигнали з верхнього та нижнього входів на вихід. Отже, стохастичний бітовий потік сигналу sel має містити однакову кількість одиничних та нульових бітів, тобто $N/2$. У дійсному діапазоні значень біполярного кодування це еквівалентно числу 0. Як було проілюстровано на рис. 3, отриманий результат $\frac{2}{8}$ становить рівно половину від правильного (очікуваного) значення $\frac{4}{8}$.

У науковій літературі зі стохастичних обчислень цей феномен відомий як проблема зменшення масштабу (scaled-down issue) при біполярному стохастичному додаванні. Для виконання операції додавання з k входами використовується деревоподібна структура, що складається з кількох двовходових

мультиплексорів. У такому випадку кінцевий результат буде масштабований (зменшений) на коефіцієнт $\frac{1}{2^m}$, де m позначає кількість мультиплексорів у відповідному дереві. Щоб математично довести проблему зменшення масштабу, припустимо, що p_a та p_b — це кількості одиничних бітів («1») у вхідних послідовностях A та B відповідно.

У вихідній послідовності мультиплексора out_{MUX} довжиною N бітів рівно половина випадковим чином є копією верхньої послідовності A. Інші випадково обрані біти є копією нижнього входу B. Завдяки властивості стаціонарності, що є фундаментальною для стохастичних послідовностей, статистичні характеристики вибраних бітів з послідовностей A та B є ідентичними до характеристик оригінальних послідовностей, з яких вони походять. Відповідно, якщо кількість одиничних бітів в оригінальній послідовності становить p_a, то кількість одиничних бітів у випадково обраній послідовності половинного розміру дорівнюватиме $\frac{p_a}{2}$. З цього випливає, що вихідний потік out_{MUX} містить $\frac{(p_a + p_b)}{2}$ одиничних бітів замість очікуваної кількості, яка мала б становити $(p_a + p_b)$ одиничних бітів.

Для наочного прикладу припустимо, що вхідні значення дорівнюють $A = 0.2$ та $B = 0.3$. За ідеальних умов очікується, що вихідний сигнал мультиплексора (out_{MUX}) повинен представляти точну алгебраїчну суму цих значень, тобто $out_{MUX} = A + B = 0.2 + 0.3 = 0.5$. Відповідна кількість одиничних бітів («1») у стохастичних послідовностях A та B розраховується з використанням рівнянь (11) і (12) відповідно. Ці формули здійснюють перехід від дійсних чисел до їхнього ймовірнісного еквівалента в біполярному форматі кодування:

$$\frac{2p_a - N}{N} = 0.2 \Rightarrow p_a = \frac{1.2N}{2}, \tag{2.7}$$

$$\frac{2p_b - N}{N} = 0.3 \Rightarrow p_b = \frac{1.3N}{2}. \tag{2.8}$$

Далі, загальна кількість одиничних бітів у вихідному потоці мультиплексора обчислюється за допомогою рівняння (2.7). Після цього відповідне дійсне число для out_{MUX} визначається згідно з рівнянням (2.8). У кінцевому підсумку, отриманий практичний результат (0.25) становить рівно половину від очікуваного теоретичного значення (0.5). Це наочно ілюструє притаманну мультиплексорам властивість масштабованого додавання, що запобігає переповненню:

$$p_{out_{MUX}} = \frac{1}{2}\left(\frac{1.2N}{2} + \frac{1.3N}{2}\right) \Rightarrow p_{out_{MUX}} = \frac{1.2N}{4} + \frac{1.3N}{4} \tag{2.9}$$

або

$$\Rightarrow p_{out_{MUX}} = \frac{2.5N}{4}. \tag{2.10}$$

Звідки:

$$\frac{2p_{out_{MUX}} - N}{N} = \frac{\frac{5}{4}N - N}{N} = \frac{1}{4} = 0.25. \tag{2.11}$$

2.2 Алгоритм множення на основі стохастичних процесів

У випадку використання уніполярного стохастичного представлення базовий логічний вентиль AND (ТА) апаратно інтерпретується як помножувач. Це зумовлено тим, що для статистично незалежних бітових потоків імовірність одночасної появи логічних одиниць на обох входах математично дорівнює добутку їхніх індивідуальних імовірностей. Наочний числовий приклад виконання такої операції наведено на рис. 2.4, а безпосередній розрахунок вихідної ймовірності p_y для вхідних стохастичних сигналів A та B здійснюється згідно з рівняннями поданими далі:

$$p_y = p_a \cdot p_b. \tag{2.12}$$

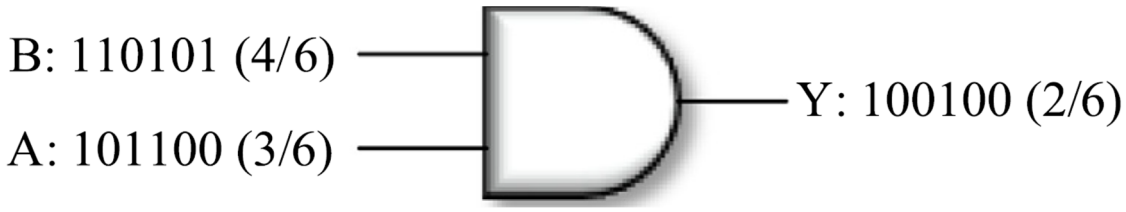


Рисунок 2.4 – Алгоритм стохастичного множення в уніполярному поданні

Припустимо, що p_y — це вихідна ймовірність результату застосування операції XNOR до двох входів, які є статистично незалежними один від одного. У наступних рівняннях ми доводимо, що виконання операції XNOR над стохастичними послідовностями A та B безпосередньо призводить до множення відповідних стохастичних чисел (SN). Згідно з принципами роботи цифрової логіки, коли на обидва входи вентиля XNOR надходить однакове логічне значення, на його виході генерується логічна одиниця, що частково описується рівнянням (2.12). Цей логічний вентиль фактично діє як апаратний детектор збігу, формуючи високий рівень сигналу виключно за умови повної ідентичності поточних станів обох вхідних потоків. Ймовірність появи одиничного (1) або нульового (0) біта у N-бітовій послідовності, яка містить p_1 одиничних бітів та $N - p_1$ нульових бітів, розраховується відповідно до рівняння (20) (див. рис. 2.5).

$$prob_{bit=1} = \frac{p_1}{N}. \tag{2.13}$$

звздки:

$$prob_{bit=0} = 1 - \left(\frac{p_1}{N} \right). \tag{2.14}$$

Отже, загальна ймовірність появи логічної одиниці («1») на виході вентиля XNOR обчислюється з використанням рівняння (2.14). Математично цей імовірнісний розрахунок базується на обчисленні суми ймовірностей двох взаємовиключних подій: імовірності того, що обидва вхідні біти одночасно є одиницями, та ймовірності того, що обидва є нулями. Цей процес покроково описується наступним ланцюжком перетворень:

$$
\begin{aligned}
prob_{y=1} &= prob_{A=1}prob_{B=1} + prob_{A=0}prob_{B=0} = \\
&= \frac{p_a}{N}\frac{p_b}{N} + \left(1 - \left(\frac{p_a}{N}\right)\right)\left(1 - \left(\frac{p_b}{N}\right)\right) = \\
&= 2\frac{(p_a p_b)}{N^2} - \frac{(p_a + p_b)}{N} + 1.
\end{aligned}
\tag{2.15}
$$

Отже, загальна кількість одиничних бітів («1») у вихідній послідовності вентиля XNOR визначається як добуток загальної довжини цієї послідовності (N) на ймовірність появи одиниці. Математично це виражається як $N \times prob_{y=1}$, що приводить до наступного рівняння (2.15):

$$
p_y = 2\frac{(p_a p_b)}{N} - (p_a + p_b) + N. \tag{2.16}
$$

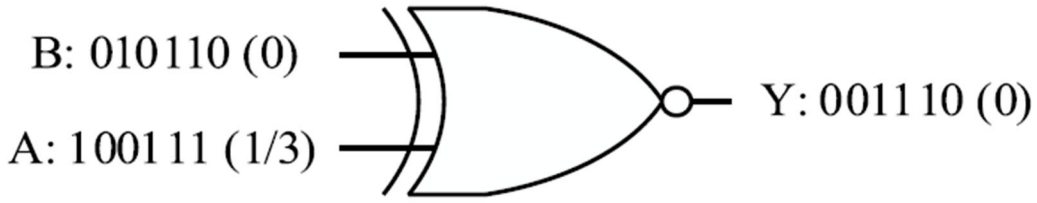


Рисунок 2.5 – Алгоритм біполярного множення з використанням логічного елемента XNOR

З огляду на це, отримане значення p_y репрезентує відповідне дійсне число у біполярному форматі кодування. Цей перехід від імовірнісного представлення до дійсного числа можна розрахувати за допомогою такого ланцюжка алгебраїчних перетворень (рівняння 2.13–216):

$$
\begin{aligned}
out &= \frac{(2p_y - N)}{N} = \frac{4p_a p_b}{N^2} - \frac{2(p_a + p_b)}{N} + 1 = \\
&= \left(\frac{2p_a}{N} - 1\right)\left(\frac{2p_b}{N} - 1\right) = \left(\frac{(2p_a - N)}{N}\right)\left(\frac{(2p_b - N)}{N}\right).
\end{aligned}
\tag{2.17}
$$

Останні дві дужки у фінальному виразі цього рівняння чітко та безпосередньо вказують на еквівалентні дійсні числа, що подаються на верхній та нижній входи логічного вентиля XNOR. Таким чином, наведене математичне доведення

переконливо підтверджує, що логічний вентиль XNOR апаратно виконує точну операцію множення еквівалентних дійсних чисел своїх вхідних сигналів.

2.3. Побудова нейрона на основі стохастичного підходу. Розширена стохастична логіка (ESL)

У цьому підрозділі детально розглянемо апаратні модулі, які були розроблені та запропоновані для побудови нейрона на основі принципів стохастичних обчислень. Цей сконструйований стохастичний нейрон у подальшому буде інтегрований як базовий обчислювальний елемент в архітектуру багатошарового перцептрона (MLP).

Як відомо, діапазон допустимих числових значень при використанні стандартного біполярного кодування є жорстко обмеженим інтервалом $[-1,+1]$. Таке суттєве обмеження динамічного діапазону створює значні перешкоди для повноцінного впровадження стохастичних обчислень у сучасні глибокі нейронні мережі (DNN), оскільки під час їхньої роботи часто виникає потреба в оперуванні набагато більшими значеннями ваг та активацій. З метою ефективного подолання цієї фундаментальної проблеми було запропоновано інноваційний метод розширеної стохастичної логіки (Extended Stochastic Logic, ESL). Цей підхід дозволяє суттєво розширити ефективний числовий діапазон до інтервалу $(-2^N+1, 2^N-1)$ для двійкового числа, яке складається з N розрядів (бітів). На додаток до цього розширення, застосування формату ESL забезпечує системі підвищену завадостійкість (noise immunity). Ця властивість апаратної стійкості базується на математичному відношенні двох комутованих (перемикальних) сигналів, які спеціально обираються для надійного кодування конкретної інформації [65]. Відповідно до парадигми ESL, кожне дійсне число репрезентується за допомогою комбінації двох окремих стохастичних сигналів, які функціонально виконують ролі чисельника та знаменника у відповідному дробовому відношенні. Розглянемо два дійсних числа x та w кодуються з

використанням чотирьох N-бітних бінарних стохастичних потоків p, q, r та s відповідно до рівнянь (29) і (30). Як зазначалося раніше, метод Extended Stochastic Logic (ESL) використовує відношення двох потоків, де один функціонально виступає чисельником, а інший — знаменником:

$$x = \frac{p}{q} \tag{2.18}$$

$$w = \frac{r}{s} \tag{2.19}$$

Для того щоб ці комплексні змінні x та w могли теоретично охоплювати нескінченний динамічний діапазон значень $(-\infty, +\infty)$, що є критично важливим для запобігання переповненню під час складних обчислень, повинні суворо виконуватися наступні математичні умови для базових потоків:

$$\begin{cases} p, r \in [-1,1] \\ q, s \in [-1,1] \end{cases} \tag{2.20}$$

тому

$$\rightarrow \forall x, w \in (-\infty, +\infty). \tag{2.21}$$

Відповідно до базового рівняння (2.3), апаратна реалізація глибоких нейронних мереж (DNN) неодмінно включає інтенсивне обчислення суми добутків вхідних значень (активацій) x_i та відповідних їм синаптичних ваг w_{ij}. Ця операція множення з накопиченням (часто відома як MAC — Multiply-Accumulate) є фундаментальним будівельним блоком будь-якої нейромережі. Наочний приклад виконання такої базової операції в рамках стохастичної парадигми наведено на рис. 6, де апаратна система повинна виконати наступні обчислення для отримання вихідного значення:

$$y = x_0 w_{00} + x_0 w_{01}. \tag{2.22}$$

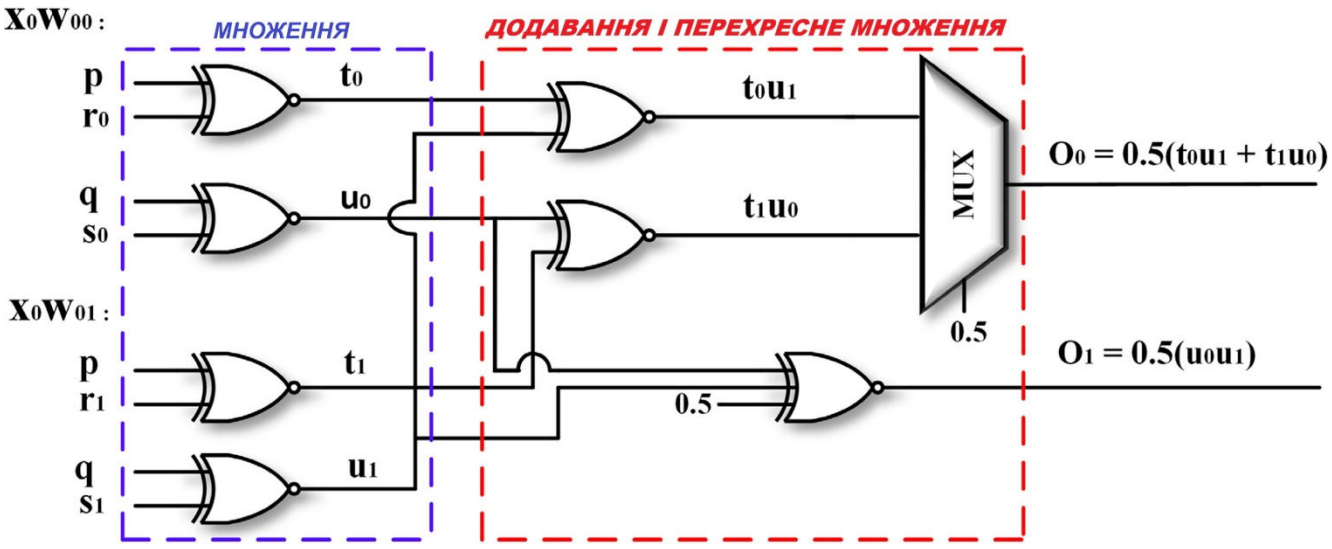


Рисунок 2.6 – Структура алгоритму додавання та множення ESL

Припустимо, що вхідне значення та вагові коефіцієнти задані у вигляді відповідних дробів: $x_0 = \frac{p}{q}$, $w_{00} = \frac{r_0}{s_0}$ та $w_{01} = \frac{r_1}{s_1}$. Перша частина структурної схеми, зображеної на рис. 6 (яка відповідає за операцію множення), виконує обчислення добутків $x_{00}w_{00}$ та $x_{00}w_{01}$ за допомогою пари спеціалізованих помножувачів на базі розширеної стохастичної логіки (ESL). Кожен такий ESL-помножувач апаратно складається з двох логічних вентилів XNOR, які паралельно перемножують чисельники та знаменники відповідних стохастичних бітових потоків. У другій частині схеми (яка реалізує ESL-додавання) знаменник результуючого значення y обчислюється з використанням тривходового вентиля XNOR. Одночасно з цим чисельник результату y формується за допомогою комбінації двох вентилів XNOR та одного мультиплексора (MUX), що дозволяє коректно виконати масштабоване підсумовування дробових значень.

При практичній апаратній реалізації нейронної мережі на базі методу ESL необхідно представляти кожні вхідні дані, синаптичні ваги та зміщення (biases) за допомогою двох окремих стохастичних потоків — p та q. Для генерації такого двійкового числа у форматі ESL застосовується спеціальний Алгоритм 1. На вхід цього алгоритму подається певне дійсне число (D_{in}). Далі, спираючись на його

абсолютне значення (модуль), система генерує два відповідні випадкові числа p та q, які формуватимуть дріб.

Процес генерації відбувається наступним чином: перш за все, згідно з рядками 5–8 алгоритму, генерується ненульове випадкове число в межах базового діапазону $[-1,+1]$.

Для першого випадку: якщо абсолютне значення вхідного числа D_{in} є більшим за 1, то згенероване значення r присвоюється змінній p (чисельнику), а значення q (знаменника) обчислюється шляхом ділення p на D_{in} (відповідно до формул 2.9–2.12).

У другому випадку: змінній q присвоюється значення r, а значення p обчислюється за допомогою безпосереднього множення q на D_{in} (відповідно до формул 2.13–2.16). Цей адаптивний математичний механізм гарантує, що стохастичні потоки чисельника та знаменника завжди залишатимуться в межах допустимого апаратного діапазону $[-1,+1]$, водночас дозволяючи системі коректно кодувати числа практично будь-якої величини.

У процесі інтеграції класичних моделей глибоких нейронних мереж (DNN) з апаратною базою стохастичних обчислень виникає фундаментальна проблема обмеження динамічного діапазону. Стандартне біполярне стохастичне кодування дозволяє оперувати значеннями виключно в межах закритого інтервалу $[-1,1]$, оскільки ймовірність появи біта фізично не може виходити за межі від 0 до 1. Однак під час інференсу нейронних мереж реальні значення синаптичних ваг (w), зміщень (biases) та активацій (x) часто перевищують ці апаратні обмеження. Для розв’язання цієї проблеми в архітектуру інтегрується метод розширеної стохастичної логіки (Extended Stochastic Logic, ESL), який представляє будь-яке дійсне число у вигляді математичного відношення двох незалежних стохастичних потоків — чисельника p та знаменника q. Запропонований Алгоритм 1 виконує функцію базового програмного енкодера (перетворювача), який адаптує числа з нескінченного діапазону до апаратних вимог стохастичного обчислювального ядра. Головне завдання алгоритму полягає у розкладанні вхідного дійсного числа

D_{in} на пару значень (p,q) таким чином, щоб забезпечити виконання умови $D_{in}=\frac{p}{q}$, при цьому жорстко гарантуючи, що обидва компоненти належатимуть допустимому апаратному діапазону: $p,q\in[-1,1]$. Процес перетворення здійснюється за наступним сценарієм:

Ініціалізація: Для заданого вхідного дійсного числа D_{in} алгоритм генерує випадкове, гарантовано ненульове дійсне число r у базовому діапазоні $[-1,1]$. • Масштабування великих значень ($|D_{in}|>1$): Якщо модуль вхідного числа перевищує одиницю, безпосередня подача такого числа на апаратні вентилі неможлива. У цьому випадку алгоритм призначає згенероване випадкове значення r чисельнику ($p=r$). Знаменник q розраховується аналітично за формулою:

$$q=\frac{p}{D_{in}}. \tag{2.23}$$

Оскільки $|D_{in}|>1$, а $|p|\le 1$, отримане значення знаменника q гарантовано потрапляє в діапазон $[-1,1]$. • Обробка дробових значень ($|D_{in}|\le 1$): Якщо вхідне число вже знаходиться в межах нормалізованого діапазону, застосовується обернений підхід для забезпечення варіативності стохастичного представлення. Змінній знаменника присвоюється випадкове значення ($q=r$), а чисельник обчислюється як:

$$p=q\cdot D_{in}. \tag{2.24}$$

Добуток двох чисел, модулі яких не перевищують одиницю, також гарантовано лежить у межах $[-1,1]$. У контексті загального пайплайну обробки даних розроблений алгоритм функціонує як сполучна ланка між програмною моделлю нейронної мережі та її апаратною реалізацією. Усі масиви вхідних даних (наприклад, пікселі зображень або сенсорні сигнали) та параметри навченої мережі (ваги) попередньо обробляються цим алгоритмом (подано далі як псевдокод).

Алгоритм 1

Algorithm 1 Gen_random (Din)

Input: Real Number (Din)

Output: Random Number (p, q)

```
1:  x = Din
2:  a = +1
3:  b = -1
4:  in = |x|
5:  While (r == 0) do
6:      r = a + (b-a) * rand(1,1)   /
7:  End While
8:  if (in > 1) then
9:      p = r
10:     q = p / x
11: else if (in <= 1) then
12:     q = r
13:     p = q * x
14: end if
```

Згенеровані дійсні значення p та q згодом передаються на апаратний рівень, де за допомогою компараторів та генераторів псевдовипадкових чисел на базі регістрів зсуву з лінійним зворотним зв'язком (LFSR) вони конвертуються у фізичні стохастичні бітові потоки. Далі ці потоки надходять на спеціалізовані ESL-помножувачі (побудовані на логічних вентилях XNOR) та суматори (на базі мультиплексорів MUX) для виконання базових операцій множення з накопиченням (MAC), що формують обчислювальну основу стохастичного нейрона.

Для забезпечення переходу від класичних дійсних чисел до формату розширеної стохастичної логіки (ESL) було розроблено відповідну програмну модель мовою Python. Базовим компонентом цієї моделі є функція generate_esl_streams, яка виконує роль первинного енкодера. Її головне завдання — розкласти будь-яке дійсне вхідне число на два дробові компоненти: чисельник p та знаменник q, при цьому жорстко гарантуючи, що обидва значення не

виходитимуть за межі апаратного діапазону [−1,1]. Процес програмного перетворення реалізовано за наступним алгоритмом: 1. Ініціалізація та генерація базового числа: Функція приймає на вхід дійсне число din (яке зберігається у змінній x). Спочатку алгоритм визначає абсолютне значення вхідного числа in_val та встановлює межі для стохастичних потоків (a = 1.0, b = -1.0). Далі за допомогою вбудованого модуля random у циклі while генерується псевдовипадкове дійсне число r у діапазоні [−1,1]. Цикл гарантує, що згенероване значення строго не дорівнює нулю, що є критично важливим для уникнення помилок ділення на нуль на наступних етапах.

```
import random
def generate_esl_streams(din: float) -> tuple[float, float]:
    x = din
    a = 1.0
    b = -1.0
    in_val = abs(x)
    r = 0.0
    while r == 0.0:
        r = a + (b - a) * random.random()
    p = 0.0
    q = 0.0

    if in_val > 1.0:
        p = r
        q = p / x
    elif in_val <= 1.0:
        q = r
        p = q * x

    return p, q
input_number = 2.5
```

```
p_val, q_val = generate_esl_streams(input_number)
print(f"Вхідне число: {input_number}")
print(f"Згенерований чисельник (p): {p_val}")
print(f"Згенерований знаменник (q): {q_val}")
print(f"Перевірка (p/q): {p_val / q_val}")
```

Формування компонентів p та q: Далі логіка розділяється залежно від величини модуля вхідного числа, щоб уникнути переповнення під час розрахунків: Масштабування великих значень (in_val > 1.0): якщо вхідне число виходить за межі допустимого діапазону, алгоритм присвоює згенероване випадкове значення чисельнику (p = r). Знаменник при цьому обчислюється як відношення чисельника до початкового вхідного числа (q= p/x). Оскільки модуль чисельника менший за одиницю, а знаменника — більший, отримане значення q також гарантовано потрапляє в діапазон $[-1,1]$.

Обробка нормалізованих значень (in_val <= 1.0): Якщо вхідне число вже знаходиться в межах робочого діапазону, застосовується обернений підхід. Згенероване випадкове значення присвоюється знаменнику (q = r), а чисельник обчислюється шляхом безпосереднього множення (p = q * x). Добуток двох чисел, модулі яких не перевищують одиницю, математично не може вийти за межі $[-1,1]$. У результаті функція повертає кортеж із двох дійсних чисел (p, q).

Запропонована програмна реалізація успішно перетворює значення з нескінченного динамічного діапазону на пару нормалізованих імовірностей. На наступних етапах симуляції ці значення використовуються для фізичної генерації стохастичних бітових послідовностей (за допомогою компараторів та LFSR-регістрів), які безпосередньо подаються на входи апаратних вентилів системи.

2.4 Програмна реалізація стохастичних логічних елементів

Цифро-імовірнісний перетворювач (Digital Probability Converter, DPC) здійснює перетворення дійсного числа у стохастичну бітову послідовність. Його

апаратну структуру наведено на рис. 2.7. Принцип роботи полягає в тому, що під час кожного тактового імпульсу двійкове представлення дійсного числа порівнюється з випадковим числом (Random Number, RN), яке генерується за допомогою регістра зсуву з лінійним зворотним зв’язком (LFSR).

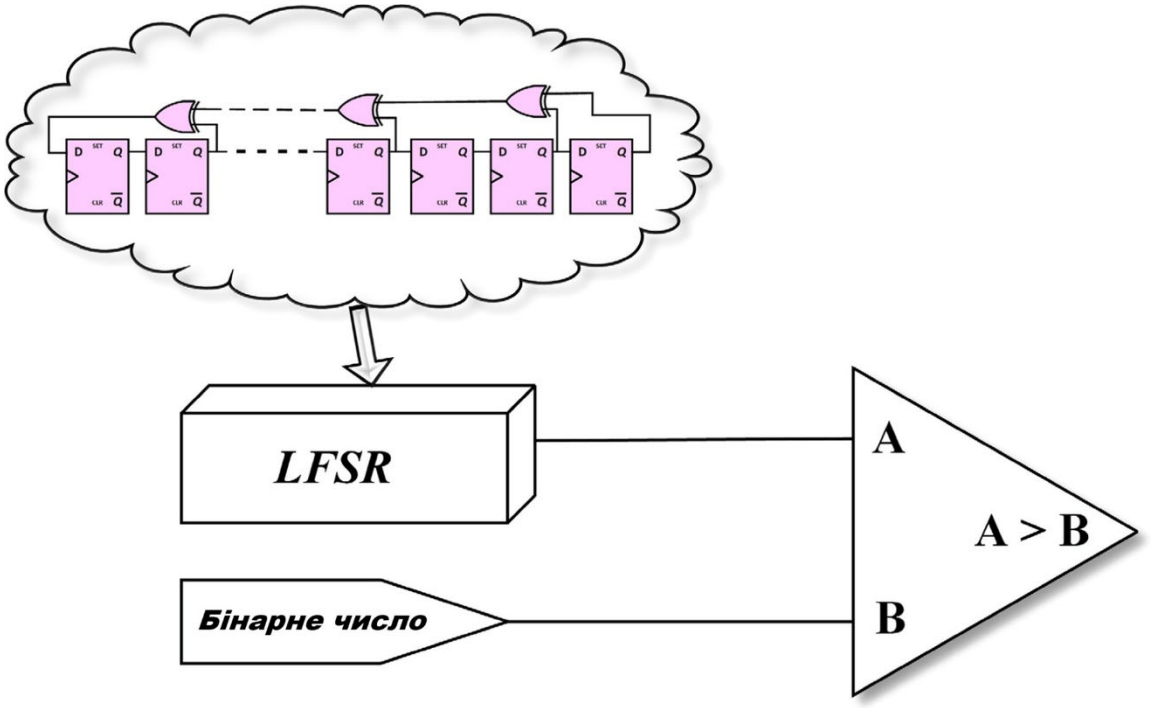


Рисунок 2.7 – Цифровий перетворювач ймовірності (ЦПІ).

Якщо згенероване значення RN перевищує вхідне двійкове число, на виході DPC формується логічна 1; у протилежному випадку генерується 0. Згенерований стохастичний потік надходить на вхід логічного вентиля XOR, що є складовою процесорного елемента (Processing Element, PE), структуру якого показано на рис. 2.8. Апаратна база PE включає реверсивний лічильник (Up/Down counter, U/D), який ініціалізується випадковим початковим значенням (seed). Поточне значення цього U/D лічильника інкрементується (збільшується) або декрементується (зменшується) залежно від того, чи дорівнює вхідний біт 1 або 0 відповідно.

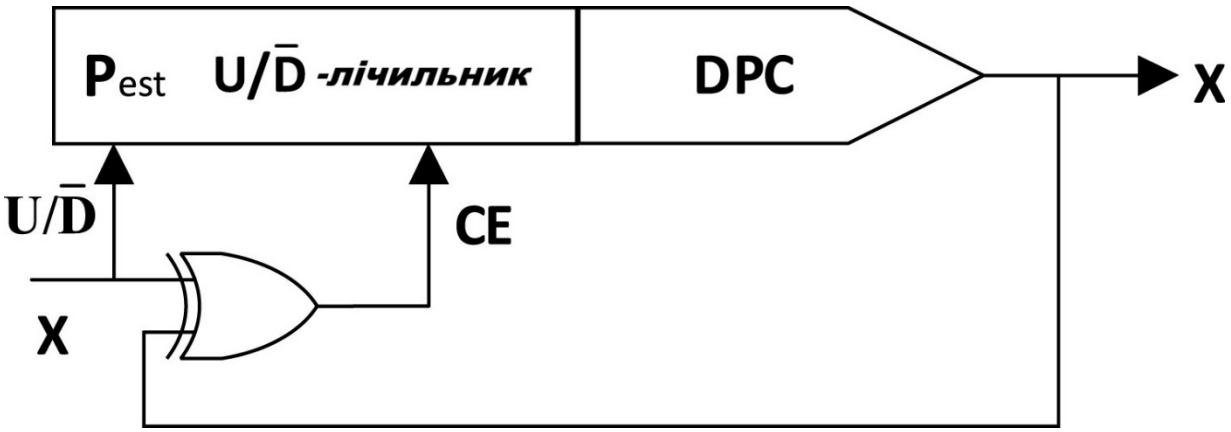


Рисунок 2.8 – Нормальний оцінювач ймовірності (PE).

Після того, як до процесорного елемента надійде достатня кількість бітів, накопичене значення в реверсивному лічильнику можна інтерпретувати як відновлене оригінальне двійкове число DPC. Збіжність процесорного елемента (PE convergence) вважається досягнутою тоді, коли порівняння вхідної стохастичної послідовності (X_{in}) та вихідної послідовності (X_{out}) призводить лише до незначних коливань поточного значення реверсивного лічильника. Як також зазначається в [16], фінальне зареєстроване число в U/D лічильнику після досягнення збіжності в ідеальному випадку дорівнює оригінальному двійковому числу, представленому на рис. 2.8.

2.5 Запропонований оцінювач ймовірності (PE)

Імовірнісний оцінювач є надзвичайно важливим елементом у структурі нейронних мереж, що базуються на стохастичних обчисленнях. Для проходження через функцію активації (наприклад, ReLU) стохастичні потоки повинні бути перетворені у двійкове значення саме за допомогою імовірнісного оцінювача. Відповідно, апаратна архітектура нейронної мережі вимагає наявності тисяч таких модулів. Як наслідок, питання енергоспоживання, займаної площі на кристалі, швидкодії та точності цього елемента є надзвичайно актуальними. Незважаючи на свою апаратну простоту, класичний процесорний елемент (PE) має суттєвий недолік — низьку швидкість збіжності. Припустимо, що початкове значення (seed)

дорівнює S ($0<S<1$), а вхідна послідовність представляє значення B ($0<B<1$). Нехай еквівалентні дійсні числа для B та S позначаються як P_B та P_S відповідно. Ймовірність того, що відповідні стохастичні послідовності будуть нерівними між собою (P_{neq}), обчислюється за допомогою рівняння (2.25):

$$P_{neq} = P_S(1-P_B)+(1-P_S)P_B. \tag{2.25}$$

Без втрати загальності припустимо, що $P_B = P_S + \Delta P$. Тоді ймовірність P_{neq} можна виразити через рівняння (34) та (35):

$$\begin{aligned} P_{neq} &= P_S(1-P_S-\Delta P)+(1-P_S)(P_S+\Delta P)= \\ &= P_S(1-P_S)+\Delta P(1-2P_S). \end{aligned} \tag{2.26}$$

Коли значення X_{in} не дорівнює X_{out} у поточному тактовому циклі, для зменшення величини ΔP вміст процесорного елемента має бути збільшений. Оскільки ймовірність появи логічної одиниці у послідовності X_{in} становить P_S, інкрементація значення процесорного елемента відбуватиметься з імовірністю P_{inc}, яка розраховується за наступною формулою:

$$P_{inc} = P_S P_{neq} \tag{2.27}$$

У такому випадку значення ΔP буде зменшено (нове значення становитиме $\Delta P - \frac{1}{2^{20}}$), після чого описаний процес повторюється. З іншого боку, якщо поточний біт вхідної послідовності X_{in} дорівнює нулю, значення ΔP, навпаки, збільшиться на величину $\frac{1}{2^{20}}$, а збіжність віддалятиметься. У найкращому випадку, щоразу, коли значення потоків X_{in} та X_{out} відрізняються, умова інкрементації виконуватиметься, і вміст процесорного елемента P_S наближатиметься до цільового значення P_B. Це призводить до зменшення ΔP, що, у свою чергу, зумовлює зниження як імовірності нерівності P_{neq}, так і ймовірності інкрементації P_{inc}. Для наочного прикладу припустимо, що початкове значення $P_S = 0{,}1$, а $\Delta P = 0{,}8$ (відповідно $P_B = 0{,}9$). Тоді розрахункові ймовірності становитимуть:

$$P_{neq} = 0,09 + 0,8(0,8) = 0,73$$

$$P_{inc} = 0,073$$

Як наслідок, значення ΔP зменшиться до величини:

$$\Delta P = 0,8 - \frac{1}{2^{20}} \approx 0,799999$$

Використовуючи це оновлене значення ΔP, отримуємо нові ймовірності: $P_{neq} = 0,729999$ та $P_{inc} = 0,0729999$. На цьому етапі, якщо умова інкрементації знову задовольняється, то ΔP продовжить зменшуватися до значення $0,799998$. Після виконання операції зменшення ΔP понад 2^{20} (більше ніж 10^6) разів, система досягне повної збіжності. У стані збіжності, коли $\Delta P = 0$, фінальні значення ймовірностей становитимуть $P_{neq} = 0,1$ та $P_{inc} = 0,01$.

Якщо реверсивний лічильник може збільшувати або зменшувати своє значення більше ніж на одну одиницю за такт, загальна кількість кроків зменшиться, однак точність кінцевого результату суттєво погіршиться. З іншого боку, якщо розрядність реверсивного лічильника зменшується, величина ΔP збільшується, що забезпечує швидше досягнення збіжності. Наприклад, при використанні 8-бітного реверсивного лічильника $\Delta P = \frac{1}{2^8} \approx 0,003906$, що значно перевищує значення ΔP для 20-бітного лічильника. Проте оцінка ймовірності за допомогою 20-бітного реверсивного лічильника є набагато точнішою. Для математичного доведення наявності великої кількості циклів збіжності у класичному процесорному елементі припустимо, що початкове значення лічильника процесорного елемента N_S є меншим за N_B (де N_B — це L_{Reg}-бітне двійкове число, яке виступає джерелом послідовності X). У такому випадку лічильник процесорного елемента дійсно інкрементується на одиницю, коли значення потоків X та X' не збігаються. Різниця між N_B та N_S визначається за допомогою рівняння поданого далі:

$$\delta = N_B - N_S. \qquad (2.28)$$

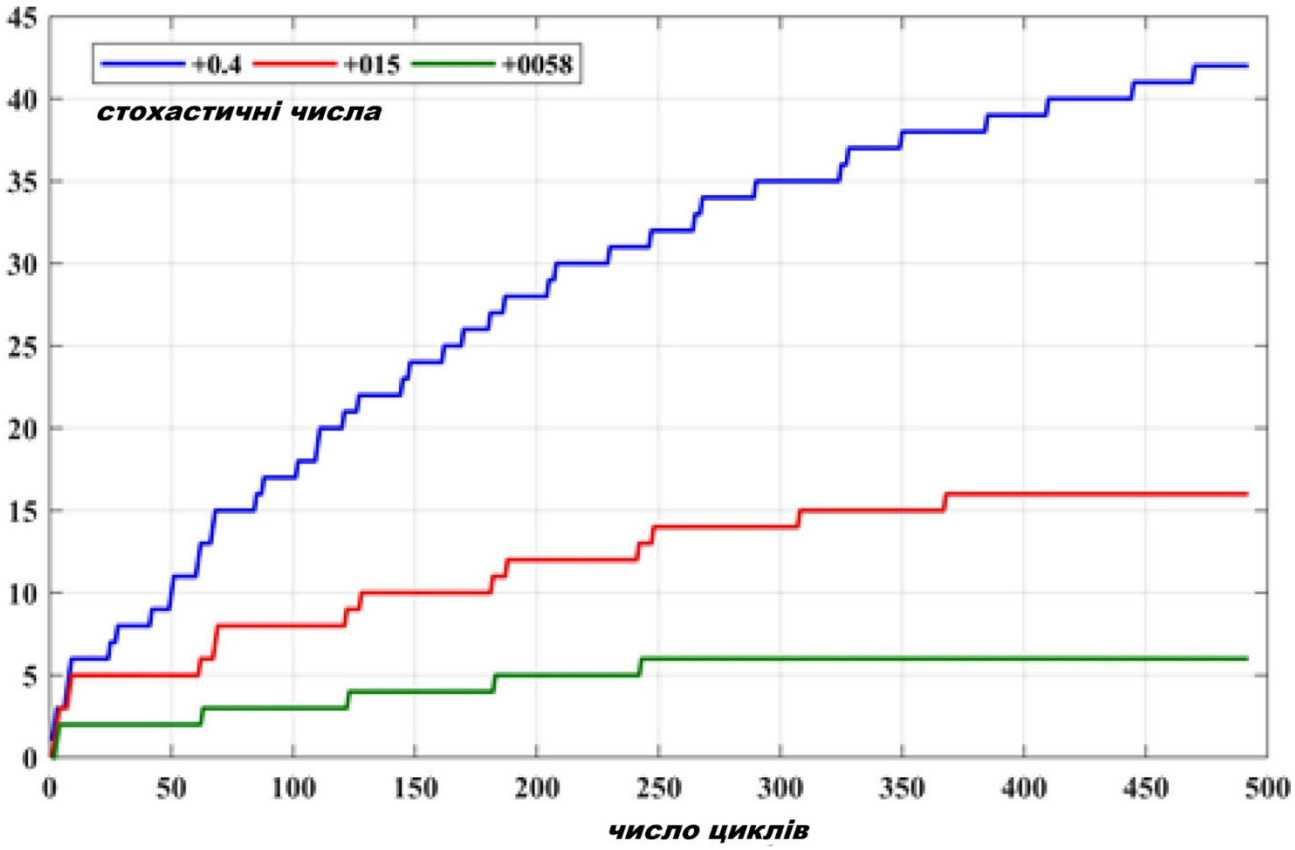


Рисунок 2.9 – Три концепції збіжності 8-бітних чисел у традиційному процесорному елементі.

Ймовірність генерації нерівних бітів у послідовностях X та X' становить P_{neq}. Оскільки поява нерівних бітів підпорядковується розподілу Бернуллі, одна успішна подія відбувається за $1/P_{neq}$ тактових циклів. Отже, для досягнення збіжності класичний процесорний елемент потребує кількості тактових циклів, що визначається за формулою (38):

$$n_{Conv-cycles} = \frac{\delta}{P_{neq}}. \tag{2.29}$$

Якщо припустити, що L_{Reg} є розрядністю лічильника процесорного елемента, а також розрядністю регістрів зсуву з лінійним зворотним зв'язком, то величину $N_{Conv-cycles}$ можна представити наступним чином:

$$N_S = P_S \times 2^{L_{Reg}}, \tag{2.30}$$

$$N_B = P_B \times 2^{L_{Reg}}, \tag{2.31}$$

$$\delta = (P_B - P_S) \times 2^{L_{Reg}}, \tag{2.32}$$

$$N_{Conv-cycles} = \frac{\delta}{P_{neq}} = \frac{(P_B - P_S)}{P_B + P_S - 2P_B P_S} \times 2^{L_{Reg}}. \tag{2.33}$$

Перепишемо вираз для P_{neq} з рівняння (2.32) у вигляді (2.33):

$$P_{neq} = P_B(1-P_S) + P_S(1-P_B) = P_B + P_S - 2P_B P_S. \tag{2.34}$$

звідки

$$P_{neq} = (P_B - P_S) + 2P_S - 2P_B P_S = \delta + 2P_S(1-P_B). \tag{2.35}$$

З рівняння (2.31) можна вивести наступну залежність :

$$P_B = P_S + \frac{\delta}{2^{L_{Reg}}}. \tag{2.36}$$

Отже, маємо:

$$P_{neq} = \frac{\delta}{2^{L_{Reg}}} + 2P_S\left(1 - P_S - \frac{\delta}{2^{L_{Reg}}}\right)$$

звідки:

$$P_{neq} = \frac{\delta}{2^{L_{Reg}}} - \frac{2P_S \times \delta}{2^{L_{Reg}}} + 2P_S(1-P_S), \tag{2.37}$$

$$P_{neq} = \frac{\delta}{2^{L_{Reg}}}(1-2P_S) + 2P_S(1-P_S). \tag{2.38}$$

Після цього ми можемо переформулювати рівняння (2.38) у вигляді виразу:

$$N_{Conv-cycles} = \frac{\delta}{\frac{\delta}{2^{L_{Reg}}}(1-2P_S) + 2P_S(1-P_S)} \times 2^{L_{Reg}}. \tag{2.39}$$

Для випадку, коли $P_S = 0$, з рівняння (46) отримуємо такий результат:

$$N_{cycles} = \frac{\delta}{\frac{\delta}{2^{L_{Reg}}}} \times 2^{L_{Reg}} = 2^{(2 \times L_{Reg})}. \tag{2.40}$$

Варто зазначити, що для такого значення P_S кількість тактових циклів, необхідних для досягнення збіжності, не залежить від величини δ. Для випадку, коли $P_S = 0{,}5$, підстановка у рівняння (46) дає наступний результат (48):

$$N_{cycles} = 2\delta \times 2^{L_{Reg}} = 2^{(L_{Reg}+1)} \times \delta;\ \delta_{max} = 2^{\frac{L_{Reg}}{2}} \tag{2.41}$$

де

$$N_{cycles} = 2^{\left(\frac{3}{2} \times L_{Reg}\right)} \quad . \tag{2.42}$$

Для випадку, коли $P_S = 0,9$, з рівняння (2.41) отримуємо вираз :

$$N_{Conv-cycles} = \frac{\delta}{\frac{\delta}{2^{L_{Reg}}}(-0,8) + 0,18} \times 2^{L_{Reg}};\ \delta_{max} = 0,1 \times 2^{L_{Reg}}, \tag{2.43}$$

або

$$N_{Conv-cycles} = \frac{0,1 \times 2^{L_{Reg}}}{0,1} \times 2^{L_{Reg}} = 2^{(2 \times L_{Reg})}. \tag{2.44}$$

На основі цих обчислень можна зробити висновок, що $N_{Conv-cycles}$ є пропорційним до $2^{L_{Reg}}$, що є дуже великим числом (наприклад, $L_{Reg} = 20$ призводить до щонайменше 1048576 циклів). На рис. 2.9 та рис. 2.10 показано кількість тактових циклів, необхідних для досягнення збіжності, для трьох різних 20-бітних та трьох 8-бітних чисел. Для всіх симуляцій початкові значення встановлено рівними нулю. Відповідно до результатів, необхідний час збіжності зростає зі збільшенням різниці між початковим значенням та двійковим числом. Для 20-бітних чисел мінімальний час збіжності становить понад 6 мільйонів тактових циклів.

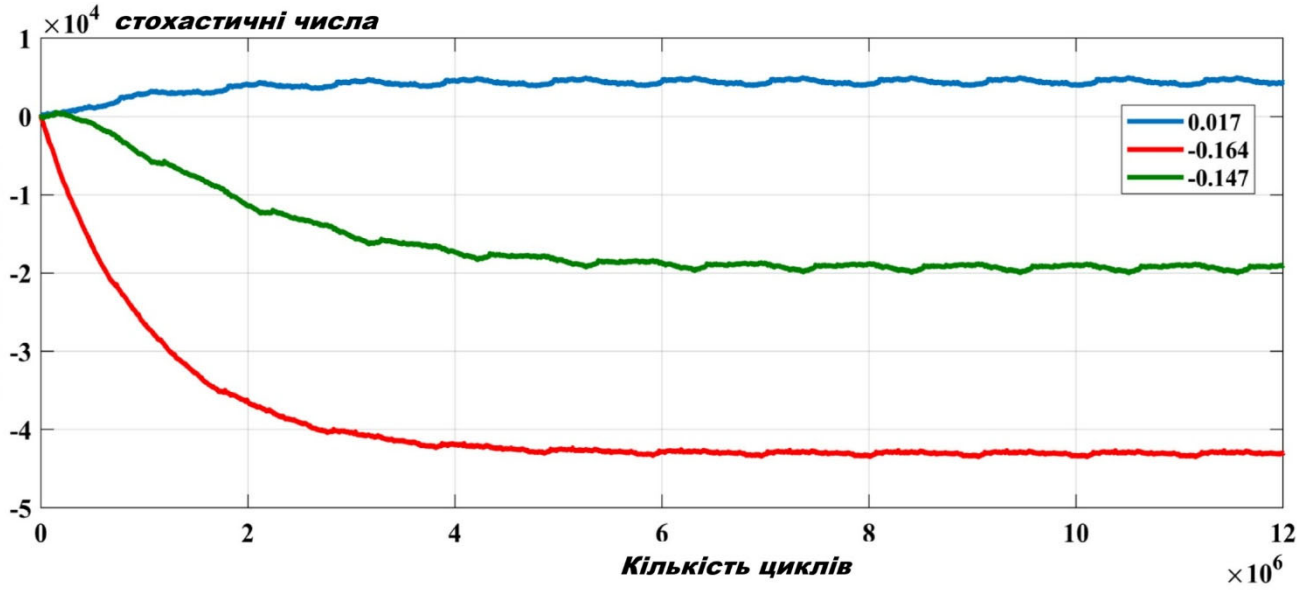


Рисунок 2.10 – Три концепції збіжності 20-бітних чисел у традиційному процесорному елементі.

Проблема повільної збіжності процесорного елемента досліджувалася у відповідній літературі, де зазначається, що кількість тактових циклів, необхідних для збіжності, варіюється від 40 000 (для ймовірності, встановленої на рівні 0,5) до 7 200 000 (для ймовірності, встановленої на рівні 1). Коли вихідне двійкове число значно відрізняється від початкового значення реверсивного лічильника, для досягнення збіжності потрібні мільйони тактових циклів, а середня довжина послідовності, необхідна для класичного процесорного елемента, становить 3 958 900 бітів. Для розв'язання цієї проблеми було запропоновано процесорний елемент на основі бінарного пошуку. У ньому замість того, щоб починати з єдиного початкового значення та поступово збільшувати вміст реверсивного лічильника, оптимальне початкове значення визначається за допомогою алгоритму бінарного пошуку. Використання такого новітнього методу дозволяє досягти збіжності в середньому за 96 939 тактових циклів.

Якщо довжина стохастичної послідовності встановлена на рівні N бітів, то дві послідовності, які містять p та $p+1$ одиничних бітів, представлятимуть два дійсних числа R_1 та R_2 згідно з рівняннями (2.45) та (2.46). Різниця між R_1 та R_2

становить $\frac{2}{N}$, що означає, що мінімальний крок представлення (роздільна здатність) для дійсного числа у біполярній формі дорівнюватиме $\frac{2}{N}$. Наприклад, для значень $N=32$, 1024 та 1048576 цей крок становить $0{,}0625$, $0{,}001953$ та $0{,}0000019076$ відповідно.

$$R_1 = \frac{2p - N}{N} \tag{2.45}$$

звідки:

$$R_2 = \frac{2p + 2 - N}{N}. \tag{2.46}$$

і

$$R_2 - R_1 = \frac{2}{N}. \tag{2.47}$$

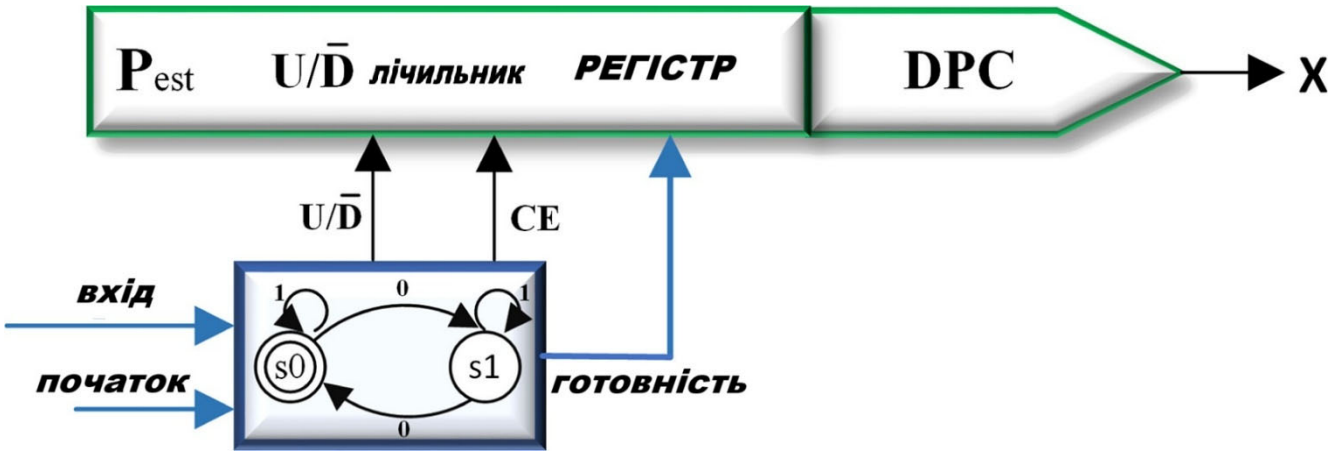


Рисунок 2.11 – Запропонована модель процесорного елемента.

Припустимо, що необхідно представити дробове число $r = 0{,}1$ за допомогою зазначених стохастичних послідовностей. Тоді відповідна абсолютна похибка становитиме $2{,}5\%$, $0{,}039\%$ та $0{,}00038\%$ (для наведених вище значень N). Отже, збільшення довжини стохастичної послідовності призводить до зменшення похибки представлення, однак це відбувається ціною зниження швидкості збіжності. Крім того, навіть для помірної довжини послідовності (наприклад,

$N=128$) абсолютна похибка є достатньо малою. На рис. 2.11 показано порівняння результатів для різних бітових потоків у запропонованому процесорному елементі.

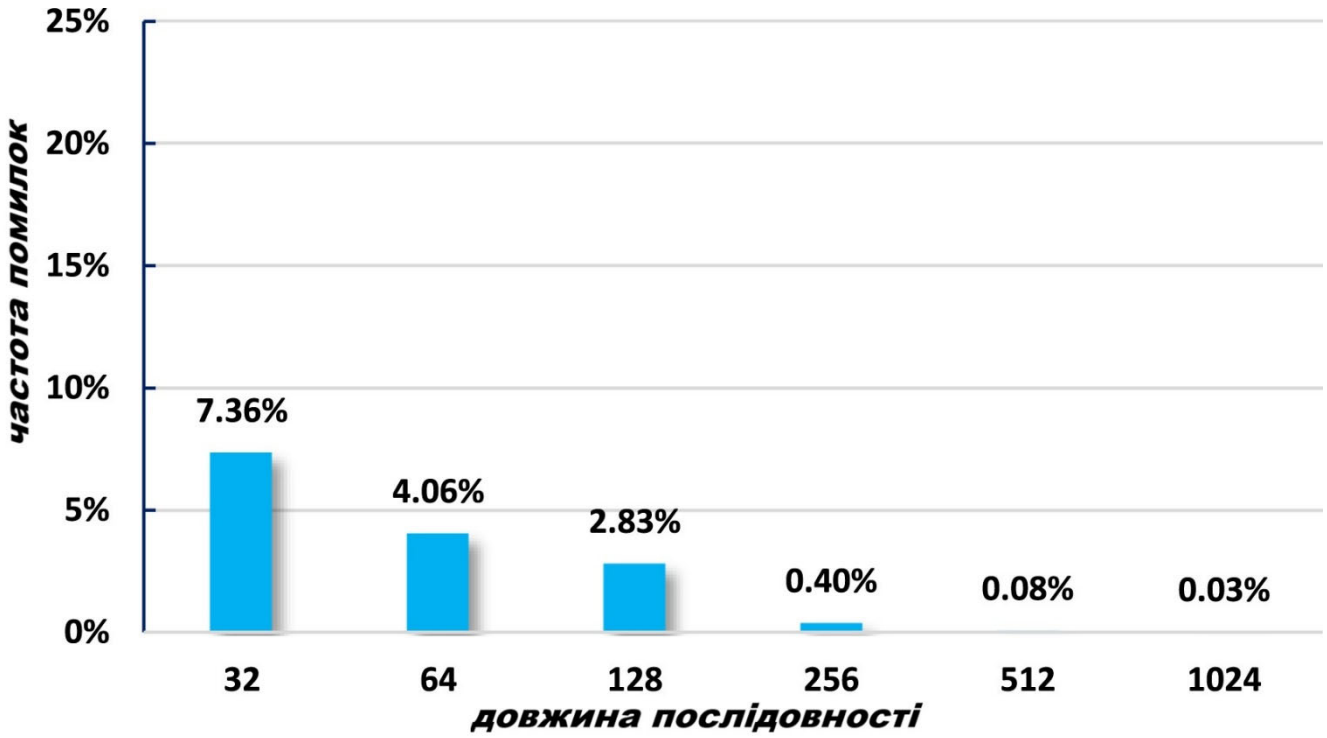


Рисунок 2.12 – Точність роботи запропонованого процесорного елемента.

З іншого боку, випадкова послідовність, згенерована цифро-імовірнісним перетворювачем, демонструє стаціонарну поведінку. Це означає, що перша N-бітна підпослідовність згенерованого стохастичного потоку представляє те саме число, що й $10\times N$-бітна підпослідовність, хоча і з вищезгаданою похибкою відкидання. Ці два спостереження стали мотивацією для розробки запропонованої архітектури імовірнісного оцінювача, яку проілюстровано на рис 2.11. У ній петля зворотного зв’язку та механізм порівняння вхідної і вихідної послідовностей замінені скінченним автоматом. Як наслідок, концепція збіжності повністю усувається, а оцінка ймовірності вхідної послідовності здійснюється виключно шляхом підрахунку нульових та одиничних бітів у N-бітній підпослідовності стохастичного потоку.

На відміну від класичного процесорного елемента, запропонована архітектура є активною протягом N тактових циклів. Початок активного стану

визначається сигналом запуску; по завершенню активного стану активується сигнал готовності, який, у свою чергу, слугує сигналом запуску для наступного процесорного елемента. Скінченний автомат має три стани: очікування, активний стан та стан виведення. У стані очікування сигналам дозволу рахунку та готовності присвоюється логічний нуль, і скінченний автомат залишається в цьому стані до моменту появи сигналу запуску. В активному стані внутрішній лічильник починає рахувати від 0 до N, а сигналу дозволу рахунку присвоюється логічна одиниця. У кожному тактовому циклі реверсивний лічильник збільшує або зменшує своє значення залежно від поточного біта вхідної стохастичної послідовності.

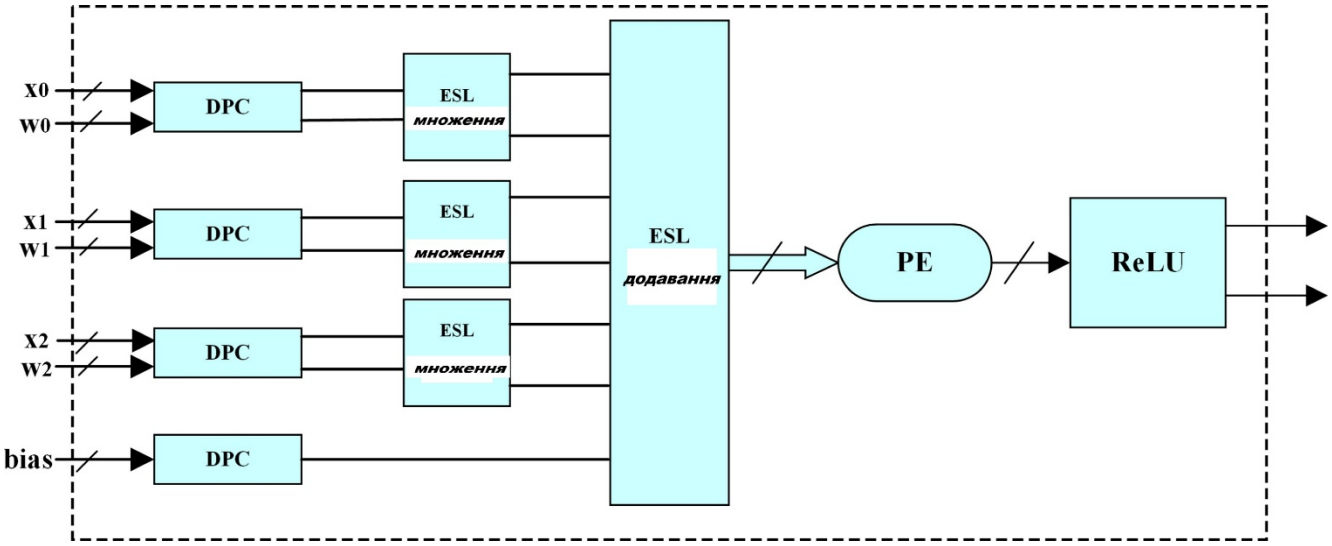


Рисунок 2.13 – Однонейронна структура (ESL) на базі розширеної стохастичної логіки.

Коли внутрішній лічильник досягає значення $N-1$, сигнал дозволу рахунку скидається в нуль, сигнал готовності змінюється на одиницю, а сам внутрішній лічильник обнуляється. Після цього скінченний автомат переходить у стан виведення, під час якого вміст реверсивного лічильника переноситься до першого регістра, що виконує роль двійкового числа для цифро-імовірнісного перетворювача. Вміст реверсивного лічильника скидається до нуля, і скінченний автомат повертається у стан очікування. Цифро-імовірнісний перетворювач залишається активним протягом наступних N тактових циклів для генерації нової N-бітної стохастичної послідовності для наступного каскаду обчислень.

Структуру окремого нейрона на базі розширеної стохастичної логіки зображено на рис. 2.13. Згідно з цією схемою, вхідні дані та вагові коефіцієнти спочатку подаються на цифро-імовірнісні перетворювачі для їх перетворення з дійсних чисел у бінарні стохастичні потоки. Потім, відповідно до рівняння (3), вони проходять через помножувач на базі розширеної стохастичної логіки, який складається з пари вентилів виключне АБО-НІ (XNOR). На наступному етапі, для виконання операції додавання з рівняння (3), суми результатів множення та сума зі значеннями зміщення отримуються з блоку додавання, зображеного на рисунку 6. Нарешті, запропонований модуль процесорного елемента застосовується для отримання ймовірності згенерованого бітового потоку. Якщо отримана ймовірність є позитивною, вихідне значення функції активації ReLU залишається незмінним; в іншому випадку вихідне значення встановлюється рівним нулю. Раніше, у підрозділі 2.1 та рівняннях (2.8)–(2.15), обговорювалася проблема зменшення масштабу при стохастичному біполярному додаванні. Для вирішення цієї проблеми у відповідній літературі пропонується в мультиплексорі з n входами масштабувати вхідне значення кінцевого результату до $\frac{z}{n}$ замість z. Як наслідок, для отримання правильної відповіді від функції активації, результат роботи мультиплексора необхідно збільшити в n разів, або ж безпосередньо помножити скінченний автомат з K станами на n.

Ще один додатковий скінченний автомат (необхідний через лінійне перетворення коефіцієнта підсилення), один вентиль XNOR, а також генератор додаткового біполярного стохастичного потоку вимагаються для виконання операції множення. Це, у свою чергу, спричиняє значні апаратні витрати. Накопичувальний паралельний лічильник використовується в деяких дослідженнях для подолання втрати точності, яка може призвести до некоректних результатів обчислень. Однак нейрон, що базується на такому лічильнику, має дуже низьку точність. Крім того, зі збільшенням розмірності вхідних даних споживана потужність, енергія та займана площа на кристалі зростають лінійно.

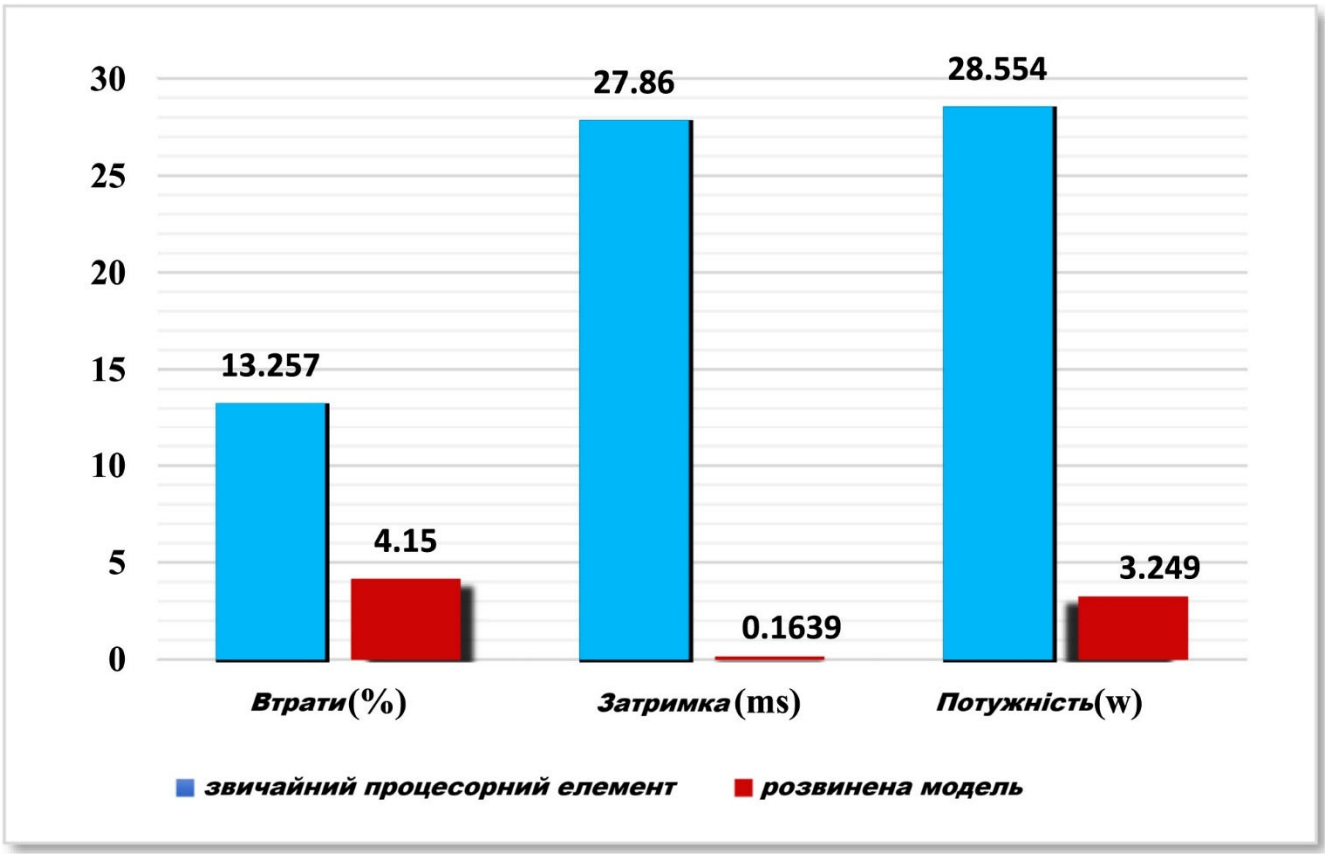


Рисунок 2.14 – 10-бітове порівняння окремих нейронів як традиційної, і розробленої структури.

У запропонованому нами процесорному елементі проблема зменшення масштабу вирішується наступним чином. Початкове значення реверсивного лічильника дорівнює нулю. N -бітна послідовність біполярного суматора (out_{MUX}) містить p_a одиничних бітів та $N - p_a$ нульових бітів, що відповідає дійсному значенню, рівному $\frac{p_a - N}{N}$. Це значення становить половину від очікуваного. Коли вся N-бітна послідовність надходить до реверсивного лічильника, кінцеве значення дорівнюватиме різниці між кількістю одиничних та нульових бітів:

$$V_{final} = p_a - (N - p_a) = 2p_a - N. \quad (2.48)$$

Якщо вміст процесорного елемента перетворюється на стохастичну послідовність за допомогою цифро-імовірнісного перетворювача, то отримана послідовність представлятиме дійсне число, яке вдвічі перевищує out_{MUX} відповідно до рівняння

(2.48). Отже, запропонований імовірнісний оцінювач процесорного елемента успішно вирішує проблему зменшення масштабу.

$$V = \frac{(2p_a - N) - N}{N} = \frac{2p_a - 2N}{N} = 2 \times \left(\frac{p_a - N}{N} \right) = 2 \times out_{MUX}. \qquad (2.49)$$

Затримка, точність та енергоспоживання класичної та запропонованої 10-бітної однонейронної структури на базі розширеної стохастичної логіки порівнюються на рис. 2.14. Окремий нейрон містить два входи, відповідні вагові коефіцієнти та одне значення зміщення. Результати демонструють підвищення точності на $9{,}107\%$, а також зниження енергоспоживання та затримки на $25{,}305\%$ і $27{,}696\%$ відповідно.

2.6 Реалізація програмної платформи на програмованій логічній інтегральній схемі

Останньою метою нашого дослідження була реалізація багатошарового персептрона (MLP) на базі стохастичних обчислень на кристалі ПЛІС (FPGA). ПЛІС мають обмежені апаратні ресурси та лімітовану кількість входів і виходів (I/O). Це спонукало нас до пошуку такого підходу до реалізації, який би максимально раціонально (із «жадібною» стратегією) використовував порти вводу-виводу, апаратні ресурси та займану площу кристала, паралельно з оптимізацією точності, швидкодії, енергоспоживання та затримки. Ми зосередилися на реалізації мережі MLP для класифікації зображень із набору даних MNIST. Модифікований Національний інститут стандартів і технологій (MNIST) розробив стандартний набір напівтонових зображень рукописних цифр розміром 28*28 (кожне зображення містить 784 пікселі), який екстенсивно використовується для оцінювання ефективності апаратних реалізацій глибоких нейронних мереж (DNN) [66, 67].

Запропонована нами реалізація багатошарового персептрона отримує значення пікселів, вагові коефіцієнти та значення зміщення через три канали. Зважаючи на структуру розширеної стохастичної логіки, кожен канал складається з пари портів, кожен з яких містить BW виводів (де BW — це розрядність значень

пікселів). Ми позначили порти p та q для вхідних даних, r та s для вагових коефіцієнтів, а k та z для значень зміщення.

Спочатку ми натренували багатошаровий персептрон у середовищі Python, використовуючи тестовий та навчальний набори даних; потім для кожного зображення з набору MNIST відповідні значення пікселів, вагові коефіцієнти та значення зміщення були згенеровані у вигляді BW-бітних двійкових чисел за допомогою MATLAB. Згенеровані значення були записані у текстові файли та зчитані тестовим стендом проекту на базі Verilog.

Припускаючи, що BW дорівнює 8, кожна стохастична операція та обчислення вимагали періоду у 255 тактових циклів, відповідно до описів у попередньому підрозділі. На рис. 2.15 показано загальний алгоритм роботи нашої розробки. Ми розглянули дві структури, A та B, з конфігурацією нейронів 784-200-100-10 та 784-100-200-10 відповідно.

Отже, у реалізованому багатошаровому персептроні було чотири різні шари. Обчислення в кожному шарі починаються, коли відповідний сигнал дозволу встановлюється в логічну одиницю. По завершенню обчислень шар встановлює сигнал готовності в 1, а власний сигнал дозволу скидає в 0. У цьому стані шар очікує, поки його сигнал дозволу знову не буде встановлений в 1 попереднім шаром. На початковому етапі задаються початкові значення для внутрішніх лічильників, регістрів та керуючих сигналів.

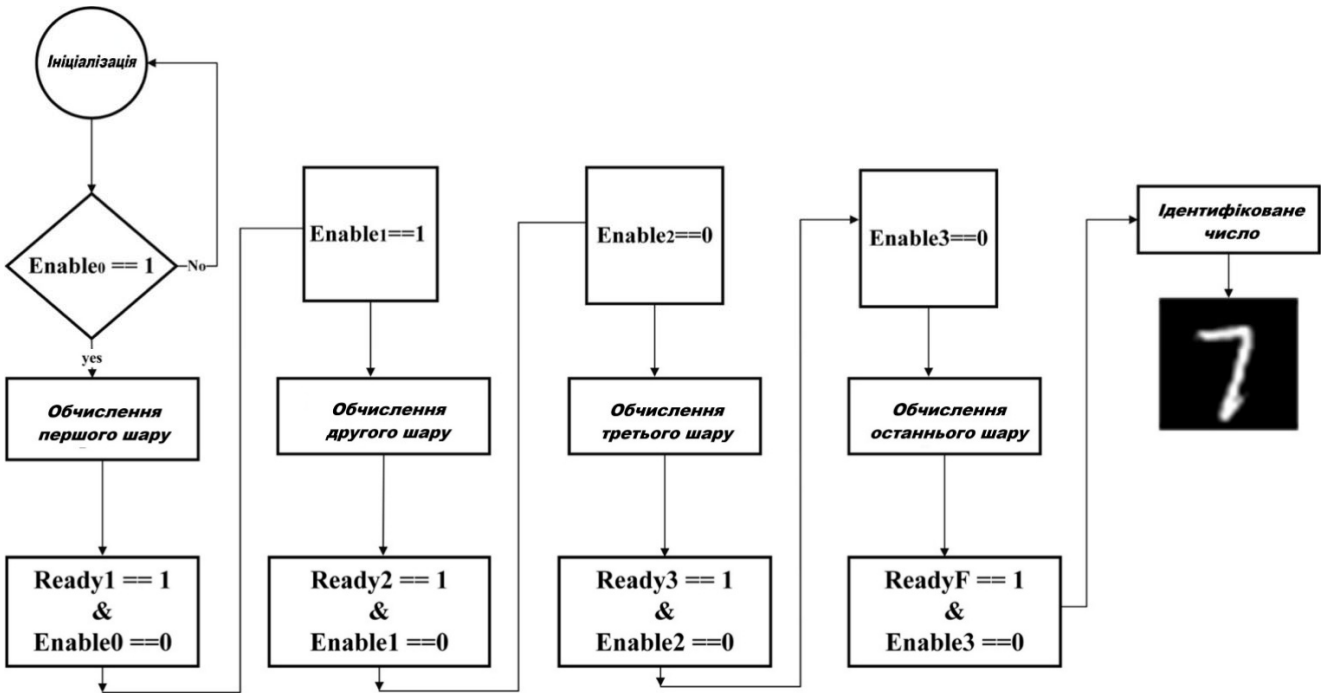


Рисунок 2.15 – Блок-схема передачі сигналів між рівнями

Коли значення першого пікселя надходить до першого шару, воно має бути помножене на кожен ваговий коефіцієнт, що відповідає зв'язкам між першим входом та всіма нейронами першого шару (W_{0i}, де $1 < i < 784$). Для кожного нейрона отриманий результат записується в накопичувач. Значення другого пікселя множиться на вагові коефіцієнти W_{1i} $(1 < i < 784)$ для кожного нейрона в першому шарі, після чого ці значення додаються до попередніх значень, що вже зберігаються в накопичувачах. Цей процес повторюється доти, доки всі 784 значення пікселів не надійдуть до ПЛІС.

Коли всі операції множення на вагові коефіцієнти та додавання завершуються, значення з накопичувачів підсумовуються зі значеннями зміщення. Обчислення в першому шарі закінчуються розрахунком вихідного значення функції активації ReLU для кожного нейрона. Після завершення процесу обробки в першому шарі накопичувачі починають виконувати роль джерела вхідних даних для другого шару, і описаний вище алгоритм застосовується для обчислення вихідних результатів другого шару. Варто зазначити, що для кожного значення пікселя відповідні 784 вагові коефіцієнти надходять до ПЛІС послідовно.

Наприклад, для значення першого пікселя спочатку до ПЛІС надходить ваговий коефіцієнт w_{00} . Потім, через 256 тактових циклів, надходить w_{01}, і так далі.

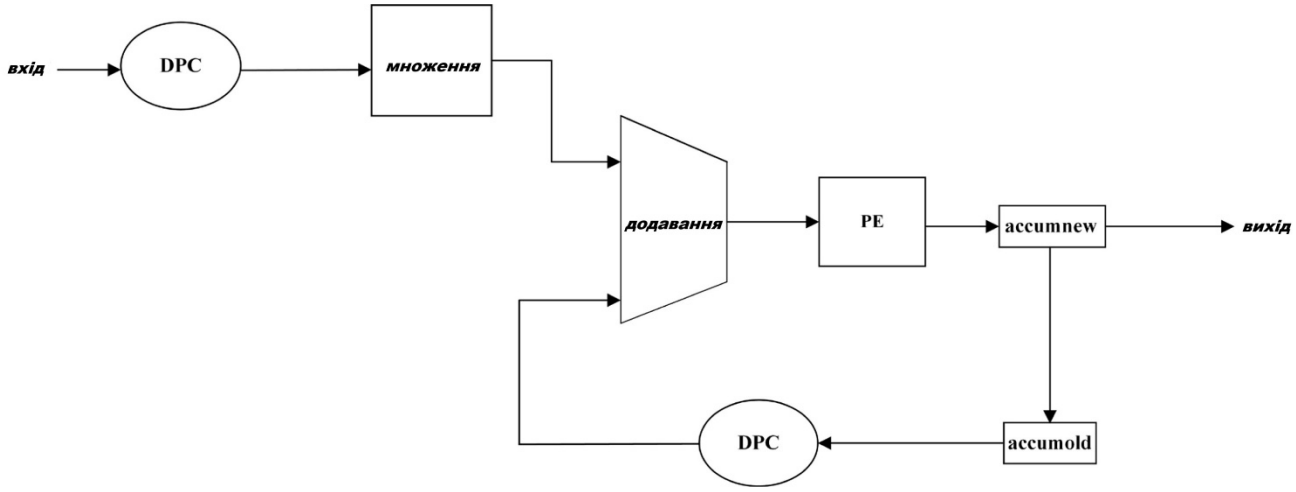


Рисунок 2.16 – Механізм акумулятора зворотного зв'язку

Внутрішню схему нейрона проілюстровано на рис. 2. 16. Вхідні двійкові числа (значення пікселів та вихідні дані попереднього шару), а також вагові коефіцієнти і значення зміщення перетворюються на 256-бітний стохастичний потік за допомогою модулів цифро-імовірнісних перетворювачів. Необхідні операції множення та додавання виконуються за допомогою помножувача та суматора на базі розширеної стохастичної логіки, які були детально описані у підрозділі 2.2. Варто зазначити, що додавання поточного добутку $x_i \times w_{ij}$ до попереднього значення, збереженого в накопичувачі, здійснюється за допомогою петлі зворотного зв’язку, яку також зображено на рисунку 16. Цифро-імовірнісний перетворювач у колі зворотного зв’язку перетворює вміст накопичувача на 256-бітну стохастичну послідовність.

Алгоритм 2 Процес обчислення запропонованого багатошарового персептрона (MLP)

Вхідні дані: X (вхідні значення пікселів), W (вагові коефіцієнти), B (значення зміщення), L (кількість шарів).

Вихідні дані: Кінцевий результат класифікації.

Алгоритм: Класифікація багатошарового перцептрона (MLP) на базі стохастичних обчислень

```
Вхідні дані: Зображення X, ваги W, зміщення B, кількість шарів L
Результат: Кінцевий результат класифікації

for Layer_idx від 1 до L do
    // Визначення вхідних даних для поточного шару
    if Layer_idx == 1 then
        Input_data = X; // Для першого шару входами є пікселі зображення
    else
        Input_data = Previous_Layer_Result; // Результати попереднього шару
    end if

    // Обчислення для кожного нейрона в поточному шарі
    for Neuron_idx від 1 до N_Layer_idx do
        Accumulator = 0; // Ініціалізація накопичувача для сумування

        // Обробка вхідних сигналів нейрона
        for Input_idx від 1 до N_Input do
            // Перетворення вхідного значення та ваги у стохастичні потоки
            Stochastic_Input = SC_Converter(Input_data[Input_idx]);
            Stochastic_Weight = SC_Converter(W[Layer_idx][Neuron_idx][Input_idx]);

            // Виконання стохастичного множення
            Product = Stochastic_Multiplier(Stochastic_Input, Stochastic_Weight);

            // Накопичення результату (стохастичне додавання)
            Accumulator = Stochastic_Adder(Accumulator, Product);
        end for

        // Додавання значення зміщення (bias)
        Stochastic_Bias = SC_Converter(B[Layer_idx][Neuron_idx]);
        Final_Sum = Stochastic_Adder(Accumulator, Stochastic_Bias);

        // Оцінка ймовірності та застосування функції активації ReLU
        Neuron_Output = ReLU(Probabilistic_Estimator(Final_Sum));

        // Збереження результату обчислення нейрона
        Layer_Result[Neuron_idx] = Neuron_Output;
    end for
```

```
    // Оновлення даних для переходу до наступного шару
    Previous_Layer_Result = Layer_Result;
end for

return Кінцевий результат класифікації;
```

Безпосередня реалізація цього алгоритму мовою Python є така:

Програмна реалізація алгоритму MLP (Python)

```python
def SC_Converter(value):
    """
    Цифро-імовірнісний перетворювач (SNG).
    Для програмної математичної моделі повертає саме значення.
    """
    return value


def Stochastic_Multiplier(input_val, weight_val):
    """
    Помножувач на базі розширеної стохастичної логіки.
    Математично відповідає звичайному множенню.
    """
    return input_val * weight_val


def Stochastic_Adder(val1, val2):
    """
    Суматор на базі розширеної стохастичної логіки.
    Математично відповідає звичайному додаванню.
    """
    return val1 + val2
```

```python
def Probabilistic_Estimator(value):
    """
    Імовірнісний оцінювач (процесорний елемент).
    Оцінює фінальне значення після накопичення.
    """
    return value
```

```python
def ReLU(value):
    """
    Функція активації ReLU (Rectified Linear Unit).
    Пропускає додатні значення, від'ємні замінює на 0.
    """
    return max(0, value)
```

```
def evaluate_proposed_mlp(X, W, B, L):
    """
    Алгоритм 1: Процес обчислення запропонованого багатошарового
персептрона (MLP)

    Параметри:
    X (list): Вхідні значення пікселів (одномірний список)
    W (list): Вагові коефіцієнти (тривимірний список: [шар][нейрон]
[вхід])
    B (list): Значення зміщення (двовимірний список: [шар][нейрон])
    L (int): Кількість шарів

    Повертає:
    list: Кінцевий результат класифікації
    """
    Previous_Layer_Result = []

    # Цикл по кожному шару (в Python індексація з 0 до L-1)
    for Layer_idx in range(L):
        Layer_Result = []

        # Визначення вхідних даних для поточного шару
        if Layer_idx == 0:
            # Для першого шару входами є пікселі зображення
            Input_data = X
        else:
            # Для наступних шарів — результати попереднього
            Input_data = Previous_Layer_Result

        N_Neurons = len(W[Layer_idx]) # Кількість нейронів у
поточному шарі
        N_Inputs = len(Input_data)    # Кількість входів до нейрона

        # Обчислення для кожного нейрона в поточному шарі
        for Neuron_idx in range(N_Neurons):
            # Ініціалізація накопичувача для сумування
            Accumulator = 0

            # Обробка вхідних сигналів нейрона
            for Input_idx in range(N_Inputs):
                # Перетворення вхідного значення та ваги
                Stochastic_Input =
SC_Converter(Input_data[Input_idx])
                Stochastic_Weight = SC_Converter(W[Layer_idx]
[Neuron_idx][Input_idx])
```

```
                # Виконання стохастичного множення
                Product = Stochastic_Multiplier(Stochastic_Input,
Stochastic_Weight)

                # Накопичення результату (стохастичне додавання)
                Accumulator = Stochastic_Adder(Accumulator, Product)

            # Додавання значення зміщення (bias)
```

```
            Stochastic_Bias = SC_Converter(B[Layer_idx][Neuron_idx])
            Final_Sum = Stochastic_Adder(Accumulator,
Stochastic_Bias)

            # Оцінка ймовірності та застосування функції активації
ReLU
            Neuron_Output = ReLU(Probabilistic_Estimator(Final_Sum))

            # Збереження результату обчислення нейрона
            Layer_Result.append(Neuron_Output)

        # Оновлення даних для переходу до наступного шару
        Previous_Layer_Result = Layer_Result

    # Повернення кінцевого результату класифікації
    return Previous_Layer_Result
```

Алгоритм 2: Даний алгоритм описує роботу головного модуля та його процедуру. Визначено всі типи внутрішніх з'єднань. Сигнал тактування, позначений як clk, та сигнал скидання визначено як однобітові вхідні дані. Вхідні дані мережі MNIST, вагові коефіцієнти та зміщення представлено у вигляді N-бітових вхідних значень. Вихідні дані функції активації $ReLU$ та акумулятор, що використовуються в розробці, задано як масив N-бітових регістрів. Розмірність цього масиву визначається кількістю нейронів наступного шару.

Усі лічильники та сигнали також мають регістровий тип. Змінні *accumnew* та $accumold$, зображені на рис. 2.16, є N-бітовими провідниками. Передача сигналів у системі є чутливою до позитивного фронту тактового імпульсу clk. У стані скидання усім регістрам присвоюється нульове значення, а сигнал дозволу першого шару встановлюється в логічну «1». В активному стані лічильник $up255$ починає

відлік від 0 до 256, що зумовлено довжиною 8-бітової послідовності. Коли лічильник $up255$ досягає значення 256, він обнуляється, а сигнали $start_{in}$ та $start_{ind}$ набувають значення 1.

Якщо сигнал прапорця встановлено в 1 у стані скидання, масив акумулятора ініціалізується нулем; в іншому випадку значення акумулятора, яке є виходом мультиплексора, присвоюється змінній $accumold$, як показано на рис. 16. Після завершення обробки других 100 нейронів для першого вхідного сигналу, подається другий вхідний сигнал. Відповідно, лічильник $up784$ збільшується на одиницю в кожному циклі, а регістри прапорця та $up100$ скидаються до нуля.

В іншому випадку, тимчасовий регістр $up100$, який випереджає основний $up100$ на один такт clk, збільшується до 100, а лічильник $up255$ знову обнуляється. Таким чином, значення тимчасового регістра $up100$ присвоюється основному регістру $up100$, а сигнал $start_{in}$ скидається до нуля. Коли у стані прапорця його значення досягає нуля, масивам акумулятора присвоюється значення *accumnew*; інакше масив акумулятора залишається без змін. П

ісля отримання результату обчислень для останнього нейрона, це значення передається до функції активації $ReLU$. У стані готовності, після виконання обчислень останнього вхідного сигналу з останнім нейроном другого шару на 255-му такті, активується сигнал готовності, що діє як сигнал скидання для другого шару та розглядається як завершення обчислень першого шару. Як наслідок, сигнал дозволу першого шару також деактивується.

Алгоритм 3: Псевдокод тестового стенду (test bench)

Псевдокод машини станів (State Machine)

```
Input state:
    Read MNIST inputs, weights, biases, registers;
    { 1. Сигналізація тестового стенду першого шару. }

Reset state:
    if (reset == 0) then
        all registers initialize to zero;
        start_b0 = 0;
    end if

Neurons counting state:
    until (achieving 784 neuron) do
        read weights in each 255-clock cycle;
        when (reach 100th neuron) do
            read inputs;
        end when
    end until

Start state:
    if (784 neuron are done) then
        start_b0 = 1;
    end if

Bias state:
    if (start_b0 == 1) then
        until (achieving 100 neuron) do
            read biases each 255-clock cycle;
        end until

        when (100 neuron are done) do
            start_b0 = 0;
            start_next_layer = 1;
        end when
    end if

{ 2. Аналогічний процес для іншого шару. }
```

Реалізація алгоритму мовою Python Оскільки псевдокод описує логіку роботи апаратного тестового стенду (цикли по тактах), у Python ми використаємо імітацію цих затримок та станів

```
import time

def run_test_bench_simulation():
    print("--- Запуск симуляції алгоритму 3 ---")

    # 1. Стан вводу
    print("[1] Стан вводу: завантаження MNIST, ваг та зміщень...")
    mnist_inputs = list(range(784))  # Імітація даних
    weights = [0.5] * 784
    biases = [0.1] * 100
    registers = {}

    # 2. Стан скидання
    reset = 0
    start_b0 = 0
    if reset == 0:
        registers = {i: 0 for i in range(10)}
        start_b0 = 0
        print("[2] Стан скидання виконано: регістри обнулені.")

    # 3. Стан підрахунку нейронів
    print("[3] Початок підрахунку нейронів...")
    neurons_processed = 0
    while neurons_processed < 784:
        # Імітація зчитування ваг (255 тактів на кожне зчитування)
        neurons_processed += 1
        # Умова для 100-го нейрона
        if neurons_processed == 100:
            print(f"    Подія: Досягнуто {neurons_processed}-й
нейрон. Зчитування вхідних даних...")

    # 4. Стан запуску
    if neurons_processed == 784:
        start_b0 = 1
        print(f"[4] Стан запуску: Оброблено {neurons_processed}
нейронів. start_b0 = 1")

    # 5. Стан зміщення
    start_next_layer = 0
    if start_b0 == 1:
        print("[5] Стан зміщення: зчитування...")
        biases_processed = 0
        while biases_processed < 100:
            biases_processed += 1
```

Алгоритм 3: Даний алгоритм описує тестове середовище головного модуля, наведеного в Алгоритмі 2. Вхідні дані, вагові коефіцієнти, зміщення та регістри

визначаються у стані введення. Використовуються три регістри: 9-бітовий регістр *upclk* для підрахунку 255 тактів *clk*, 8-бітовий регістр *upw* для підрахунку вагових коефіцієнтів та 10-бітовий регістр *upx* для 784 вхідних значень. Як і в основному коді, вони є чутливими до позитивного фронту тактового імпульсу.

Стан скидання — це стан, у якому всім регістрам як початкове значення присвоюється нуль. Після цього стану значення *upclk* збільшується від 0 до 256. Значення регістра *upx* збільшується, доки не досягне 784; коли вводиться 783-й вхідний сигнал і значення *upclk* досягає 256, відбувається зчитування вагових коефіцієнтів, а *upclk* скидається до нуля. Регістр *upw* призначений для підрахунку 100 нейронів другого шару; отже, коли він досягає значення 100 під час кожного зчитування вхідних даних, *upw* обнуляється, а лічильник *upx* активується після введення всіх 784 вхідних значень.

Сигнал $start_{b0}$ у початковому стані вказує на завершення процесу зчитування вагових коефіцієнтів та вхідних даних. По-друге, зміщення повинні зчитуватися, коли сигнал $start_{b0}$ дорівнює 1. При цьому значення *upw* збільшується, а *upclk* досягає 256 тактів *clk*; зчитуються зміщення, *upclk* скидається до нуля, а регістр *upw* збільшується на одиницю у стані зміщення. Коли всі зміщення введено і обчислення завершено, сигнал $start_{b0}$ скидається до нуля, і активується сигнал $start_{b1}$ для другого шару.

Розміщення вхідних файлів, що включають вхідні дані x_{ij}, вагові коефіцієнти w_{lij} та зміщення b_{ij}, здійснюється наступним чином:

1. Відповідно до Алгоритму 1, усі 784 значення пікселів перетворюються на два числа в діапазоні від -1 до $+1$. Отримані числа записуються в рядок вхідних файлів за допомогою програмного коду MATLAB.

2. Для вагових коефіцієнтів подібні текстові файли генеруються за допомогою MATLAB. Слід зауважити, що тестовий стенд Verilog зчитує вагові коефіцієнти кожні 255 тактів сигналу тактування.

3. Значення зміщень записуються в текстовий файл, аналогічний файлам із вхідними даними та ваговими коефіцієнтами. Введення значень зміщень виконується відповідно до відповідних рядків Алгоритму 3.

Усі обчислення та операції тривають до завершення обробки всіх нейронів. Таким чином, для останніх 10 нейронів отримується найбільше значення; мітка цього нейрона вказує на розпізнану цифру від 0 до 9.

2.7 Тестування роботи програмної системи

Для оцінки ефективності розробленого обчислювального елемента та тестування запропонованого методу реалізації отримані результати було розділено на дві частини. У першій частині здійснено реалізацію та аналіз розробленої моделі на базі простої мережі багатошарового перцептрона. Далі проведено дослідження складнішої структури з використанням вхідних даних набору MNIST, паралельно з вирішенням завдань апаратної реалізації на базі програмованих логічних інтегральних схем, а також виконано порівняння отриманих результатів з іншими методами та архітектурами. Нижче першочергово розглядається проста мережа.

Проста мережа конфігурації 1-3-3-1 Розглянемо мережу, зображену на рис. 2. 17, яка містить один вхідний нейрон, перший та другий приховані шари по 3 нейрони в кожному, а також один вихідний нейрон. Детальне пояснення математичних операцій та функцій кожного елемента у стохастичній структурі такої мережі було раніше наведено в Розділі 2. На першому етапі за допомогою середовища MATLAB було створено набір вхідних даних, що складається з 1000 випадкових вибірок; після цього проведено навчання мережі з використанням фреймворку PyTorch.

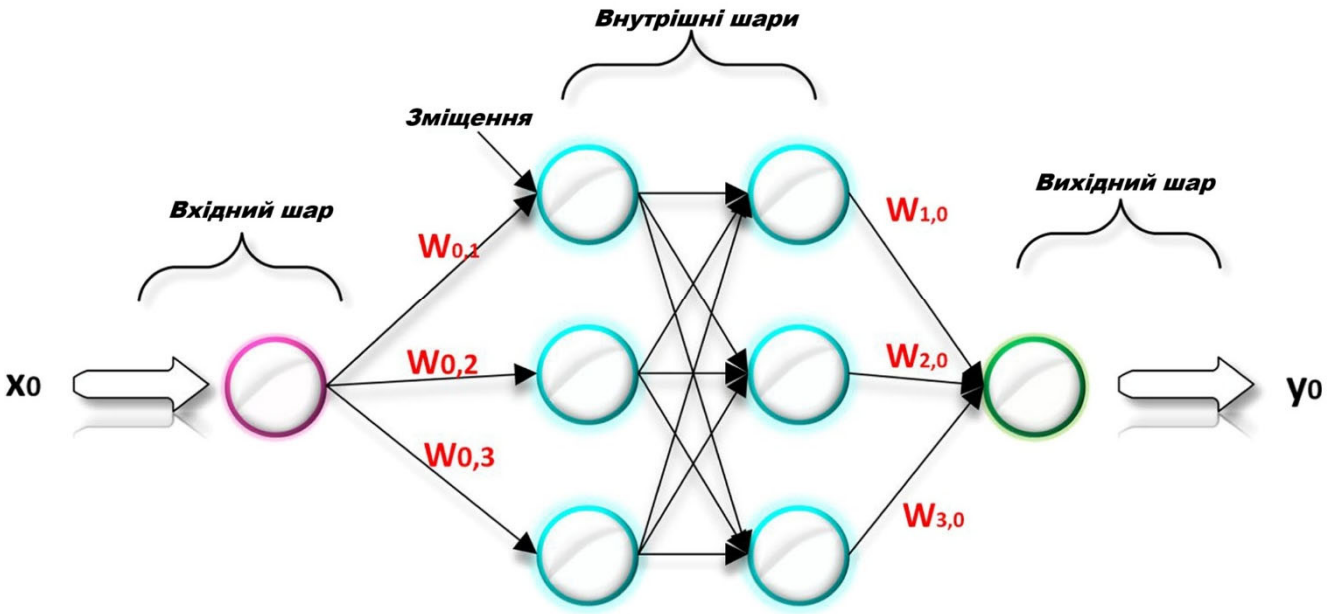


Рисунок 2.17 – Мережа MLP 1-3-3-1.

У результаті було отримано вагові коефіцієнти та зміщення для багатошарового перцептрона конфігурації 1-3-3-1, які наведено в табл. 2.1. З метою порівняння точності запропонованої стохастичної моделі з традиційною стохастичною моделлю, обидві архітектури було реалізовано на базі програмованої логічної інтегральної схеми Xilinx Virtex-7 серії xc7vx980t. Кінцеві значення вагових коефіцієнтів та зміщень для розробленої та традиційної моделей представлено в табл. 2.1. Відповідно до отриманих значень вагових коефіцієнтів та зміщень, запропонований метод дозволяє генерувати більш точні числові значення.

Слід зазначити, що довжина застосованої послідовності для запропонованого методу становить 128 біт, що значно менше порівняно з довжиною послідовності, яка використовується в традиційних методах на основі стохастичних обчислень. Для 128-бітової послідовності спостерігається середня похибка на рівні 3 %. Водночас у традиційному методі цей показник похибки досягає 7,5 %, оскільки він залежить від кількості бітів, близькості числа до нуля та тривалості тактів тактового сигналу. Таким чином, запропонований метод є не лише точнішим, але й значно швидшим за традиційні методи на базі стохастичних обчислень.

Таблиця 2.1 – Вагові коефіцієнти та зміщення багатошарового перцептрона конфігурації 1-3-3-1 у середовищі PyTorch, для запропонованого обчислювального елемента та традиційного обчислювального елемента

Вагові коефіцієнти 1-го прихованого шару	Вихід запропонованого ОЕ	Вихід традиційного ОЕ	Вагові коефіцієнти 2-го прихованого шару	Вихід запропонованого ОЕ	Вихід традиційного ОЕ	Вихідні вагові коефіцієнти	Вихід запропонованого ОЕ	Вихід традиційного ОЕ
-0,081	-0,0511	-0,159	+0,1786	+0,1415	+0,1023	-0,4	-0,344	-0,3028
-0,3286	-0,2843	-0,2366	+0,3142	+0,3651	+0,3965	+0,4	+0,3440	+0,3028
-0,3284	-0,2843	-0,2366	-0,1206	-0,09	-0,132	+0,0610	+0,031	+0,02621
+0,2160	+0,1885	+0,1541	+0,3004	+0,270	+0,4020	-	-	-
-0,1417	-0,1628	-0,1852	+0,2204	+0,1885	+0,1541	-	-	-
-0,0729	-0,0511	-0,159	+0,4	+0,3440	-0,3028	-	-	-
-0,0318	-0,0314	0	-0,2265	-0,1885	-0,1541	-	-	-
-0,4	-0,344	-0,3028	+0,0569	+0,031	+0,02621	-	-	-
+0,4	+0,3440	+0,3028	-0,4	-0,344	-0,3028	-	-	-
Зміщення			Зміщення			Зміщення		
+0,4	+0,3440	+0,3028	-0,4	-0,344	-0,3028	+0,1550	+0,1415	+0,1128
-0,4	-0,344	-0,3028	+0,2754	+0,2351	+0,3412	-	-	-
+0,2702	+0,2473	+0,3312	+0,4	+0,3440	+0,3028	-	-	-

Таблиця 2.2 – Реалізація 128-бітових послідовностей для традиційного та запропонованого багатошарового перцептрона конфігурації 1-3-3-1 на базі стохастичних обчислень з використанням програмованої логічної інтегральної схеми Virtex7-xc7vx980t

Метод	Динамічне енергоспоживання (мВт)	Затримка критичного шляху (нс)	Точність (%)
Традиційний обчислювальний елемент	9,872	1,685	82,705
Запропонований обчислювальний елемент	4,6042	1,423	99,33

Результати дослідження динамічного енергоспоживання, часових характеристик та точності реалізованого багатошарового перцептрона конфігурації 1-3-3-1 для традиційного обчислювального елемента та запропонованого обчислювального елемента наведено в табл. 2.2. На основі отриманих результатів встановлено, що розроблена архітектура забезпечує підвищення точності на $16,625\,\%$, зменшення затримки критичного шляху на $0,262$ нс та суттєве зниження динамічного енергоспоживання на $5,2628$ мВт.

Для отримання кінцевих результатів було використано набір даних MNIST. Цей набір даних складається з навчальної вибірки обсягом $60\,000$ зразків та тестової вибірки обсягом $10\,000$ зразків. Зображення в наборі даних MNIST є чорно-білими (у градаціях сірого) з розміром 28×28 пікселів і містять рукописні цифри з мітками від 0 до 9.

Таблиця 2.3 – Реалізація трьох різних структур для порівняння на базі програмованої логічної інтегральної схеми Virtex-7.

Метод	Пристрій	Розмір мережі	Довжина послідовності	Похибка	Площа: Тригери	Площа: Таблиці пошуку	Площа: Ввід/вивід	Затримка
Запропонований	Virtex-7	784-100-200-10	256	2,24%	5935 (0,24%)	14 964 (1,22%)	74 (8,71%)	15,3 (мкс)
Запропонований	Virtex-7	784-100-200-10	512	1,86%	65675 (0,27%)	18 033 (1,48%)	83 (9,76%)	30,660 (мкс)
Запропонований	Virtex-7	784-100-200-10	1024	1,48%	7279 (0,30%)	19 276 (1,58%)	92 (10,82%)	61,380 (мкс)
[31]	Virtex-7	784-100-200-10	256	2,33%	-	1 013 002 (83%)	-	1,705 (мкс)
[36]	Virtex-7	784-100-200-10	1024	5,72%	-	144 450 (11,82%)	-	8,561 (мкс)
[44]	ARTIX-7	784-20-15-10	Не стохастичний	-	-	62 695 (46,58%)	-	819 (тактів)
[45]	ZC706	784-512-512-10	Не стохастичний	1,60%	-	78 679 (35,99%)	-	210 (тис. кадрів/с)
[46]	xc7z010clg400	400-25-15-10	Не стохастичний	2,26%	-	37%	-	-

З метою отримання результатів та проведення порівняння з попередніми дослідженнями було розглянуто дві структури нейронних мереж. Структура A, що являє собою багатошаровий перцептрон конфігурації 784-200-100-10, складається з вхідного шару, який містить 784 нейрони, та двох прихованих шарів: перший прихований шар налічує 200 нейронів, а другий — 100 нейронів. Вихідний шар складається з 10 нейронів, які мають мітки від 0 до 9.

Структура B є багатошаровим перцептроном конфігурації 784-100-200-10. Єдиною відмінністю від попередньої архітектури є кількість нейронів у першому та другому прихованих шарах. Структура A використовується для порівняння точності з результатами дослідження [42], тоді як друга структура застосовується для отримання результатів порівняння з методом, запропонованим у роботі [31].

Крім того, результати порівняння з чотирма нестохастичними апаратними реалізаціями нейронних мереж на різних платах програмованих логічних інтегральних схем наведено в табл. 2.3. Модулі стохастичного багатошарового перцептрона реалізовано мовою опису апаратури Verilog у програмному середовищі Vivado спільно з ваговими коефіцієнтами та зміщеннями, екстрагованими з нейронної мережі, попередньо навченої за допомогою мови програмування Python.

На рис. 18 зображено результати порівняння багатошарового перцептрона на базі стохастичних обчислень (структура A) з архітектурою, запропонованою у дослідженні Y. Liu, S. Liu та ін. (2018) [42], для різних довжин послідовності — від 32 до 2048 біт.

У згаданій роботі відсутні дані щодо енергоспоживання, тому запропонований багатошаровий перцептрон порівнюється з нею лише за показником точності. В обох методах для розширення діапазону використовується структура ESL, проте точність запропонованого методу є вищою, особливо для довших послідовностей.

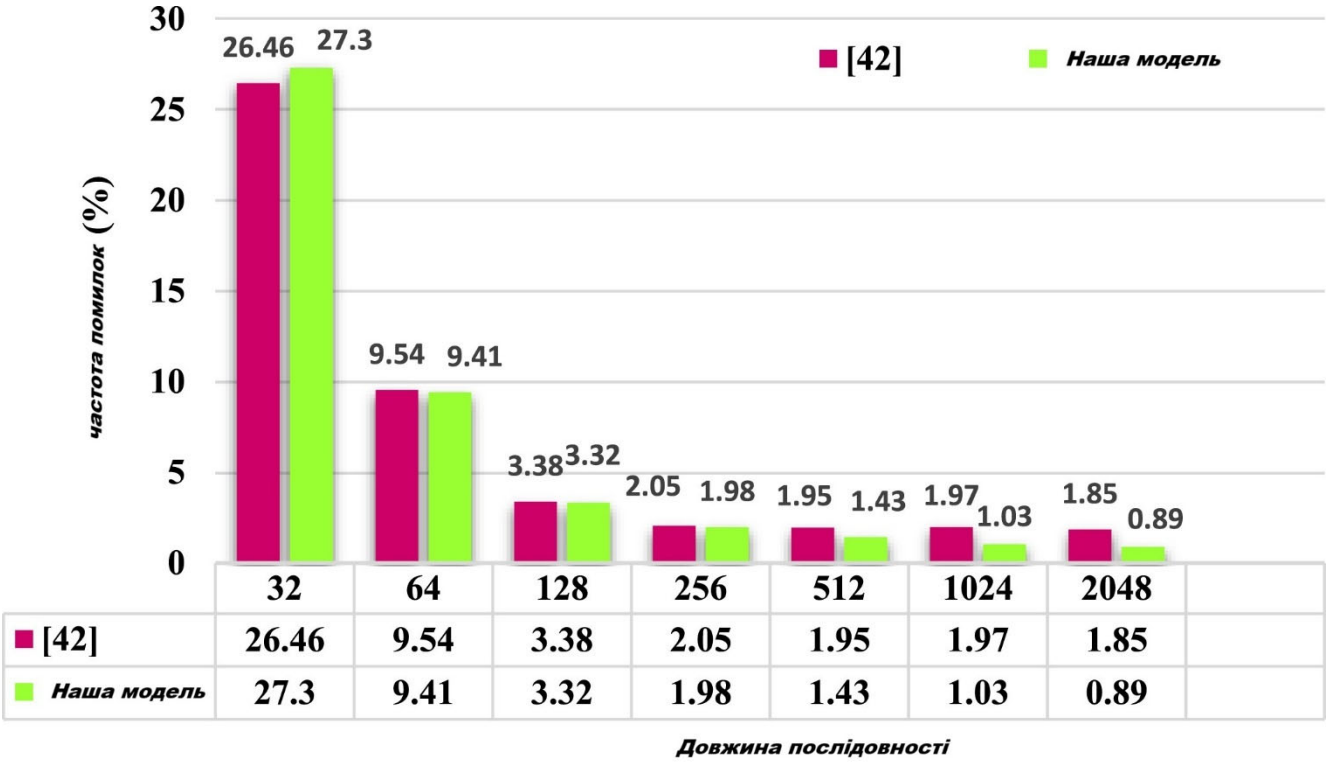
Таблиця 2.4. Порівняння енергоспоживання для розробленого методу та інших нестохастичних методів.

Метод	Довжина послідовності	Динамічне енергоспоживання (Вт)	Енергоспоживання сигналів (Вт)	Логічне енергоспоживання (Вт)	Енергоспоживання вводу/виводу (Вт)	Загальне енергоспоживання на кристалі (Вт)
Запропонований	256	0,323	0,016	0,006	0,301	0,626
	512	0,361	0,017	0,007	0,336	0,664
	1024	0,381	0,018	0,008	0,355	0,684
[45]	Не стохастичний	-	-	-	-	3,286
[44]	Не стохастичний	-	-	-	-	4,868

Основною причиною вищої точності є краща продуктивність розробленого обчислювального елемента порівняно з традиційним. Як зазначалося раніше, традиційний обчислювальний елемент не досягає збіжності за таких малих довжин послідовностей; отже, точність традиційного обчислювального елемента в архітектурі Y. Liu, S. Liu та ін. (2018) [42] є обмеженою.

Комплексне порівняння запропонованого методу зі стохастичними та нестохастичними методами наведено в табл. 2.3. Перші три моделі [31, 36] та розроблений метод є архітектурами на базі стохастичних обчислень, які реалізовані на програмованій логічній інтегральній схемі Xilinx Virtex-7 моделі xc7v2000t із робочою частотою 100 МГц. Показник похибки в запропонованому методі суттєво

покращився порівняно з обома попередніми підходами. У дослідженні [31] для розширення діапазону дійсних чисел замість структури ESL було застосовано інтегральні стохастичні обчислення. Порівняно з ESL, інтегральні стохастичні обчислення потребують меншої довжини послідовності для досягнення високої точності. Однак це призводить до збільшення займаної площі на кристалі через складнішу структуру.



	32	64	128	256	512	1024	2048
[42]	26.46	9.54	3.38	2.05	1.95	1.97	1.85
Наша модель	27.3	9.41	3.32	1.98	1.43	1.03	0.89

Рисунок 2.18 – Порівняння точності запропонованого нами методу з [42] для різних значень довжини послідовності.

В інтегральних стохастичних обчисленнях, якщо вхідне значення не перевищує 0,6, точність обчислень є аналогічною до традиційних стохастичних обчислень. В іншому випадку спостерігається значна втрата точності.

З іншого боку, запропонований обчислювальний елемент не потребує використання множника для досягнення кінцевого результату в блоці обчислення функції вартості. Як наслідок, у розробленому методі повністю усунуто необхідність у додатковому апаратному забезпеченні та модулях, що зазвичай потрібні для вирішення проблеми масштабування значень.

Моделювання показує, що точність обчислень у запропонованому нами підході покращилася на 2Затримка (латентність) у нашій структурі розраховується таким чином:

$$\text{latency} = \text{delayinourmethod} \times \text{periodofclock}, \tag{2.50}$$

і

$$\text{latency(8bit)} = (6 \times 255) \times 10\text{ns} = 15300\text{ns}. \tag{2.51}$$

У розробці [31] використовується повністю конвеєрна паралельна архітектура, яка поділена на невеликі етапи навчання для прискорення тренування мережі. Ця модель приймає обмежену кількість вузлів та входів. Концепція конвеєрної обробки (pipelining) також застосовується у структурі B. Li та ін. (2015) [36].

Таким чином, затримка в цих методах значно зменшується ціною збільшення використання площі кристала та ресурсів, енергоспоживання і складності. У запропонованій нами архітектурі концепція конвеєрної обробки безпосередньо не використовується. Проте, завдяки величезному зменшенню апаратних витрат, в одному чипі FPGA можна одночасно реалізувати кілька мереж багатошарового перцептрона (MLP).

У майбутніх дослідженнях структуру кожного шару можна реорганізувати для паралельної реалізації. Відповідно до даних про використання площі, наведених у табл. 2.3, на одній FPGA можна одночасно реалізувати приблизно 10 мереж MLP зі структурою 784-200-100-10 (що обмежується кількістю контактів вводу-виводу — IO pins). Натомість в інших двох методах через високий рівень використання таблиць пошуку (LUT) можна реалізувати лише одну мережу.

Зрештою, рівень затримки в нашому методі компенсується можливістю обробки кількох зображень MNIST за допомогою одного чипа FPGA. У роботі [31] відсутні дані щодо рівня енергоспоживання. Тому в табл. 23 порівнюються лише точність, використання площі кристала та затримка.

Три нестохастичні методи наведено в останніх трьох рядках табл. 2.3. Порівняно з усіма трьома методами, запропонований нами підхід використовує в

кілька разів меншу площу, проте час обчислень у нестохастичних методах є меншим, ніж в архітектурах на базі стохастичних обчислень.

У табл. 2.4 наведено дані щодо енергоспоживання розробленого методу порівняно з двома нестохастичними архітектурами. Значення динамічного енергоспоживання, а також енергоспоживання сигналів, логіки та блоків вводу-виводу для запропонованого методу наведено для трьох довжин послідовностей: 256, 512 та 1024 біти. У роботі [45] представлено нестохастичний метод, який реалізує архітектуру 784-512-512-10 на програмованій логічній інтегральній схемі ZC706. П

Порівняно із запропонованим нами методом (для довжини послідовності 1024 біти), тут досягається покращення точності на 0,12У дослідженні H. Mittal та ін. (2019) [46] реалізовано мережу багатошарового перцептрона зі структурою 400-25-15-10 на базі плати Xilinx Zybo XC7Z010CLG400-1. Наш метод є в середньому на 0,78У роботі J. Park та W. Sung (2016) [44] запропоновано глибоку нейронну мережу з фіксованою комою на базі програмованої логічної інтегральної схеми з використанням зовнішньої динамічної пам’яті (DRAM).

Ми порівняли загальне енергоспоживання нашого розробленого методу (для довжини послідовності 1024 біти) з результатами цього дослідження. Як показано в табл. 2.4, за допомогою їхнього методу було досягнуто зниження загального енергоспоживання на кристалі на 4,184 Вт. Зрештою, запропонована нами модель є вдалим рішенням для застосувань, які вимагають структури з мінімальним використанням площі та ресурсів, найменшим енергоспоживанням, а також належним рівнем швидкодії та точності.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

В розділі проаналізовано умови праці розробника програмного забезпечення, зокрема вимоги до безпечної експлуатації обчислювальної техніки та організації ергономічного робочого місця. Додатково досліджено вплив динамічних природних явищ на поверхні землі та визначено принципи ведення робіт в умовах надзвичайних і аварійних ситуацій.

3.1 Динамічні явища на поверхні землі

Динамічні явища на поверхні Землі включають широкий спектр природних процесів, які можуть мати значний вплив на безпеку людей та інфраструктури. Ці явища охоплюють геологічні, метеорологічні та гідрологічні зміни, кожне з яких має свої специфічні ризики та можливості для запобігання катастрофам [68].

Усі надзвичайні ситуації в Україні регулюються відповідно до національного законодавства, зокрема згідно з наступними ключовими документами: Кодекс цивільного захисту України, положення про підсистему реагування на надзвичайні ситуації та проведення аварійно-рятувальних та інших невідкладних робіт у рамках єдиної державної системи цивільного захисту, положення про штаб з ліквідації наслідків надзвичайної ситуації та організацію оперативно-технічної і звітної документації, наказ Державної служби України з надзвичайних ситуацій [68-72].

До динамічних явищ на поверхні землі належать абіотичні та біотичні небезпеки. Одним із проявів природних небезпек є стихійні лиха.

Стихійні лиха - це природні явища, які мають надзвичайний характер та призводять до порушення нормальної діяльності населення, загибелі людей, руйнування і знищення матеріальних цінностей.

За причиною виникнення стихійні лиха поділяють на: тектонічні (пов'язані з процесами, які відбуваються в надрах земної кори), топологічні (пов'язані з процесами, які відбуваються на поверхні землі), метеорологічні (пов'язані з процесами, які відбуваються в атмосфері).

Види абіотичних небезпек: літосферні, гідросферні, атмосферні, космічні. Серед літосферних небезпек розрізняють землетруси, вулкани, зсуви, селі.

— Землетрус — підземні поштовхи та коливання земної поверхні, зумовлені раптовими зміщеннями і розривами в корі або у верхній частині мантії, які передаються на великі відстані у вигляді пружних коливань.

— Зсув - сповзання мас гірських порід вниз по схилу, яке виникає через порушення рівноваги. Зсуви бувають повільні (см/доба), середньої швидкості (м/год), швидкі (км/год).

— Селі - це паводки з великою концентрацією ґрунту, мінеральних частин, каміння, уламків гірських порід (від 10-15 до 75% об'єму потоку). За складом матеріалу, що переносить потік, розрізняють: грязьові, грязекам'яні, водокам'яні.

Гідросферні небезпеки - повені, снігові лавини, шторми, цунамі.

— Повінь — значне затоплення місцевості внаслідок підйому рівня води в річці, озері, водосховищі. В Україні повені спостерігаються в басейнах Дніпра, Дністра, Прип'яті. В останні роки найчастіше спостерігаються в Закарпатті (Західний Буг та Тиса).

— Снігова лавина - величезна маса снігу, яка зсувається або падає зі стрімких гірських схилів, захоплюючи різні об'єкти, що трапляються на шляху. Лавина супроводжується утворенням передлавинної поверхневої хвилі, що має найбільшу руйнівну силу. Розрізняють сухі (зимові) та мокрі (весняні) снігові лавини.

— Шторм - тривалий, дуже сильний вітер, що спричиняє значні руйнування на суші та велике хвилювання на морі.

— Цунамі - великі хвилі, що виникають на поверхні океану під час підводних землетрусів. Атмосферні небезпеки - бурі, урагани, тайфуни, цунамі, смерчі, морози, засуха тощо.

— Ураган - вітер руйнівної сили зі швидкістю 35 м/с.

— Буря - шторм, тривалий, дуже сильний вітер (понад 20 м/с), спричинений зазвичай циклоном.

— Смерч - сильний локальний атмосферний вихор (діаметр до 1000м), в якому повітря обертається зі швидкістю до 100 м/с.

3.2 Заходи, що запобігають впливу на людину агресивних та токсичних речовин, які використовуються в технологічному процесі

У сучасному технологічному виробництві широко використовуються речовини, що мають агресивні або токсичні властивості. Їх наявність у виробничому середовищі створює потенційну небезпеку для здоров’я та життя працівників. Тому надзвичайно важливим завданням є розроблення і впровадження комплексу заходів, спрямованих на запобігання шкідливому впливу таких речовин. Ці заходи повинні бути інтегрованими у систему управління охороною праці підприємства та включати технічні, організаційні, адміністративні, інформаційні та медико-профілактичні складові. Поряд з цим важливою є роль відповідного нормативного регулювання, оскільки вимоги щодо безпеки праці, роботи з хімічними речовинами, оцінювання ризиків та впровадження системи контролю повинні відповідати сучасним стандартам, зокрема ДСТУ. Виконання вимог стандартів забезпечує єдиний підхід до організації безпеки та охорони здоров’я працівників.

Ідентифікація речовин, які є потенційно небезпечними, є першочерговим етапом у формуванні системи безпеки. Відповідно до ДСТУ ISO 14698-1:2008 [72], підприємства мають здійснювати контроль повітряного середовища на робочих місцях з метою оцінки рівнів забруднення та встановлення джерел викиду. Аналіз хімічних речовин у технологічному процесі дозволяє визначити рівень ризику для працівника та необхідні заходи контролю. До типових небезпечних речовин, що використовуються в технологічному процесі, належать: кислоти (сірчана, азотна, хлоридна), які можуть викликати опіки, подразнення дихальних шляхів; луги (гідроксид натрію, калію), що становлять загрозу при потраплянні на шкіру або в очі; леткі органічні сполуки (бензол, толуол), що мають канцерогенні та мутагенні властивості; пари важких металів (ртуть, свинець), які накопичуються в організмі

та мають хронічний токсичний ефект; гази (аміак, хлор, сірководень), які викликають гостре отруєння при перевищенні допустимих концентрацій. На основі переліку використаних речовин формується реєстр небезпек, визначаються гігієнічні нормативи та розробляються карти ризиків.

Технічний захист базується на мінімізації або усуненні контактів працівника з токсичними речовинами за рахунок застосування безпечних технологій. До основних принципів належать: герметизація технологічного обладнання, що виключає витік шкідливих речовин (ДСТУ EN ISO 14123-1:2016) [73]; заміна небезпечної речовини на менш токсичну, за можливості; використання замкнених технологічних циклів, де робоче середовище повністю ізольоване від навколишнього простору; впровадження автоматизованих систем подачі, змішування та транспортування реагентів; застосування систем аварійного відключення, автоматичних клапанів та сповіщувачів витоку. Крім того, важливим аспектом технічного захисту є регулярна діагностика обладнання та перевірка його герметичності, проведення профілактичного технічного обслуговування та модернізація систем з урахуванням новітніх досягнень у галузі безпеки праці.

Колективні засоби захисту передбачають створення інженерних систем, що ізолюють або нейтралізують небезпечні речовини до їхнього контакту з людиною. До таких засобів належать: вентиляційні системи з локальними витяжками, які забезпечують виведення шкідливих парів безпосередньо з місця утворення (ДБН В.2.5-67:2013) [74]; витяжні шафи, які встановлюються в лабораторіях та на робочих місцях для роботи з леткими речовинами; фільтраційні установки з багатоетапними системами очищення повітря (вуглецеві, хімічні, HEPA-фільтри); фізичні бар'єри, перегородки та герметичні камери для відокремлення працівника від джерела небезпеки; системи рециркуляції повітря з виведенням очищених газів за межі робочої зони. Ефективність колективного захисту залежить від правильного проектування, монтажу та обслуговування обладнання. Необхідно забезпечувати регулярну перевірку систем, зміну фільтрів, очищення повітропроводів.

Засоби індивідуального захисту застосовуються тоді, коли інженерні заходи недостатні або не повністю виключають контакт із шкідливою речовиною. До них відносяться: респіратори, фільтруючі півмаски та протигази (ДСТУ EN 149:2017) [75], вибір яких залежить від типу речовини (аерозоль, газ, пара); захисні окуляри, екрани, маски для захисту органів зору від агресивних парів (ДСТУ EN 166:2017) [76]; захисний одяг, комбінезони з кислотостійких і лугостійких матеріалів (ДСТУ EN 14605:2017) [77]; рукавиці з гуми, латексу, неопрена, нітрилу залежно від речовини; захисне взуття з покриттям, стійким до дії агресивних речовин. Працівники мають проходити навчання щодо правильного використання, зберігання та догляду засобі індивідуального захисту. Усі засоби повинні регулярно перевірятись, а несправні — замінюватись.

Організаційні заходи включають розробку нормативної документації, проведення навчань та моніторинг дотримання вимог безпеки. Основні складові: розроблення та впровадження інструкцій з охорони праці для кожної операції з токсичними речовинами; регулярне проведення вступного, первинного, повторного та позапланового інструктажу; організація роботи відповідальних осіб за безпеку хімічних речовин на підприємстві; ведення документації щодо обліку, зберігання та використання небезпечних речовин; забезпечення наочності: плакати, сигнальні таблички, евакуаційні схеми (ДСТУ EN ISO 7010:2019) [48]; проведення періодичних аудитів стану охорони праці. Ці заходи сприяють формуванню культури безпеки, що є запорукою зниження ризиків у повсякденній роботі.

Контроль рівня концентрації токсичних речовин у повітрі виконується згідно з методиками, визначеними в ДСТУ ISO 16000 [79]. Він включає: автоматизовані системи виявлення перевищення ГДК; індикатори кольорової зміни для виявлення наявності газів; портативні прилади для експрес-аналізу повітря; лабораторне дослідження повітря, води, поверхонь обладнання; аналіз та документування результатів з подальшим коригуванням заходів безпеки. Контрольні заходи мають виконуватись систематично, згідно з графіками та у випадках надзвичайних ситуацій (виток, аварія).

Для підтримання здоров’я працівників необхідно: проводити попередній медичний огляд при прийомі на роботу; здійснювати періодичні огляди згідно з чинними нормативами; створити медичний пункт з доступом до первинної медичної допомоги; мати засоби надання допомоги при хімічних ураженнях (антидоти, нейтралізатори); забезпечити працівників інформацією про симптоми отруєння, методи самодопомоги, профілактику хімічних уражень. У разі виникнення нещасного випадку необхідно вжити заходів евакуації, ізоляції джерела небезпеки та викликати медичну допомогу.

Основні нормативні документи, що регулюють роботу з токсичними речовинами: ДСТУ ISO 45001:2019 — Системи управління охороною здоров’я та безпекою праці [80]; ДСТУ EN 60068-2-1:2022 — Оцінювання впливу хімічних речовин, що вдихаються [81]; ДСТУ Б В.2.5-67:2013 — Вентиляція та кондиціонування повітря [74]; ДСТУ EN ISO 14123-1:2018 — Безпечність машин [82]. Захист від шкідливих речовин; ДСТУ EN 149:2017 — Респіратори фільтрувальні [75]; ДСТУ EN 943-1:2021— Захисний одяг для газів [83]; ДСТУ EN ISO 7010:2019— Знаки безпеки [78]. Також враховуються інші нормативно-правові акти, включаючи Накази МОЗ України, санітарні правила, технічні регламенти.

Комплексна система запобігання впливу токсичних речовин у технологічному процесі повинна включати технічні, організаційні, медичні та нормативні компоненти. Виконання вимог чинних ДСТУ дозволяє ефективно виявляти та нейтралізувати загрози, що виникають при використанні небезпечних речовин. Рекомендується: регулярно оновлювати та переглядати програми безпеки; проводити аналіз ризиків при впровадженні нових технологій; впроваджувати сучасні системи автоматичного контролю; забезпечувати ефективну комунікацію між службами безпеки, керівництвом та працівниками; інвестувати в навчання персоналу та модернізацію ЗІЗ. Тільки системний підхід дозволить створити умови праці, що забезпечують здоров’я, безпеку та ефективність роботи у технологічному середовищі, де присутні агресивні або токсичні речовини.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було досягнуто основної мети — розроблено та реалізовано програмно-апаратну архітектуру нейронної мережі типу багатошаровий перцептрон на основі стохастичних обчислень із використанням ПЛІС Xilinx Virtex-7. Запропоноване рішення забезпечує оптимізацію обчислювальних процесів у нейронних мережах, підвищуючи ефективність використання апаратних ресурсів кристала при збереженні необхідної точності класифікації. Важливим аспектом дослідження стало впровадження інноваційного процесорного елемента, побудованого на основі підходу скінченних автоматів і механізму сигналізації, що дало змогу уникнути традиційних проблем збіжності результатів та масштабованості, характерних для стохастичних суматорів.

Для досягнення поставлених завдань було запропоновано архітектурний підхід, що базується на використанні контролера сигналів, який забезпечує послідовну передачу вхідних даних. Це дало змогу реалізувати складні нейронні мережі з великою кількістю вузлів і шарів із використанням мінімальної кількості апаратних ресурсів — лише двох стохастичних модулів множення та додавання. Порівняно з традиційними процесорними елементами, розроблена модель не потребує виконання мільйонів тактів для досягнення стабільного результату, що суттєво підвищує швидкодію та точність обчислень у межах специфічних режимів функціонування ПЛІС.

Практична реалізація та результати симуляції підтвердили переваги запропонованого методу над існуючими аналогами. Зокрема, вдалося досягти:

- скорочення площі використання кристала на понад **82%**;
- підвищення точності розпізнавання на **2%**;
- суттєвого зниження енергоспоживання системи.

Попри те, що стохастичні обчислення належать до методів наближених обчислень і характеризуються дещо більшою затримкою порівняно з детермінованими архітектурами, розроблена система є універсальною та

придатною до адаптації для інших типів нейронних мереж, зокрема згорткових і рекурентних. Подальші перспективи розвитку проєкту полягають у розробці стратегії конвеєризації обчислень для мінімізації часових затримок, що дасть змогу створювати більш продуктивні та енергоефективні інтелектуальні системи для вбудованих програмно-апаратних рішень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain // Psychol. Rev. – 1958. – Vol. 65, no. 6. – P. 386–408. – DOI: http://dx.doi.org/10.1037/h0042519.
2. Feng T., Wang C., Chen X., Fan H., Zeng K., Li Z. URNet: A U-net based residua network for image dehazing // Appl. Soft Comput. – 2021. – Vol. 102. – P. 106884. – DOI: http://dx.doi.org/10.1016/j.asoc.2020.106884.
3. Liu P., Wang J., Guo Z. Multiple and complete stability of recurrent neural networks with sinusoidal activation function // IEEE Trans. Neural Netw. Learn. Syst. – 2021. – Vol. 32, no. 1. – P. 229–240. – DOI: http://dx.doi.org/10.1109/TNNLS.2020.2978267.
4. Hartpence B., Kwasinski A. CNN and MLP neural network ensembles for packet classification and adversary defense // Intell. Converg. Netw. – 2021. – Vol. 2, no. 1. – P. 66–82. – DOI: http://dx.doi.org/10.23919/ICN.2020.0023.
5. Lokwon K., Asaad S., Linsker R. A fully pipelined FPGA architecture of a factored restricted Boltzmann machine artificial neural network // ACM Trans. Reconfigurable Technol. Syst. – 2014. – Vol. 7. – DOI: http://dx.doi.org/10.1145/2539125.
6. Haykin S. S. Neural Networks and Learning Machines. – 3rd ed. – Upper Saddle River, NJ, USA: Pearson, 2009.
7. Varnava C. Phase-change memory devices for on-chip neural networks // Nature Electron. – 2021. – Vol. 4. – P. 454. – DOI: http://dx.doi.org/10.1038/s41928-021-00627-4.
8. Yeo I., Gi S. G., Wang G., Lee B. G. A hardware and energy-efficient online learning neural network with an RRAM crossbar array and stochastic neurons // IEEE Trans. Ind. Electron. – 2021. – Vol. 68, no. 11. – P. 11554–11564. – DOI: http://dx.doi.org/10.1109/TIE.2020.3032867.
9. Gaines B. R. Stochastic computing systems // Advances in Information Systems Science. – 1969. – P. 37–172.

10. He X., Chu Z. Stochastic circuit design based on exact synthesis // Proc. 2021 China Semiconductor Technology International Conference (CSTIC). – 2021. – P. 1–3. – DOI: http://dx.doi.org/10.1109/CSTIC52283.2021.9461462.
11. Brown B. D., Card H. C. Stochastic neural computation. I. Computational elements // IEEE Trans. Comput. – 2001. – Vol. 50, no. 9. – P. 891–905.
12. Brown B. D., Card H. C. Stochastic neural computation. II. Soft competitive learning // IEEE Trans. Comput. – 2001. – Vol. 50, no. 9. – P. 906–920.
13. Alaghi A., Hayes J. P. Dimension reduction in statistical simulation of digital circuits // Proc. Symp. Theory Modeling Simulation (DEVS Integrative M&S Symp.). – 2015. – P. 1–8.
14. Hayes J. P. Introduction to stochastic computing and its challenges // Proc. DAC. – 2015. – P. 59.
15. Ardakani A. et al. Hardware implementation of FIR/IIR digital filters using integral stochastic computation // Proc. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – 2016. – Vol. 3. – P. 6540–6544.
16. Liu Y., Liu S., Wang Y., Lombardi F., Han J. A survey of stochastic computing neural networks for machine learning applications // IEEE Trans. Neural Netw. Learn. Syst. – 2021. – Vol. 32, no. 7. – P. 2809–2824. – DOI: http://dx.doi.org/10.1109/TNNLS.2020.3009047.
17. Wang R. et al. Stochastic circuit design and performance evaluation of vector quantization for different error measures // IEEE Trans. Very Large Scale Integr. (VLSI) Syst. – 2016. – Vol. 24, no. 11. – P. 3169–3183.
18. Gonzalez-Guerrero L. P. et al. ASC-FFT: Area-efficient low-latency FFT design based on asynchronous stochastic computing // Proc. 2019 Latin American Symposium on Circuits and Systems (LASCAS). – 2019.
19. Zhang Q., Chen Y., Li S., Zeng X., Parhi K. K. A high-performance stochastic LDPC decoder architecture designed via correlation analysis // IEEE Trans. Circuits Syst. I. Regul. Pap. – 2020. – Vol. 67, no. 12. – P. 5429–5442. – DOI: http://dx.doi.org/10.1109/TCSI.2020.3003457.

20. Mahesh V. V., Shahana T. K. Design and synthesis of FIR filter banks using area and power efficient stochastic computing // Proc. 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4). – 2020. – P. 662–666. – DOI: http://dx.doi.org/10.1109/WorldS450073.2020.9210403.
21. Wang Y., Li H., Li X. Real-time meets approximate computing: An elastic cnn inference accelerator with adaptive trade-off between QoS and QoR // Proc. DAC. – 2017. – P. 1–6.
22. Mavridis C. N., Baras J. S. Vector quantization for adaptive state aggregation in reinforcement learning // Proc. 2021 American Control Conference (ACC). – 2021. – P. 2187–2192. – DOI: http://dx.doi.org/10.23919/ACC50511.2021.9483052.
23. Monardo V., Iyer A., Donegan S., Graef M. D., Chi Y. Plug-and-play image reconstruction meets stochastic variance-reduced gradient methods // Proc. 2021 IEEE International Conference on Image Processing (ICIP). – 2021. – P. 2868–2872. – DOI: http://dx.doi.org/10.1109/ICIP42928.2021.9506021.
24. Li P., Lilja D., Qian W., Bazargan K., Riedel M. D. Computation on stochastic bit streams digital image processing case studies // IEEE Trans. Very Large Scale Integr. (VLSI) Syst. – 2014. – Vol. 22, no. 3. – P. 449–462.
25. Chen K. C., Wu C. H. High-accurate stochastic computing for artificial neural network by using extended stochastic logic // Proc. 2021 IEEE International Symposium on Circuits and Systems (ISCAS). – 2021. – P. 1–4. – DOI: http://dx.doi.org/10.1109/ISCAS51556.2021.9401418.
26. Xia Z., Chen J., Huang Q., Luo J., Hu J. Neural synaptic plasticity-inspired computing: A high computing efficient deep convolutional neural network accelerator // IEEE Trans. Circuits Syst. I. Regul. Pap. – 2021. – Vol. 68, no. 2. – P. 728–740. – DOI: http://dx.doi.org/10.1109/TCSI.2020.3039346.
27. Wang H., Luo Y., An W., Sun Q., Xu J., Zhang L. PID controller-based stochastic optimization acceleration for deep neural networks // IEEE Trans. Neural Netw. Learn. Syst. – 2020. – Vol. 31, no. 12. – P. 5079–5091. – DOI: http://dx.doi.org/10.1109/TNNLS.2019.2963066.

28. Liu X., Parhi K. K. Molecular and DNA artificial neural networks via fractional coding // IEEE Trans. Biomed. Circuits Syst. – 2020. – Vol. 14, no. 3. – P. 490–503. – DOI: http://dx.doi.org/10.1109/TBCAS.2020.2979485.
29. Zhang Y., Wang R., Zhang X., Wang Y., Huang R. Parallel hybrid stochastic-binary-based neural network accelerators // IEEE Trans. Circuits Syst. II. – 2020. – Vol. 67, no. 12. – P. 3387–3391. – DOI: http://dx.doi.org/10.1109/TCSII.2020.2994464.
30. Li J. et al. Towards acceleration of deep convolutional neural networks using stochastic computing // Proc. 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). – 2017. – P. 115–120. – DOI: http://dx.doi.org/10.1109/ASPDAC.2017.7858306.
31. Ardakani A., Leduc-Primeau F., Onizawa N., Hanyu T., Gross W. J. VLSI implementation of deep neural network using integral stochastic computing // IEEE Trans. Very Large Scale Integr. (VLSI) Syst. – 2017. – Vol. 25, no. 10. – P. 2688–2699. – DOI: http://dx.doi.org/10.1109/TVLSI.2017.2654298.
32. Carter W. et al. A user programmable reconfigurable gate array // Proc. Custom Integr. Circuits Conf. – 1986. – P. 233–235.
33. Trimberger S. M. Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology // Proc. IEEE. – 2015. – Vol. 103, no. 3. – P. 318–331. – DOI: http://dx.doi.org/10.1109/JPROC.2015.2392104.
34. Zhou Z., Berger M. S., Yan Y. Mapping TSN traffic scheduling and shaping to FPGA-based architecture // IEEE Access. – 2020. – Vol. 8. – P. 221503–221512. – DOI: http://dx.doi.org/10.1109/ACCESS.2020.3043887.
35. Liu C., Wang C., Luo J. Large-scale deep learning framework on FPGA for fingerprint-based indoor localization // IEEE Access. – 2020. – Vol. 8. – P. 65609–65617. – DOI: http://dx.doi.org/10.1109/ACCESS.2020.2985162.
36. Li B., Najafi M. H., Lilja D. J. An FPGA implementation of a restricted Boltzmann machine classifier using stochastic bit streams // Proc. IEEE 26th Int. Conf. Appl.-Specific Syst. Archit. Process (ASAP). – 2015. – P. 68–69.

37. Salehi S. A. Low-cost stochastic number generators for stochastic computing // IEEE Trans. Very Large Scale Integr. (VLSI) Syst. – 2020. – Vol. 28, no. 4. – P. 992–1001. – DOI: http://dx.doi.org/10.1109/TVLSI.2019.2963678.
38. Ichihara H., Ishii S., Sunamori D., Iwagaki T., Inoue T. Compact and accurate stochastic circuits with shared random number sources // Proc. 2014 IEEE 32nd International Conference on Computer Design (ICCD). – Seoul, 2014. – P. 361–366. – DOI: http://dx.doi.org/10.1109/ICCD.2014.6974706.
39. Joe H., Kim Y. Novel stochastic computing for energy-efficient image processors // Electronics. – 2019. – Vol. 8, no. 6. – P. 720.
40. Lee V. T., Alaghi A., Hayes J. P., Sathe V., Ceze L. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing // Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE). – Lausanne, 2017. – P. 13–18. – DOI: http://dx.doi.org/10.23919/DATE.2017.7926951.
41. Liu S., Gross W. J., Han J. Introduction to dynamic stochastic computing // IEEE Circuits Syst. Mag. – 2020. – Vol. 20, no. 3. – P. 19–33. – DOI: http://dx.doi.org/10.1109/MCAS.2020.3005483.
42. Liu Y., Liu S., Wang Y., Lombardi F., Han J. A stochastic computational multi-layer perceptron with backward propagation // IEEE Trans. Comput. – 2018. – Vol. 67, no. 9. – P. 1273–1286. – DOI: http://dx.doi.org/10.1109/TC.2018.2817237.
43. Valencia D., Fard S. F., Alimohammad A. An artificial neural network processor with a custom instruction set architecture for embedded applications // IEEE Trans. Circuits Syst. I. Regul. Pap. – 2020. – Vol. 67, no. 12. – P. 5200–5210. – DOI: http://dx.doi.org/10.1109/TCSI.2020.3003769.
44. Park J., Sung W. FPGA based implementation of deep neural networks using on-chip memory only // Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – 2016. – P. 1011–1015. – DOI: http://dx.doi.org/10.1109/ICASSP.2016.7471828.
45. Abdelsalam A. M., Boulet F., Demers G., Pierre Langlois J. M., Cheriet F. An efficient FPGA-based overlay inference architecture for fully connected DNNs // Proc.

International Conference on ReConFigurable Computing and FPGAs (ReConFig). – 2018. – P. 1–6. – DOI: http://dx.doi.org/10.1109/RECONFIG.2018.8641735.

46. Mittal H., Sharma A., Perumal T. FPGA implementation of handwritten number recognition using artificial neural network // Proc. 2019 IEEE 8th Global Conference on Consumer Electronics (GCCE). – 2019. – P. 1010–1011. – DOI: http://dx.doi.org/10.1109/GCCE46687.2019.9015236.
47. Salapura V., Gschwind M., Maischberger O. A fast FPGA implementation of a general-purpose neuron // Proc. FPL. – 1994.
48. Szabo T., Antoni L., Horvath G., Feher B. A full-parallel digital implementation for pre-trained NNs // Proc. IJCNN. – 2000.
49. Izeboudjen N., Farah A., Titri S., Boumeridja H. Digital implementation of artificial neural networks: From VHDL description to FPGA implementation // Proc. IWANN. – 1999.
50. Gschwind M., Salapura V., Maischberger O. A generic building block for Hopfield neural networks with on-chip learning // Proc. ISCAS. – 1996.
51. Omondi A. R., Rajapakse J. C., Bajger M. FPGA neurocomputers // FPGA Implementations of Neural Networks / Ed. by A. R. Omondi, J. C. Rajapakse. – Boston, MA: Springer, 2006. – DOI: http://dx.doi.org/10.1007/0-387-28487-7_1.
52. Eldredge J. G., Hutchings B. L. RRANN: A hardware implementation of the backpropagation algorithm using reconfigurable FPGAs // Proc. IEEE World Conference on Computational Intelligence. – 1994.
53. Gadea R., Cerda J., Ballester F., Mocholi A. Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation // Proc. ISSS. – 2000. – P. 225–230.
54. Salapura V. Neural networks using bit-stream arithmetic: A space efficient implementation // Proc. IEEE Int. Conf. on Circuits and Systems. – 1994.
55. van Daalen M., Jeavons P., Shawe-Taylor J. A stochastic neural architecture that exploits dynamically reconfigurable FPGAs // Proc. IEEE Workshop on FPGAs for Custom Computing Machines. – 1993. – P. 202–211.
56. Maeda Y., Tada T. FPGA implementation of a pulse density neural network using simultaneous perturbation // Proc. IJCNN. – 2000.

57. Guimarães C. J. B. V., Fernandes M. A. C. Real-time neural networks implementation proposal for microcontrollers // Electronics. – 2020. – Vol. 9. – P. 1597. – DOI: http://dx.doi.org/10.3390/electronics9101597.
58. McCulloch W. S., Pitts W. A logical calculus of the ideas immanent in nervous activity // Bull. Math. Biophys. – 1943. – Vol. 5. – P. 115–133. – DOI: http://dx.doi.org/10.1007/BF02478259.
59. Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press, 2016. – P. 196.
60. Joshi V. et al. ESSOP: Efficient and scalable stochastic outer product architecture for deep learning // Proc. IEEE International Symposium on Circuits and Systems (ISCAS). – 2020. – P. 1–5. – DOI: http://dx.doi.org/10.1109/ISCAS45731.2020.9180872.
61. Alaghi A., Hayes J. P. Survey of stochastic computing // ACM Trans. Embed. Comput. Syst. – 2013. – Vol. 12, no. 2. – P. 92.
62. Dickson J. A., McLeod R. D., Card H. C. Stochastic arithmetic implementations of neural networks with in situ learning // Proc. IEEE Int. Conf. Neural Netw. – 1993. – Vol. 2. – P. 711–716.
63. Ting P. S., Hayes J. P. Stochastic logic realization of matrix operations // Proc. 2014 17th Euromicro Conference on Digital System Design (DSD). – 2014. – P. 356–364.
64. Kim K., Lee J., Choi K. Approximate derandomizer for stochastic circuits // Proc. ISOCC. – 2015.
65. Canals V., Morro A., Oliver A., Alomar M. L., Rossello J. L. A new stochastic computing methodology for efficient neural network implementation // IEEE Trans. Neural Netw. Learn. Syst. – 2016. – Vol. 27, no. 3. – P. 551–564.
66. LeCun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition // Proc. IEEE. – 1998. – Vol. 86, no. 11. – P. 2278–2324.
67. Lecun Y., Cortes C. The MNIST database of handwritten digits. – 2010. – URL: http://yann.lecun.com/exdb/mnist/ (дата звернення: 05.05.2026).
68. Запорожець О. І., Халмурадов В. І., Применко В. І. Безпека життєдіяльності : підручник. Київ : Центр учбової літератури, 2013. С. 84–130.
69. Кодекс цивільного захисту України. URL: https://zakon.rada.gov.ua/laws/show/5403-17#Text (дата звернення: 05.05.2026).

70. Про затвердження Положення про підсистему реагування на надзвичайні ситуації, проведення аварійно-рятувальних та інших невідкладних робіт єдиної державної системи цивільного захисту : наказ Міністерства внутрішніх справ України від 04.05.2016 № 356.
71. Про затвердження Положення про штаб з ліквідації наслідків надзвичайної ситуації та видів оперативно-технічної і звітної документації штабу з ліквідації наслідків надзвичайної ситуації : наказ Міністерства внутрішніх справ України від 26.12.2014 № 1406.
72. ДСТУ ISO 14698-1:2008 Якість повітря. Чисті приміщення та відповідні контрольовані середовища. Контролювання біозабруднень. Частина 1. Загальні принципи та методи (ISO 14698-1:2003, IDT).
73. ДСТУ EN ISO 14123-1:2016 Безпечність машин. Зниження ризику для здоров’я, спричинюваного небезпечними речовинами, виділюваними машинами.
74. ДБН В.2.5-67:2013 Опалення, вентиляція та кондиціонування.
75. ДСТУ EN 149:2017 Засоби індивідуального захисту органів дихання. Фільтрувальні півмаски для захисту від аерозолів. Вимоги, випробування, маркування (EN 149:2001+A1:2009, IDT).
76. ДСТУ EN 166:2017 Засоби індивідуального захисту очей. Технічні умови (EN 166:2001, IDT).
77. ДСТУ EN 14605:2017 Одяг захисний. Захист від рідких хімічних речовин.
78. ДСТУ EN ISO 7010:2019 Графічні символи. Кольори та знаки безпеки. Зареєстровані знаки безпеки
79. ДСТУ ISO 16000. Повітря в приміщенні.
80. ДСТУ ISO 45001:2019 Системи управління охороною здоров’я та безпекою праці. Вимоги та настанови щодо застосування (ISO 45001:2018, IDT).
81. ДСТУ EN 60068-2-1:2022 Випробування на вплив зовнішніх чинників.
82. ДСТУ EN ISO 14123-1:2018 Безпечність машин. Зниження ризику для здоров’я від небезпечних речовин, що виділяють машини
83. Левченко О. Г., Полукаров О. І., Арламов О. Ю., Полукаров Ю. О., Землянська О. В. Охорона праці та цивільний захист : підручник для студентів бакалаврату. Київ : Каравела, 2021. 352 с.

ДОДАТКИ

IX Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.942
Момотюк В.–ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОЇ ПЛАТФОРМИ ДЛЯ МОНІТОРИНГУ РИЗИКІВ СТОХАСТИЧНИХ СИСТЕМ З ВИКОРИСТАННЯМ НЕЙРОМЕРЕЖЕВИХ МОДУЛІВ

Науковий керівник: к.т.н., доцент Стоянов Ю. М.

Momotiuk V.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF A SOFTWARE PLATFORM FOR RISK MONITORING OF STOCHASTIC SYSTEMS USING NEURAL NETWORK MODULES

Supervisor: Ph.D. in Engineering, associate professor Stoianov Ju .M.

Ключові слова: програмна платформа, моніторинг ризиків, стохастичні системи, штучні нейронні мережі, стохастичні обчислення, глибоке навчання
Keywords: software platform, risk monitoring, stochastic systems, artificial neural networks, stochastic computing, deep learning

Сьогодні штучні нейронні мережі (ШНМ) та алгоритми глибокого навчання є основою для вирішення складних завдань класифікації, розпізнавання образів та аналітики в системах штучного інтелекту. Проте розмір сучасних глибоких нейронних мереж часто перевищує тисячі вузлів, що робить суто програмні рішення на стандартних процесорах надто повільними для застосунків, які потребують тривалого навчання та миттєвого використання (інференсу). Серйозним викликом у застосунках реального часу стає апаратна реалізація багатошарових нейронних мереж (наприклад, багатошарових перцептронів).

Традиційні архітектури вимагають використання десятків тисяч складних апаратних множників, що призводить до величезного споживання енергії та надмірного використання ресурсів кристала, тому не підходять для портативних рішень. Серед різноманітних методів оптимізації величезну увагу дослідників привернули обчислювальні архітектури на базі стохастичних обчислень (Stochastic Computing, SC).

Цей підхід дозволяє замінити складні арифметичні операції на прості логічні вентилі (наприклад, вентилі XNOR замість великих мультиплікаторів), що кардинально знижує апаратні витрати та робить систему енергоефективною. Однак, функціонування систем на базі стохастичних обчислень містить у собі серйозні внутрішні ризики та обмеження.

Головною проблемою є повільна збіжність: для генерації відносно точного результату системі та традиційним оцінювачам ймовірності (PE) потрібні мільйони тактових імпульсів. Крім того, виникають проблеми "масштабування вниз" (scale-down) та висока ймовірність флуктуаційних похибок при математичних операціях. Ідеальною платформою для розгортання таких нейромереж є програмовані логічні інтегральні