МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

На правах рукопису
УДК 681.325

Івасьєв Степан Володимирович

МЕТОДИ ТА ОБЧИСЛЮВАЛЬНІ ЗАСОБИ РІШЕННЯ ЗАДАЧ ТЕОРІЇ ЧИСЕЛ У БАЗИСАХ РАДЕМАХЕРА - КРЕСТЕНСОНА

Спеціальність 05.13.05 – комп'ютерні системи та компоненти

Дисертація на здобуття наукового ступеня
кандидата технічних наук

Ідентичність всіх примірників дисертації засвідчую:
Вчений секретар спеціалізованої вченої ради К 57.082.02
(Ясній В.В.)

Науковий керівник
доктор технічних наук, професор
Николайчук Ярослав Миколайович

ТЕРНОПІЛЬ – 2016

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ……. 5
ВСТУП…… 6
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ ОПРАЦЮВАННЯ БАГАТОРОЗРЯДНИХ ІНФОРМАЦІЙНИХ КОДІВ ДЛЯ ЗАДАЧ ТЕОРІЇ ЧИСЕЛ……. 13
1.1 Характеристики архітектур процесорів опрацювання багаторозрядних інформаційних потоків……..… 13
1.2 Аналіз структур та системних характеристик багаторозрядних перемножувачів у базисі Радемахера……..……... 18
1.3 Аналіз архітектур та характеристик спецпроцесорів кореляційного, спектрального та ентропійного опрацювання даних ……...…… 25
1.4 Способи кодування інформаційних потоків в комп'ютерних системах на основі теоретико-числових базисів Радемахера та Крестенсона…….. 29
1.5 Аналіз існуючих бібліотек для роботи з багаторозрядними числами, постановка задачі дослідження ……..…… 32
ВИСНОВКИ ДО РОЗДІЛУ 1……..……. 36
РОЗДІЛ 2 РОЗРОБКА ТЕОРЕТИЧНИХ ЗАСАД, МЕТОДІВ ТА УДОСКОНАЛЕНИХ АЛГОРИТМІВ РІШЕННЯ ЗАДАЧ ТЕОРІЇ ЧИСЕЛ У ТЕОРЕТИКО-ЧИСЛОВИХ БАЗИСАХ РАДЕМАХЕРА ТА КРЕСТЕНСОНА…….. 37
2.1 Метод знаходження залишку багаторозрядного числа у базисі Радемахера……..……..... 37
2.2 Векторно-модульний метод модулярного множення багаторозрядних чисел ……....…… 40
2.3 Метод перевірки багаторозрядних чисел на простоту…….. 44
2.4 Дослідження обчислювальних процесів пошуку квадратичних

лишків багаторозрядних чисел……..……. 51
2.5 Метод компактного кодування простих багаторозрядних чисел…….. 56
ВИСНОВКИ ДО РОЗДІЛУ 2…….. 66
РОЗДІЛ 3 РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДІВ ОПРАЦЮВАННЯ БАГАТОРОЗРЯДНИХ ЧИСЕЛ У ЗАДАЧАХ ФАКТОРИЗАЦІЇ НА ОСНОВІ ТЕОРЕТИКО – ЧИСЛОВОГО БАЗИСУ ХААРА – КРЕСТЕНСОНА…….……..... 67
3.1 Метод визначення околу рішення задачі факторизації ……...…….. 67
3.2 Дослідження збіжності екстремумів залишкової функції в околі розв’язку задачі факторизації…….. 76
3.3 Метод факторизації на основі використання квадратичних лишків…. 88
3.4 Розробка методу та моделі локалізації розрядності розв’язку задачі факторизації багаторозрядних чисел……..……... 99
ВИСНОВКИ ДО РОЗДІЛУ 3…….. 105
РОЗДІЛ 4 РЕАЛІЗАЦІЯ ПРОГРАМНИХ ТА АПАРАТНИХ ЗАСОБІВ ОПРАЦЮВАННЯ БАГАТОРОЗРЯДНИХ ЧИСЕЛ У БАЗИСАХ РАДЕМАХЕРА ТА ХААРА – КРЕСТЕНСОНА…….. 106
4.1 Проектування програмних рішень опрацювання багаторозрядних чисел……....…… 106
4.2 Реалізація апаратних компонентів виконання операцій піднесення до квадрату за модулем в теоретико - числовому базисі Хаара - Крестенсона..... 121
4.3 Розробка алгоритму та реалізація програми компактного кодування багаторозрядних простих чисел ……..……...….... 128
4.4 Пристрій компактного кодування багаторозрядних простих чисел……...……... 132
4.5 Апаратна реалізація методу факторизації багаторозрядних чисел …. 138
ВИСНОВКИ ДО РОЗДІЛУ 4…….. 146
ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ…… 147

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……..…….. 149
Додаток А. Структура цифрового автокорелятора……..... 168
Додаток Б. Схема пристрою спектрального-косинусного перетворення... 169
Додаток В. Функціональна схема пристрою перетворення чисел з позиційної системи в систему залишкових класів...…..…….…….. 170
Додаток Г. Структурна схема однорозрядного рандомізатора – мультиплексора ..…….. 171
Додаток Д. Теоретичні засади пошуку символів Якобі та Лежандра …… 172
Додаток Е. Аналіз методів факторизації та їх обчислювальної складності ……. 177
Додаток Ж. Лістинг бібліотеки для опрацювання багаторозрядних чисел…….. 182
Додаток И. Лістинг додатку «Імовірнісний тест простоти»...…… 200
Додаток К. Лістинг додатку «Факторизація багаторозрядних чисел»…… 202
Додаток М. Моделі фракталів в околі рішень задачі факторизації……..... 207
Додаток Н. Реалізація пристрою факторизації на ПЛІС……. 211
Додаток П. Технологічна схема пристрою факторизації……. 215
Додаток Р. Результати тестування пристрою факторизації……..... 216
Додаток Т. Акти впровадження результатів дисертаційної роботи……... 217

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІП – інформаційні потоки;

ТЧБ – теоретико-числові базиси;

БРЧ – багаторозрядні числа;

БПЧ – багаторозрядні прості числа;

АЦК – аналого-цифровий кодер;

АЦП – аналого-цифровий перетворювач;

БОС – багатопроцесорні обчислювальні системи;

ДФ СЗК – досконала форма систем залишкових класів;

ЕЦП – електронно-цифровий підпис;

КМ – комп'ютерні мережі;

КС – комп'ютерні системи;

КТЗ – Китайська теорема про залишки;

НСД — найбільший спільний дільник;

ПКД – пам'ять колективного доступу;

ПЛІС – програмовані логічні інтегральні схеми;

ПОКС – проблемно-орієнтовані комп'ютерні системи;

СР – спецпроцесори;

РКС – розподілена комп'ютеризована система;

СЗК – система залишкових класів;

СКС – спеціалізовані комп'ютерні системи;

СО – системні об'єкти;

СОІ – середовище обробки інформації;

СПІ – середовище передавання інформації;

ФІП – формувачі інформаційних потоків;

RSA — алгоритм Рівеста-Шаміра-Адлемана;

MIPS – мільйон повідомлень в секунду (Million Instruction Per Second).

ВСТУП

Актуальність теми. Важливим напрямком розвитку досліджень у галузі методів та програмно-апаратних засобів опрацювання цифрових даних є вдосконалення алгоритмів, які використовуються в сучасних комп'ютерних системах (КС), та розвиток математичних основ теорії чисел на базі кодових систем різних теоретико-числових базисів (ТЧБ), до яких належать: унітарний, Хаара, Радемахера, Крестенсона, Уолша, Галуа тощо. Вдосконаленням цього класу обчислень на основі названих ТЧБ є створення високопродуктивних компонентів та спецпроцесорів міжбазисних перетворень та рішення задач теорії чисел. Розробка відповідних програмних та апаратних обчислювальних засобів дозволяє підвищити продуктивність, швидкодію, зменшити апаратну та обчислювальну складності методів опрацювання багаторозрядних чисел (БРЧ) при знаходженні набору модулів досконалої та модифікованої досконалої форм системи залишкових класів (СЗК) в різних прикладних задачах обчислювальної математики.

Однією з найбільш складних задач такого класу методів та спецпроцесорів є реалізація алгоритмів опрацювання БРЧ, модульних операцій, модулярного множення та експоненціювання, пошуку квадратичних лишків, пошуку багаторозрядних простих чисел (БПЧ), тестів на простоту тощо.

Успіхи сучасного розвитку мікроелектронної, комп'ютерної техніки та використання потужних багатоядерних суперпроцесорів та кластерів дозволяє ефективно вирішити багато прикладних задач теорії чисел на основі двійкової арифметики ТЧБ Радемахера шляхом створення відповідних алгоритмічних, програмних та мікропроцесорних інструментальних засобів.

Аналіз стану застосування в сучасній комп'ютерній техніці арифметики різних ТЧБ свідчить, що в цілому стан цієї задачі далекий від вирішення. Значний вклад у розвиток теорії методів та високопродуктивних процесорів опрацювання БРЧ на основі ТЧБ Крестенсона внесли відомі українські та

зарубіжні вчені: Акушський І.Й., Палагін О.В., Брюхович Є.І., Романов С.І., Тарасенко В.П., Николайчук Я.М., Мельник А.О., Червяков В.П., Краснобаєв В.А., B. Omondi, N. Szabo, M. Hosseinzadeh, Lenstra H. W., Lakhani G., L.-L. Yang, L. Hanzo та інші.

В той же час, вирішення наукової задачі розвитку обчислювальних методів та засобів на основі математичних засад теорії чисел знаходиться в стадії становлення та направленості до створення і реалізації високопродуктивних обчислювальних засобів та спецпроцесорів, які б забезпечували необхідну в даний час та в майбутньому швидкодію опрацювання БРЧ в умовах постійного зростання вимог до системних характеристик КС.

Тому розробка підходів, методів, алгоритмів та обчислювальних засобів на основі розвитку алгоритмів та математичних положень теорії чисел у різних ТЧБ є актуальною науковою задачею.

Зв'язок роботи з науковими програмами, планами і темами.

Дисертаційна робота виконувалася у рамках науково-дослідних робіт кафедри спеціалізованих комп'ютерних систем Тернопільського національного економічного університету "Розробка теоретичних засад методів формування та цифрового опрацювання даних у розподілених спеціалізованих комп'ютерних системах" (Державний реєстраційний номер 0112U008458) та кафедри комп'ютерної інженерії "Опрацювання багаторозрядних чисел в системі залишкових класів" (Державний реєстраційний номер 0115U001607).

Мета і задачі дослідження.

Метою досліджень є зменшення складності, підвищення продуктивності методів та швидкодії алгоритмів, спеціалізованого програмного і апаратного забезпечення у прикладних задачах теорії чисел.

Для досягнення поставленої мети у дисертаційній роботі необхідно розв'язати наступні задачі:

1) провести аналіз:

- характеристик архітектур процесорів опрацювання БРЧ;

- архітектур та характеристик спецпроцесорів кореляційного та спектрального опрацювання чисел у різних ТЧБ;

- методів кодування БПЧ в КС на основі ТЧБ Радемахера та Крестенсона;

- методів пошуку квадратичних лишків, модулярного множення, факторизації БРЧ в задачах теорії чисел;

2) розробити:

- метод компактного кодування масивів БПЧ;

- метод факторизації БРЧ у базисі Радемахера та Крестенсона;

- схемотехнічні рішення генератора квадратів БРЧ у базисі Хаара - Крестенсона, пристрою компактного кодування БПЧ та процесора факторизації БРЧ у базисі Хаара - Крестенсона;

- програмне забезпечення факторизації БРЧ та дослідження задачі факторизації БРЧ для виявлення околу рішення, компактного кодування БПЧ, множення БРЧ;

3) удосконалити:

- метод визначення околу рішення задачі факторизації;

- векторно-модульний алгоритм модулярного множення;

- метод пошуку квадратичних лишків.

Об'єкт дослідження – процеси програмного та апаратного опрацювання багаторозрядних чисел у спеціалізованих комп’ютерних системах з використанням теоретико – числового базису Радемахера – Крестенсона.

Предмет дослідження – методи, алгоритми та засоби зменшення апаратної та часової складностей при опрацюванні багаторозрядних чисел на основі використання теоретико – числового базису Радемахера – Крестенсона.

Методи дослідження. Для дослідження та аналізу методів, алгоритмів і спеціалізованого програмного, апаратного забезпечення у прикладних задачах теорії чисел використані математичні основи теорії чисел,

арифметика виконання обчислювальних операцій в ТЧБ Радемахера, Хаара та Крестенсона, теорія інформації, теорія алгоритмів, методи програмованого синтезу та реалізації мікропроцесорних засобів на кристалах.

Наукова новизна отриманих результатів полягає в наступному:

Вперше розроблено:

- метод компактного кодування масивів багаторозрядних простих чисел, який, на відміну від існуючих, ґрунтується на зберіганні обмеженого числа динамічно кодованих молодших розрядів їх двійкового представлення у базисі Радемахера та інкрементно розрядно-позиційному нарощенні їх більш високих розрядів, що дало можливості зменшити на порядок обсяг необхідної пам'яті;

- метод факторизації багаторозрядних чисел на основі арифметики теоретико - числового базису Радемахера – Крестенсона шляхом представлення цифрових даних у системі залишкових класів, застосування модульної арифметики, виключення операції добування кореня квадратного та здійснення способу розв'язання задач шляхом асоціативного сканування множини розв'язків в околі рішення, що, в порівнянні з відомими методами, дало можливість зменшити розрядності операндів, спростити алгоритм пошуку факторизованих чисел та підвищити швидкодію алгоритму обчислень.

Удосконалено:

- метод визначення околу рішення задачі факторизації шляхом обчислення двійкового логарифму різниці між відомим числом, що факторизується, та добутком, який обчислюється ітераційно, що дало можливість підвищити швидкодію визначення двійкової розрядності числа ітерацій з поліноміальною складністю в порівнянні з існуючими алгоритмами експоненційної складності.

Отримав подальший розвиток*:*

- векторно-модульний метод модулярного множення, який, на відміну від існуючих, реалізований в теоретико-числовому базисі Радемахера –

Крестенсона і характеризується меншою кількістю операцій додавання та, відповідно, меншою обчислювальною складністю в порівнянні з існуючими алгоритмами множення;

- метод пошуку квадратичних лишків у кодовій системі базису Крестенсона, який, на відміну від існуючих, характеризуються підвищеною швидкодією та меншою обчислювальною складністю.

Практичне значення одержаних результатів.

1. Розроблено високопродуктивні алгоритми факторизації на основі теореми Ферма, модулярного множення, пошуку квадратичних лишків, компактного кодування БПЧ, перевірки БРЧ на простоту згідно арифметики ТЧБ Хаара – Крестенсона та Радемахера - Крестенсона.

2. Розроблено схемотехнічні рішення пристрою кодування БПЧ, що дало можливість зменшити об’єм використання пам’яті при зберіганні БПЧ.

3. Розроблено схемотехнічні рішення високопродуктивного генератора квадратів БРЧ у ТЧБ Радемахера та Крестенсона, що спрощує реалізацію запропонованого способу факторизації згідно удосконаленого алгоритму Ферма.

4. Розроблена функціональна та структурна схема спецпроцесора рішення задач факторизації у базисі Хаара – Крестенсона на основі запропонованих високопродуктивних методів.

Результати досліджень використані при виконанні науково-дослідних робіт, а також у навчальному процесі на кафедрах комп’ютерної інженерії та спеціалізованих комп’ютерних систем Тернопільського національного економічного університету при викладанні дисциплін «Комп’ютерні системи», «Комп’ютерна криптографія» та «Захист інформації в комп’ютерних системах». Отримані результати впроваджені на ТОВ «Стріла» та ТОВ «Інтеграл» для опрацювання інформаційних потоків у дистрибутивних та корпоративних комп’ютерних мережах.

Особистий внесок.

У друкованих працях, опублікованих у співавторстві, автору належить:

[40, 45, 155, 170] - дослідження обчислювальних складностей алгоритмів опрацювання БПЧ та принципів рішення прикладних задач теорії чисел для реалізації спецпроцесорів, [31, 34, 56] - розроблено метод кодування та зберігання БПЧ, [35, 44, 46, 53, 140, 151] - досліджено та реалізовано алгоритм модульного множення з використанням ТЧБ Радемахера-Крестенсона, [30, 56] - розроблено метод локалізації розв'язку задачі факторизації БРЧ, [31, 36, 35, 57, 64, 121, 122, 111] - розроблено метод факторизації БРЧ, [29] – розроблено метод визначення квадратичного лишку у СЗК.

Апробація результатів дисертації. Основні результати дисертаційної роботи апробовані на: проблемно-науковій міжгалузевій конференції «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління», Бучач, Україна, 2010, 2011, 2014; міжнародній конференції «Досвід проектування і застосування САПР в мікроелектроніці», (CADSM-2011, CADSM-2015), Львів-Поляна, Україна, 2011, 2015; міжнародній молодіжній математичній школі "Питання оптимізації обчислень, (ПОО-XXXVII)", Київ, Україна, 2011, 2014, 2015; міжнародній конференції "Сучасні проблеми радіотехніки, телекомунікації та комп'ютерні науки", (TCSET'2012, TCSET'2014), Львів, Славськ, Україна, 2012, 2014; II Всеукраїнській науково-практичній конференції молодих учених та студентів «Інтелектуальні технології в системному програмуванні», (ІТСП-2013), Хмельницький, Україна, 2013; міжнародній конференції "Захист інформації і безпека інформаційних систем - 2014", Львів, Україна, 2014; міжнародній науково-практичній конференції «Сучасні інформаційні та електронні технології», Одеса, Україна, 2014; IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS–2015), Warsaw, Poland, 2015; V Всеукраїнській школі-семінарі молодих вчених та студентів «Сучасні комп'ютерні інформаційні технології» (АСІТ-2015), Тернопіль, Україна, 2015.

Публікації. За матеріалами дисертації опубліковано 20 друкованих праць: 7 статей, з яких 5 у фахових виданнях (1-одноосібна), 13 публікацій - у матеріалах та тезах доповідей конференцій, з яких 3 - у наукових виданнях, що індексуються наукометричною базою Scopus.

Структура дисертації. Дисертаційна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел та додатків. Загальний обсяг дисертації становить 222 сторінки, з них 148 сторінок основного тексту, містить 70 рисунків, 29 таблиць, 14 додатків, список використаних джерел із 172 найменувань.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ОПРАЦЮВАННЯ БАГАТОРОЗРЯДНИХ ІНФОРМАЦІЙНИХ КОДІВ ДЛЯ ЗАДАЧ ТЕОРІЇ ЧИСЕЛ

1.1 Характеристики архітектур процесорів опрацювання багаторозрядних інформаційних потоків

Аналіз літературних джерел методів та алгоритмів розв'язання задач теорії чисел [6, 7, 16, 17] дозволяє класифікувати наступні програмно–апаратні засоби, які використовуються в якості інструментарію реалізації відповідних обчислень.

Найбільш широке застосування при реалізації алгоритмів задач теорії чисел отримали універсальні програмно-апаратні засоби, персональні комп'ютери та потужні обчислювальні кластери, які виконують операції двійкової арифметики ТЧБ Радемахера. Класичну основу комп'ютерних засобів ТЧБ Радемахера складають компоненти, що утворюють базову структуру процесора, яка представлена на рисунку 1.1.

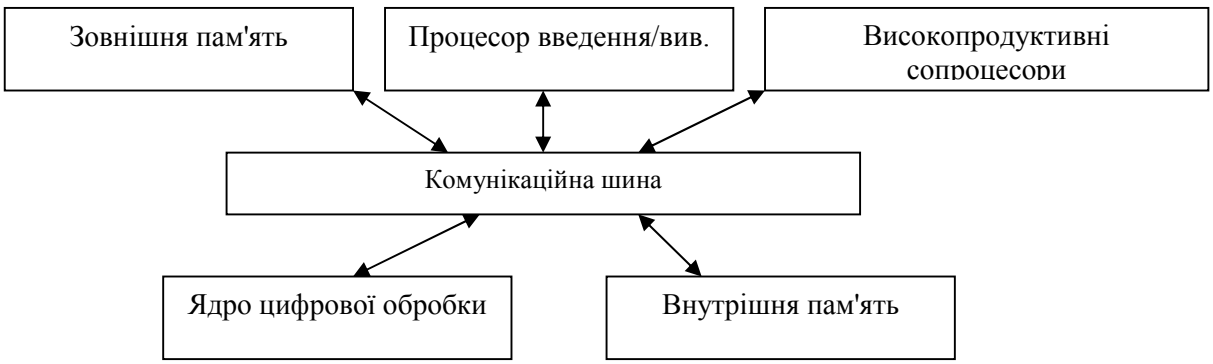


Рисунок 1.1 – Структура процесора цифрової обробки даних.

Як показано в багатьох наукових працях [50, 51, 103, 93], суттєвим недоліком обчислень на основі двійкової арифметики є наявність наскрізних переносів операції додавання, віднімання, піднесення до квадрату, ділення та добування кореня квадратного. Особливо це стосується задач теорії чисел,

які передбачають реалізацію алгоритмів опрацювання БРЧ. Структура компонента “Процесор введення/вив.”, що присутній в структурній схемі рисунка 1.1 деталізовано зображена на рисунку 1.2.

При цьому важливими компонентами такої структури, які дозволяють суттєво підвищити швидкодію обчислень, є високопродуктивні сопроцесори, наприклад, матричного множення, модульної арифметики, конвеєрної обробки, оперативної та асоціативної пам’яті тощо [50, 51].

Досягнення високої продуктивності універсальних програмно-апаратних засобів також забезпечується глибоким розпаралеленням опрацювання потоків даних у процесі введення/виведення [27].

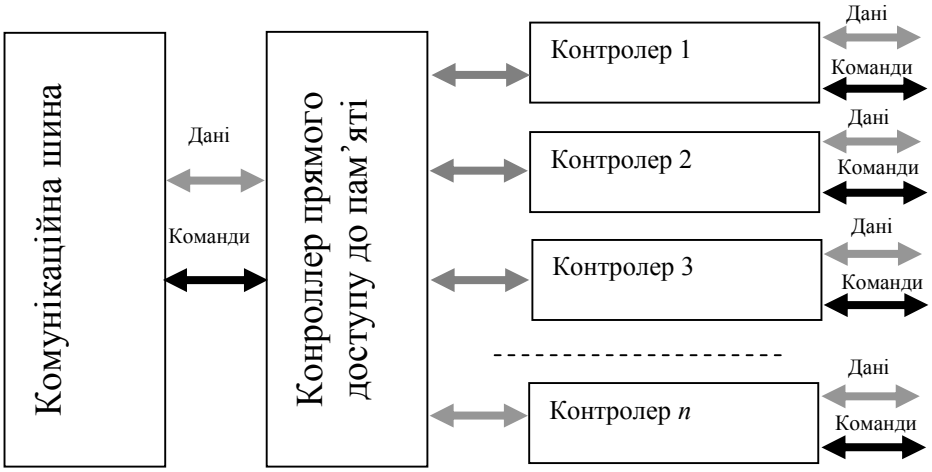


Рисунок 1.2 – Структура процесора введення/виведення

У процесі розвитку універсальних комп’ютерних засобів широке застосування отримали процесори на основі фон Нейманівської та Гарвардської архітектур. При цьому більшими обчислювальними можливостями володіють пристрої гарвардської архітектури завдяки розпаралеленню опрацювання даних. Класифікація гарвардської архітектури процесорів паралельної обробки даних представлена на рисунку 1.3.

MIMD (multiple instruction stream / multiple data stream) реалізують опрацювання багатомірного потоку команд та даних [51, 51, 103, 162, 163].

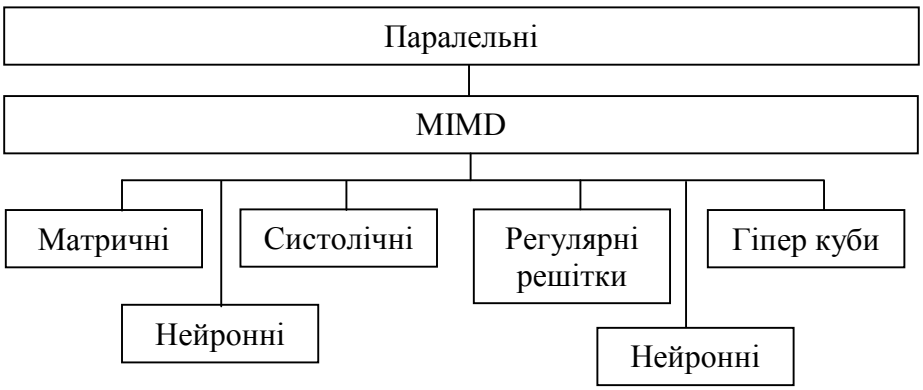


Рисунок 1.3 – Класифікація MIMD - процесорів гарвардської архітектури паралельної обробки даних

Іншим класом високопродуктивних універсальних процесорів є послідовно-паралельні процесори гарвардського класу архітектур, яка зображена на рисунку 1.4.

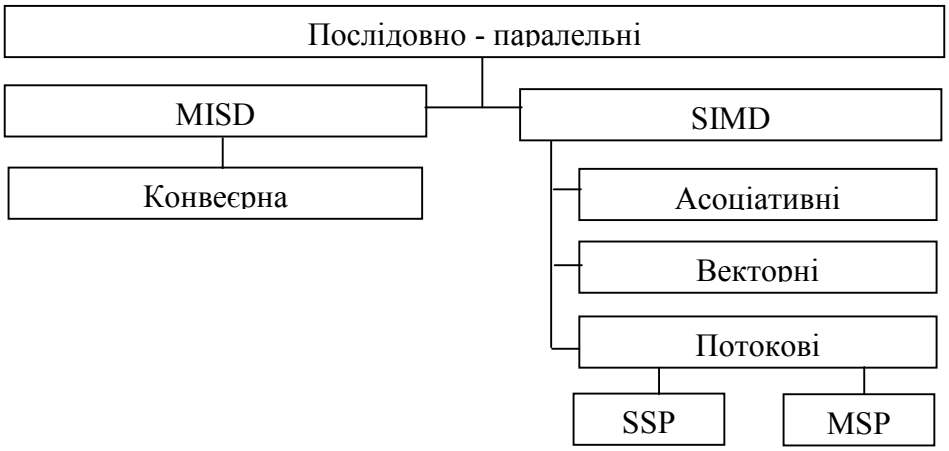


Рисунок 1.4 – Класифікація гарвардської архітектури послідовно – паралельної обробки даних

Аналіз характеристик продуктивності даного класу процесорів послідовно – паралельного опрацювання даних показує, що розподіл на MISD (Multiple Instruction stream, Single Data stream) і SIMD (single instruction, multiple data) відрізняються числом модифікацій. Процесори, які мають архітектуру SIMD, можна класифікувати ще на три різновиди згідно способів опрацювання та типу даних.

На відміну від фон Неймановської архітектури, процесори Гарвардської архітектури забезпечують швидке виконання арифметичних операцій за один машинний такт, що є важливим при реалізації алгоритмів опрацювання задач теорії чисел над багаторозрядними операндами.

На відміну від процесорів архітектури MISD, що використовуються для конвеєрного опрацювання даних, SIMD – процесори можуть бути векторного, асоціативного та потокового типу, в яких всі елементи одночасно виконують одну команду над різними даними – однократний потік команд над багатократним потоком даних.

Такі процесори містять велику кількість операційних пристроїв, які одночасно дозволяють проводити обробку багатьох потоків даних. Особливістю асоціативних процесорів є наявність асоціативної пам'яті, дані з якої вибираються за змістом, а не за адресою [50]. Структура векторного процесора показана на рисунку 1.5.

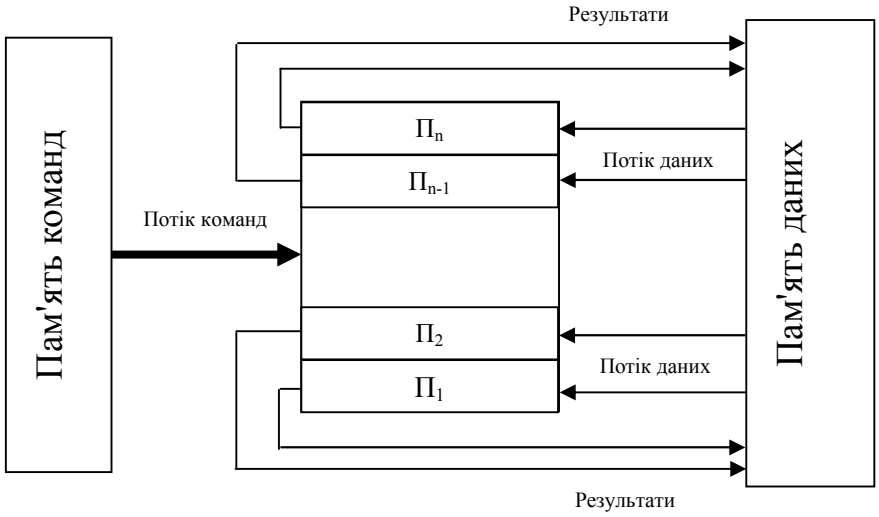


Рисунок 1.5 - Структура векторного процесора

Векторні процесори мають високу продуктивність опрацювання даних за рахунок паралельної обробки великого набору даних однією командою [50]. До даного класу процесорів відносяться процесори фірм NEC, Hitachi [166] та інші.

Існують два класи процесорів: однопотокові (Single-Streaming Processor, SSP [101]) та багатопотокові (Multi-Streaming Processor, MSP

[166]).

До даного класу відносяться процесори фірми Intel, в яких реалізується технологія Streaming SIMD Extensions (SSE, потокова обробка по принципу “одна команда – багато даних”) [103]. Дані процесори знайшли широке застосування при опрацюванні мови спектрального та кореляційного аналізу, шифрування та дешифрування даних, а також при обробці зображень. Прикладом такого класу процесорів є: ILLIAC, IV, ICL, DAP, Goodyear Aerosoace MPP, Connectio Machine-1 [47, 50].

На рисунку 1.6 показана структура конвеєрного процесора MISD, в якому використовуються багато послідовно з’єднаних процесорів, які обробляють один потік даних. Такі процесори представлені типом Cisc Pentium.

Паралельні та матричні архітектури універсальних процесорів типу MIMD реалізують опрацювання багатомірного потоку команд та даних [47, 48, 49].

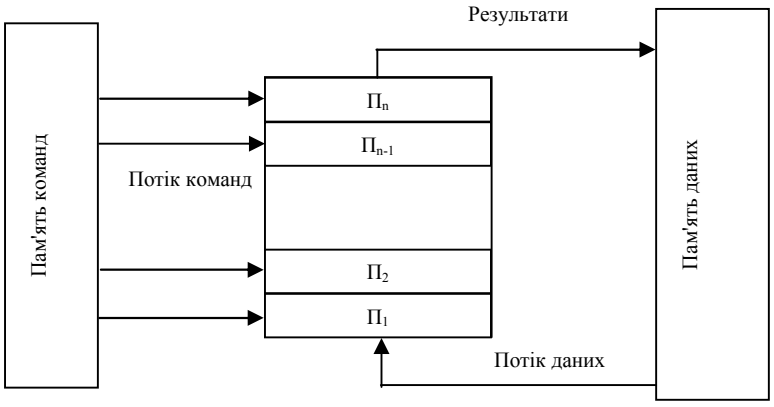


Рисунок 1.6 - Структура конвеєрного процесора MISD

Даний тип процесорів найчастіше використовується в якості сопроцесорів для високопродуктивного виконання операцій множення та додавання.

У таблиці 1.1 приведені характеристики швидкодії універсальних процесорів при виконанні базових операцій двійкової арифметики у ТЧБ

Радемахера (v – кількість вентилів).

Таблиця 1.1 – Швидкодія виконання арифметико-логічних операцій універсальних процесорів у ТЧБ Радемахера

Базові операції	Базис Радемахера
Зсув	v
Рівності	v
По модулю 2	$5\ v$
Логічне множення	v
Логічне додавання	$2v$
Додавання	$2nv$
Віднімання	$(3n+5)v$
Множення	$2v(2n+1)$
Ділення	n^2v
Знакова (старшинства)	nv

Виконана систематизація архітектур універсальних процесорів, які використовуються в персональних комп'ютерах, кластерах та багатоядерних системах показує широкі можливості паралельного опрацювання інформації, які можна використовувати при обробці БРЧ в прикладних задачах теорії чисел.

З таблиці 1.1 видно, що найнижчою швидкодією характеризуються операції додавання, множення та ділення над n – розрядними числами, що суттєво обмежує можливості ефективного застосування універсальних процесорів при вирішенні задач теорії чисел, що базуються на багаторозрядних цифрових даних (512-1024 біт).

1.2 Аналіз структур та системних характеристик багаторозрядних перемножувачів у базисі Радемахера

В алгоритмах опрацювання інформаційних потоків розповсюдженою є операція множення, яка в багатьох випадках визначає загальну швидкодію

алгоритму або ж пристрою.

Значне зниження швидкодії універсальних процесорів обумовлене виконанням операції множення над двійковими числами, які виконуються згідно виразу [47]:

$$Z = A \times B = \left(\sum_{i=0}^{n-1} a_i \times 2^i \right) \times \left(\sum_{i=0}^{n-1} b_i \times 2^j \right) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j \times 2^{i+j} \qquad (1.1)$$

де $a_i, b_j \in \{0,1\}, i, j \in \{0, n-1\}$.

Алгоритм множення виконується шляхом зсуву окремих операндів парних добутків згідно виразу (1.1):

$$\begin{array}{r} 2^{-5} a_4 b_1 + 2^{-6} a_3 b_1 + 2^{-7} a_2 b_1 + 2^{-8} a_1 b_1 \\ 2^{-4} a_4 b_2 + 2^{-5} a_3 b_2 + 2^{-6} a_2 b_2 + 2^{-7} a_1 b_2 \\ 2^{-3} a_4 b_3 + 2^{-4} a_3 b_3 + 2^{-5} a_2 b_3 + 2^{-6} a_1 b_3 \\ 2^{-2} a_4 b_4 + 2^{-3} a_3 b_4 + 2^{-4} a_2 b_4 + 2^{-5} a_1 b_4 \\ \hline 2^{-1} Z_8 + 2^{-2} Z_7 + 2^{-3} Z_6 + 2^{-4} Z_5 + 2^{-5} Z_4 + 2^{-6} Z_3 + 2^{-7} Z_2 + 2^{-8} Z_1 \end{array} \qquad (1.2)$$

Відомий ряд класичних структур матричних перемножувачів [50]. На рисунку 1.7 наведено структуру матричного перемножувача Брауна.

Перемножувач складається з n^2 операцій логічного добутку. Також така схема реалізації перемножувача містить $n^2 - n$ операцій однорозрядного двійкового додавання. Кількість операцій повного двійкового додавання становить $(n^2 - 2n)$. Додатково схема виконує n операцій неповного двійкового додавання.

Для виконання операцій за основу взято базис І-НЕ. Кількість елементів, що виконані в перемножувачі згідно схеми, що на рисунку 1.7, буде рівна $n^2+5n+9(n^2-2n)=10n^2-13n$. Найбільш довгий маршрут проходження інформаційного сигналу для запропонованої схеми складає $(2n-2)$.

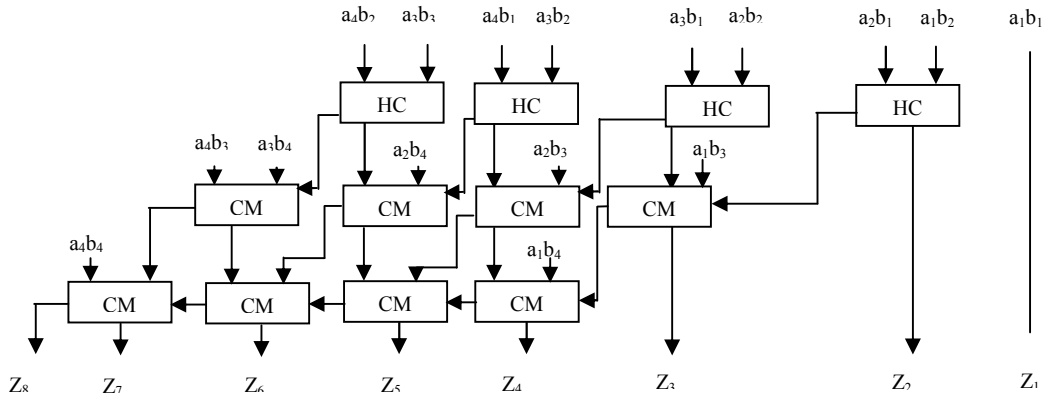


Рисунок 1.7- Структура перемножувача Брауна

Існує також матричний перемножувач з горизонтальним розповсюдженням переносу (рисунок 1.8) [50].

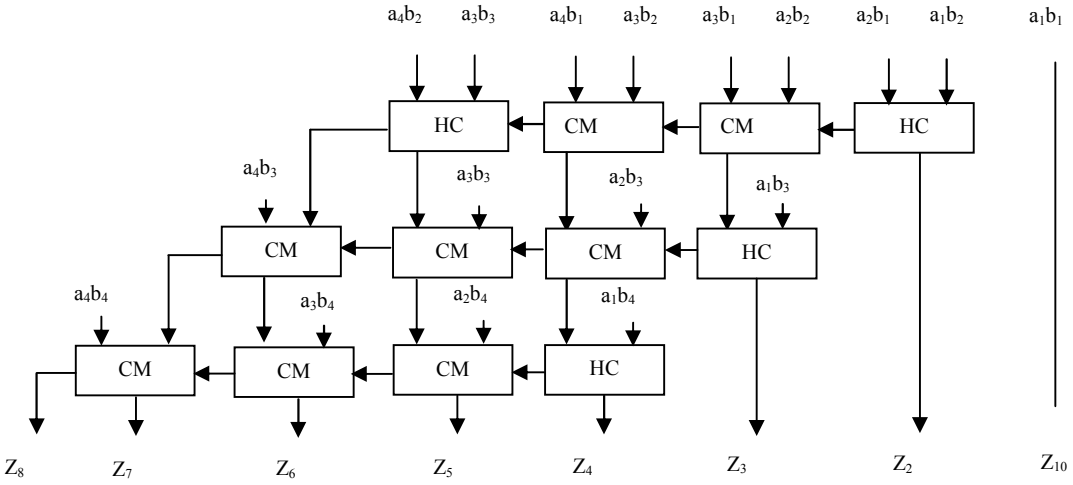


Рисунок 1.8 – Структура матричного перемножувача з горизонтальним розповсюдженням переносу

Даний пристрій складається з n^2 операцій логічного добутку. Містить $(n^2- n)$ операцій однорозрядного двійкового додавання. Кількість операцій, що реалізують повне двійкове додавання, становить $(n^2- 2n)$. Також, згідно схеми пристрою, кількість операцій неповного додавання складає n. Швидкодія реалізації перемножувача складає $(3n-4)$.

Реалізувавши структуру деревоподібного перемножувача, наприклад Даада або Уоллеса, можна зменшити найбільший шлях проходження сигналу

і тим самим пришвидшити виконання операції множення БРЧ.

Перемножувач із структурою дерева Уоллеса вважається одним з найбільш швидкодіючих [50], проте запропонована схема характеризується рядом недоліків, серед яких виявлено непостійність його структури. Даний пристрій складається з n^2 операцій логічного добутку. Кількість операцій однорозрядного двійкового додавання становить $(n^2- n)$.

Щодо операцій повного двійкового додавання, то їх кількість становить $(n^2- 2n)$. Пристрій, реалізований згідно наведеної схеми, виконує також n операцій неповного двійкового додавання. Швидкодія пристрою, реалізованого за подібною схемою, становить $(2n- 2)$ [50].

Схема деревоподібної структури Уоллеса наведена на рисунку 1.9.

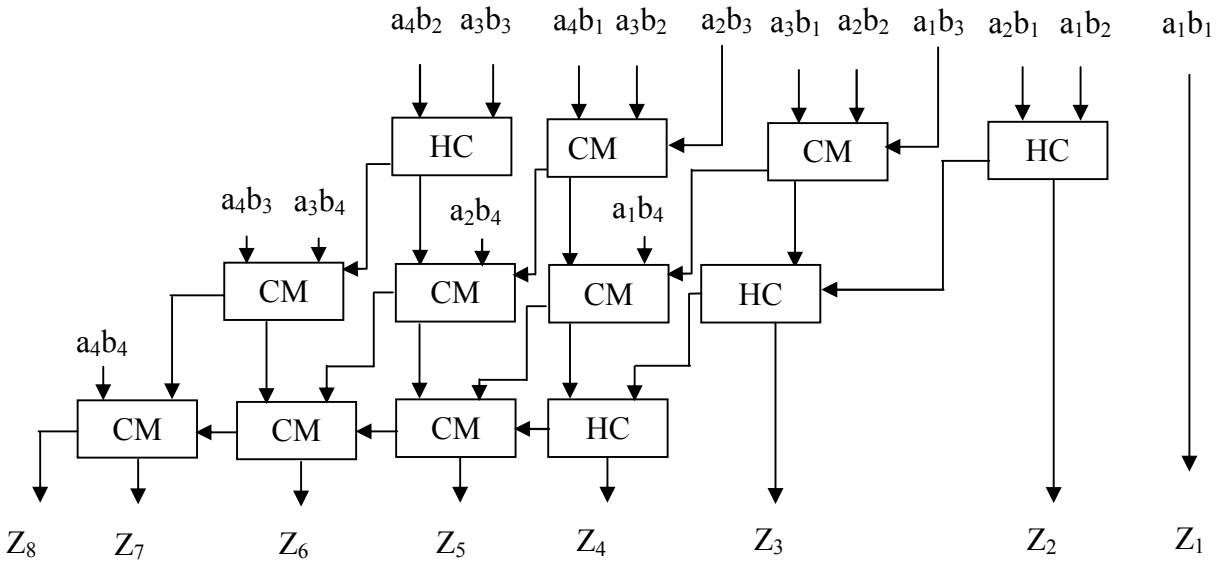


Рисунок 1.9 – Структура перемножувача із структурою дерева Уоллеса

При множенні чисел з невеликою кількістю розрядів, наприклад 16 – 32 біт, ефективним є використання схеми пристрою, запропонованої в [50]. Структура перемножувача Даада представлена на рисунку 1.10.

Перемножувач, реалізований згідно схеми Даада, подібний до схеми Уоллеса, проте реалізований з мінімальною кількістю суматорів, що

приводить до збільшення швидкодії подібної схеми при опрацюванні чисел до 32 біт.

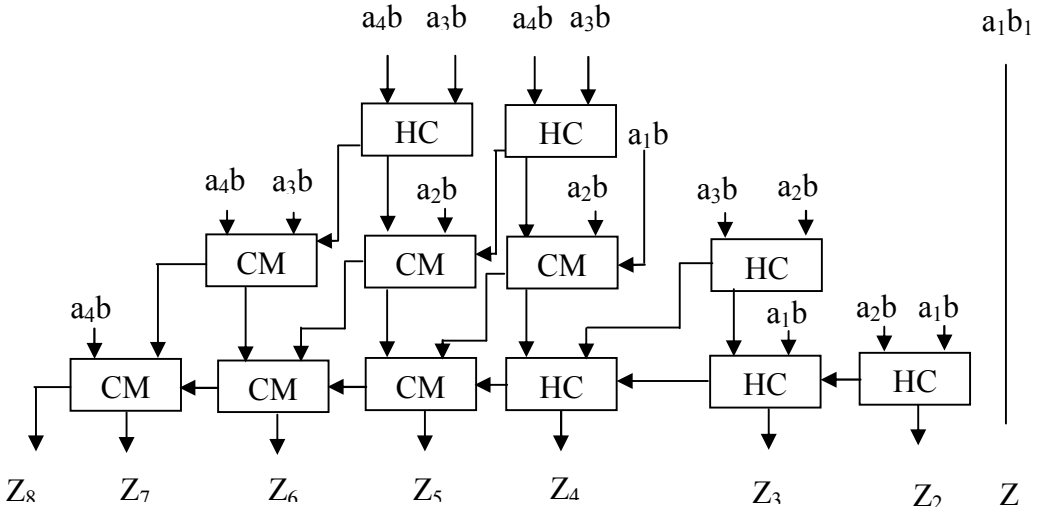


Рисунок 1.10 – Структура перемножувача дерева Даада

Цей пристрій буде складатись з n^2 операцій логічного добутку. Кількість операцій однорозрядного двійкового додавання, як і в перемножувачі, реалізованому згідно схеми Уоллеса, становить $(n^2 - n)$. Кількість повних двійкових додавань становить $(n^2 - 2n)$. Пристрій, як і попередені схеми, буде містити n операцій неповного двійкового додавання. Швидкодія подібного пристрою, як і реалізованого за схемою Уоллеса, складає $(2n - 2)$.

Реалізація перемножувача на основі модульної арифметики ТЧБ Радемахера – Крестенсона або Хаара – Крестенсона має структуру, приведену на рисунку 1.11. Вона реалізована для опрацювання чисел, які відповідають 32-бітному процесору у базисі Радемахера.

Оскільки показаний набір взаємно простих модулів відповідає умовам однозначного кодування даних для 32-бітного процесора перемноження 16-бітних двійкових чисел у базисі Радемахера.

Для виконання умов однозначного кодування даних добуток вибраних взаємно – простих модулів повинен перевищувати 2^{32}.

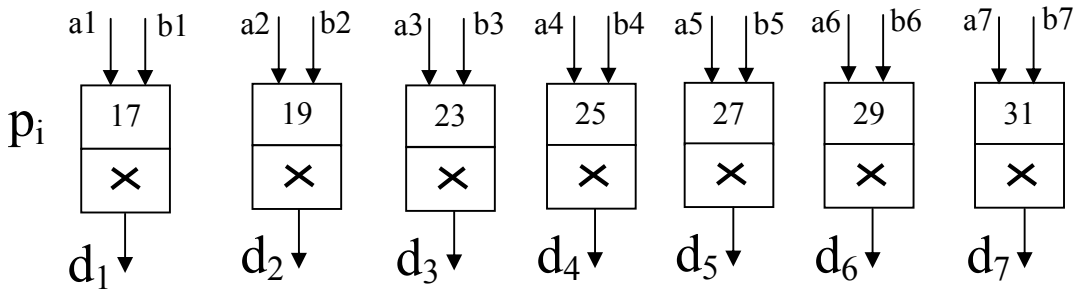


Рисунок 1.11 – Структура перемножувача у базисі Радемахера-Крестенсона

Отже, маємо: $P_1 \cdot P_2 \cdot P_3 \cdot P_4 \cdot P_5 \cdot P_6 \cdot P_7 = P_0$; $P_0 = 17 \cdot 19 \cdot 23 \cdot 25 \cdot 27 \cdot 29 \cdot 31 = 4508102925$.

$2^{32}=4294967296<4508102925$;

Розглянемо приклад перемноження двох 16-бітних чисел у базисі Радемахера – Крестенсона. Нехай $A=21845_{10}=101010101010101_2$, $B=60379_{10} = 1110101111011011_2$, а їх добуток $D=1318979255_{10}=100\ 1110\ 1001\ 1110\ 0000\ 0110\ 1011\ 0111_2$. Для реалізації операції перемноження у базисі Радемахера згідно схеми Брауна отримаємо матрицю:

```
                                1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
                              1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _
                            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 _ _
                          1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _ _ _
                        1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _ _ _ _
                      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 _ _ _ _ _
                    1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _ _ _ _ _ _
                  1 0 1 0 1 0 1 01 0 1 0 1 0 1 _ _ _ _ _ _ _
                1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _ _ _ _ _ _ _ _
              1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _ _ _ _ _ _ _ _ _
            0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 _ _ _ _ _ _ _ _ _ _
          1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _ _ _ _ _ _ _ _ _ _ _
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 _ _ _ _ _ _ _ _ _ _ _ _
      1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _ _ _ _ _ _ _ _ _ _ _ _ _
    1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _
  ---------------------------------------------------------------
  1 0 0 1 1 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 1 1
```

Для приведеного прикладу процесора множення у базисі Радемахера часова складність становить $\tau_{\times} = 2n \cdot \tau_{\sum}$; $\tau_{\sum} = 5\upsilon.$. Тобто $\tau_{\times} = 32 \cdot 5\upsilon = 160\upsilon$,

де υ - швидкодія переключення логічного вентиля мікроелектронної реалізації кристала перемножувача.

Апаратна складність розраховується згідно виразу [48]:

$$A = n^2 + (n^2 - 2n) A_{CM} + 2\mathrm{n} \cdot A_{HC};\ A_{CM} = 12;\ A_{HC} = 5.$$

Апаратна складність такого перемножувача складається з 256 операцій логічного добутку, 240 операцій однорозрядного двійкового додавання, 224 – повного двійкового додавання та 16 операцій неповного двійкового додавання. Отже, апаратна складність перемножувача для приведеного прикладу $A=(256-32)12\upsilon+32\cdot5\upsilon+256\upsilon=3104\ \upsilon$, де υ - логічний елемент або інвертор. Переведемо числа A і B у систему числення залишкових класів ТЧБ Радемахера – Крестенсона згідно заданого набору модулів p_i.

p_i	17	19	23	25	27	29	31
A_i =	0	14	18	20	2	8	21
×	×	×	×	×	×	×	×
B_i =	12	16	4	7	4	1	22
D_i =	0	15	3	15	8	8	28

$d_i = (a_i \times b_i) \bmod p_i;\ i \in 1..7;$

При реалізації алгоритму множення у ТЧБ Радемахера – Крестенсона на універсальному комп'ютері необхідно виконати сім звертань до пам'яті. Оскільки кожен модуль є 5-бітним, то звертання будуть 10-бітні. Так як 32-бітний процесор дозволяє одночасно отримувати 3 результати 5-бітних кодів модульного множення, то загальне число звертань не перевищує 2.

При апаратній реалізації перемножувача в ТЧБ Радемахера – Крестенсона часова складність буде рівна 2υ, незалежно від розрядності перемножуваних чисел, а апаратна складність розраховується згідно виразу:

$$A = \sum_{i=1}^{7} p_i + \sum_{i=1}^{7} p_i^2 = \sum_{i=1}^{7} (p_i^2 + p_i)\ \upsilon.$$

де p_i^2 - число логічних елементів І-НЕ в матриці перемноження;

p_i - число інверторів на виходах матриці перемноження.

Для розглянутого прикладу 32-бітного процесора у ТЧБ Радемахера – Крестенсона апаратна складність буде така:

$(17^2 + 19^2 + 23^2 + 25^2 + 27^2 + 29^2 + 31^2) + (17 + 19 + 23 + 25 + 27 + 29 + 31) =$
$= 289 + 361 + 529 + 625 + 729 + 841 + 961 + 171 = 4506.$

Таким чином, перевищення апаратної складності пристрою множення у ТЧБ Радемахера–Крестенсона по відношенню до апаратної складності матричного перемножувача з горизонтальним розповсюдженням переносу буде рівне 1,45, враховуючи, що матричні модульні перемножувачі ТЧБ Радемахера – Крестенсона мають більш регулярну однорідну структуру по відношенню до структур повних та неповних суматорів, які є компонентами перемножувачів у ТЧБ Радемахера, що спрощує практичну реалізацію таких схем на ПЛІС.

Проведений аналіз архітектур, апаратної складності та швидкодії матричних перемножувачів, які використовуються в універсальних процесорах у якості сопроцесорів прискорювачів, показує, що час виконання елементарних арифметичних операцій, число яких при рішенні задач теорії чисел може складати 2^{100}-2^{1024} у двійковій арифметиці ТЧБ Радемахера, збільшується квадратично. Таким чином, застосування двійкової арифметики при розв'язанні задач теорії чисел є не ефективним та низькошвидкісним. Наприклад, при виконанні задач шифрування інформації, факторизації чисел, знаходження символів Якобі, тестів простоти і швидкодії універсального процесора з тактовою частотою 10-100 ГГц факторизація 1024 бітів зайняла б більше 1000 років.

1.3 Аналіз архітектур та характеристик спецпроцесорів кореляційного, спектрального та ентропійного опрацювання даних.

Особливістю відомих архітектур такого класу спецпроцесорів є їх реалізація на основі арифметики різних ТЧБ: унітарного, Хаара, Радемахера,

Крестенсона та Галуа, оскільки дані ТЧБ породжують відповідні системи числення [54].

Широке практичне застосування отримали мультибазисні системи числення: Радемахера-Крестенсона, Хаара – Крестенсона та Радемахера – Галуа [55].

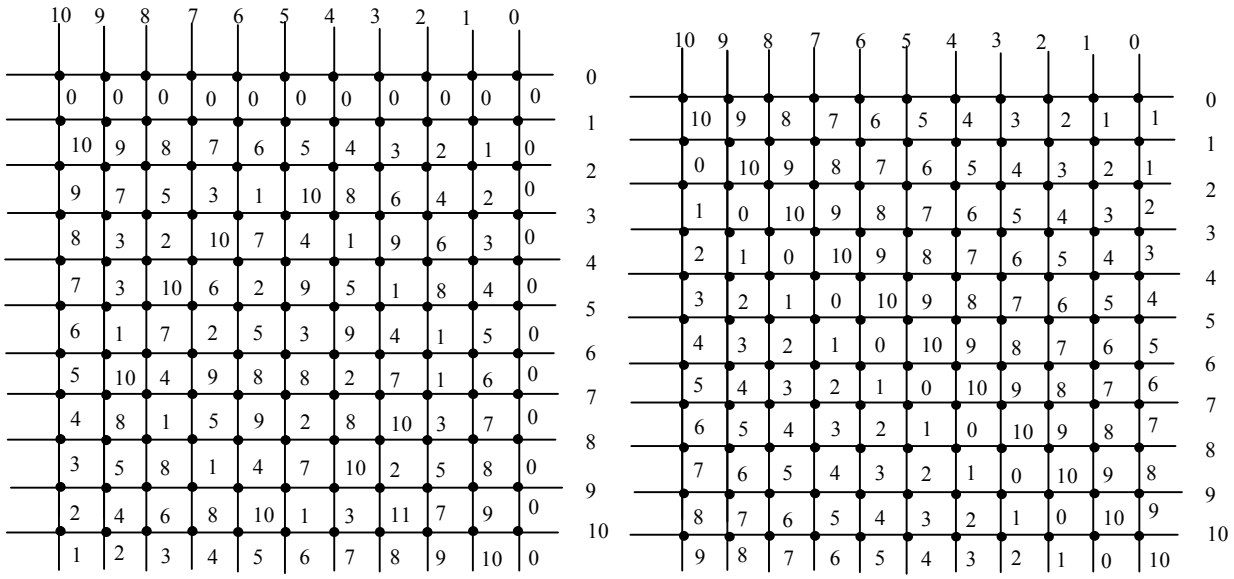
На сьогодні велика увага приділяється пристроям кореляційного опрацювання багаторозрядних інформаційних потоків, які широко використовуються у системах зв'язку, медичних приладах, радіотехнічних системах спостереження за рухомими об'єктами та ін. Це стосується так званих широкосмугових псевдошумових сигналів.

Відома структура швидкодіючого цифрового автокорелятора, в якому опрацювання чисел виконується у СЗК базису Крестенсона. Структура цифрового автокорелятора зображена в додатку А. Коди чисел представляються у ТЧБ Хаара – Крестенсона [70].

Реалізація принципу кодування інформаційних потоків дозволяє ефективно використати переваги швидкодіючої модульної арифметики [70]. Використання кодів Хаара-Крестенсона для представлення інформаційних даних у цифровому автокореляторі дозволяє глибоко розпаралелити обчислювальні процеси і реалізувати виконання операцій додавання та множення матричним методом з часовою затримкою 2 мікротакти.

Згідно [70], пристрій містить: 1 - АЦП паралельного типу з вихідним кодом базису Хаара, 2 - блок пам'яті, 3 - генератор імпульсів, 4 - вентильні матриці перемноження по модулю, 5 - вентильні матриці сумування по модулю, 6 - регістр пам'яті, 7 - шифратор. В пристрої використовуються матриці модульного множення, які наведені на рисунку 1.12.

В результаті аналізу пристроїв кореляційної обробки даних встановлена перевага застосування ТЧБ Хаара – Крестенсона для кодування інформаційних потоків, яка забезпечує прискорення обчислень незалежно від розрядності вхідних інформаційних потоків на два-три порядки [54].



а) б)

Рисунок 1.12 - Матриця модульного множення а) та додавання б) у базисі Хаара – Крестенсона

Відомий пристрій для обчислення спектрального косинусного перетворення [95] в залишкових класах, що складається з блоків множення, суматорів, аналогово-цифрового перетворювача, блоку пам'яті, генератора імпульсів, накопичувальних суматорів, дешифратора. Схема пристрою представлена в додатку Б.

Швидкодія проаналізованого пристрою визначається виразом:

$$\tau_w = (\tau_{АЦП} + 2\tau_x + m\tau\sum + \tau_ш)\upsilon$$

де τ_w - швидкодія пристрою спектрального косинусного перетворення;

υ - час затримки переключення мікроелектронного вентиля ПЛМ або ПЛІС;

$\tau_{АЦП}$=5υ - часова складність паралельного АЦП;

$\tau_x = 2\upsilon$ - часова складність матричного модульного перемножувача;

$m\tau = (m^{+2}) \cdot 2\upsilon$ - часова складність лінійки модульних матричних суматорів з врахуванням нульового та накопичувальних суматорів;

При m=32, часова складність становить 24υ+64υ=88υ.

Серед відомих аналогів пристрої для перетворення десяткового коду в СЗК висвітлені в працях [14, 126, 127]. Недоліком таких схем реалізації є низька швидкодія та висока апаратна складність, яка обумовлена великою кількістю арифметичних пристроїв, відповідно числу модулів СЗК.

Серед пристроїв перетворення чисел з позиційної системи в СЗК з найбільш ефективними параметрами є пристрій, що описаний в роботі [71]. Функціональна схема такого пристрою приведена в додатку В.

Наведена схема організації пристрою відрізняється від аналогів тим, що у ній введені K комутаційних мультиплексорів по кожному модулю СЗК P_j. Перші входи пристрою підключені до відповідних шин вхідного K-розрядного двійкового числа. Другі ж входи i-тих комутаційних мультиплексорів підключені до виходів i-1-ших комутаційних мультиплексорів, починаючи з K-2 до нульового розряду. Серед особливостей пристрою є те, що на одиничний вхід K-1-го комутаційного мультиплексора подається старший розряд, а виходи нульового комутаційного мультиплексора є виходами чисел залишків b_j по модулю P_j у СЗК.

У додатку Г приведена схема компонента пристрою перетворення чисел з позиційної системи в СЗК, а саме, однорозрядного рандомізатора – мультиплексора.

Викладені схемотехнічні та алгоритмічні рішення реалізації спецпроцесорів кореляційного, спектрального та ентропійного опрацювання інформаційних потоків у різних ТЧБ показують, що спецпроцесори у ТЧБ Радемахера–Крестенсона та Хаара–Крестенсона характеризуються найвищою швидкодією і забезпечують підвищення швидкодії опрацювання даних на 2 – 3 порядки у порівнянні з відомими аналогами [62]. Тому застосування ТЧБ Радемахера–Крестенсона та Хаара–Крестенсона для алгоритмічного розв’язання складних задач теорії чисел, побудови відповідних спецпроцесорів є перспективною, актуальною теоретичною та прикладною науковою задачею.

1.4 Способи кодування інформаційних потоків в комп'ютерних системах на основі теоретико - числових базисів Радемахера та Крестенсона

Способи кодування інформаційних потоків визначаються ТЧБ, які застосовуються для їх представлення [6, 38, 39]. Найбільш поширеними ТЧБ в сучасних КС є наступні: унітарний, Хаара, Грея, Радемахера, Крестенсона та Галуа.

Світовий досвід створення процесорів для КС, поряд з застосуванням ТЧБ Радемахера, що призводить до породження двійкової системи числення, за останні роки демонструє тенденцію застосування інших ТЧБ. Реалізація спеціалізованих, кореляційних, спектральних, ентропійних, спецпроцесорів на базі Галуа та проблемно-орієнтованих процесорів цифрової обробки даних часто виконується на базі сумісного використання комбінацій названих ТЧБ, наприклад Радемахера - Крестенсона, Радемахера - Хаара, Крестонсона - Галуа та ін [94].

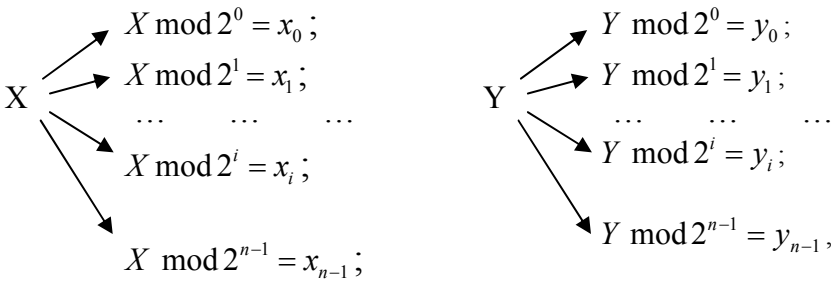
У зв'язку з цим існує проблема глибокого дослідження характеристик різних ТЧБ та граничних можливостей їх застосування для реалізації компонентів як спеціалізованих, так і універсальних процесорів. При цьому перспективним, крім розповсюдженого одновимірного (векторного) представлення чисел та виконання арифметико-логічних операцій у базисі Радемахера, є застосування змішаного базису Радемахера - Крестенсона [89, 94, 98].

Арифметичні операції над двома числами у двійковій системі числення базису Радемахера описуються наступними виразами:

$$X = \sum_{i=0}^{n-1} x_i 2^i \ , \ x_i \in \overline{0,1}; Y = \sum_{i=0}^{n-1} y_i 2^i \ , \ y_i \in \overline{0,1}.$$

Тобто двійкові коди чисел X і У : $X = (x_{n-1}, x_{n-2}, ..., x_i, ..., x_0)$; $Y = (y_{n-1}, y_{n-2}, ..., y_i, ..., y_0)$ визначаються на основі модульних операцій згідно

аналітичних виразів:



Приведені характеристики кодових матриць ТЧБ Радемахера, Крестенсона, які найширше використовуються для кодування та цифрової обробки даних в інформаційних системах, мають властивості мінімальної надлишковості по відношенню до наступних базисів: унітарного, Хаара, Крейга, Уолша та Грея [25].

У таблиці 1.2 N – діапазон представлення чисел, $p_1, p_2, \ldots, p_i, \ldots, p_m$ – набір взаємо простих модулів СЗК базису Крестенсона, $a_i=p_i-1$.

Система числення залишкових класів базису Крестенсона, детально описана Акушським І.Я. та Юдіцьким Д.І. [7].

Таблиця 1.2 – Характеристики кодових матриць ТЧБ

	Радемахера	Крестенсона
Кодові матриці	$M_{Rad} = \begin{vmatrix} 000\ldots00 \\ 000\ldots01 \\ 000\ldots10 \\ 000\ldots11 \\ \ldots\ldots\ldots \\ 111\ldots11 \end{vmatrix}$	$M_{Cres} = \begin{vmatrix} P_1 & P_2 & \ldots & P_m \\ 0 & 0 & \ldots & 0 \\ 1 & 1 & \ldots & 1 \\ 2 & 2 & \ldots & 2 \\ 0 & 3 & \ldots & 3 \\ \ldots & \ldots & \ldots & \ldots \\ 2 & 4 & \ldots & 6 \end{vmatrix}$
n - число активних кодових елементів	$n = \frac{\mathrm{N} \cdot \log_2 N}{2}$	$n = \prod_{i=1}^{m} P_i$
V – об'єм кодової матриці	$\mathrm{V}=\mathrm{N}\cdot\log_2 \mathrm{N}$	$V = \sum_{i=1}^{m} \log_2 (P_i - 1)$

Результати досліджень часової складності реалізації операції множення над числами в базисах Радемахера (O1(n)) та Радемахера - Крестенсона (O2(n)) приведені на рисунку 1.13 [83].

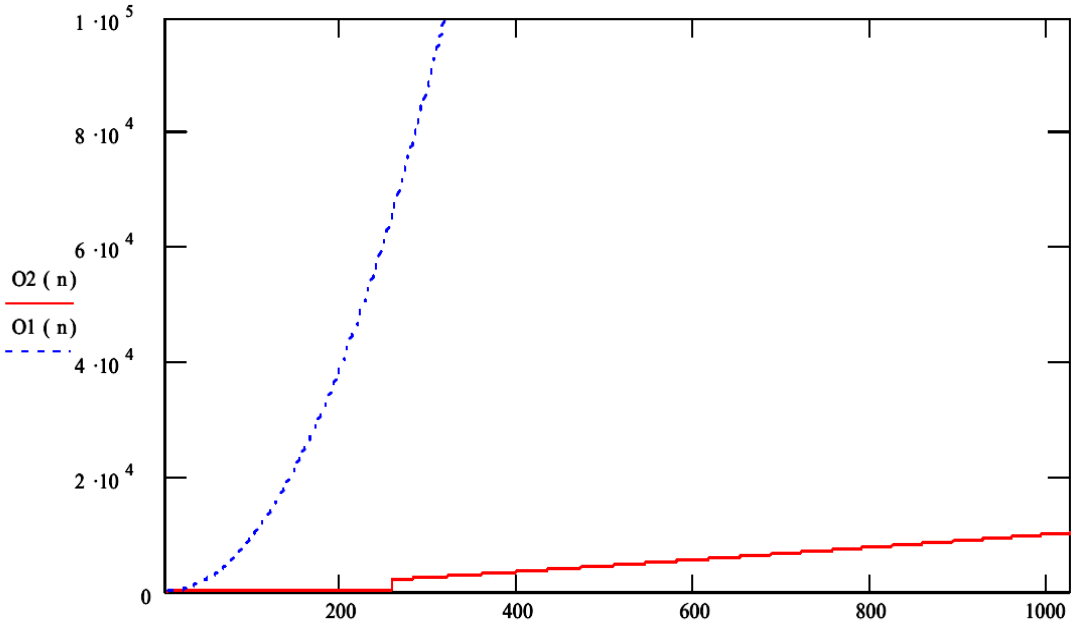


Рисунок 1.13 – Часова складність реалізації операції множення в базисах Радемахера (O1(n)) та Радемахера - Крестенсона (O2(n))в залежності від розмірності

Нормалізована форма СЗК, запропонована науковою школою проф. Николайчука Я.М., найбільш поширена в телекомунікаційних процесорах інформаційних систем нафтогазової промисловості [62].

З рисунка 1.13 видно, що методи множення у базисі Радемахера - Крестенсона характеризується суттєвим збільшенням швидкодії, що є важливою перевагою його застосування шляхом використання спеціалізованих процесорів та контролерів.

В роботі [83] проведено дослідження та розробка операції експоненціювання (рисунок 1.14).

В результаті у роботі [83] розроблено метод модулярного експоненціювання, який з допомогою використання особливостей базису Радемахера – Крестенсона (O4(n)) призводить до кардинального зменшення обчислювальної складності порівняно з базисом Радемахера (O3(n)), що ілюструє рисунок 1.14.

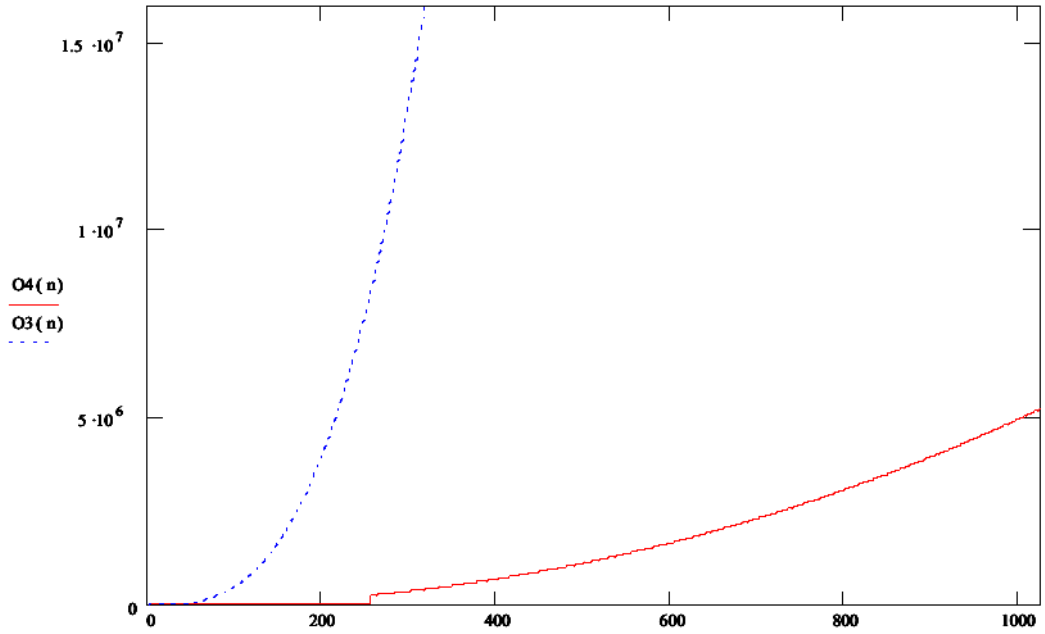


Рисунок 1.14 – Порівняння характеристики виконання модулярного експоненціювання БРЧ в базисах Радемахера (O3(n)) та Радемахера – Крестенсона (O4(n))

Проведені дослідження показують перспективність використання мультибазисних методів опрацювання багаторозрядних інформаційних потоків в прикладних задачах теорії чисел.

1.5 Аналіз існуючих бібліотек для роботи з багаторозрядними числами, постановка задачі дослідження

Серед сучасних програмних засобів опрацювання БРЧ виділяються ряд бібліотек для реалізації відповідних алгоритмів [8], а саме: бібліотека для роботи з БРЧ Ленстра, Arageli, NTL, GMP, Crypto++ та інші. Наведемо особливості організації деяких з них.

В роботі [8] наведений порівняльний аналіз швидкодії реалізації алгоритмів опрацювання БРЧ з використанням різних бібліотек (таблиця 1.3). В результаті дослідження отримані відповідні значення RANK[8] та наведені в таблиці 1.3. Для кожного алгоритму було виконано 100 тестів та встановлено відносні показники продуктивності.

Таблиця 1.3 Характеристики швидкодії алгоритмів з використанням різних бібліотек опрацювання БРЧ

Бібліотека	*RANK(ALG$_i$)*				AES-192	TDES	RSA (2048)		ECDSA (F2m=283)		HMAC SHA-1	SHA-1 (/&=256)	RANK
	MUL	POWMOD	xGCD	ECMUL			*sign*	*verf*	*sign*	*Verf*			
NTL	1,01	1,18	1,0	-	-	-	-	-	-	-	-	-	1,06
MIRACLE	3,58	2,62	3,15	1	1,00	-	2,29	1,72	2,02	2,76	-	1,12	1,40
Botan	3,41	5,21	1,09	1,42	2,16	1,98	1,00	1,00	1,60	1,95	1,15	1,13	1,67
Crypto++	4,49	5,04	16,82	4,35	2,68	1,90	1,12	1,15	1,00	1,00	1,05	1,07	2,19
OpenSSL	2,80	2,43	12,49	1,15	4,49	3,39	1,68	1,51	1,85	2,53	1,00	1,00	2,26
OpenPGP	2,87	2,31	3,11	1,41	1,12	1,37	2,35	1,73	1,54	2,00	-	1,06	1,79
GNU Crypto	1,0	1,0	1,01	1,24	1,38	1,00	2,71	1,80	1,75	2,13	1,06	1,08	1,35
CryptLib	5,25	4,17	8,59	1,7	1,03	1,47	1,09	1,08	1,07	2,05	1,02	1,06	1,82
LenstraLib	1,02	1,15	1,03	1,3	-	-	-	-	-	-	-	-	1,12

Для роботи з БРЧ бібліотека Lenstra є досить гнучким інструментом, що забезпечує найбільш ефективне використання обчислювальних ресурсів, а також реалізацію основних арифметичних та криптографічних алгоритмів, тому для реалізації розроблених методів обрана саме вона.

Факторизацію доцільно застосовувати для підбору модулів у СЗК [86], яка на даний час є однією з альтернатив двійковій системі числення, що дозволяє застосовувати нові підходи до організації обчислювальних систем при виконанні елементарних математичних операцій [86]. Хоча СЗК не позбавлена недоліків, до яких відносяться, зокрема, відсутність ділення та порівняння чисел, необхідність визначення умов переповнення розрядної сітки, однак її успішно можна застосовувати для додавання, віднімання та множення цілих багаторозрядних чисел [86, 87]. Безсумнівною перевагою СЗК є можливість виконання операцій над числами, які менші за вибрані модулі, розпаралелення процесу обчислень та відсутність міжрозрядних переносів.

СЗК – це непозиційна система числення [39], десяткові числа в якій представляються невід'ємними залишками від ділення на кожен з системи взаємно простих модулів p_i. Додавання, віднімання і множення в СЗК відбуваються незалежно по кожному модулю без переносів між розрядами. Зворотне перетворення з СЗК у десяткову систему числення ґрунтується на використанні китайської теореми про залишки [39] і є досить громіздким процесом, що є ще одним недоліком СЗК, який стримував її розвиток і поширення:

$$N = \left(\sum_{i=1}^{n} b_i B_i \right) mod\ P, \tag{1.3}$$

де $B_i = M_i m_i$, $M_i = \frac{P}{p_i}$, базисні числа m_i шукаються з виразу $(M_i m_i) mod\ p_i = 1$.

Необхідність обчислення базисних чисел $m_i = M_i^{-1} mod\ p_i$ істотно збільшує складність переведення чисел з СЗК у десяткову систему. Спрощення цієї задачі відбувається у досконалій формі СЗК (ДФ СЗК), коли модулі p_i підібрані таким чином, що усі m_i=1. У роботах [38, 39] була розвинута теорія ДФ СЗК і запропоновано метод для визначення системи модулів ДФ СЗК. Однак у випадку обмеженої кількості модулів або необхідності використання модулів, які мало відрізняються один від одного, цей метод непридатний. У роботі була запропонована модифікована ДФ СЗК (МДФ СЗК), у якій базисні числа m_i=±1, що також виключає необхідність пошуку оберненого числа.

У [39] показано, що після відповідних математичних перетворень можна отримати умову, яка повинна виконуватися для визначення набору модулів для ДФ та МДФ СЗК:

$$(p_2p_3..p_{n-2}+p_1p_3..p_{n-2}+...+p_1p_2..p_{n-3}-p_1p_2..p_{n-2})+(p_1p_2..p_{n-2})^2=ab. \quad (1.4)$$

$$(p_2p_3..p_{n-2}+p_1p_3..p_{n-2}+...+p_1p_2..p_{n-3}-p_1p_2..p_{n-2})+(p_1p_2..p_{n-2})^2=\pm ab. \quad (1.5)$$

Це означає, що ліва частина (1.4), (1.5) повинна бути факторизована, на основі чого визначаються параметри *a* та *b* для визначення будь-якої кількості модулів.

Отже, вирішення задач теорії чисел дозволить спростити процес перетворення з СЗК у позиційну систему числення, що, в свою чергу, можна ефективно використовувати в ряді прикладних задач обчислювальної техніки.

Таким чином, для досягнення поставленої мети в дисертаційній роботі необхідно розв'язати наступні задачі:

1) розробити:

- метод компактного кодування масивів БПЧ;

- метод факторизації БРЧ у базисі Радемахера та Крестенсона;

- схемотехнічні рішення генератора квадратів БРЧ у базисі Хаара - Крестенсона, пристрою компактного кодування БПЧ та процесора факторизації БРЧ у базисі Хаара - Крестенсона;

- програмне забезпечення факторизації БРЧ та дослідження задачі факторизації БРЧ для виявлення околу рішення, компактного кодування БПЧ, множення БРЧ;

2) удосконалити:

- метод визначення околу рішення задачі факторизації;

- векторно-модульний алгоритм модулярного множення;

- метод пошуку квадратичних лишків;

ВИСНОВКИ ДО РОЗДІЛУ 1

1. Проведена систематизація архітектур універсальних процесорів, що використовуються у персональних комп'ютерах, кластерах, яка показує широкі можливості паралельного опрацювання інформації для обробки БРЧ в прикладних задачах теорії чисел.

2. Проведений аналіз структур та системних характеристик багаторозрядних перемножувачів у базисі Радемахера та обґрунтовано необхідність розробки перемножувачів та суматорів в ТЧБ Радемахера – Крестенсона.

3. Проведений аналіз архітектур та характеристик спецпроцесорів кореляційного, спектрального та ентропійного опрацювання потоків даних, який показує, що спецпроцесори опрацювання інформаційних потоків у ТЧБ Радемахера-Крестенсона та Хаара-Крестенсона характеризуються найвищою швидкодією.

4. Проаналізовано способи кодування інформаційних потоків в комп'ютерних системах на основі ТЧБ Радемахера та Крестенсона, обґрунтовано перспективність використання мультибазисних методів опрацювання багаторозрядних інформаційних потоків в прикладних задачах теорії чисел .

5. Проведений аналіз існуючих бібліотек для роботи з БРЧ, обґрунтовано актуальність розв'язання задач теорії чисел при перетворенні з СЗК у позиційну систему числення і виконана постановка задач дослідження.

РОЗДІЛ 2
РОЗРОБКА ТЕОРЕТИЧНИХ ЗАСАД, МЕТОДІВ ТА УДОСКОНАЛЕНИХ АЛГОРИТМІВ РІШЕННЯ ЗАДАЧ ТЕОРІЇ ЧИСЕЛ У ТЕОРЕТИКО-ЧИСЛОВИХ БАЗИСАХ РАДЕМАХЕРА ТА КРЕСТЕНСОНА

2.1 Метод знаходження залишку БРЧ у базисі Радемахера

Класичні алгоритми пошуку залишку базуються на використанні багаторозрядного базису Радемахера, який має певні недоліки та функціональні обмеження [33]. Загальним недоліком розглянутих в [72] структурних схем пошуку залишків є отримання не завжди найменших залишків та надлишковість порівнянь.

Тому доцільною є розробка удосконаленого методу, який базується на застосуванні особливостей ТЧБ Радемахера – Крестенсона [33].

В основу запропонованого методу покладено представлення БРЧ *P* та модуля *b* у вигляді двійкових чисел, тобто:

$$P \bmod b = K, \tag{2.1}$$

де $P = \sum_{i=0}^{n-1} p_i 2^i$, *де* $\text{p}_\text{i} = 0,1$; $b = \sum_{i=0}^{k-1} b_i 2^i$, *де* $b_\text{i} = 0,1$.

На першому етапі виділяється k-1 старших розрядів числа P, в результаті чого отримується вектор $K = (P_{n-1}, P_{n-2}, ..., P_{n-k-1})$, якщо $K \geq B$, то знаходиться $K \bmod b = 0, K - b = S$ і записується $S = \sum_{i=1}^{k-2} S_i 2^i, S_i = 0,1$, тоді $S = (S_{k-2}, S_{k-3}, ..., S_1, S_0)$. Формується вектор:

$$S_1 = (S^1{}_{k-2}, S^1{}_{k-3}, ..., S^1{}_1, S^1{}_0, p_{n-k-3}) \tag{2.2}$$

шляхом дописування в молодший розряд $p_{n\text{-}k\text{-}3}$. Перевіряється нерівність

$S_1 > B$, якщо справджується, то шукається значення $S_1 \ mod \ b = S_1 - b$, в іншому випадку дописується в молодший розряд p_{n-k-4} і отримується:

$$S_2 = (S^2_{k-2}, S^2_{k-3}, ..., S^2_1, S^2_0, p_{n-k-3}, p_{n-k-4}), \quad (2.3)$$

обчислюється значення $S_2 \ mod \ b$. Дана процедура продовжується доти, поки в молодшому розряді не отримається p_0, тобто вектор:

$$S_n = (S^n_{k-2}, S^n_{k-3}, ..., S^n_1, S^n_0, p_0). \quad (2.4)$$

В результаті для знаходження залишку в базисі Радемахера необхідно знайти значення [33]:

$$P \ mod \ b = S_n \ mod \ b = K. \quad (2.5)$$

Знаходження залишку БРЧ в базисі Радемахера виконується згідно алгоритму[33]:

1.Вхід x,P;

2.Число P представляється в базисі Радемахера: $P(p_n \ldots p_{0)}$;

3.Зменшується розмір n, вектора P на кількість одиниць від P_n до P_i (поки $P_i=0$), який рівний нулю, і записується у вектор $K(K_m \ldots K_0)$;

4. Змінюється x=x-n-i;

5.У базисі Радемахера до числа P додається вектор K, беручи до уваги позицію бітів;

6.Зменшується розмір m вектора K на кількість одиниць від K_m до K_i, який рівний нулю, і записується у вектор K;

7.Змінюється x=x-n-i;

8.Виконується перехід на крок 5, доки $x \geq 0$;

9.Вихід K=res x mod p;

Основними перевагами даного алгоритму, в порівнянні з описаним в роботі [72], є зменшення надлишкового використання пам'яті та кількості

порівнянь.

На рисунку 2.1 подано розроблену блок-схему алгоритму пошуку залишків БРЧ по простих модулях [33].

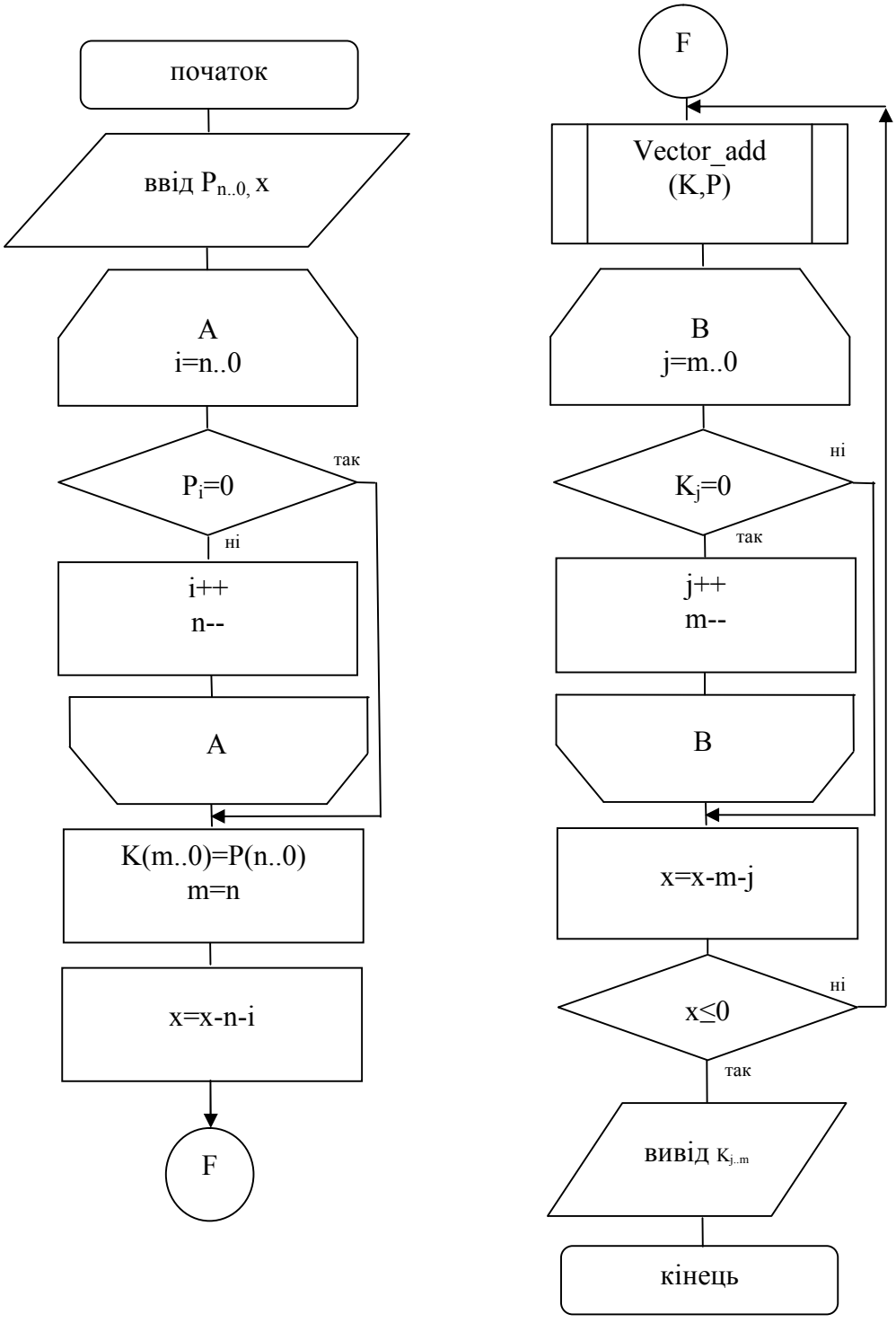


Рисунок 2.1 - Блок – схема алгоритму пошуку залишків БРЧ по простих модулях

Особливістю даного алгоритму є використання базису Радемахера і зменшення кількості операцій додавання, яка пропорційна розрядності чисел [33].

Наведемо приклад:

$10989_{10} \bmod 7_{10} = 2_{10}$

$10101011101101_2 \bmod 111_2 = 10_2$

$10101011101101_2 - 1110000000000_2 = 111011101101_2$

$111011101101_2 - 111000000000_2 = 11101101_2$

$11101101_2 - 11100000_2 = 1101_2$

$1101_2 - 111_2 = 110_2$

$110_2 < 111_2$

Обчислювальна складність алгоритму становить $O(n) = \frac{n \cdot \log_2 n}{\log_2 p}$ [33].

2.2 Векторно-модульний метод модулярного множення багаторозрядних чисел

При вирішенні багатьох задач теорії чисел, які виконуються над багаторозрядними числами, актуальним питанням є необхідність суттєвого прискорення та зменшення обчислювальної складності операції модулярного множення, піднесення чисел до квадрату, визначення квадратичності, обчислення символів Якобі, визначення оберненого елемента за модулем у системі числення залишкових класів базису Крестенсона, факторизації БРЧ, дискретного логарифмування.

У [46] викладено теоретичні засади розробленого нового методу векторно-модульного множення n–розрядних чисел $a = \sum_{i=0}^{n-1} a_i \cdot 2^i$ та $b = \sum_{j=0}^{n-1} b_j \cdot 2^j$, де a_i, b_j=0, 1, n–розрядність модуля p. Суть методу полягає у тому, що для знаходження результату операції модулярного множення $a \cdot b \bmod p$ будується два вектор-рядки, перший з яких складається з елементів

$c_0 = 2^0 \cdot b \bmod p$, $c_i = 2 \cdot c_{i-1} \bmod p$, другий - з a_i, що показано в таблиці 2.1.

Таблиця 2.1– Представлення вектор-рядків модулярного множення

c_{n-1}		c_i	…	c_1	c_0
a_{n-1}	…	a_j	…	a_1	a_0

Отримання результату модулярного множення двох n – розрядних чисел відбувається згідно формули:

$$a \cdot b \bmod p = \left(\sum_{i=0}^{n-1} a_i \cdot c_i \right) \bmod p, \qquad (2.6)$$

Розроблений метод характеризується меншою часовою та апаратною складністю порівняно з матрично-модульним у базисі Радемахера - Крестенсона за рахунок зменшення кількості операцій додавання з $2 \cdot \log_2 n$ до $\log_2 n$, тобто в два рази [46]. Оскільки при вирішенні окремих названих задач теорії чисел число операцій додавання може перевищити 2^{32}, то збільшення ефективності в два рази є суттєвим і значно розширює функціональні можливості опрацювання БРЧ, а також спрощує мікроелектронну реалізацію спецпроцесорів.

Розглянемо приклад. Нехай потрібно знайти значення $7973 \cdot 8921 \bmod 135$. Згідно алгоритму представимо 8921 у двійковій системі числення, тобто: $8921_2 = 10001011011001$. Після чого для знаходження операції модулярного множення потрібно скласти таблицю 2.2. В першому її рядку записується двійкове представлення $8921_2 = 10001011011001$, а в другому - значення $c_0 = 2^0 \cdot 7973 \bmod 135$; $c_1 = 2 \cdot c_0 \bmod 135$; …; $c_i = 2^i \cdot c_{i-1} \bmod 135$. Для знаходження $c_0 = 2^0 \cdot 7973 \bmod 135$ доцільно скористатися методом знаходження залишку БРЧ на основі використання ТЧБ Радемахера - Крестенсона, запропонованого в [33].

Для цього в перший рядок таблиці 2.1 записується двійкове представлення числа 7973_2=1111100100101, а в другий - $2^i \bmod 135$, де i=0,…,n-1, n- розрядність числа 7973. Результат пошуку залишку $c_0 = 2^0 7973 \bmod 135 = (1+4+32+121+107+79+23+46) \bmod 135=8$.

Таблиця 2.2 - Знаходження операції модуля в ТЧБ Радемахера - Крестенсона

1	7973	1	1	1	1	1	0	0	1	0	0	1	0	1
2	$2^i \bmod 135$	46	23	79	107	121	128	64	32	16	8	4	2	1

Отже, c_0=8, c_1=16, $c_2 = 32$, $c_3 = 64$; $c_4 = 128$; c_5=121, c_6=107, c_7=79, c_8=23, c_9=46, c_{10}=92, c_{11}=49, c_{12}=98,c_{13}=61 (таблиця 2.3).

Таблиця 2.3 - Модулярне множення в ТЧБ Радемахера - Крестенсона

1	8921	1	0	0	0	1	0	1	1	0	1	1	0	0	1
2	$2 \cdot c_{i-1} \bmod 135$	61	98	49	92	46	23	79	107	121	128	64	32	16	8

Отже, 7973·8921 mod 135 = (8+64+128+107+79+46+61) mod 135=88. В таблиці 2.4 приведено порівняння складностей відомих та розробленого алгоритмів множення БРЧ.

Таблиця 2.4 – Складність відомих та розробленого алгоритмів модулярного множення

Позначення	Назва	Складність
O(n)	Шонхаге-Штрасена	$n \cdot \log n \cdot \log(\log n)$
O1(n)	Стандартний	n^2
O2(n)	Матрично-модульний	$\begin{cases} 2\log n, \textit{ якщо } n \le 64 \\ n\log n, \textit{ інакше} \end{cases}$
O3(n)	Алгоритм Карацуби та Тома-Кука	$n^{1,585}$
O4(n)	Карацуби	$O4(n) = n^{1,465}$
O5(n)	Радемахера-Крестенсона	$\log n$

Отже, розроблений метод модулярного множення з використанням векторно-модульних операцій в ТЧБ Радемахера-Крестенсона доцільно використовувати при опрацюванні інформаційних потоків з БРЧ, включаючи перевірку чисел на простоту, факторизацію та інші.

Таким чином, в результаті проведених досліджень було встановлено часові складності відомих методів модулярного множення та розробленого. Графічні залежності часової складності відомих алгоритмів представлені на рисунку 2.2, а запропонованого алгоритму – на рисунку 2.3 [46].

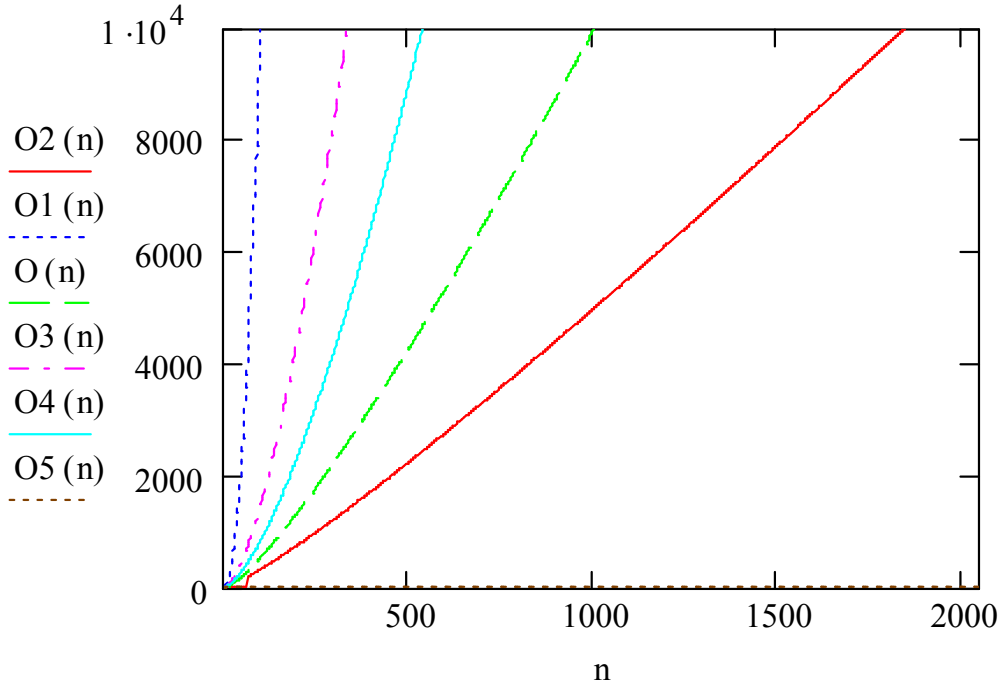


Рисунок 2.2 – Складність операції модулярного множення відомих алгоритмів.

Рисунок 2.3 відображає складність операції модулярного множення запропонованого алгоритму та ілюструє перевагу над наведеними алгоритмами.

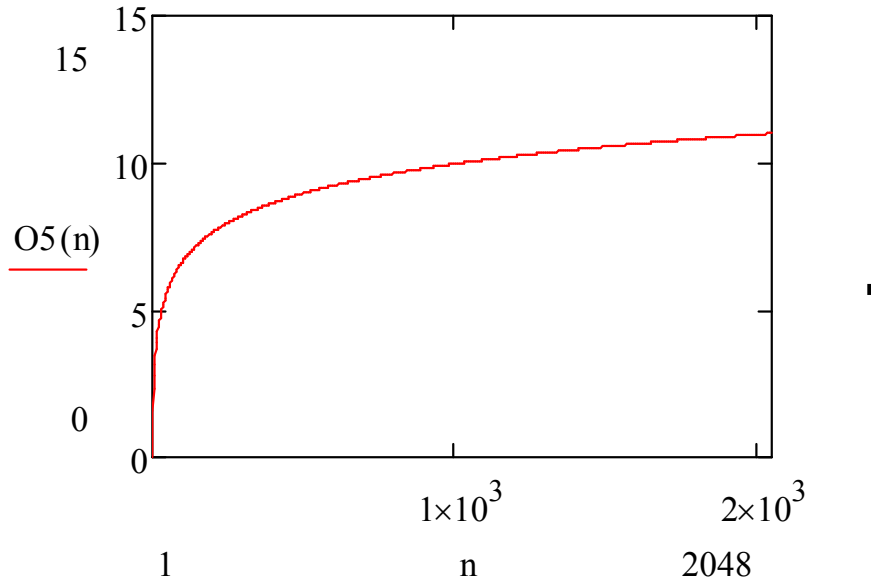


Рисунок 2.3 – Складність операції модулярного множення запропонованого алгоритму

Результати досліджень показали, що в основу часової складності модулярного множення входять складності операцій пошуку залишків по модулю та додавання. З рисунків 2.2 та 2.3 видно, що використання розробленого векторно-матричного методу, який ґрунтується на використанні ТЧБ Крестенсона-Радемахера, дозволяє на порядок зменшити часову складність модулярного множення відносно класичного методу і на 50 % - в порівнянні з матрично-модульним алгоритмом в ТЧБ Радемахера-Крестенсона [46].

2.3 Метод перевірки багаторозрядних чисел на простоту

Розроблений метод перевірки БРЧ на простоту ґрунтується на фундаментальній математичній основі малої теореми Ферма та розробленого в 2.2 векторно-модульного алгоритму множення в базисі Крестенсона.

Нехай маємо *n*-розрядне число *p*. Тоді згідно теореми Ферма повинно виконуватися наступне співвідношення:

$$m=2^P \bmod p=2. \qquad (2.7)$$

Введемо позначення: $2^p = M_{(2)}$ і представимо його у такому вигляді::

$$\underbrace{M_{(2)}=100000\ldots00}_{P\text{-розрядів}} \qquad (2.8)$$

,

Відповідно, $p=\sum_{i=0}^{n-1} p_i 2^i$, де p_i=0,1, причому p>>n. Далі потрібно здійснити розклад $M_{(2)}$ у такий добуток:

$$\underbrace{M_{(2)}=100..00\ldots0}_{\left[\frac{P}{n+1}\right]} \times \underbrace{100..00\ldots000}_{\left[\frac{P}{n+1}\right]} \times \ldots \times \underbrace{100..00..000}_{\left[\frac{P}{n+1}\right]} \times \underbrace{100..00\ldots000}_{P-l\left[\frac{P}{n+1}\right]} \qquad (2.9)$$

У (2.9) $l=\left\lceil \frac{p}{n} \right\rceil$ – кількість однакових множників у розкладі $M_{(2)}$ з розрядністю n+1.

Для знаходження $2^p \bmod p$ треба обчислити залишки кожного з множників рівняння (2.9) шляхом віднімання, тобто якщо l – парне, то на наступному етапі залишки $M^2_{(2)}$ групуються попарно і перемножуються, тобто шукаються квадрати $\left(M^2_{(2)}\right)^2$.

$$M^{12}_{(2)}= \underbrace{100..00\ldots000}_{\left[\frac{P}{n+1}\right]} -P_{(2)} \qquad (2.10)$$

$$M_{(2)}^{2} = \underbrace{100..00\ldots000}_{P-l\left[\frac{P}{n+1}\right]} - P_{(2)} = \underbrace{100..00\ldots000}_{P-l\left[\frac{P}{n+1}\right]} \qquad (2.11)$$

де $p_{(2)}$ – двійкове представлення числа p.

В результаті таких операцій отримується l залишків $M_{(2)}^{2}$ і останній множник у записі (2.11).

У випадку, коли l – непарне, то по 2 групується l-1 залишків $M_{(2)}^{2}$, які, аналогічно до попереднього, підносяться до квадрату, і один добуток $M_{(2)}^{2}$ з останнім множником у формулі (2.11).

Отже, для знаходження $2^p \ mod\ p$ потрібно $log_2 l$ описаних вище кроків, на кожному з яких відбувається перевірка на парність кількості залишків. Така процедура призводить до виконання на кожному кроці двох операцій модулярного множення, які можна виконати векторно-модульним методом, описаним в [46], з часовою складністю O(2 $\log_2$n).

В порівнянні з відомими, розроблений метод характеризується високою швидкодією та меншою обчислювальною складністю, що дозволяє ефективно застосовувати його при перевірці БРЧ на простоту.

Оскільки відомі методи є, переважно, ймовірнісними, то вони однозначно не вказують на необхідні умови простоти числа. Проведені дослідження показують ефективність та високу ймовірність виявлення простоти числа.

На рисунку 2.4 представлена розроблена структурна схема алгоритму перевірки БРЧ на простоту, яка показує схему розрядного розбиття при знаходженні залишку БРЧ, який вказує на ймовірну простоту числа.

Перевірка натуральних чисел на простоту характеризується обчислювальною складністю $O(n\,Log_2 n)$ з врахуванням використання векторно-модульного алгоритму модулярного множення.

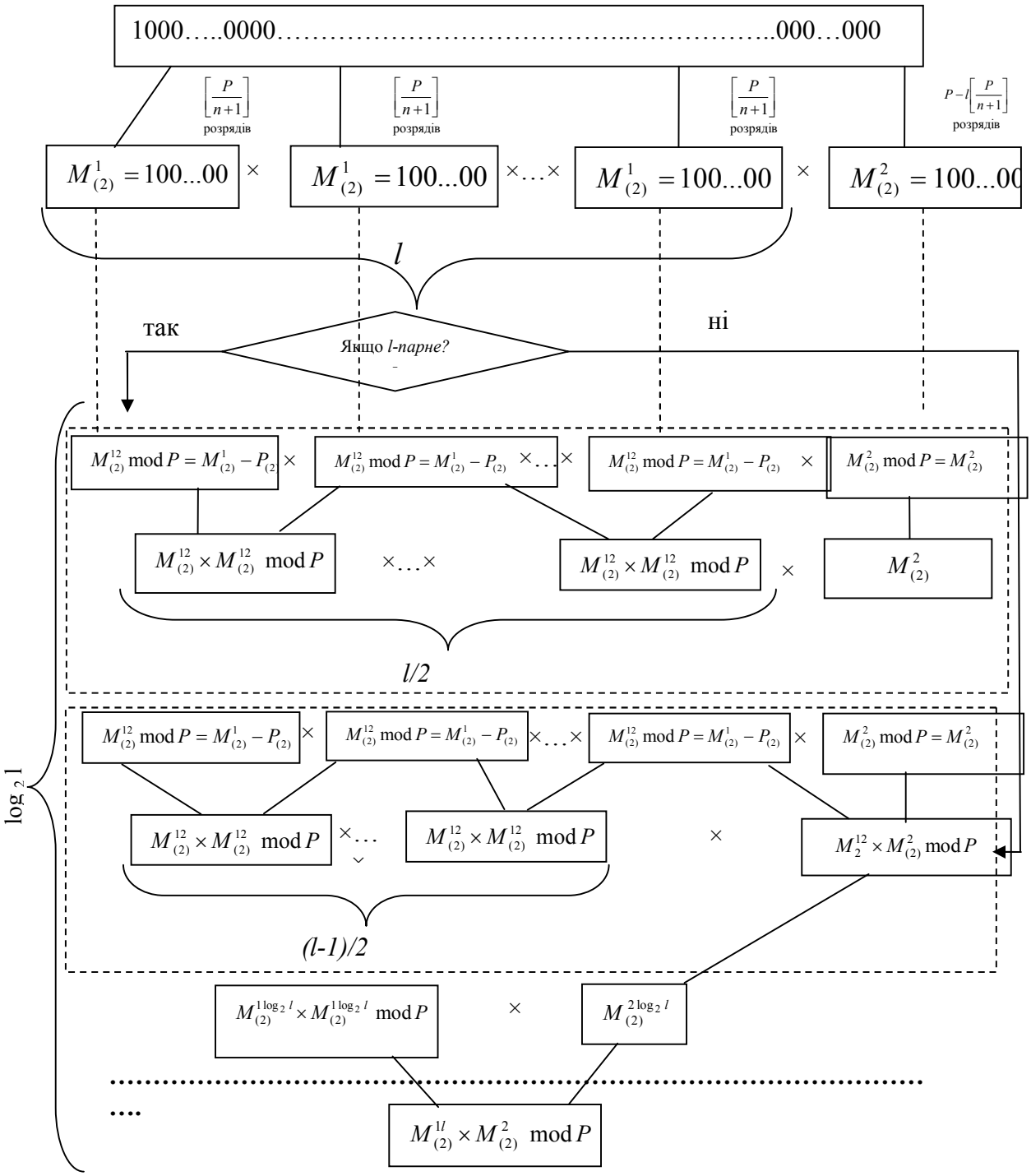


Рисунок 2.4 – Схема перевірки багаторозрядних чисел на простоту

Блок-схема роботи алгоритму перевірки БРЧ на простоту представлена на рисунку 2.5.

Покрокове виконання запропонованого алгоритму реалізується таким чином:

1. Вхід: n-розрядне число p.

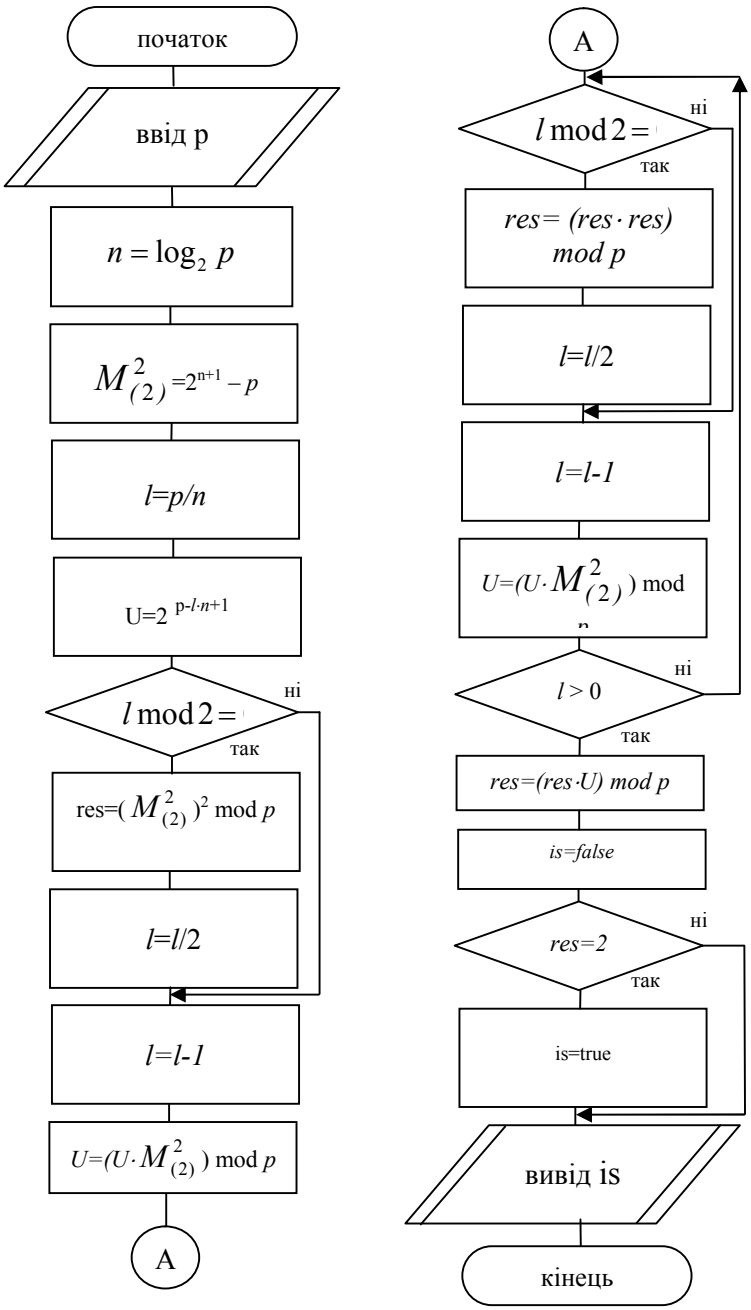


Рисунок 2.5 – Блок-схема перевірки *n*-розрядних чисел на простоту

2. Шукається різниця $M^2_{(2)} = 2^{n+1} - p$.
3. Шукається ціла частина від ділення *l=p/n* та U=2 $^{p-l\cdot n+1}$.
4. Якщо *l* - парне, то обчислюється res=($M^2_{(2)}$)² mod *p* та відбувається побітовий зсув змінної *l,* тобто *l=l*/2. Якщо *l* - непарне, тоді *l=l-1, U=(U*·$M^2_{(2)}$) mod *p*.
5. Якщо *l* – парне, тоді res= (res· res) mod p та виконується побітовий

зсув змінної *l,* тобто *l=l*/2. Якщо *l* – непарне, тоді *l=l-1*, $U=(U\cdot M_{(2)}^{2})$ *mod p*.

6. Якщо *l* > 0, тоді відбувається перехід на крок 5.
7. Відбувається операція модулярного множення та присвоєння *res=(res·U) mod p*.
8. Якщо *res=2*, то алгоритм повертає значення true, якщо ні, то false.

У відомому тесті на простоту Соловея – Штрасена [28] використовується критерій Ейлера: якщо n – просте, то $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$ для всіх значень *a*, для яких НСД(*a*, n) = 1. Порівняння обчислювальних складностей розробленого та відомого тестів простоти наведені на рисунку 2.6.

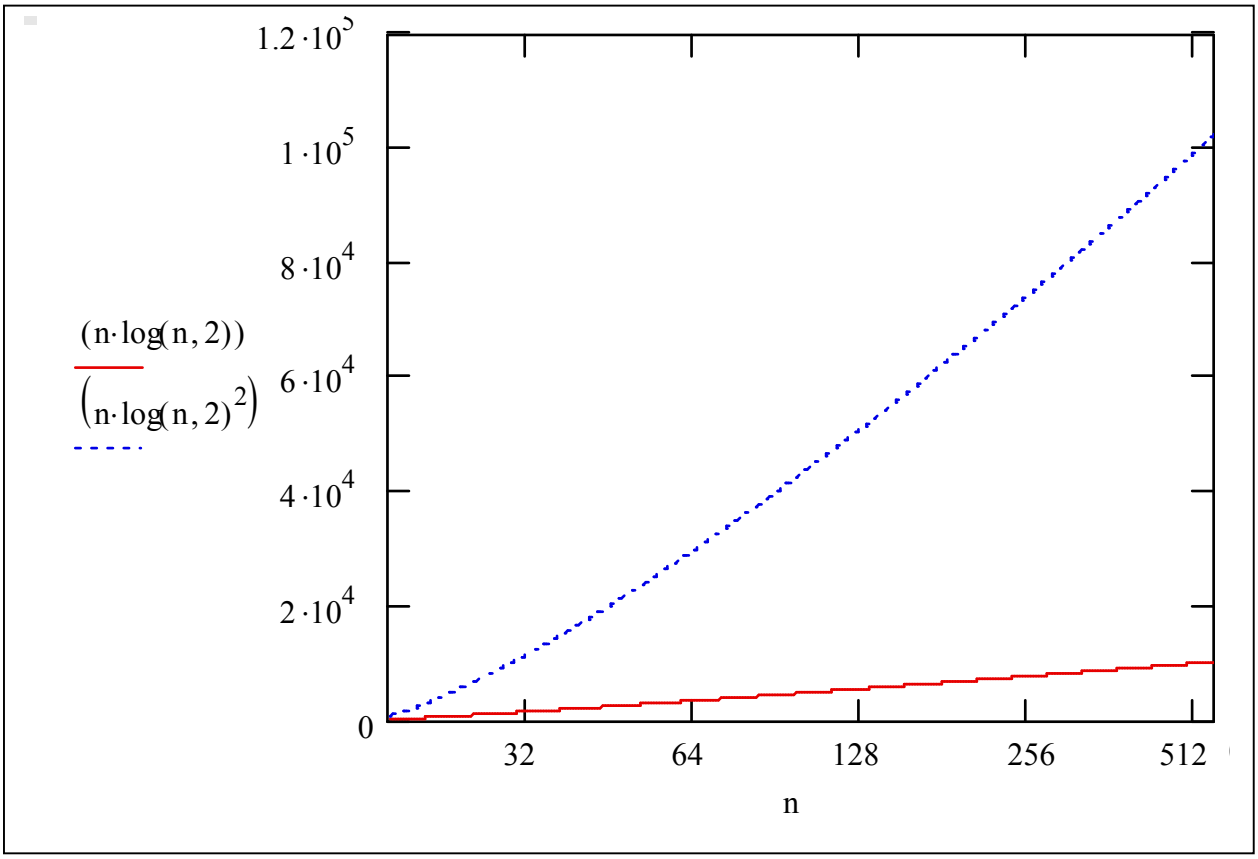


Рисунок 2.6 – Порівняння складностей відомого та розробленого тесту простоти.

Слід зазначити, що в даному підході потрібно перевіряти, чи $\left(\frac{a}{n}\right)$ буде квадратичним лишком, тобто обчислювати символ Якобі [25], що призводить до часової складності $O(n \cdot Log_2^2 n)$.

Якщо *res = 2*, то число ймовірнісно просте і алгоритм повертає значення true, якщо ж ні, то значення false.

В результаті таких обчислень буде отримано значення $2^p \bmod p$ і якщо воно рівне *2*, то *p* є простим числом.

Слід зазначити, що дуже рідко зустрічаються числа, які задовільняють умові 2.7, але вони не є простими. Нижня границя співпадіння при цьому знаходиться в співвідношенні 1:1000, а верхня - 3:1000.

В таблиці 2.5 подані результати експериментального дослідження розподілу чисел, які задовольняють умову 2.7 і є складеними.

Таблиця 2.5 – Розподіл складених чисел, які задовільняють умову $2^p \bmod p = 2$

Номер по порядку	Діапазон чисел	Складені числа *P*, які задовільняють умову $2^p \bmod p = 2$
1	[1..1000]	341
2	[1000..2000]	1105
3	[1000..2000]	1387
4	[1000..2000]	1729
5	[2000..3000]	2047

Результати чисельного експерименту показують, що використання даного методу дозволить однозначно визначати прості числа з врахуванням таблиці 2.5, тобто чисел, які є винятками. Кількість винятків співвідноситься, як один до тисячі і свідчить про високу ймовірність виявлення простого числа.

2.4 Дослідження обчислювальних процесів пошуку квадратичних лишків багаторозрядних чисел

Пошук квадратичних лишків є важливою задачею теорії чисел, результати вирішення якої мають широке застосування при вдосконаленні алгоритмів вибору системи модулів для досконалої форми СЗК та опрацювання БРЧ за умови реалізації модульних операцій.

Для пошуку квадратичних лишків в праці [28] розглянуто методи на основі використання символів Якобі та Лежандра, які дозволяють однозначно вказувати, чи число є квадратичним лишком. Відомі методи визначення квадратичного лишку наведені в додатку Д.

Метод знаходження квадратичних лишків широко використовується для опрацювання та перетворення даних і може бути застосований в системах передавання інформації, а також для захисту даних від помилок та несанкціонованого доступу.

Проведений аналіз існуючих методів визначення квадратичного лишку за модулем говорить про те, що розв'язання даної задачі далеке від досконалості. В умовах опрацювання БРЧ перспективними є алгоритми, побудовані на базі СЗК [29].

Розробка ефективного методу пошуку квадратичного лишку за модулем з використання СЗК є актуальною задачею у теорії чисел.

Розглянемо залишки квадратів цілих чисел по декількох простих модулях p_j, тобто $a_1(p_1, p_2, ..., p_m) = b_1^1, b_2^1, \ldots, b_m^1$, $a_2(p_1, p_2, ..., p_m) = b_1^2, b_2^2, \ldots, b_m^2$, $a_n(p_1, p_2, ..., p_m) = b_1^n, b_2^n, \ldots, b_m^n$, де $a_i = i^2$, $b_j^i = a_i (\mathrm{mod}\, p_j), 1 \le i \le n,\ 1 \le j \le m$, m – кількість модулів.

Квадрати цілих чисел можна представити у вигляді суми непарних чисел, кількість яких дорівнює даному числу [29]:

$$n^2 = \sum_{i=1}^{n} (2i - 1). \qquad (2.12)$$

Використовуючи властивість (2.12), шукані залишки можна отримувати за допомогою рекурентної формули

$$b_j^i = \left(b_j^{i-1} + z_i^j\right) \bmod p_j, \qquad (2.13)$$

де $z_j^i = z_i \bmod p_j$, z_i=2i-1.

Відповідні результати по модулях 3, 5, 7, 11 представлені в таблиці 2.6 і відображають циклічні властивості квадратичних лишків по простих модулях. Ці властивості дозволяють скоротити процес ітераційного перебору наполовину і виконуються для будь-яких модулів [29].

З таблиці 2.6 видно, що кількість квадратичних лишків для кожного модуля становить (p_j+1)/2 (включаючи 0). Це випливає з рівності $n^2 \, mod \, p_j = (-n^2) \, mod \, p_j = (p_j - n)^2 \, mod \, p_j$.

Таблиця 2.6 – Пошук залишків квадратів по простих модулях

N	z_i	a_n	p_1=3		p_2=5		p_3=7		p_4=11	
			z_1^i	b_1^i	z_2^i	b_2^i	z_3^i	b_3^i	z_4^i	b_4^i
1	1	1	1	1	1	1	1	1	1	1
2	3	4	0	1	3	4	3	4	3	4
3	5	9	2	0	0	4	5	2	5	9
4	7	16	1	1	2	1	0	2	7	5
5	9	25	0	1	4	0	2	4	9	3
6	11	36	2	0	1	1	4	1	0	3
7	13	49	1	1	3	4	6	0	2	5
8	15	64	0	1	0	4	1	1	4	9
9	17	81	2	0	2	1	3	4	6	4
10	19	100	1	1	4	0	5	2	8	1
11	21	121	0	1	1	1	0	2	10	0

Алгоритм визначення квадратичності залишку числа за певним модулем складається з таких кроків [29]:

1) зберігається число a (залишок для перевірки на квадратичність) та

простий модуль числа *p*;

2) ініціалізується лічильник *i=1*;

3) встановлюється бінарна ознака *is=false*;

4) якщо *a =0,* то встановлюється властивість *is = true* і відбувається перехід на крок 12, оскільки залишок 0 завжди є залишком квадрату;

5) якщо *a =1*, то встановлюється властивість *is = true* та відбувається перехід на крок 12, оскільки залишок *0* завжди є залишком квадрату;

6) ініціалізується лічильник *pN=1* та лічильник *sN=1;*

7) оскільки кожен квадрат змінюється на крок з різницею *2*, то, відповідно, збільшується pN=pN+2 та sN= sN+ pN;

8) на лічильник накладаються обмеження за модулем *p*, тобто відбувається переприсвоєння pN=(pN mod *p*); sN =(sN mod *p*);

9) збільшується значення лічильника *i* на *1*;

10) якщо *pN = a*, то встановлюється *is=true* та відбувається перехід на крок 11;

11) якщо лічильник i<(*p* /2), то відбувається перехід на крок 7;

12) алгоритм повертає властивість is, яка може набувати лише два різних значення: 1 або 0, що, як і в символах Якобі, служить індикатором належності залишку числа до квадратичного лишку за певним модулем.

Алгоритм повертає лише два різних значення 1 або 0, що, як і при застосуванні символів Якобі, служить індикатором належності залишку числа до цілого квадрату за певним модулем.

Слід зазначити, що система залишків по відповідних модулях утворює циклічну групу, що дозволяє наполовину скоротити відповідний перебір.

Метод визначення символів Якобі дозволяє зменшити часову складність за рахунок заміни операції множення додаванням і при цьому підвищити швидкодію на 1 порядок [29].

Запропоноване рішення дозволяє виконувати виявлення квадратичних лишків, яке характеризується меншою обчислювальною складністю відносно

відомих способів, яка зростає в $O(\frac{e^{2\sqrt{\ln n \ln\ln n}} + \log^2 n}{(\frac{n^2}{2} - n)\cdot \log_2 n})$ разів.

На рисунку 2.7 представлена розроблена блок-схема роботи алгоритму визначення квадратичності лишку, найскладнішою операцією якого є додавання за модулем, а кількість ітерацій буде дорівнювати (m-1)/2.

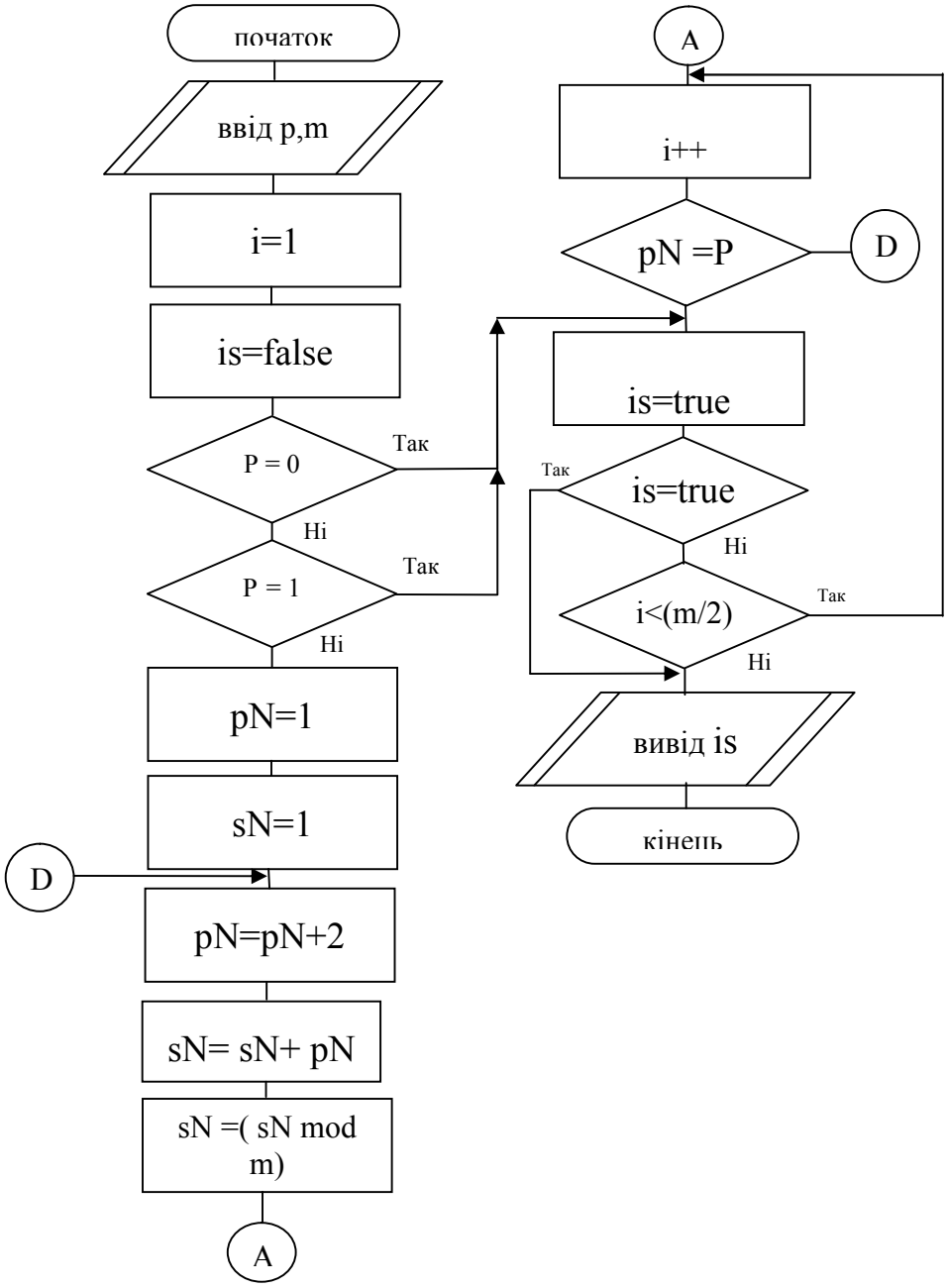


Рисунок 2.7 – Блок схема визначення квадратичності лишку

Результатом роботи алгоритму може бути лише два значення 1, 0, які вказують на можливу квадратичність або ж гарантовану неквадратичність лишку, для гарантованого визначення квадратичності лишку можна скористатись декількома модулями.

Запропонований метод має ефективне застосування, приведене в підрозділі 3.2, та високу практичну цінність при реалізації методів опрацювання багаторозрядних простих чисел в задачах теорії чисел та сучасної обчислювальної техніки, оскільки він є ітераційним та не містить в собі операцій, складних для виконання апаратним шляхом.

Графічні залежності даних складностей в логарифмічній шкалі представлені на рисунку 2.8.

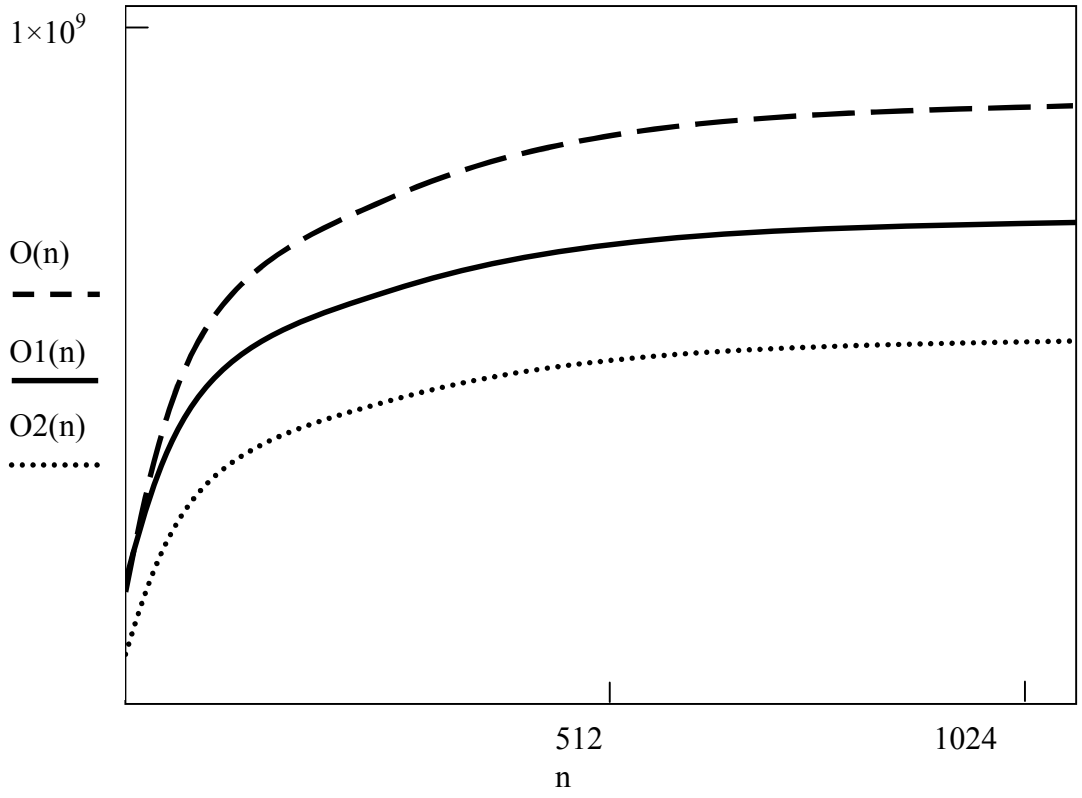


Рисунок 2.8 – Порівняльна характеристика обчислювальної складності пошуку символу Лежандра (O(n)), символів Якобі (O1(n)) та розробленого методу визначення квадратичності лишку (O2(n))

Оскільки у відомих методах пошуку символів Якобі основною трудомісткою операцією є модулярне експоненціювання, то розрахунки показують, що запропонований алгоритм на основі використання системи залишкових класів (основною операцією якого є пошук модуля від числа) дозволяє зменшити складність з $O2(n^3)$ або $O1(n^2 \log n)$ (Монтгомері метод) до $O(n \cdot \log_2 n)$, тобто ефективність зростає в $E2(n) = n$ разів[29].

Отже, в результаті даних міркувань, розроблено алгоритм пошуку квадратичних лишків, який характеризується меншою обчислювальною складністю відносно відомих методів, (зокрема, знаходження символів Якобі та Лежандра) і дозволяє ефективно визначати повні квадрати по модулю *P*.

2.5 Метод компактного кодування простих багаторозрядних чисел

При реалізації алгоритмів опрацювання багаторозрядних простих чисел в задачах вибору системи взаємно простих модулів для процесорів ТЧБ Крестенсона, пошуку найбільшого спільного дільника, виявлення квадратичного лишку, виконання арифметичних операцій модульної арифметики виникає необхідність зберігання та генерування великих масивів багаторозрядних простих чисел.

Генерування та зберігання багаторозрядних простих чисел, представлених повнорозрядними двійковими кодами, є неефективним у зв'язку з тим, що потребує великих об'ємів пам'яті.

Теорема про розподіл простих чисел стверджує, що кількість $\pi(n)$ простих чисел на відрізку від 1 до *n* зростає із зростанням *n,* як $\frac{n}{\ln n}$, тобто $\frac{\pi(n)}{n / \ln n} \to 1,\ n \to \infty$. Оцінка об'ємів пам'яті згідно наведеної теореми приведена в таблиці 2.7.

Зростання верхньої оцінки об'єму пам'яті для повнорозрядного

зберігання простих двійкових чисел показано на рисунку 2.9.

Таблиця 2.7 – Верхня оцінка розподілу простих чисел

Розрядність	Кількість	Об’єм пам’яті
2^8	64	64 байти
2^{10}	256	2 КБ
2^{16}	16384	32 КБ
2^{32}	1073741824	4 Гб
2^{64}	4611686018427387904	36 Гб
2^{128}	8,5070591730234615865843651857942e+37	2^{213} Терабайт
2^{256}	2,8948022309329048855892746252172e+76	2^{426} Йотабайт
2^{512}	3,3519519824856492748935062495515e+153	*
2^{1024}	4,4942328371557897693232629769726e+307	*

Побудований графік із змінною шкалою, в якому відображені верхні оцінки необхідного об’єму для збереження простих чисел.

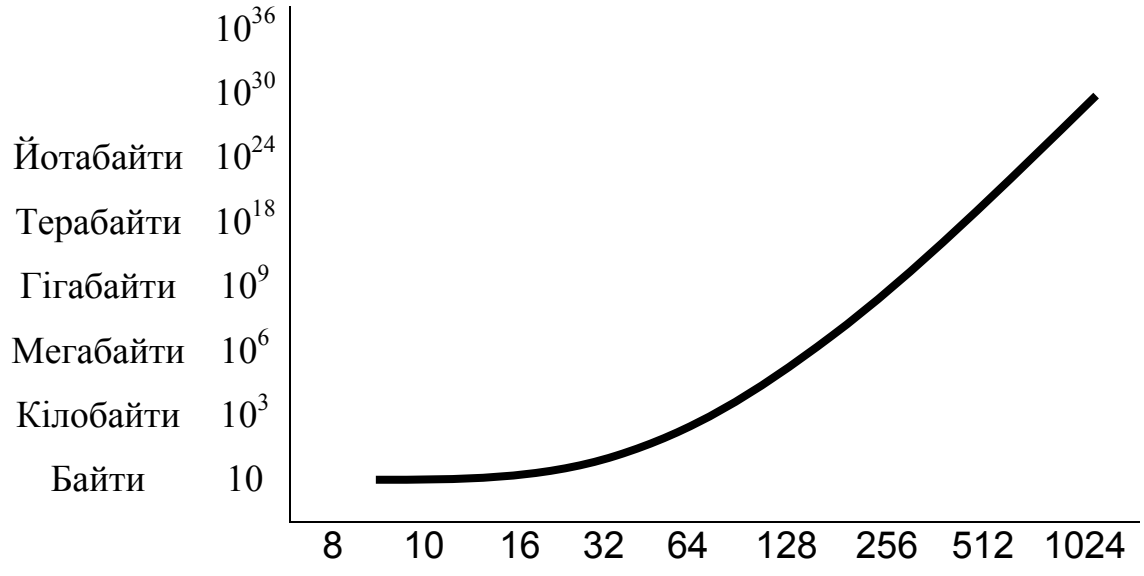


Рисунок 2.9 - Оцінка об’єму пам’яті для повнорозрядного зберігання простих двійкових чисел

З графіка видно, що об’єм пам’яті, необхідний для зберігання n-

розрядних простих чисел до 32 розрядів, становить 4 Гб , для 64 розрядів - 36 Гб, а при 128 - 2^{213} Терабайт, оскільки при опрацюванні окремих задач теорії чисел розрядність простих чисел може сягати 512 біт, що відповідає надзвичайно великим об'ємам необхідної пам'яті, тому актуальною є наукова задача більш компактного зберігання великих масивів багаторозрядних простих чисел.

Можливість більш компактного зберігання простих БРЧ може бути реалізована за рахунок надлишковості багаторозрядних двійкових кодів, оскільки різниця між двома простими БРЧ може не перевищувати 10-20 біт. Причому інформативна частина простого числа знаходиться в 20-30 бітах молодших розрядів при ідентичності сотень старших бітів, починаючи з 32 - 64 біт. Для прикладу візьмемо числа з інтервалу $2^{20} - 2^{64}$:

1052629_{10} =10000000111111010101 (2^{20});

2147487701_{10}=10000000000000000000111111010101 (2^{32});

9223372036854779861_{10}=10000000000000000000000000000000000000000000000000000000000111111010101(2^{64});

При цьому в двійкових кодах БРПЧ при активній зміні бітів молодших розрядів в старших розрядах велике число одиниць або нулів може обчислюватись тисячами, що є основою методу компактного кодування БРПЧ.

В роботах [56, 34] запропонований метод компактного кодування БРПЧ, суть якого полягає в тому, що в пристроях пам'яті запам'ятовується певне число молодших розрядів, а 1 біт використовується для ідентифікації наскрізних переносів у старші розряди, а коди старших розрядів визначаться шляхом підрахунку числа переносів, які ідентифіковані бітом синхронізації, розподіл якого в середньому до 20 простих чисел. Тобто у діапазоні до 2^{1024} буде знаходитись 2^{51} бітів синхронізації.

Таким чином, для зберігання об'єму простих чисел розрядністю до 1024 за попередніми оцінками потрібно 1048576 байт. Для збереження цілого числа в двійковій системі використовуються всі байти, які несуть значення

числа. В результаті проведених досліджень та аналізу переліку простих чисел отримано результати (див. рисунок 2.9), які свідчать про те, що числа з однаковими молодшими бітами можна представити у вигляді трьох частин. Їх розміри залежать від розрядності простого числа та кількості бітів в обраному закінченні (таблиця 2.8) [56].

Таблиця 2.8 – Послідовність простих чисел з однаковим закінченням

4194389 100000000000 0000101 0101	4195493 100000000000 1001010 0101
4194581 100000000000 0010001 0101	4195573 100000000000 1001111 0101
4194661 100000000000 0010110 0101	4195589 100000000000 1010000 0101
4194677 100000000000 0010111 0101	4195621 100000000000 1010010 0101
4194917 100000000000 0100110 0101	4195861 100000000000 1100001 0101
4195157 100000000000 0110101 0101	4195973 100000000000 1101000 0101
4195189 100000000000 0110111 0101	4196149 100000000000 1110011 0101
4195253 100000000000 0111011 0101	4196341 100000000000 1111111 0101
4195349 100000000000 1000001 0101	

В результаті проведеного аналізу представлення простих чисел в базисі Радемахера виявлено, що для чисел з однаковими молодшими розрядами можна зберігати лише частину бітів.

Нехай P_n – числа з однаковими молодшими розрядами, тобто:

$$P_1 = \sum_{i=1}^{n-1} P_i^1 2^i, \text{ де } \mathrm{P}_\mathrm{i}^1 = 0{,}1$$

$$P_2 = \sum_{i=1}^{n-1} P_i^2 2^i, \text{ де } \mathrm{P}_\mathrm{i}^2 = 0{,}1 \qquad (2.14)$$

$$P_n = \sum_{i=1}^{n-1} P_i^n 2^i, \text{ де } \mathrm{P}_\mathrm{i}^\mathrm{n} = 0{,}1$$

Отже, отримаємо розклад чисел в базисі Радемахера:

$$P_{1(2)}^1 = (P_{n-1}^1, P_{n-2}^1, ..., P_1^1, P_0^1)$$

$$P_{2(2)}^2 = (P_{n-1}^2, P_{n-2}^2, ..., P_1^2, P_0^2) \qquad (2.15)$$

$$P^n_{n(2)} = (P^n_{n-1}, P^n_{n-2}, ..., P^n_1, P^n_0)$$

Причому $P^i_3; P^i_2; P^i_1; P^i_0$, $i = 1, n$ є однаковими. Також слід відмітити, що старші розряди змінюються дуже повільно, в результаті чого можна зробити висновок, що їх зберігати не обов'язково, а використовувати біт синхронізації, який вказує, коли в старші розряди $P^i_{n-1}; P^i_{n-2}; ...; P^i_{n-k}$, i=1,..,n додавати 1.

Для збереження простих чисел доцільно проводити аналіз 8 бітів, тобто від 4 до n-k-1 розряду, для яких виконується умова: n-k-5=8 біт.

В подальшому зберігаються тільки представлення від P^i_{n-k-1} до P^i_4, а зміна в старших розрядах на 1 контролюється бітом синхронізації, який також зберігається і вказує порядковий номер простого числа, в якому додається 1. В таблиці 2.9 приведено розбиття кодів простих чисел в базисі Радемахера на групи розрядів та конкатенацію з бітом синхронізації.

Таблиця 2.9 - Розбиття простих чисел з закінченням 0101 в базисі Радемахера на частини [34]

Десяткове представлення	Двійкове представлення			Біт синхронізації
	Верхні розряди числа	7-ми бітне закінчення числа	Закінчення	
142789	1000101	1011100	0101	0
142837	1000101	1011111	0101	0
142949	1000101	1100110	0101	0
142981	1000101	1101000	0101	0
143093	1000101	1101111	0101	0
143141	1000101	1110010	0101	0
143333	1000101	1111110	0101	0
143413	1000110	0000011	0101	1
143461	1000110	0000110	0101	0

Продовження таблиці 2.9

Десяткове представлення	Двійкове представлення			Біт синхронізації
	Верхні розряди числа	7-ми бітне закінчення числа	Закінчення	
143477	1000110	0000111	0101	0
143509	1000110	0001001	0101	0
142789	1000101	1011100	0101	0
143653	1000110	0010010	0101	0
143669	1000110	0010011	0101	0
143797	1000110	0011011	0101	0
143813	1000110	0011100	0101	0
143909	1000110	0100010	0101	0
144037	1000110	0101010	0101	0

Розроблено алгоритм збереження великорозрядних простих чисел, що дозволяє з використанням біта синхронізації забезпечити економію дискового простору.

1. Вхід P[0..j], Limit;
2. перевірка простоти P;
3. якщо P не просте, то P++ крок 2;
4. C[]= P[4..11];
5. P++;
6. перевірка простоти;
7. якщо Pi..j не просте, то крок 5;
8. C[]= P[4..11];
9. D[]= P[12..j];
10. C[8]=0;
11. якщо D ≠ P[12..j], тоді C[8]=1;
12. запис C в файл;
13. якщо P≠Limit тоді 7;

14. вихід.

Дослідження показали, що ефективність методу збереження багаторозрядних простих чисел зростає у відповідності із збільшенням розрядності чисел (рисунок 2.10).

Попередній аналіз методу кодування БПЧ показує, що, у порівнянні з відомими алгоритмами в базисі Радемахера, він характеризуються лінійно-логарифмічною обчислювальною складністю [56].

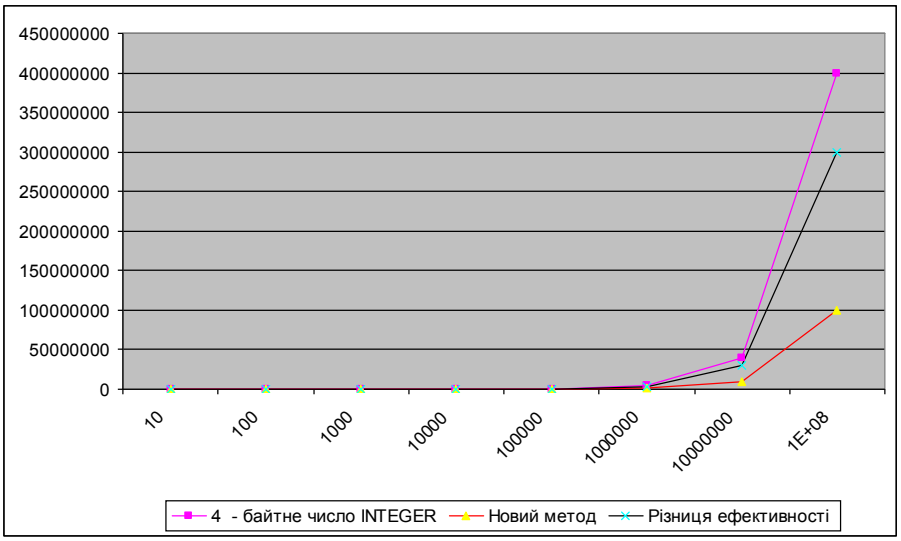


Рисунок 2.10 - Графіки порівняння використання пам’яті

Розподіл простих чисел не є рівномірним, тому проведено дослідження розподілу простих чисел з однаковими закінченнями для виявлення збіжності кількості простих чисел між бітами синхронізації. Таблиця 2.10 ілюструє мінімальні кількості простих чисел між бітами синхронізації при зростаючих розрядностях послідовності простих чисел.

Таблиця 2.10 - Кількість мінімальних повторень старших бітів в залежності від розрядності простого числа з відомим закінченням

Мінімальна кількість повторень	Розрядність
31	13
24	14

Продовження таблиці 2.10

Мінімальна кількість повторень	Розрядність
20	16
21	14
19	16
14	17
12	18
11	19
10	20
9	20
7	21
6	23
5	25
4	26
2	26

В результаті аналізу кількості повторень закінчень простих чисел на мінімальні значення було виявлено, що при обробці всіх 32-розрядних простих чисел кількість повторів не менше 2 (таблиця 2.10).

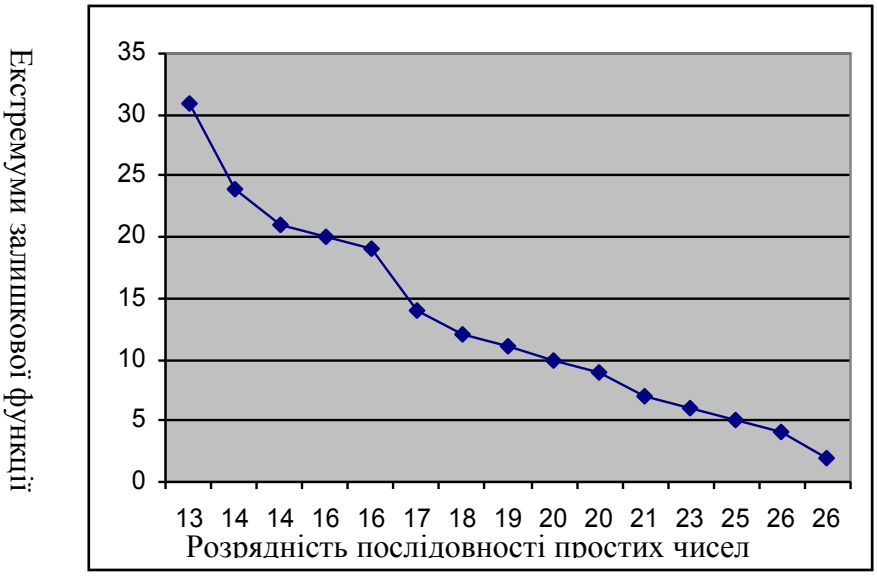


Рисунок 2.11 - Збіжність послідовності простих чисел між бітами синхронізації

Для детальнішої оцінки ефективності методу на рисунку 2.12 приведений графік, що ілюструє кількість простих чисел, що знаходяться між зростаннями групи верхніх розрядів (бітами синхронізації).

Наведені дослідження ефективності проведені для схеми, при якій відбувається поділ на 4 нижніх розряди послідовності простих чисел з однаковими закінченнями, наступних 7 бітів та біту синхронізації, кількість яких виражає значення верхніх розрядів такого поділу [56].

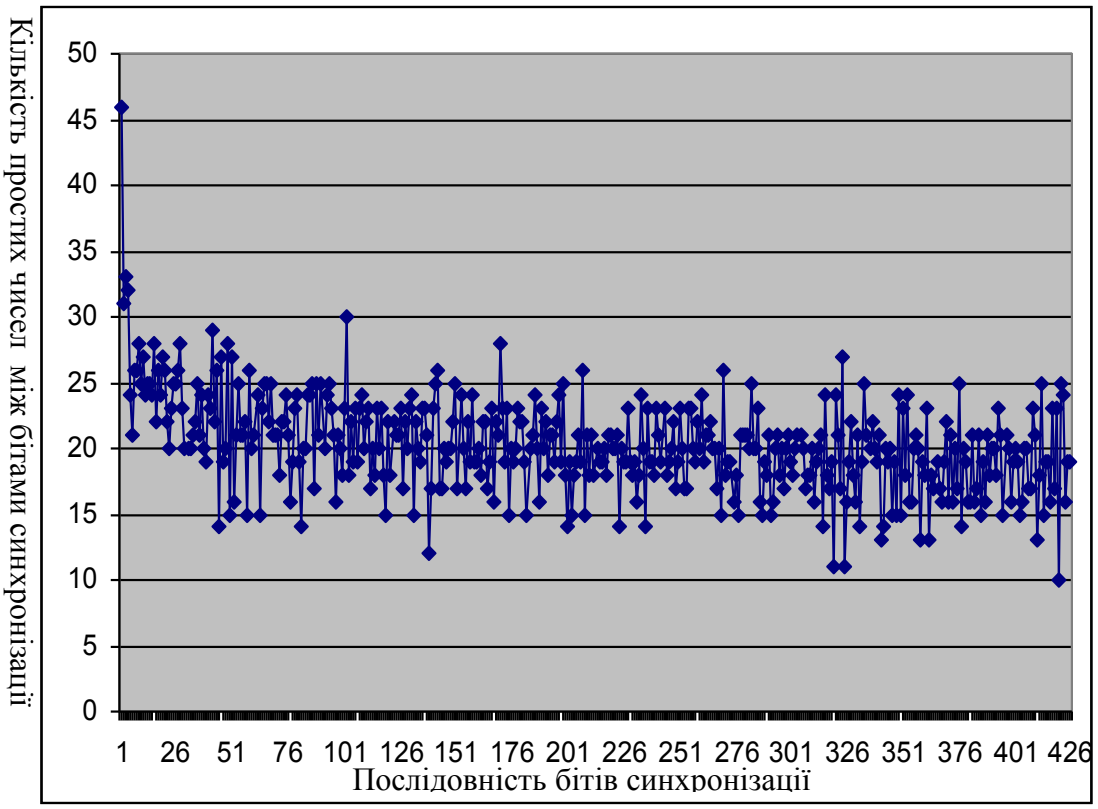


Рисунок 2.12 – Кількість екстремумів між бітами синхронізації

Для розрядностей 1024 біти і більше доцільна реалізація, при якій зберігається 15 біт та біт синхронізації. Якщо не поділяти послідовність простих чисел на групи з різними двійковими закінченнями, то кількість простих чисел між бітами синхронізації буде значно більшою в порівнянні з схемою, де числа групуються по 4-бітних двійкових закінченнях, та збільшиться в 15 раз, що підвищить ефективність зберігання БПЧ.

Дослідження на збіжність кількості простих чисел з однаковими

закінченнями між бітами синхронізації показують, що для ефективного збереження послідовності багаторозрядних простих чисел, більших 64 біт, доцільно дещо змінити схему кодування, використавши 15 біт молодших розрядів з послідовності та біт синхронізації, який буде супроводжувати наростання бітів в старших 15 розрядах.

Отже, розроблений метод дозволяє в декілька порядків збільшити швидкодію доступу та збереження інформації, оскільки для запису 32 – бітного числа використовуються лише семибітне закінчення та біт синхронізації. Для збереження послідовності багаторозрядних чисел до 1024 для кожного простого числа зазначеної розрядності необхідно 128 байт, а при використанні розробленого методу - лише 2 байти.

ВИСНОВКИ ДО РОЗДІЛУ 2

1. Розроблений метод знаходження залишку БРЧ, основними перевагами якого є зменшення надлишкового використання пам’яті та кількості порівнянь, що дозволяє зменшити обчислювальну складність на 1-2 порядки.

2. Розроблено ефективний метод модулярного множення на основі використання векторно-модульних операцій в ТЧБ Радемахера - Крестенсона, який дозволяє в два рази зменшити кількість суматорів при виконанні даної операції та на 50 % зменшити часову складність в порівнянні з матрично-модульним алгоритмом.

3. Досліджено обчислювальні процеси пошуку квадратичних лишків БРЧ та розроблено алгоритм пошуку квадратичного лишку з використанням ТЧБ Крестенсона, який, на відміну від існуючих, характеризується меншою обчислювальною складністю і дозволяє ефективно визначати повні квадрати по модулю P.

4. Розроблено метод компактного кодування багаторозрядних простих чисел, який, на відміну від існуючих, характеризується лінійно-логарифмічною обчислювальною складністю та дозволяє в декілька порядків збільшити швидкодію доступу та збереження інформації.

РОЗДІЛ 3

РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДІВ ОПРАЦЮВАННЯ БАГАТОРОЗРЯДНИХ ЧИСЕЛ У ЗАДАЧАХ ФАКТОРИЗАЦІЇ НА ОСНОВІ ТЕОРЕТИКО – ЧИСЛОВОГО БАЗИСУ ХААРА - КРЕСТЕНСОНА

3.1 Метод визначення околу рішення задачі факторизації

Метод Ферма ґрунтується на розв’язанні діофантового рівняння виду:

$$F_k - P_0 = \dot{\Delta}^2 \qquad (3.1)$$

де P_0 – відоме число, яке рівне добутку двох БРПЧ, Fk - повний квадрат:

$$P_0 = P_1 \times P_2 \qquad (3.2)$$

Метод пошуку P_1 і P_2 є складним, оскільки потрібно здійснювати операцію ділення над БРЧ, що призводить до експоненційного зростання складності:

$$\frac{P_0}{\breve{P}_1} = P_2 \ , \ P_1 < P_2 \qquad (3.3)$$

При розрядності 100 – 1000 біт P_0 і відповідно 50 – 500 біт P_1 та P_2 приводить до пошуку тільки одного розв’язку у цілих числах у діапазоні 2^{50} – 2^{500}.

Дослідження операції множення багаторозрядних двійкових чисел (768 біт) на основі матриць (рисунок 3.1) кодового представлення $P_1 \times P_2$ показують розподіл наскрізних переносів, що, в свою чергу, вказує кількість одиниць або нулів в добутку.

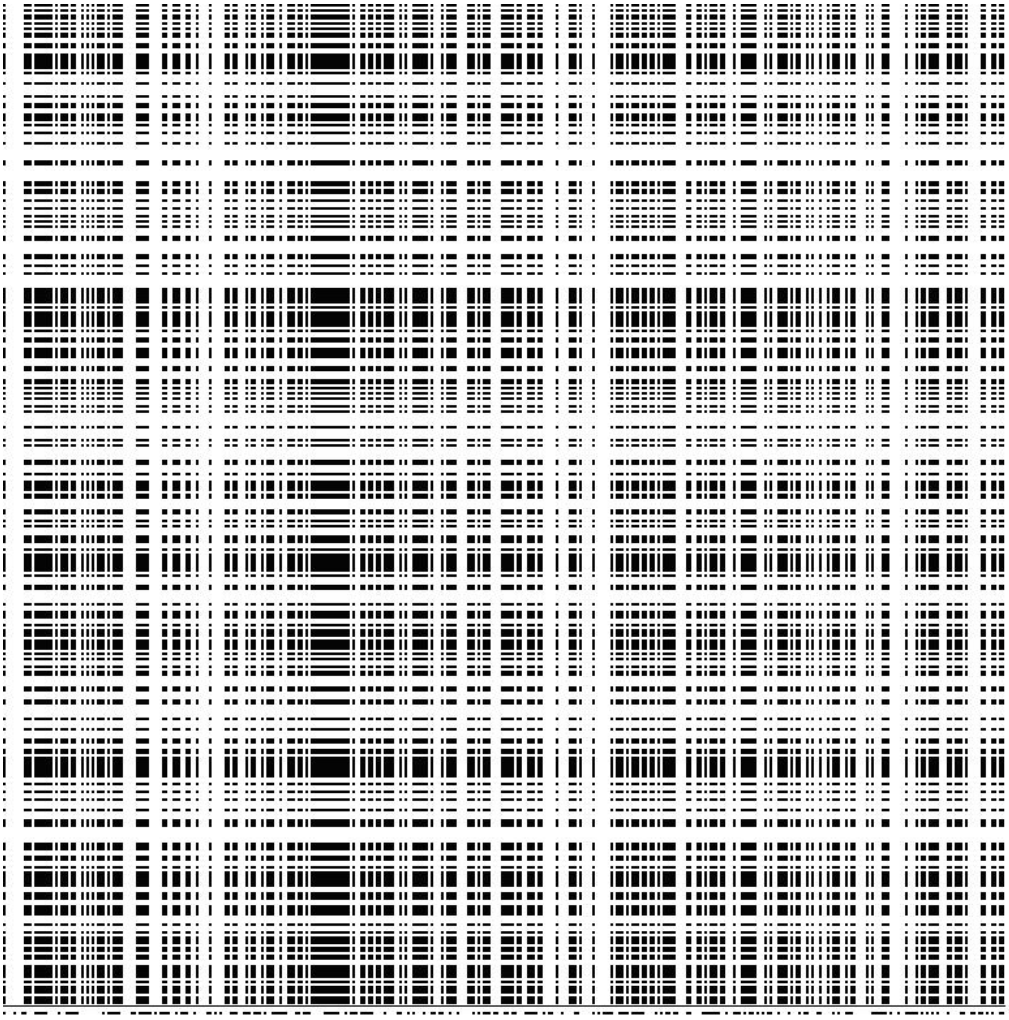


Рисунок 3.1 – Матриця кодового представлення множення багаторозрядних двійкових чисел (768 біт)

Згідно методу Ферма, для факторизації числа P_0 виконуються наступні операції: обчислюється $\sqrt{P_0}$, яке округлюється до більшого цілого P_c^*. Підноситься P_c^* до квадрату $F_1 = P_c^{*2}$, після чого формується послідовність квадратів згідно співвідношень $F_1 = (P_c^*+1)^2$, $F_2 = (P_c^*+2)^2$,…, $F_k = (P_c^*+k)^2$. Перевіряється, чи добувається цілочисельний корінь з різниці $\dot{\Delta} = \sqrt{F_k - P_0}$. Якщо добувається, то числа P_1 і P_2 знаходяться згідно виразу:

$$P_1 = \dot{\Delta} - P * c + k + \dot{\Delta} = P_2 \qquad (3.4)$$

Обчислювальна складність методу Ферма для БРЧ експоненційна. Із збільшенням розрядності вона відповідно зростає, оскільки число процесів k може складати 2^{300} - 2^{400} і тільки на єдино правильному кроці можливе однозначне рішення задачі факторизації. Слід зазначити, що при використанні даного методу необхідно підносити до квадрату числа P_c+k з розрядністю 300-500 біт, що приводить до необхідності кожного разу знаходити різницю $F_к$-P_0 та добувати корені квадратні з цієї різниці.

Графічно модель факторизації такого методу представлено на рисунку 3.2.

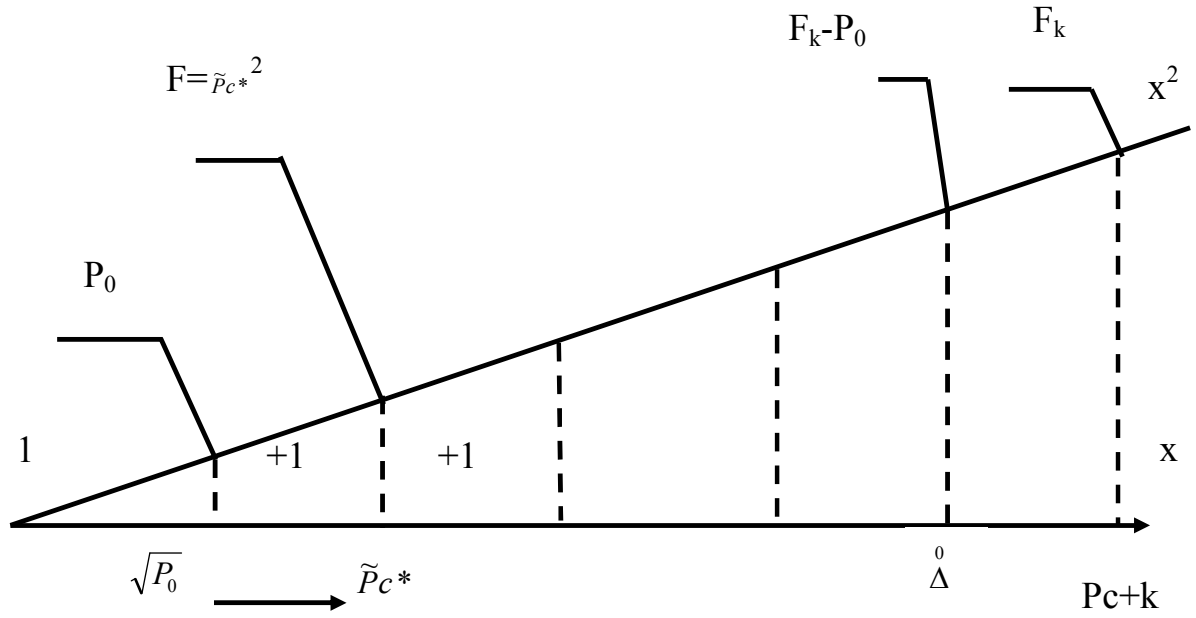


Рисунок 3.2 - Граф – модель спрощеного алгоритму факторизації на основі теореми Ферма

Отже, з врахуванням недоліків методу Ферма, розроблено метод, в основі якого лежать наступні операції: добувається $\sqrt{P_0}$ і округлюється до більшого цілого $\left|\sqrt{P_0}\right| = \widetilde{P}_c$. Піднімається $\widetilde{P}_c$ один раз до квадрату і обчислюються значення S_k згідно співвідношення:

$$S_k = k(2\widetilde{P}_c + k) + \Delta_0 . \qquad (3.5)$$

Значення $\sqrt{S_k} = \overset{0}{\Delta}$ перевіряється на існування цілого кореня і для єдиного знайденого k з рівняння (3.5) визначаються шукані P_1 і P_2.

Оскільки k – багаторозрядне число, то метод, в якому відсутні операції піднесення $\widetilde{P}c + k$ до квадрату, буде мати меншу обчислювальну складність. Числа, що використовуються в методі, мають значно меншу розрядність, ніж в алгоритмі Ферма, як зображено на рисунку 3.3 граф–моделі вдосконаленого методу відображення кроків S_k.

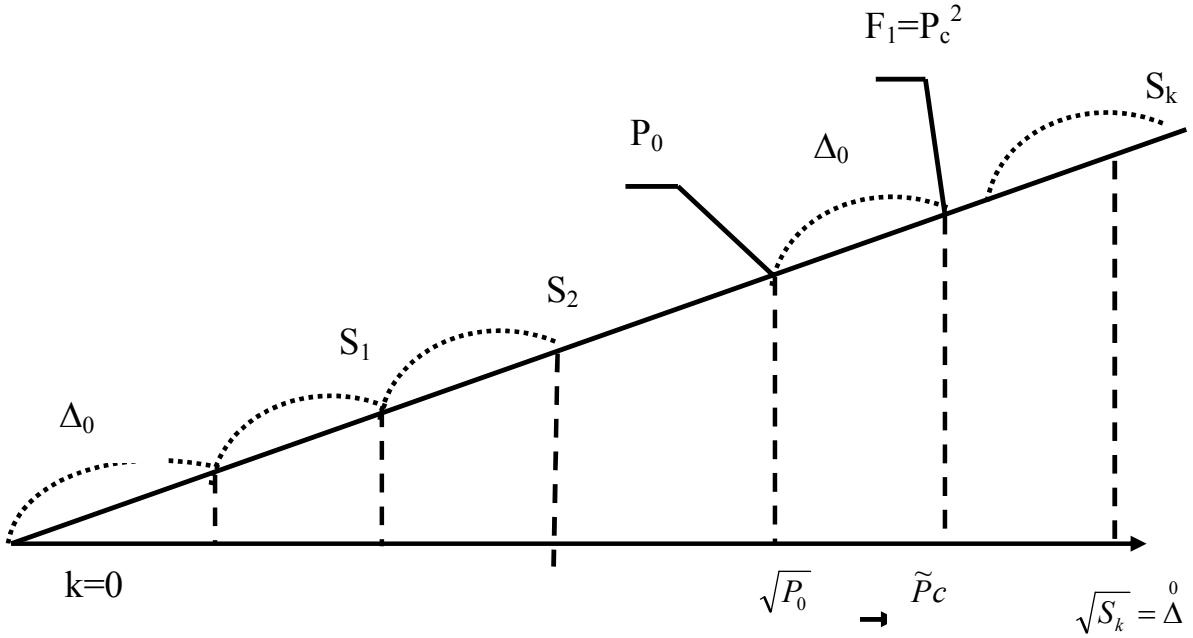


Рисунок 3.3 – Граф–модель спрощеного алгоритму факторизації на основі теореми Ферма

Це дозволяє уникнути операцій піднесення до степеня та зменшити розрядність операндів на декілька порядків, як показано в таблиці 3.1, а саме на P_0.

З таблиці 3.1 видно, що у вдосконаленому методі виключається операція піднесення до квадрату. Крім того, арифметичні дії виконуються

над числами, розмірність яких на декілька порядків менша, ніж у класичному методі.

Таблиця 3.1 – Приклад факторизації класичним та удосконаленим методом

k	n	$(\Delta_n)^2$, класичний метод	$(\Delta_n)^2$, вдосконалений метод
1	62	62^2-3811=33	62^2-3811=33
2	63	63^2-3811=158	33+125=158
3	64	64^2-3811=285	158+127=285
4	65	65^2-3811=414	285+129=414
5	66	66^2-3811=545	414+131=545
6	67	67^2-3811=678	545+133=678
7	68	68^2-3811=813	678+135=813
8	69	69^2-3811=950	813+137=950
9	70	70^2-3811=1089=33^2	950+139=1089=33^2

Таким чином, отримано розклад числа 3811 на прості множники:

$$3811 = 70^2 - 33^2 = (70 + 33)(70 - 33)= 103\cdot37.$$

Кількість ітерацій в обох випадках буде однаковою, а найскладнішою залишається операція перевірки квадратичності лишку. Для зменшення її обчислювальної складності можна використати СЗК.

Слід зазначити, що число кроків k, як показали тестові приклади, для відомих RSA приблизно на 6-8 двійкових розрядів менші від числа розрядів шуканих чисел P_1 і P_2, тобто розрядність їх однакова і рівна половині розрядності P_0. Запропоновано метод пошуку розрядності k згідно виразів:

$$\widetilde{\Delta} = \sqrt{\widetilde{S}k} = \sqrt{\widetilde{k}(2\widetilde{P}c + k)} \tag{3.6}$$

Звідки:

$$\widetilde{P}_1 = \widetilde{\Delta} - \widetilde{P}c + \widetilde{k} + \widetilde{\Delta} = \widetilde{P}_2 \tag{3.7}$$

Якщо $\hat{E}[\log_2(P_0 - \widetilde{P}_0)]npu\ \hat{E}[\log_2 k]$, тоді k змінюється двійково-логарифмічно 1, 2, 4, 8, 16, 32 … 2^i. У результаті для відомих (тестових) багаторозрядних чисел RSA отримуються наступні результати досліджень змінної k (рисунок 3.4.).

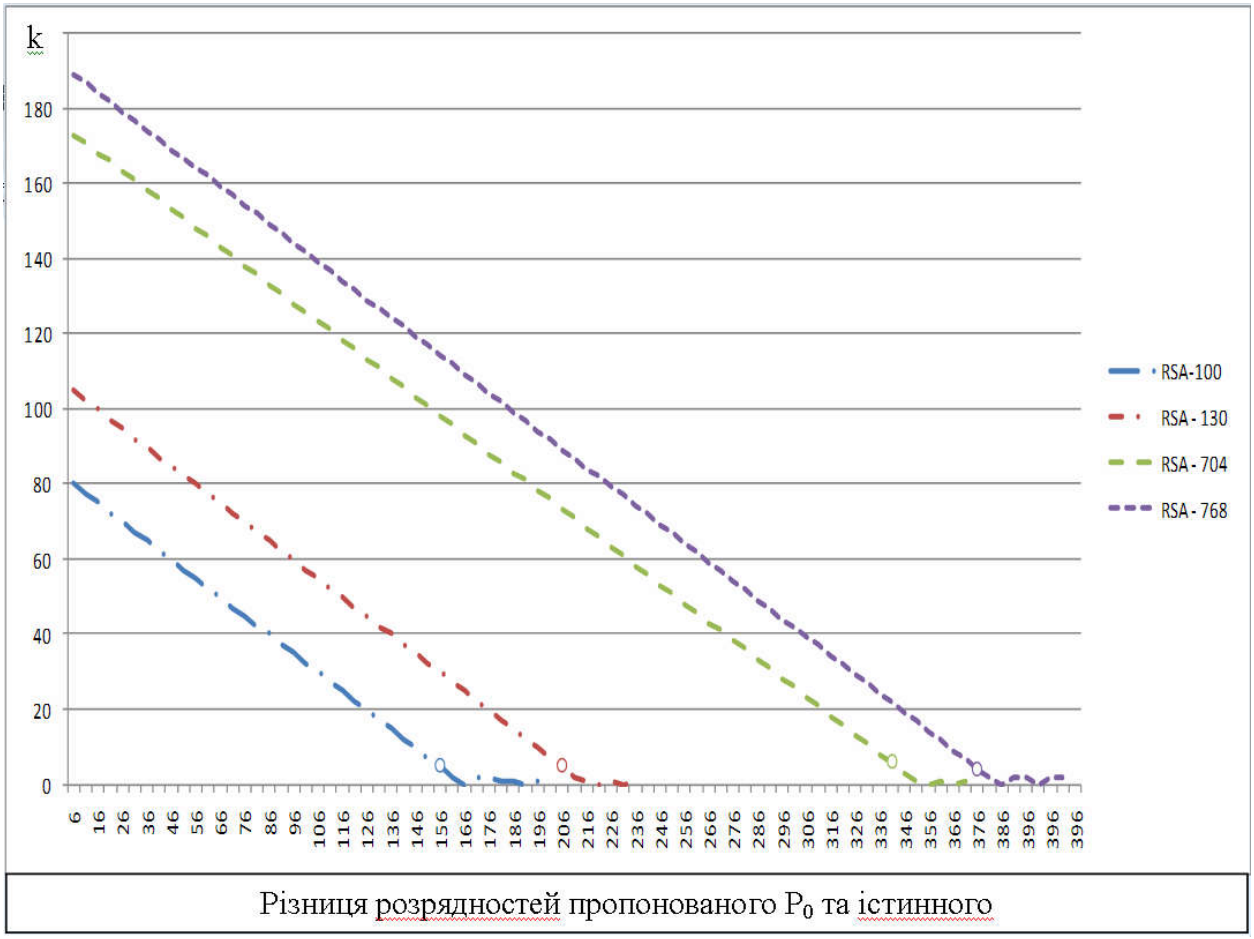


Рисунок 3.4 - Граф – модель пошуку розрядності $k=2^i$ для відомих RSA

Деталізація порозрядного дослідження k для відомих RSA 768 та

згенерованого випадковим чином БРЧ 795 бітів, добутку двох чисел розрядністю 397 бітів приведена на рисунку 3.5.

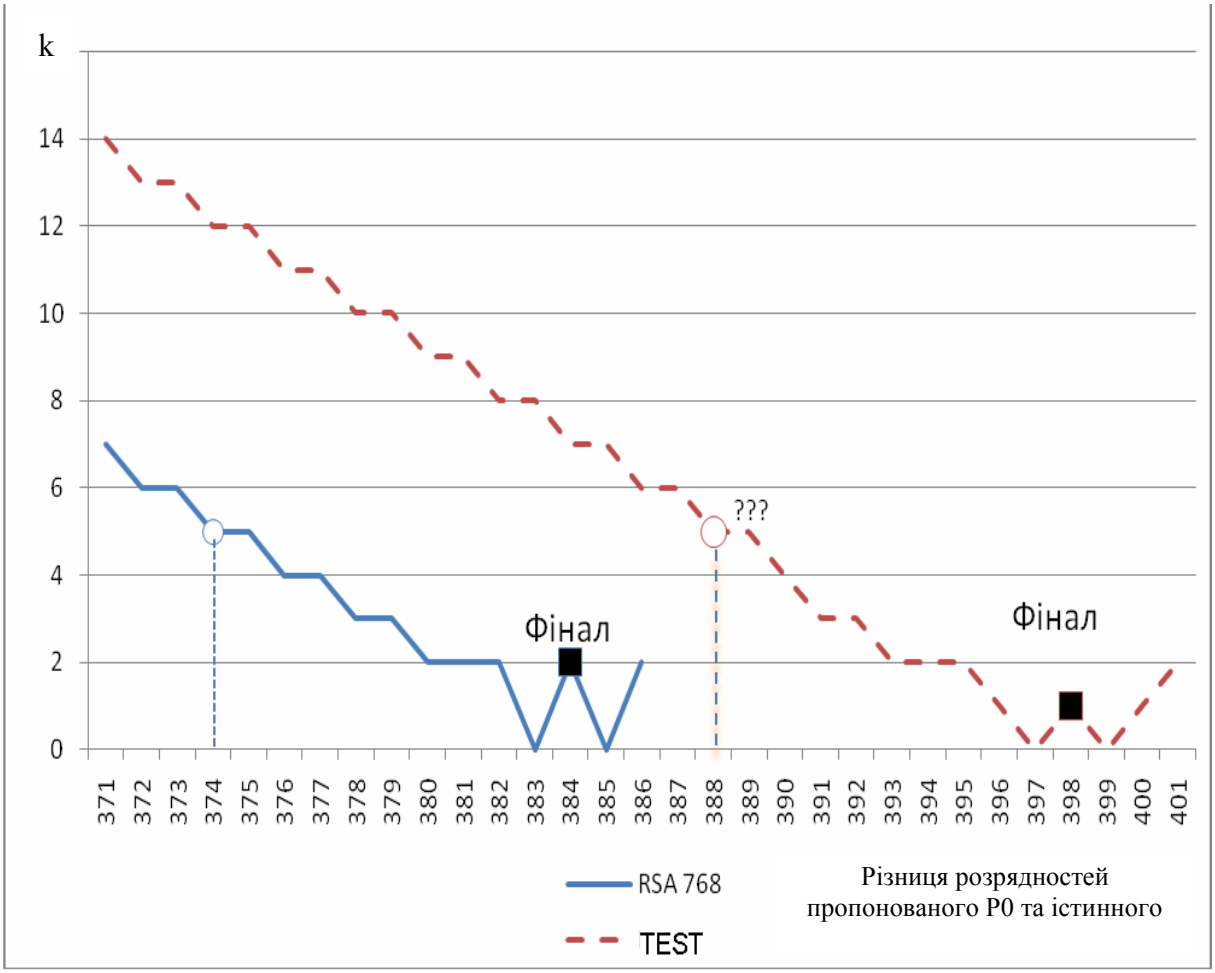


Рисунок 3.5 – Деталізована модель пошуку розрядності $k=2^i$ для тестових відомих RSA

Деталізована модель пошуку розрядності невідомих множників, що на рисунку 3.5, ілюструє різницю двійкових логарифмів добутків для шуканого розв'язку та числа, що факторизується.

Метод реалізується на основі виразу (3.4), визначаються пропоновані множники та добуток для порівняння з числом, що факторизується, причому число k змінюється ітераційно та порозрядно логарифмічно зростає. На кожній ітерації методу обчислюється логарифм різниці пропонованого добутку та числа, що факторизується, причому процес продовжується доти, поки пропонований добуток для заданого k не буде дорівнювати 0 або ж не

стане від'ємним.

Властивість "сильного" двійкового коду *k* дозволяє звузити границі можливих його значень згідно властивостей двійкових кодів чисел (таблиця 3.2)

Таблиця 3.2 - Звуження діапазону можливих значень *k*

K min				K max			
P0	P1	P2	k	P0	P1	P2	k
10101000000	11000	11100	10011	1000000000	100000	11111	10101
00000000000	00000	00000	10111	0000000000	000000	11111	11110
00000000000	00000	00000	00010	0000000000	000000	11111	11000
00000000000	00000	00000	11010	0000000000	000000	11111	01100
00000000000	00000	00000	11101	0000000000	000000	11111	11001
00000000000	00000	00000	10011	0000000000	000000	11111	10000
00000000000	00000	00000	11011	0000000000	000000	11111	00011
00000000000	00000	00000	01100	0000000000	000000	11111	00010
00000000000	00000	00000	11100	0000000000	000000	11111	00011
00000000000	00000	00000	00101	0000000000	000000	11111	00110
00000000000	00000	00000	01010	0000000000	000000	11111	11011
00000000000	00000	00000	00000	0000000000	000000	11111	11011
00000000000	00000	00000	11000	0000000000	000000	11111	10100
00000000000	00000	00000	10010	0000000000	000000	11111	11010
00000000000	00000	00000	00100	0000000000	000000	11111	00001
00000000000	00000	00000	11011	0000000000	000000	11111	00111
00000000000	00000	00000	10100	0000000000	000000	11111	01100
00000000000	00000	00000	10100	0000000000	000000	11111	10011
00000000000	00000	00000	00011	0000000000	000000	11111	00100
00000000000	00000	00000	10011	0000000000	000000	11111	01010
00000000000	00000	00000	11110	0000000000	000000	11111	10110
00000000000	00000	00000	11100	0000000000	000000	11111	10101
00000000000	00000	00000	00111	0000000000	000000	11111	00000
00000000000	00000	00000	00110	0000000000	000000	11111	10110
00000000000	00000	00000	00010	0000000000	000000	11111	00001
00000000000	00000	00000	01100	0000000000	000000	11111	11000
00000000000	00000	00000	01001	0000000000	000000	11111	10100
00000000000	00000	00000	01111	0000000000	000000	11111	10000
00000000000	00000	00000	00111	0000000000	000000	11111	10011
00000000000	00000	00000	00000	0000000000	000000	11111	11101
00000000000	00000	00000	00011	0000000000	000000	11111	00010
00000000001	00000	00000	11000	0000000000	000000	11111	10111
10100000000	00000	00000	00101	0000000000	000000	11111	01101

Продовження таблиці 3.2

K min				K max			
P0	P1	P2	k	P0	P1	P2	k
00000000000	00000	00000	11100	00000000000	0000000	11111	10001
00000000000	00000	00000	11000	00000000000	0000000	11111	00010
00000000000	00000	00000	01011	01111111111	0000000	11111	10111
00000000000	00000	00000	11110	11111111111	0000000	11111	10110
00000000000	00000	00000	11011	11111111111	0000000	11111	01100
00000000000	00000	00000	11011	11111111111	0000000	11111	01001
00000000000	00000	00000	10011	11111111111	0000000	11111	01110
00000000000	00000	00000	11000	11111111111	0000000	11111	10000
00000000000	00000	00000	10001	11111111111	0000000	11111	10100
00000000000	00000	00000	01010	11111111111	0000000	11111	11011
00000000000	00000	00000	10111	11111111111	0000000	11111	11010
00000000000	00000	00000	11101	11111111111	0000000	11111	01111
00000000000	00000	00000	00110	11111111111	0000000	11111	10011
00000000000	00000	00000	01100	11111111111	0000000	11111	00110
00000000000	00000	00000	10000	11111111111	0000000	11111	00110
00000000000	00000	00000	01010	11111111111	0000000	11111	01101
00000000000	00000	00000	00010	11111111111	0000000	11111	11010
00000000000	00000	00000	11100	11111111111	0000000	11111	10101
00000000000	00000	00000	11111	11111111111	0000000	11111	10101
00000000000	00000	00000	00111	11111111111	0000000	11111	00000
00000000000	00000	00000	11111	11111111111	0000000	11111	01101
00000000000	00000	00000	11101	11111111111	0000001	11111	11110
00000000000	00000	00000	10011	11111111111		11111	01111
00000000000	00000	00000	01100	11111111111		11111	10111
00000000000	00000	00000	10000	11111111111		11111	10001
00000000000	00000	00000	10101	11111111111		11111	01010
00000000000	00000	00000	10011	11111111111		11111	00111
00000000000	00000	00000	01001	11111111111		11111	01011
00000000000	00000	00000	00011	11111111111		11111	10100
00000000000	00000	00000	10101	11111111111		11111	11110
00000000001	00000	00000	11101	11111111111		11111	00101
	00000	00000	01010	11111111111		11111	01110
	00000	00000	00100	11111111111		11111	10011
	00000	00000	01001	11111111111		11111	10000
	00000	00000	00000			11111	00001
	00000	00000	0011			11111	10001
	00000	00000				11111	111
	01	01				11	

Аналіз двійкових кодів $\tilde{k}$ показує, що вони близькі до класу ортогональних кусково – постійних функцій ТЧБ Радемахера (меандри), Грея

(фазо-зсувні меандри), Уолша (добутки фрагментів функцій Радемахера), а в окремих випадках рекурентних функцій базису Галуа або їх фрагментів з різними ключами.

Запропонований метод дозволяє виявити розрядності множників в процесі дослідження числа, що факторизується, та розрядність числа k, кількості кроків до наступного цілого квадрату як єдиного розв'язку задачі факторизації, що дозволяє проводити перебір Ферма - подібними методами в розрядності розв'язку.

3.2 Дослідження збіжності екстремумів залишкової функції в околі розв'язку задачі факторизації

Проведений аналіз існуючих методів розкладу на множники та тестів простоти говорить про те, що розв'язання даної задачі далеке від досконалості. З розвитком інформаційних технологій зростають розмірності вхідних параметрів криптосистем, що в свою чергу призводить до збільшення обчислювальної складності при генерації та опрацюванні багаторозрядних простих чисел. Тому розробка ефективного методу пошуку простих чисел та методів знаходження околу розв'язків факторизації є актуальною задачею, що дозволить зменшити часову складність RSA-подібних асиметричних алгоритмів шифрування.

В результаті дослідження процесу факторизації числа P_0, яке розкладається на множники прямим перебором, потрібно виконати наступні операції: знайти $P^* = \left|\sqrt{P_0}\right|$ - непарне число, тоді обчислити значення згідно співвідношень $P_1 = P_0 \bmod P^*$, $P_n = P_0 \bmod P^* + 2_n$, $n \in \mathrm{Z}$ [30].

На наступному кроці будуються послідовності:

$$P_2 = P_1 + Q_1, P_3 = P_2 + Q_2, \ldots, P_n = P_{n-1} + Q_{n-1}, \qquad (3.8)$$

де $Qi = 2i\,, i = 1, \ldots, n-1$

Аналогічним чином знаходиться значення $P_{1*} = P_0 \bmod(P*-2i), i = 1, \ldots, n-1$.

$$P_1* = P_0 \bmod P* = P_1$$
$$P_2* = P_0 \bmod(P*-2) \qquad (3.9)$$
$$P_3* = P_0 \bmod(P*-4)$$

де P_i* - залишкова функція.

Слід відмітити, що в результаті таких перетворень значення залишків мають пилкоподібну характеристику, яка представлена на рисунку 3.6.

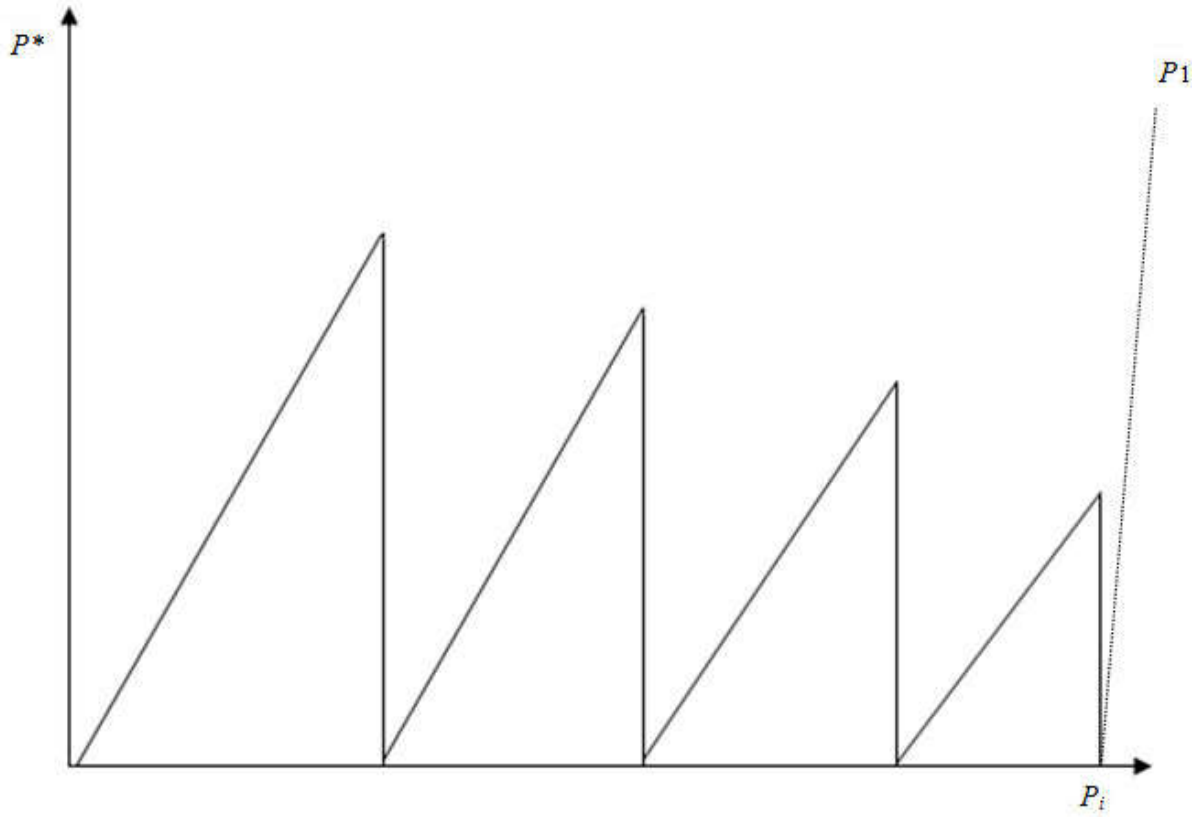


Рисунок 3.6 – Пилкоподібна залишкова функція

В результаті експериментів отримано, що кількість зростань залишків стабільно наближається до зони розв’язку і це, в свою чергу, дозволяє її

локалізувати. Даний підхід зменшує діапазон та обчислювальну складність пошуку розв'язання задачі факторизації БРЧ [30].

Наприклад:

$P_0 = 445023604703_{10} = 1100111100111010111100101001111111011111_2$

$P* = 667101_{10} = 10100010110111011101_2$

$P_1 = 10010101001111100001_2$

$P_2 = 10110001101110111111_2$

Далі від $P*$ потрібно відняти 2.

Даний процес продовжується на основі аналітичних співвідношень доти, поки не локалізується діапазон розв'язку, тобто:

$$P*_n = P_0 \bmod(P*-2n), n \in Z \tag{3.10}$$

$$P_2* = P_1* + Q_1*, P_3* = P_2* + Q_2*, \ldots, P_n* = P_{n-1}* + Q_{n-1}*$$

$$Q_1 = |P_2 - P|, Q_2* = |P_3 - P_2|, \ldots, Q_{n-1} = |P_n - P_{n-1}|$$

$$Q_1* = |P_2* - P_1*|, Q_2* = |P_3* - P_2*|, \ldots, Q_{n-1}* = |P_n* - P_n* - 1|$$

$$Q_1 = |P_0 \bmod(P*+2) - P_0 \bmod P*|$$

$$Q_2 = |P_0 \bmod(P*+4) - P_0 \bmod(P*+2)|$$

$$Q_{n-1} = |P_0 \bmod(P*+2_n) - P_0 \bmod(P*+2_n - 2)|$$

$$Q_1* = |P_0 \bmod(P*-2) - P_0 \bmod P*|$$

$$Q_2* = |P_0 \bmod(P*-4) - P_0 \bmod(P*-2)|$$

...

$$Q_{n-1}* = \left| P_0 \bmod(P*-2_n) - P_0 \bmod\left(P*-2n+2\right) \right| \qquad (3.11)$$

В результаті проведених досліджень отримаємо таблицю 3.3 значень кількості змін залишкової функції $P*_n$ на початку перебору при заданих вхідних даних.

Таблиця 3.3 - Кількості змін залишкової функції на початку перебору [30]

1	186	261	157	123	105	93	85	77	73	68	64	61	58	56	54	52	51	48	48
46	44	44	43	41	41	40	39	38	38	36	37	35	35	35	34	33	33	32	32
32	31	31	30	30	30	29	29	29	28	28	28	27	27	27	27	26	26	26	26
25	25	25	25	25	24	24	24	24	23	24	23	23	23	23	22	22	23	22	22
21	22	21	22	21	21	21	21	21	20	20	21	20	20	20	20	20	19	20	19
19	20	19	19	19	18	19	19	18	19	18	18	18	19	18	17	18	18	18	17
18	17	17	18	17	17	17	17	17	17	16	17	17	16	17	16	17	16	16	16
16	16	16	16	16	16	15	16	16	15	16	15	16	15	15	15	16	15	15	15
15	15	15	14	15	15	14	15	15	14	15	14	14	15	14	14	15	14	14	14
14	14	14	14	14	13	14	14	14	13	14	13	14	13	14	13	14	13	13	14
13	13	13	13	14	13	13	13	13	12	13	13	13	13	13	12	13	13	12	13
12	13	12	13	12	13	12	12	13	12	12	12	13	12	12	12	12	12	12	12
12	12	12	12	12	12	11	12	12	12	11	12	12	11	12	12	11	12	11	12
11	11	12	11	12	11	11	11	12	11	11	11	11	12	11	11	11	11	11	11
11	11	11	11	11	11	10	11	11	11	11	10	11	11	10	11	11	10	11	11
10	11	10	11	10	11	10	11	10	10	11	10	10	11	10	10	11	10	10	10
10	11	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	9	10	10	10	10	9	10	10	10	9	10	10	9	10	9	10	10	9	10
9	10	9	10	9	10	9	10	9	9	10	9	9	10	9	9	10	9	9	10
9	9	9	9	10	9	9	9	9	9	10	9	9	9	9	9	9	9	9	9
9	9	9	9	9	9	9	8	9	9	9	9	9	8	9	9	9	9	8	9
9	9	8	9	9	8	9	9	8	9	9	8	9	8	9	8	9	8	9	9
8	9	8	8	9	8	9	8	9	8	8	9	8	9	8	8	9	8	8	8
9	8	8	9	8	8	8	8	9	8	8	8	8	9	8	8	8	8	8	8
8	8	9	8	8	8	8	8	8	8	8	8	8	8	8	7	8	8	8	8
8	8	8	8	8	7	8	8	8	8	8	7	8	8	8	7	8	8	8	7
8	8	8	7	8	8	7	8	8	7	8	7	8	8	7	8	7	8	8	7
8	7	8	7	8	7	8	7	8	7	8	7	8	7	8	7	7	8	7	8
7	7	8	7	7	8	7	8	7	7	7	8	7	7	8	7	7	7	8	7
7	7	8	7	7	7	8	7	7	7	7	7	8	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7	6	7	7	7	7	7	7	7	6	7	7
7	7	7	6	7	7	7	7	6	7	7	7	6	7	7	7	6	7	7	7

Для того, щоб знайти значення P_1 і P_2 $(P_0 = P_1 \cdot P_2)$, потрібно, щоб послідовності Q_{n-1}* збігались до 0. Результати дослідження можна відобразити у вигляді діаграми, представленої на рисунку 3.7.

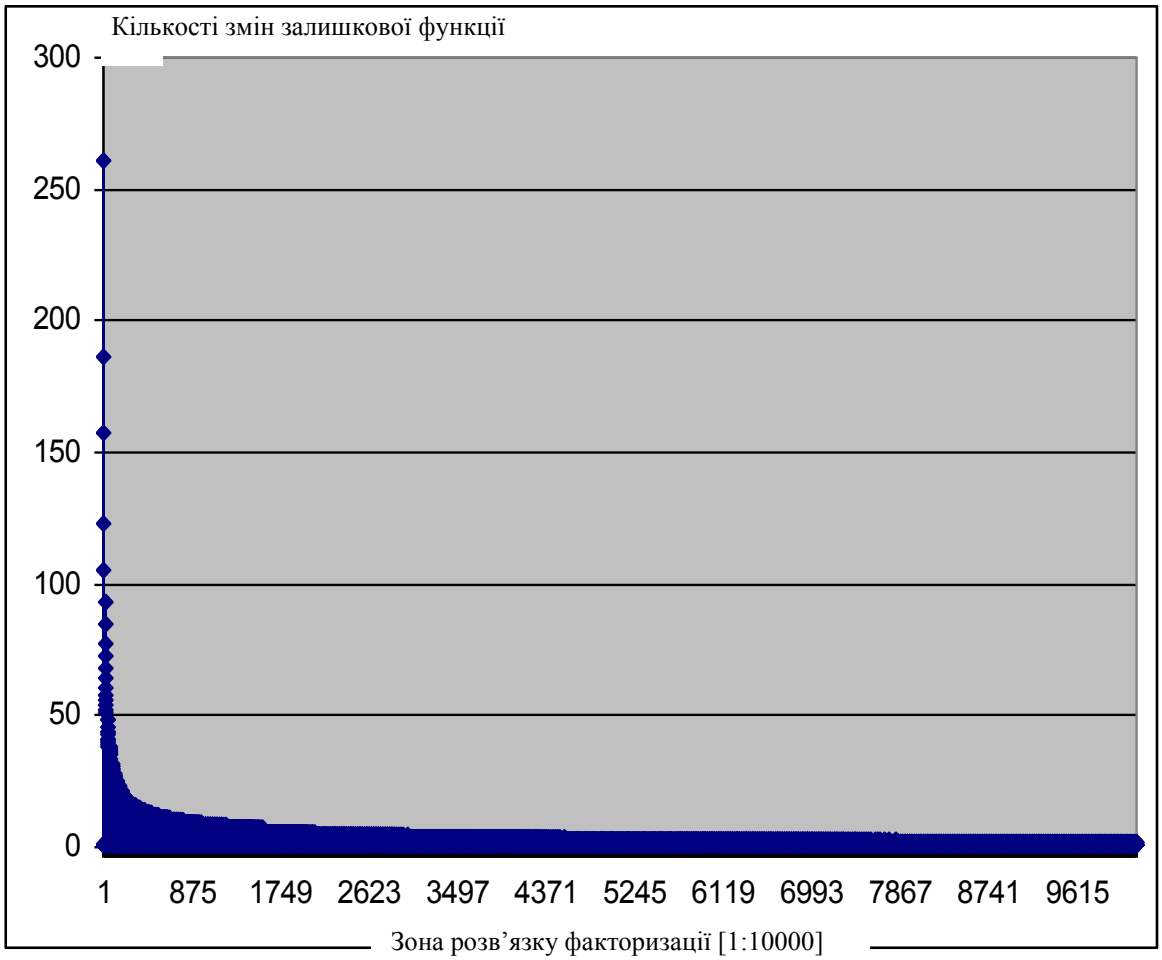


Рисунок 3.7 - Збіжність значення залишків в зоні розв'язку для прикладу з таблиці

В загальному випадку крок, при якому послідовності будуть збігатися, можна змінювати, тобто додавати і віднімати не тільки 2, а й інші значення. Тоді дані послідовності можна переписати наступним чином:

$$Q_{1k} = \left| P_0 - (P* + k) - P_0 \bmod P* \right|$$

$$Q_{2k} = \left| P_0 - (P^* + 2k) - P_0 \bmod P^* + k \right|$$

…

$$Q_{(n-1)k} = \left| P_0 - (P^* + kn) - P_0 \bmod (P^* + kn - k) \right| \qquad (3.12)$$

Відповідно

$$Q_{1k}{}^* = \left| P_0 - (P^* + k) - P_0 \bmod P^* \right|$$

$$Q_{2k}{}^* = \left| P_0 - (P^* + 2k) - P_0 \bmod (P^* - k) \right|$$

...

$$Q_{(n-1)k}{}^* = \left| P_0 - (P^* + kn) - P_0 \bmod (P^* + kn + k) \right| \qquad (3.13)$$

Даний перебір здійснюється до тих пір, поки $P^* \to 0, n \to Q_i$, де Q_i – значення, при якому $P^* - Q_i = P_1$ – розв'язок задачі факторизації.

В результаті проведених чисельних експериментів спостерігається симетричність екстремумів залишкової функції P^*_n відносно розв'язку задачі факторизації (рисунок 3.9).

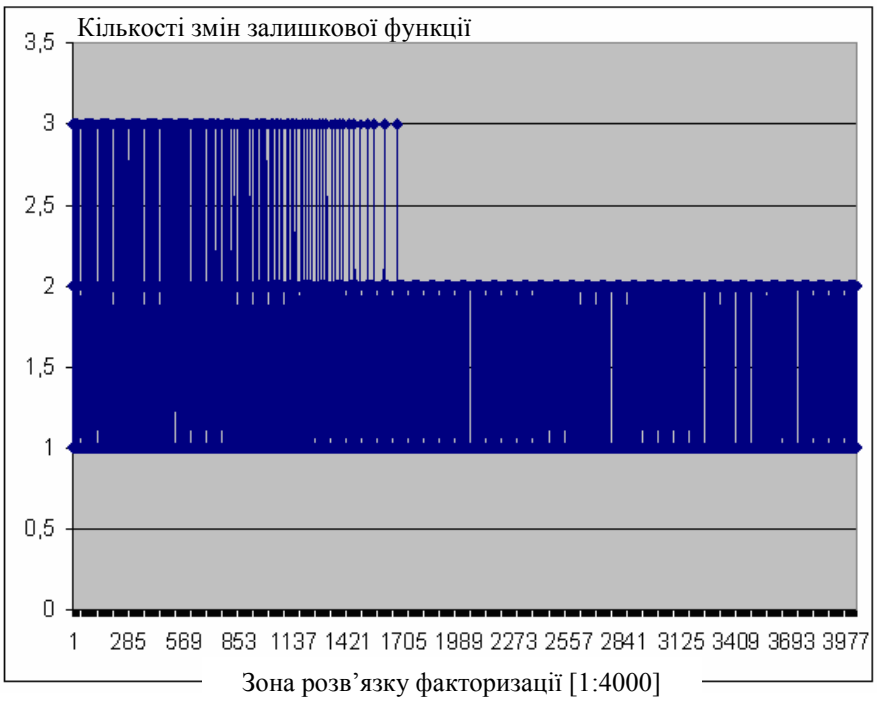


Рисунок 3.8 - Збіжність значення залишків в зоні розв'язку

В таблиці 3.4 представлено значення змін залишкової функції $P*_n$ в зоні розв'язку [30].

Таблиця 3.4 - Кількості змін залишкової функції в зоні розв'язку

1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2
2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	2	2	2	2
2	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2
2	1	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	2	2	2
2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	1	2
2	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	1	2
2	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	1	2
2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	2	1	2	2	2
2	2	2	2	2	1	2	2	2	2	2	2	2	2	1	2	2	2	2	2
2	2	1	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2	1	2
2	2	2	2	2	2	1	2	2	2	2	2	2	1	2	2	2	2	2	2
2	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1	2	2	2	2
2	2	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1	2	2	2
2	2	2	1	2	2	2	2	2	1	2	2	2	2	2	2	1	2	2	2
2	2	1	2	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2
2	1	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	2	1
2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	1	2	2	2
2	2	1	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	1	2
2	2	2	2	1	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2
1	2	2	2	2	1	2	2	2	2	1	2	2	2	2	1	2	2	2	2
1	2	2	2	2	1	2	2	2	2	1	2	2	2	2	1	2	2	2	2
1	2	2	2	2	1	2	2	2	2	1	2	2	2	2	1	2	2	2	1
2	2	2	2	1	2	2	2	2	1	2	2	2	1	2	2	2	2	1	2
2	2	2	1	2	2	2	1	2	2	2	2	1	2	2	2	1	2	2	2
2	1	2	2	2	1	2	2	2	2	1	2	2	2	1	2	2	2	1	2
2	2	2	1	2	2	2	1	2	2	2	1	2	2	2	1	2	2	2	2
1	2	2	2	1	2	2	2	1	2	2	2	1	2	2	2	1	2	2	2
1	2	2	2	1	2	2	2	2	1	2	2	2	1	2	2	2	1	2	2
2	1	2	2	2	1	2	2	2	1	2	2	1	2	2	2	1	2	2	2
1	2	2	2	1	2	2	2	1	2	2	2	1	2	2	2	1	2	2	2
1	2	2	1	2	2	2	1	2	2	2	1	2	2	2	1	2	2	1	2
2	2	1	2	2	2	1	2	2	1	2	2	2	1	2	2	2	1	2	2
1	2	2	2	1	2	2	2	1	2	2	1	2	2	2	1	2	2	1	2
2	2	1	2	2	1	2	2	2	1	2	2	1	2	2	2	1	2	2	1
2	2	2	1	2	2	1	2	2	2	1	2	2	1	2	2	2	1	2	2
1	2	2	1	2	2	2	1	2	2	1	2	2	1	2	2	2	1	2	2
1	2	2	1	2	2	2	1	2	2	1	2	2	1	2	2	2	1	2	2
1	2	2	1	2	2	1	2	2	2	1	2	2	1	2	2	1	2	2	1
2	2	1	2	2	2	1	2	2	1	2	2	1	2	2	1	2	2	1	2

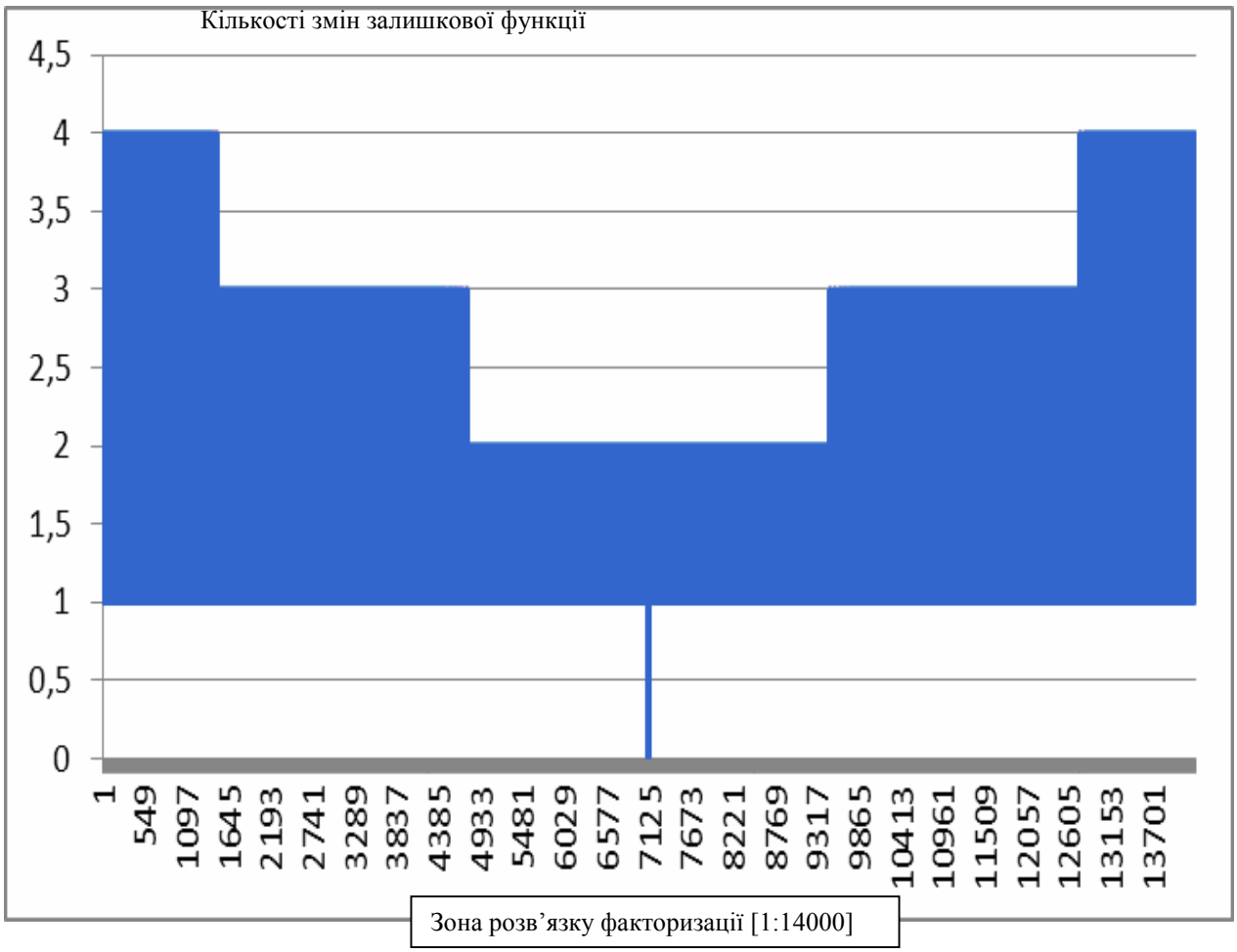


Рисунок 3.9 - Симетричність екстремумів залишкової функції в зоні розв'язку для заданого прикладу

В результаті таких обчислень отримується відображення значень залишків функції $P*_n$ в околі розв'язку, які проілюстровані на рисунку 3.8.

Інші дослідження показують аналогічну збіжність в зоні розв'язку. Це дозволяє значно спростити процес факторизації прямим перебором.

При досягненні розв'язку, тобто $P*_n = 0$, процес отримання залишків продовжується для $P*_{n+1}, \ldots, P*_{n+k}$.

В результаті реалізації запропонованого методу факторизації БРЧ на основі використання ТЧБ Крестенсона отримано збіжність кількості змін залишкової функції в зоні розв'язку для чисел RSA 100.

Результати чисельного експерименту свідчать про те, що кількість екстремумів до розв'язку симетрична кількості екстремумів після розв'язку. На рисунку 3.10 зображена графічна модель, яка відображає цю

властивість [30].

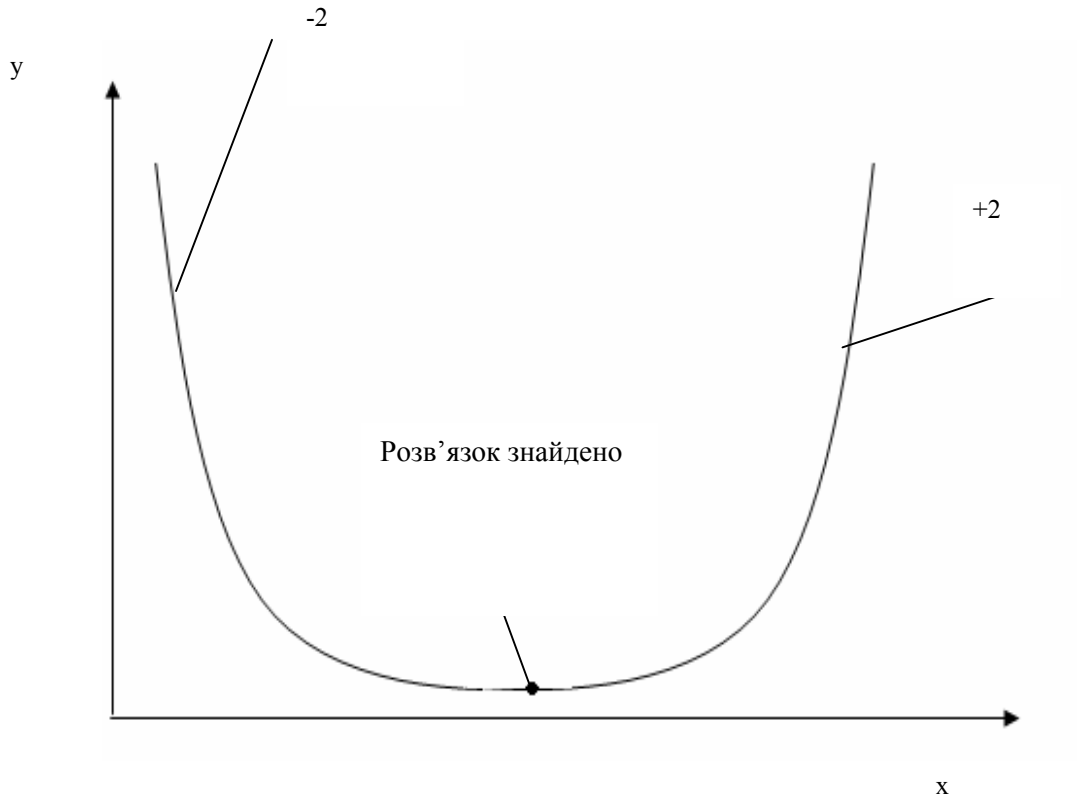


Рисунок 3.10 - Модель симетричності кількості екстремумів в зоні рішення RSA 100

Таким чином, можна розпочати пошук вірогідного відрізка з розв’язком, тим самим зменшивши вибірку перебору на декілька порядків.

Отже, якщо знайти два симетричних відрізки на однаковій відстані від розв’язку, то вони вкажуть на сам розв’язок.

Розглянемо ще декілька прикладів з багаторозрядними числами.

P_0 = 1110000001000000111011100010101010100111001101110010000 10101011001000111111011

$P*$=10101001011011001000011000011111000110 11

P_1 =11001110001010101111001010011011110 0101

P_2=10001011001110101000101010011010101 1111

Результати представлені на рисунку 3.11.

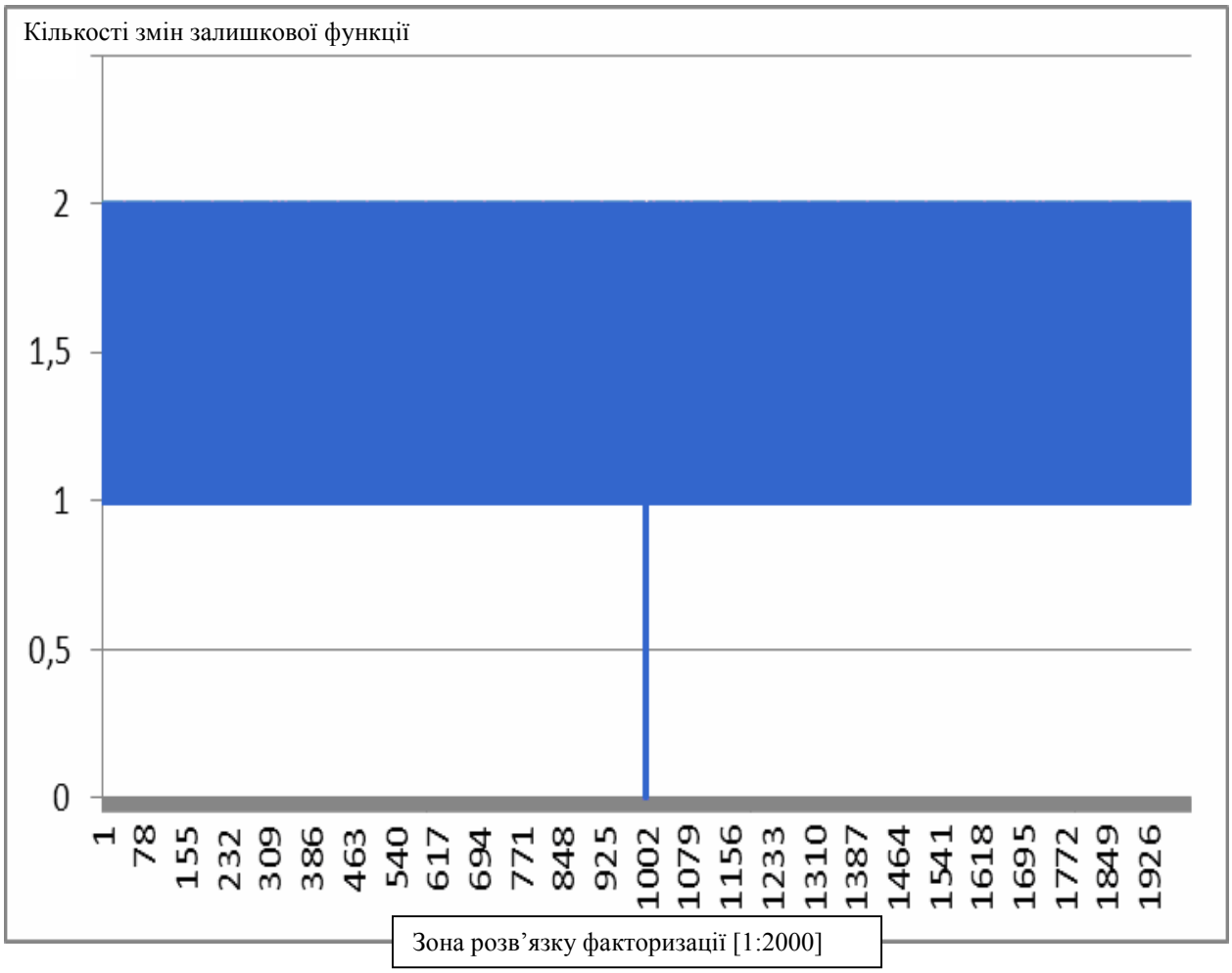


Рисунок 3.11 - Збіжність екстремумів залишкової функції для прикладу

P_0 = 1110000100000111001011010110100001110011001011001101001101010011010101000001011011001111111000000011

$P*$=10101001101101110101100011101001010001010001010000

P_1= 1100111001110001010101111001010011011110010101000 1

P_2= 1000101110000101111111000000111101000111100001001 1

Результати досліджень залишкової функції, представлені на рисунках 3.11, 3.12, 3.13, показують, що в результаті таких перетворень значення залишків характеризуються пилкоподібною функцією та дозволяють локалізувати зону розв’язку задачі факторизації, виявляти границі пошуку цього розв’язку.

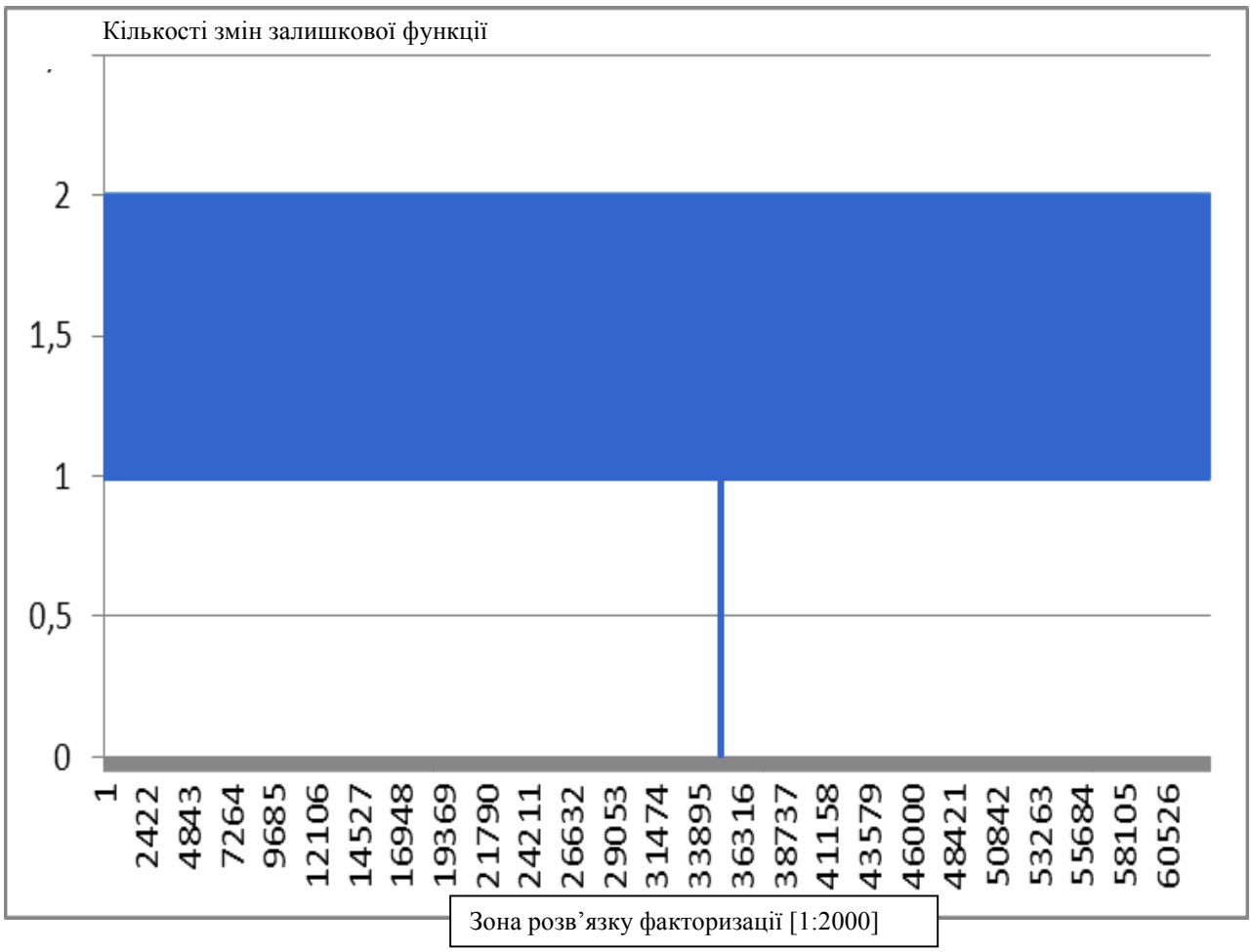


Рисунок 3.12 - Збіжність екстремумів залишкової функції для прикладу

Зона розв’язку, локалізована за допомогою дослідження екстремумів залишкової функції для БРЧ, може бути великого діапазону, тому дослідження доцільно проводити з багаторозрядним кроком. Виявивши симетричне зростання та спадання екстремумів залишкової функції, діапазон розв’язку буде знаходитись в межах 2 значень кількості зростань виявлених симетричних екстремумів залишкової функції [30].

Для БРЧ подібні дослідження провести досить складно, тому доцільно також дослідити суми екстремумів залишкових функцій з багаторозрядним кроком. Наприклад, розглянемо RSA 100.

Як показано на рисунку 3.13, в результаті проведених досліджень сум екстремумів залишкових функцій з багаторозрядним кроком виявлено зону зростання їх значення, яку необхідно вивчати на предмет симетричності значень екстремумів кількості зростань залишкової функції.

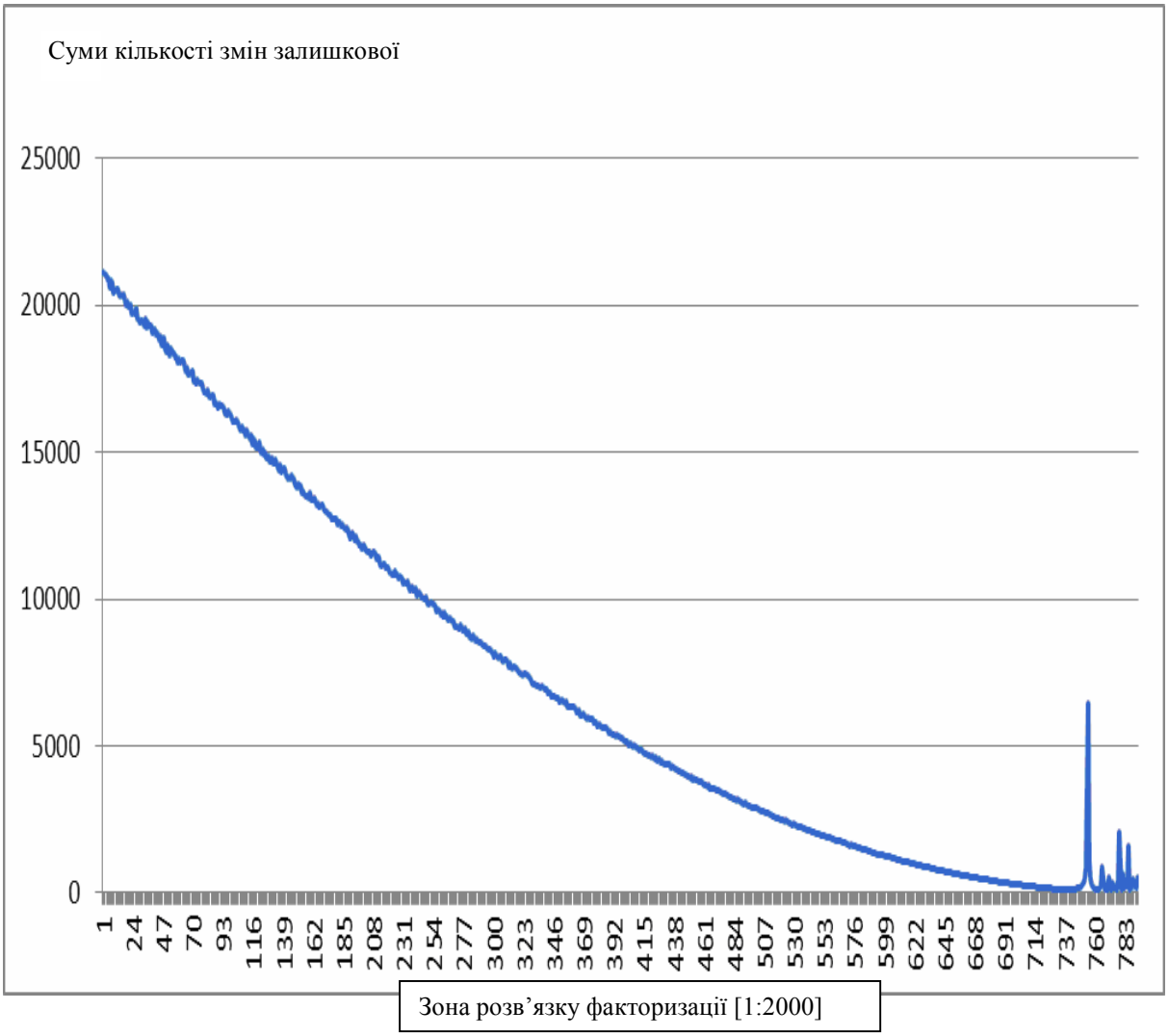


Рисунок 3.13 - Результати дослідження екстремумів залишкової функції для RSA 100 з використанням багаторозрядного кроку

Аналіз чисельного експерименту рисунків 3.10, 3.11, 3.12 свідчить, що розв'язок задачі факторизації запропонованим методом дозволяє зменшити діапазон пошуку, а саме окіл точки, при якому функція $P^* \to 0, n \to Q_i$.

$$P_0 \bmod P^* = X_1 \tag{3.14}$$

$$P_0 \bmod (P^* - 2) = X_2$$

$$P_0 \bmod (P^* - 2n) = X_n$$

Шукаємо доти, поки $X_n \le X_{n+1}$, тоді переходиться на наступний етап, на якому початкове значення лічильника $i = 0$. Даний процес продовжується доти, поки $P^* \to 0, n \to Q_i$ згідно співвідношення (3.11).

Такий ітераційний процес показує збіжність даної послідовності до розв'язку поставленої задачі факторизації і відображає симетричність залишкової функції $f(Q^*)$.

Метод дозволяє виявити та локалізувати зону розв'язку задачі факторизації для БРЧ за рахунок симетричності екстремумів залишкової функції, що, в свою чергу, забезпечить значне зменшення експоненційної складової процесу факторизації на декілька порядків.

3.3 Метод факторизації на основі використання квадратичних лишків

Основна ідея запропонованого методу ґрунтується на алгоритмі Ферма та полягає у пошуку пар натуральних чисел *A* і *B*, для яких виконується співвідношення [31]:

$$P_0 = A^2 - B^2.$$

Алгоритм Ферма описується таким виразом:

$$\Delta_n = \sqrt{n^2 - P_0}\,, \tag{3.15}$$

де $n = \lfloor \sqrt{P_0} \rfloor + k$, k=1, 2, 3, … .

Кількість ітерацій k дорівнюватиме значенню, коли параметр Δ_n буде цілим числом. Звідси можна знайти шуканий розклад на множники:

$$P_0 = (n - \Delta_n)(n + \Delta_n) \tag{3.16}$$

Недоліком цього методу є експоненційна обчислювальна складність методу Ферма ($O(\exp(n))$), тому для БРЧ факторизація обмежується функціональними можливостями апаратних засобів, оскільки кількість ітерацій k може становити 2^{300} - 2^{400} і тільки на єдиному кроці можливе однозначне рішення задачі. Причому операції необхідно виконувати над числами з розрядністю 300-500 біт.

Також запропоновано удосконалений метод факторизації для криптографічних систем захисту інформації [120], який ґрунтується на використанні символів Якобі для виявлення цілого квадрату на кожній ітерації методу. Недоліком даного методу є пошук символу Якобі з обчислювальною складністю $O(\log^2 n)$, яке відбувається на кожній ітерації методу факторизації і призводить до зменшення ефективності обчислень.

В основу запропонованого методу поставлена задача створити метод факторизації, який на основі використання СЗК дозволить уникнути виконання операцій з великою обчислювальною складністю (обчислення символу Якобі, знаходження квадратного кореня) та призведе до зменшення розрядності операндів за рахунок використання властивостей квадратичних лишків, тобто операцій бінарного порівняння з ключами факторизації, генерованими згідно запропонованого методу.

Основна ідея запропонованого методу полягає в тому, що квадрати цілих чисел можна представити у вигляді суми непарних чисел, кількість яких дорівнює даному числу [29]:

$$n^2 = \sum_{i=1}^{n} (2i-1). \qquad (3.17)$$

Тому, знайшовши Δ_n за формулою (3.15) при k=1, наступні ітерації виконуються згідно виразу $S_k = \sqrt{(\Delta_0)^2 + (2n-1)}$. Ітерації продовжуються до тих пір, поки параметр S_k не буде цілим числом.

Відобразимо залишки квадратів цілих чисел по декількох простих

модулях p_j, тобто $a_1(p_1, p_2, ..., p_m) = b_1^1, b_2^1, \ldots, b_m^1$, $a_2(p_1, p_2, ..., p_m) = b_1^2, b_2^2, \ldots, b_m^2$, $a_n(p_1, p_2, ..., p_m) = b_1^n, b_2^n, \ldots, b_m^n$, де $a_i = i^2$, $b_j^i = a_i (mod\ p_j), 1 \le i \le n,\ 1 \le j \le m$, m – кількість модулів.

Використовуючи властивість (3.16), шукані залишки отримуємо за допомогою рекурентної формули:

$$b_j^i = \left(b_j^{i-1} + z_i^j\right) mod\ p_j,$$

де $z_j^i = z_i\ mod\ p_j$, z_i=2i-1.

Відповідні результати по модулях 3, 5, 7, 11 представлені в таблиці 3.5 і показують, що кількість квадратичних лишків для кожного модуля становить (p_j+1)/2 (включаючи 0). Це випливає з рівності $n^2 mod\ p_j$=(-n^2) mod p_j=(p_j-n)^2 $mod\ p_j$.

Далі використовується властивість, коли квадрат числа по будь-якому простому модулю є квадратичним лишком по цьому ж модулю [111].

Відповідно, знайшовши залишки даного числа за простим модулем p_j, можна отримати для нього одне із значень часткового ключа факторизації y_n^j, яке дорівнює 0 або 1, причому значенню 1 відповідає випадок, коли $\Delta_n^j = \left(\Delta_n\right)^2\ mod\ p_j = \left(\Delta_{n-1}^j + z_j^i\right) mod\ p_j$ є квадратичним лишком по p_j і тоді Δ_n може бути цілим числом.

Таблиця 3.5 - Результати по модулях 3, 5, 7, 11

N	z_i	a_n	p_1=3		p_2=5		p_3=7		p_4=11	
			z_1^i	b_1^i	z_2^i	b_2^i	z_3^i	b_3^i	z_4^i	b_4^i
1	1	1	1	1	1	1	1	1	1	1
2	3	4	0	1	3	4	3	4	3	4
3	5	9	2	0	0	4	5	2	5	9
4	7	16	1	1	2	1	0	2	7	5

Продовження таблиці 3.5

N	z_i	a_n	p_1=3		p_2=5		p_3=7		p_4=11	
			z_1^i	b_1^i	z_2^i	b_2^i	z_3^i	b_3^i	z_4^i	b_4^i
5	9	25	0	1	4	0	2	4	9	3
6	11	36	2	0	1	1	4	1	0	3
7	13	49	1	1	3	4	6	0	2	5
8	15	64	0	1	0	4	1	1	4	9
9	17	81	2	0	2	1	3	4	6	4
10	19	100	1	1	4	0	5	2	8	1
11	21	121	0	1	1	1	0	2	10	0

Значення ключа факторизації для числа n визначається так: $Y_n = y_n^1 \wedge y_n^2 \wedge \ldots \wedge y_n^m$. Отримані результати представлено в таблиці 3.6.

Таблиця 3.6 - Ознаки квадратичності по модулях 3, 5, 7, 11

k	N	z_i	$(\Delta_n)^2$	p_1=3			p_2=5			p_3=7			p_4=11			Y_n
				z_1^i	Δ_n^1	y_n^1	z_2^i	Δ_n^2	y_n^2	z_3^i	Δ_n^3	y_n^3	z_4^i	Δ_n^4	y_n^4	
1	62	123	33	0	0	1	3	3	0	4	5	0	2	0	1	0
2	63	125	158	2	2	0	0	3	0	6	4	1	4	4	1	0
3	64	127	285	1	0	1	2	0	1	1	5	0	6	10	0	0
4	65	129	414	0	0	1	4	4	1	3	1	1	8	7	0	0
5	66	131	545	2	2	0	1	0	1	5	6	0	10	6	0	0
6	67	133	678	1	0	1	3	3	0	0	6	0	1	7	0	0
7	68	135	813	0	0	1	0	3	0	2	1	1	3	10	0	0
8	69	137	950	2	2	0	2	0	1	4	5	0	5	4	1	0
9	70	139	1089	1	0	1	4	4	1	6	4	1	7	0	1	1

Вектор $Y(Y_1, Y_2, \ldots, Y_n)$, в якому Y_i=0 або 1, утворює загальний ключ факторизації. В даному прикладі Y(000000001). Це означає, що на дев'ятій ітерації можливий випадок, коли Δ_n буде цілим числом, що і підтверджується обчисленням: $\sqrt{1089} = 33$.

Слід відмітити, що частковий ключ факторизації y_n^j володіє властивістю циклічності, період якої дорівнює модулю p_j, тому при розрахунках немає необхідності визначати всі поточні значення $(\Delta_n)^2$ [111]. Число $(\Delta_n)^2$, корінь квадратний з якого може бути цілим числом (Y_n=1), отримується згідно формули обчислення суми арифметичної прогресії, яку утворює послідовність z_i, за таким виразом:

$$(\Delta_n)^2 = (z_{i\ min} + k)(k-1) + (\Delta_n)^2_{min}. \qquad (3.18)$$

Крім того, можна використати іншу формулу, отриману з (3.15) для класичного алгоритму Ферма:

$$(\Delta_n)^2 = (n_{min} + k - 1)^2 - P_0, \qquad (3.19)$$

Однак обчислення згідно (3.19) повинні виконуватися над набагато більшими числами.

Метод реалізується згідно таких кроків:

1) вхід: число P_0;
2) обчислюється $\sqrt{P_0}$;
3) заокруглюється до більшого цілого $\left\lceil \sqrt{P_0} \right\rceil$;
4) шукається $\left\lceil \sqrt{P_0} \right\rceil^2 - P_0$;
5) присвоюється $i=1$;
6) шукається $pN = (2\sqrt{P_0} - 1) \bmod m$;

7) присвоюється $sN=1$;

8) обчислюється $pN=pN+2$;

9) шукається $sN= sN+ pN$;

10) обчислюється $pN=(pN \bmod m)$; $sN =(sN \bmod m);$

11) присвоюється $i =i+1$;

12) записується ознака приналежності залишку sN множині залишків цілих квадратів, якщо sN є залишком квадрату, тоді в ключі факторизації відбувається присвоєння Y_i=1, інакше Y_i=0;

13) якщо $i<m$, то перехід на крок 10;

14) присвоюється $pN = (2\sqrt{P_0} - 1) \bmod m$;

15) присвоюється $K=1$;

16) якщо $Y[K \bmod Y.\text{size}] =0$, то відбувається перехід на крок 25;

17) STEP=$2\lfloor\sqrt{P0}\rfloor + 1$;

18) присвоюється $i=0;$

19) присвоюється $sum=0;$

20) присвоюється $sum=sum+step;$

21) присвоюється $step=step+2;$

22) i++;

23) якщо $i<K$, то відбувається перехід на крок 20;

24) якщо корінь квадратний з sum є цілим числом, тоді число факторизовано, крок 27;

25) присвоюється $K= K +1$;

26) відбувається перехід на крок 16;

27) вихід

Приклад факторизації на основі властивостей залишків квадратів:

P_0=3811 , для генерації ключа обирається модуль $m=7$:

$\sqrt{3811} \approx 62 + 1 = 63$;

$63^2 = 3969$;

$3969 - 3811 = 33$;

$pN = 2 \cdot 63 - 1$;

$pN = 125$;

$sN = 33 \bmod 7 = 5$;

$pN \ \ mod \ \ m = 125 \ mod 7 = 6$;

$sN=(6+5) \ mod \ 7=4;$

Оскільки sN є лишком квадрату, то Y[1]=1;

$pN = pN + 2 \ \ mod \ \ m = (6 + 2) \ mod 7 = 1$;

sN=(sN+pN) mod 7=(1+4) mod 7=5;

Оскільки sN не є лишком квадрату, то Y[2]=0;

$pN = pN + 2 \ \ mod \ \ m = (1 + 2) \ mod 7 = 3$;

sN=(sN+pN) mod 7=(5+3) mod 7=1;

Оскільки sN є лишком квадрату, то Y[3]=1;

$pN = pN + 2 \ \ mod \ \ m = (3 + 2) \ mod 7 = 5$;

sN=(sN+pN) mod 7=(1+5) mod 7=6;

Оскільки sN не є лишком квадрату, то Y[4]=0;

$pN = pN + 2 \ \ mod \ \ m = (5 + 2) \ mod 7 = 0$;

sN=(sN+pN) mod 7=(0+6) mod 7=6;

Оскільки sN не є лишком квадрату, то Y [5]=0;

$pN = pN + 2 \ \ mod \ \ m = (0 + 2) \ mod 7 = 2$;

sN=(sN+pN) mod 7=(6+2) mod 7=1;

Оскільки sN не є лишком квадрату, то Y[6]=1;

$pN = pN + 2 \ \ mod \ \ m = (2 + 2) \ mod 7 = 4$;

sN=(sN+pN) mod 7=(1+4) mod 7=5;

Оскільки sN не є лишком квадрату, то Y[7]=0;

$pN = pN + 2 \ \ mod \ \ m = (4 + 2) \ mod 7 = 6$.

Алгоритм генерації ключа для факторизації БРЧ представлений на рисунку 3.14.

Обчислювальна складність алгоритму становить $O(n) = \log_2 n$.

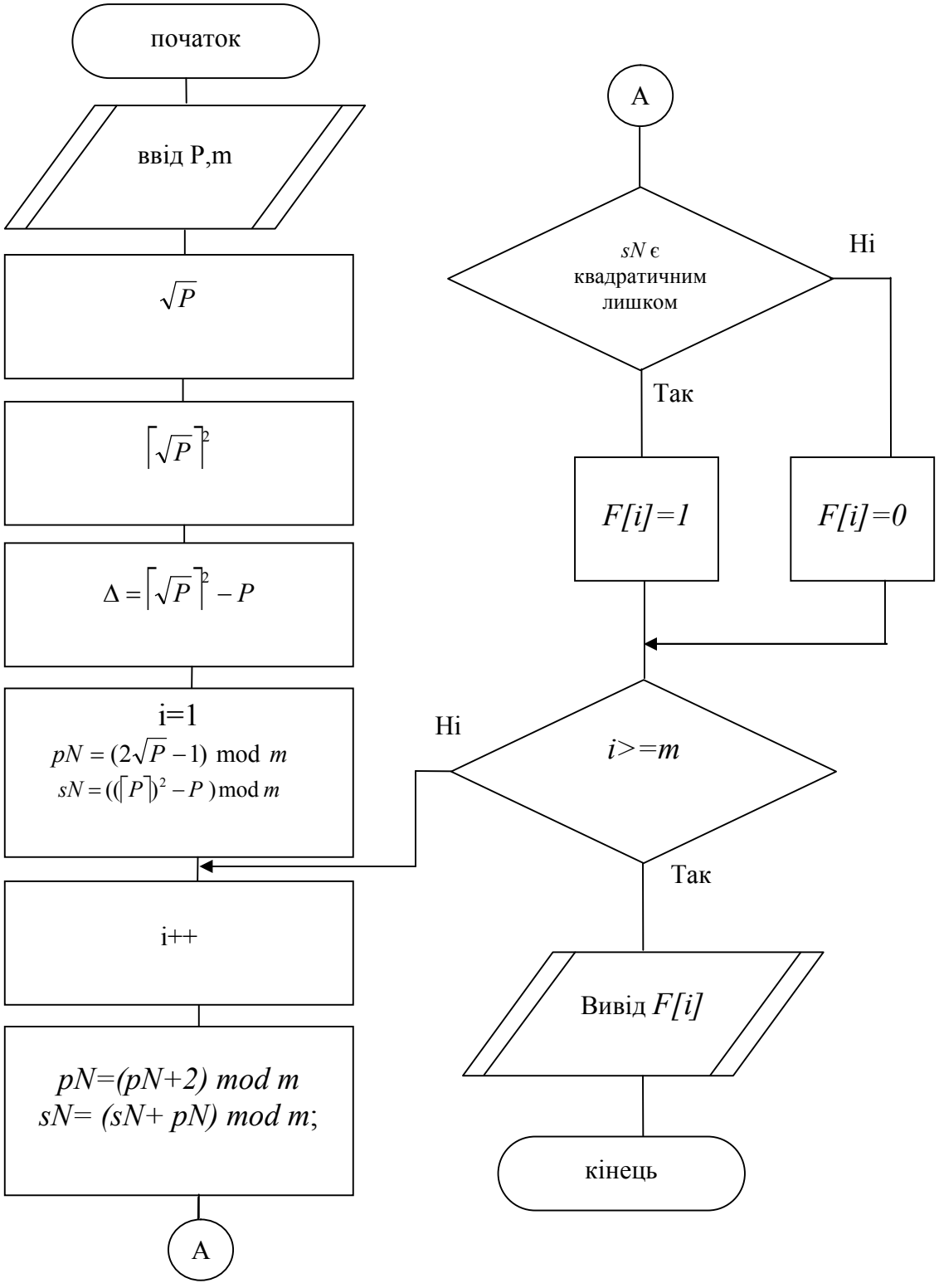


Рисунок 3.14 – Блок-схема алгоритму генерації ключа для факторизації на основі властивостей залишків квадратів

В результаті отримується вектор *Y*[]=0100101, наступні елементи якого

можна формувати, використовуючи властивість циклічності.

На початку виконання циклу встановлюється K=1.

Якщо $Y[K \bmod m]$=1, то тоді необхідно перевіряти, чи корінь квадратний із знайденого числа є цілим числом:

$Y[1 \bmod 7]$=1 , $\sqrt{1(2\cdot 62+1)+33}=\sqrt{158}=12{,}569$;

K=K+1;

$Y[2 \bmod 7]$=0;

K=K+1;

$Y[3 \bmod 7]$=1, $\sqrt{3(2\cdot 62+3)+33}=\sqrt{414}=20{,}346$;

K=K+1;

$Y[4 \bmod 7]$=0;

K=K+1;

$Y[5 \bmod 7]$=0;

K=K+1;

$Y[6 \bmod 7]$=1, $\sqrt{6(2\cdot 62+6)+33}=\sqrt{813}=28{,}513$;

K=K+1;

$Y[7 \bmod 7]$=0;

K=K+1;

$Y[8 \bmod 7]$=1, $\sqrt{8(2\cdot 62+8)+33}=\sqrt{1089}=33$.

Розроблений алгоритм факторизації буде мати вигляд, зображений на рисунку 3.15.

Запропоноване технічне рішення при достатньо великому модулю m, який складений з великої кількості простих модулів, дозволяє уникнути операції перевірки кореня квадратного, а вектор Y з ознаками однозначно вкаже на розв'язок рівняння (3.1). Ефективність використання запропонованого методу факторизації представлена на рисунку 3.16.

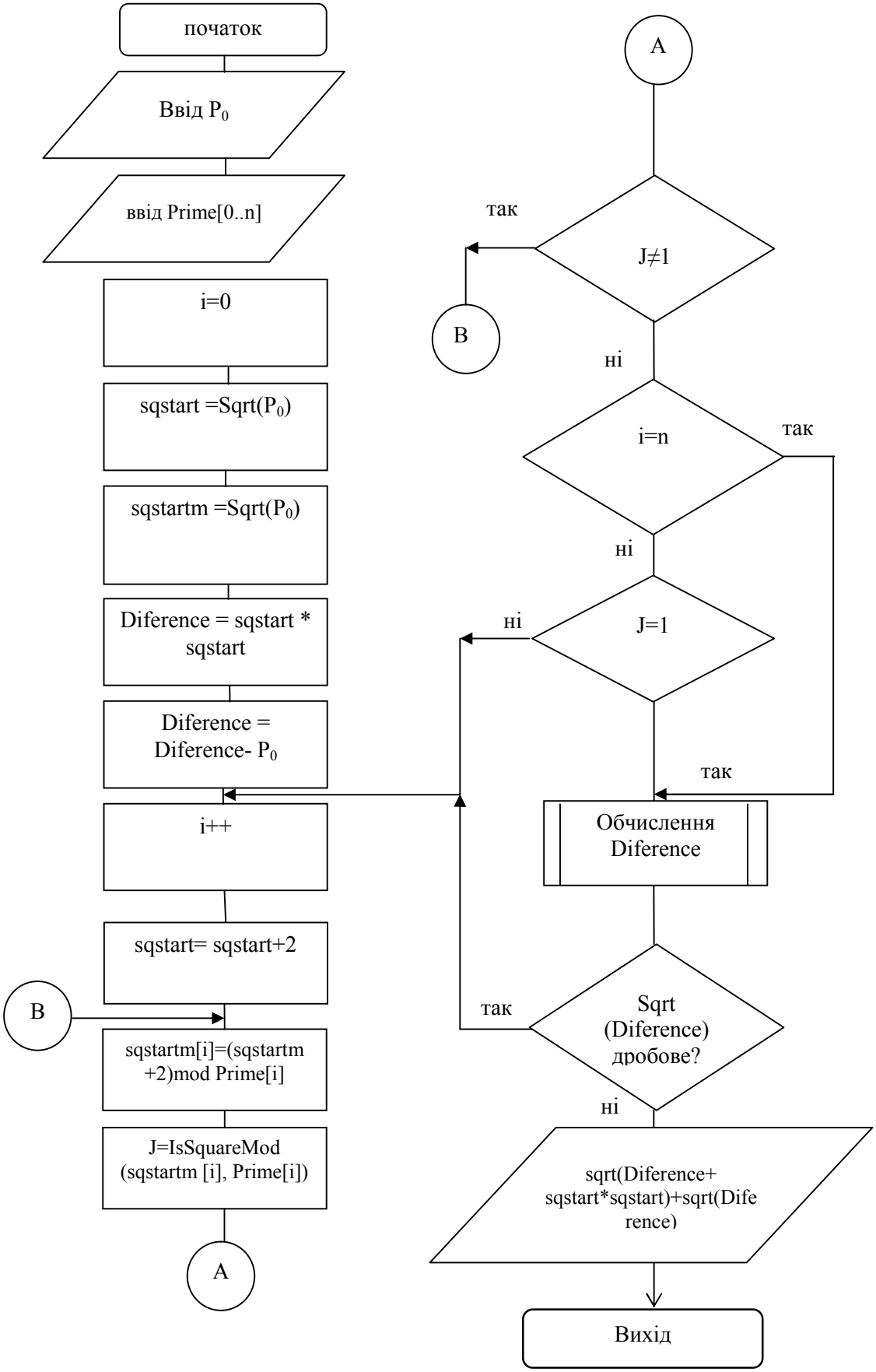


Рисунок 3.15 – Блок-схема алгоритму факторизації на основі властивостей залишків квадратів

Класичний алгоритм факторизації Ферма є подібним до методу «пробних ділень», тобто методу перебору всіх дільників. Тому складність даного методу оцінюється $O(n_0(\log n)^2)$ [111].

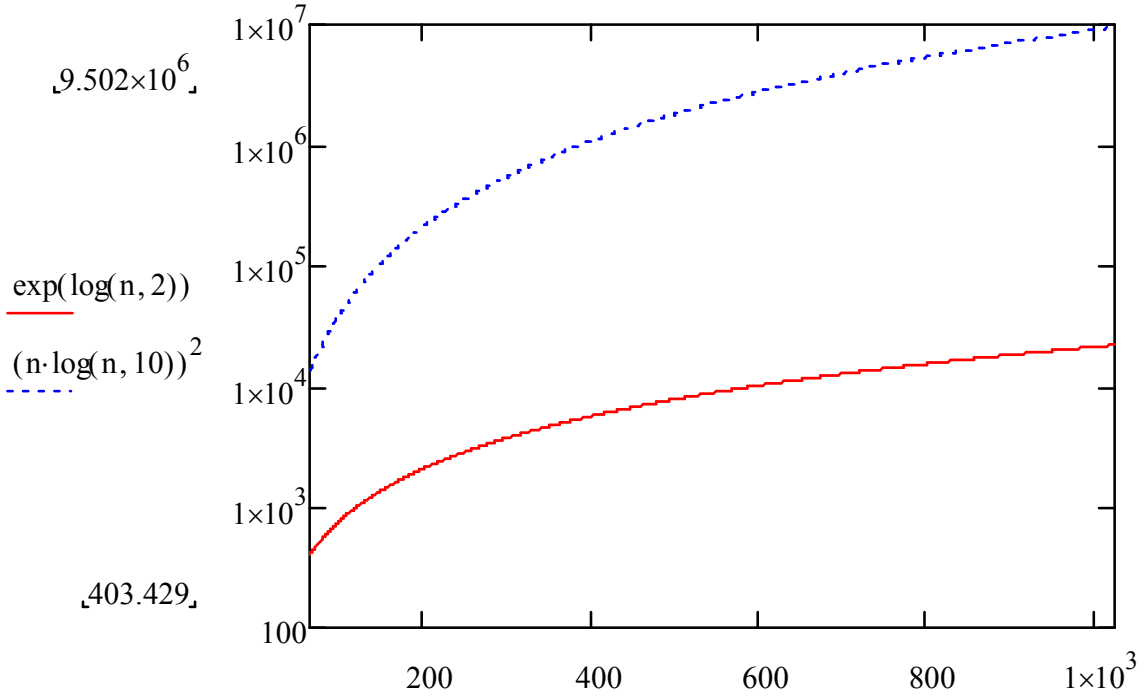


Рисунок 3.16 – Порівняльна характеристика розробленого та існуючого методів факторизації

Таким чином, метод факторизації з використанням СЗК дозволяє змінити зону розрядностей обчислювальних ресурсів на декілька порядків нижче по шкалі квадратів та замінити операцію знаходження кореня квадратного, на якій базується обчислювальна складність алгоритму Ферма, на операцію порівняння згенерованого ключа факторизації, який формується на основі ознак квадратичності.

Отже, розроблений метод факторизації дозволяє виконувати операції в СЗК над залишками невеликої розрядності.

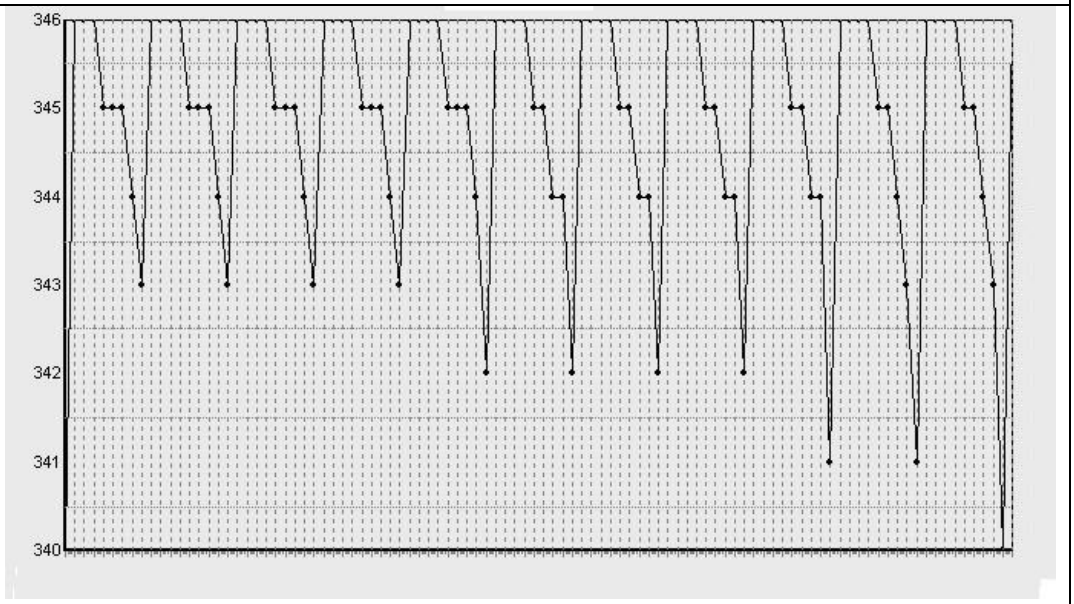
3.4 Розробка методу та моделі локалізації розрядності розв’язку задачі факторизації багаторозрядних чисел

У результаті реалізації запропонованого в 3.1 методу звуження околу рішення задачі для єдиного k отримуємо моделі – фрактали, які відображають поведінку стрибків розрядностей в залежності від значення k, що змінюється лінійно, тобто додавання 1 до молодшого розряду. Оскільки k є єдиним розв’язком задачі факторизації P_0, то дослідження саме його властивостей є актуальною задачею.

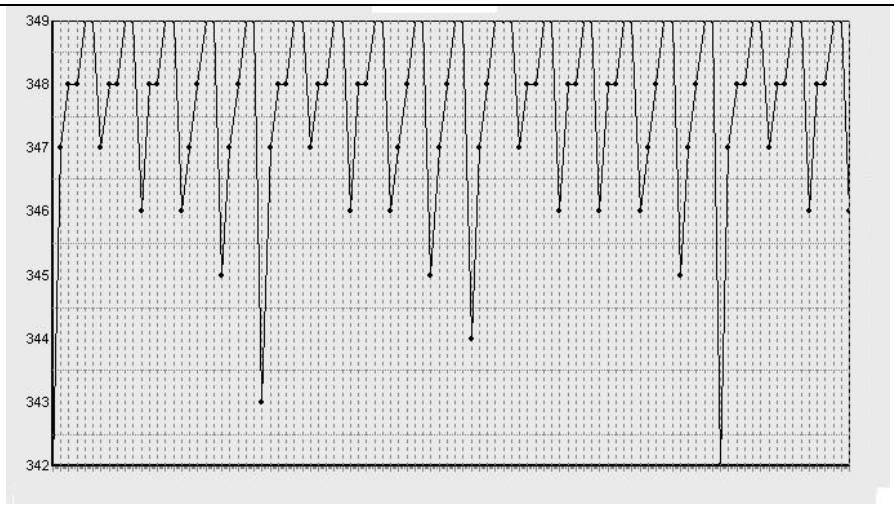
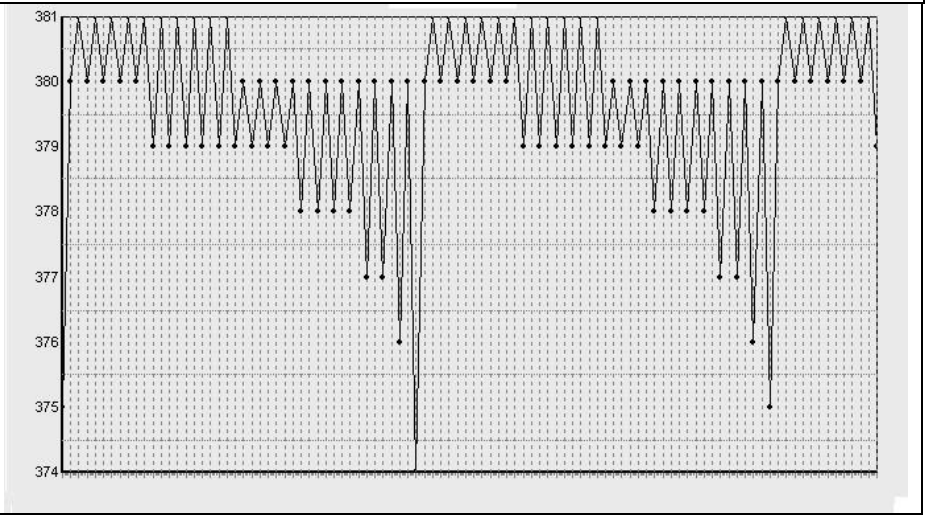
У таблиці 3.7 подані моделі фракталів в околі рішення задачі факторизації БРЧ, що дозволяють графічно ідентифікувати та відобразити двійковий логарифм різниці P_0, обчисленого відповідно до k_i, та числа, що факторизується. Експериментальні дослідження при різних комбінаціях значень k для чисел RSA 704 наведені в таблиці 3.8.

Відповідно, до кожної зміни k та P* обчислюються значення пропонованого P_0, що в разі розв’язку однозначно повинно відповідати рівнянню (3.1).

Таблиця 3.7 - Моделі фракталів в околі рішень

RSA	Біт	Моделі фракталів в околі рішень
210	696	

Продовження таблиці 3.7

RSA	Біт	Моделі фракталів в околі рішень
704	704	
768	768	

Значення пропонованого P_0 обчислюють, отримавши новий добуток P_1 та P_2 відповідно виразу (3.4).

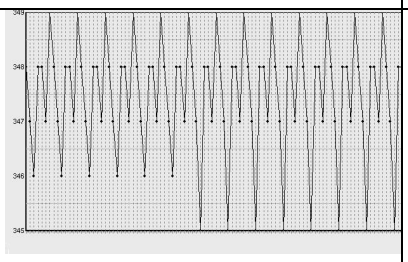
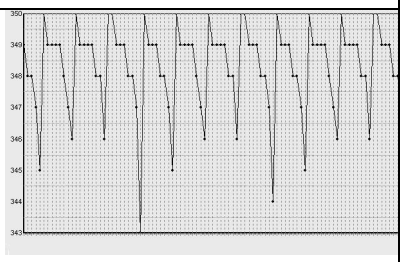
Оскільки різниця між P_0 та пропонованим P_0 (у відповідності до зміненого k) є БРЧ, то доцільно відобразити коливання розрядності різниці, що у зоні розв'язку буде відображена чіткою структурою.

При проведенні досліджень було виявлено, що моделі фракталів в околі рішення задачі факторизації набувають чіткої структури при зміні старших бітів БРПЧ, що відповідають старшим бітам істинних добутків.

Проведені експериментальні дослідження на встановлення можливого числа різних моделей - фракталів на прикладі тесту RSA 704 шляхом варіації бітів різних розрядностей для встановлення залежності структурованості

отриманої логарифмічної функції [30].

Таблиця 3.8 – Моделі фракталів при різних варіативних значеннях k для чисел RSA 704.

	$K+2^{i+0}$	$K+2^{i-1}$	$K+2^{i+1}$
RSA 704	1011101000100010111 1001011000110101101 0001110101000110011 1101111111001100001 0111001100001011111 0101100101100110100 0000100111011011010 0111101011000111000 1111101001000000101 0110011001101110101 1001110101010101111 1111111010000110101 1110011010100000100 0111101001010110101 0001111100111101101 0110010101100100001 0110010100101101100 0000001011101000111 1	K:1011101000100010111 1100101100011010110 1000111010100011001111 1011111110011000010 1100110000101111101 0110010110011010000 0010011101101101001111 0101100011100011111 0100100000101011001 1001110101100111010 1010101111111111110 1000011010111100110 1000001000111101001 0101101000111111001 1101101011001010110 0100010111001010011 0110000000010111010 0111	1011101000100010111 0010110001101011010 0111010100011001111 1111111001100001011 0011000010111110101 0010110011010000001 0111011011010011110 1 0110001110001111101 0100000010101100110 0111011010110011101 0101011111111111010 0011010111100110101 0000100011110100101 1101010001111100111 1 0110101100101011001 0001011001010010110 1 1000000001011101000 11110
Моделі фракталів			

При створенні моделі графічного відображення моделі розв’язку задачі розроблено алгоритм побудови моделей фракталів представлений на рисунку 3.17.

В додатку М наведені моделі фракталів в околі рішення задачі факторизації для БРЧ RSA, які були обрані для дослідження у зв’язку з відомими розв’язками задачі факторизації та дозволяють експериментальним

шляхом перевірити структурованість логарифмічної функції в околі рішення факторизації.

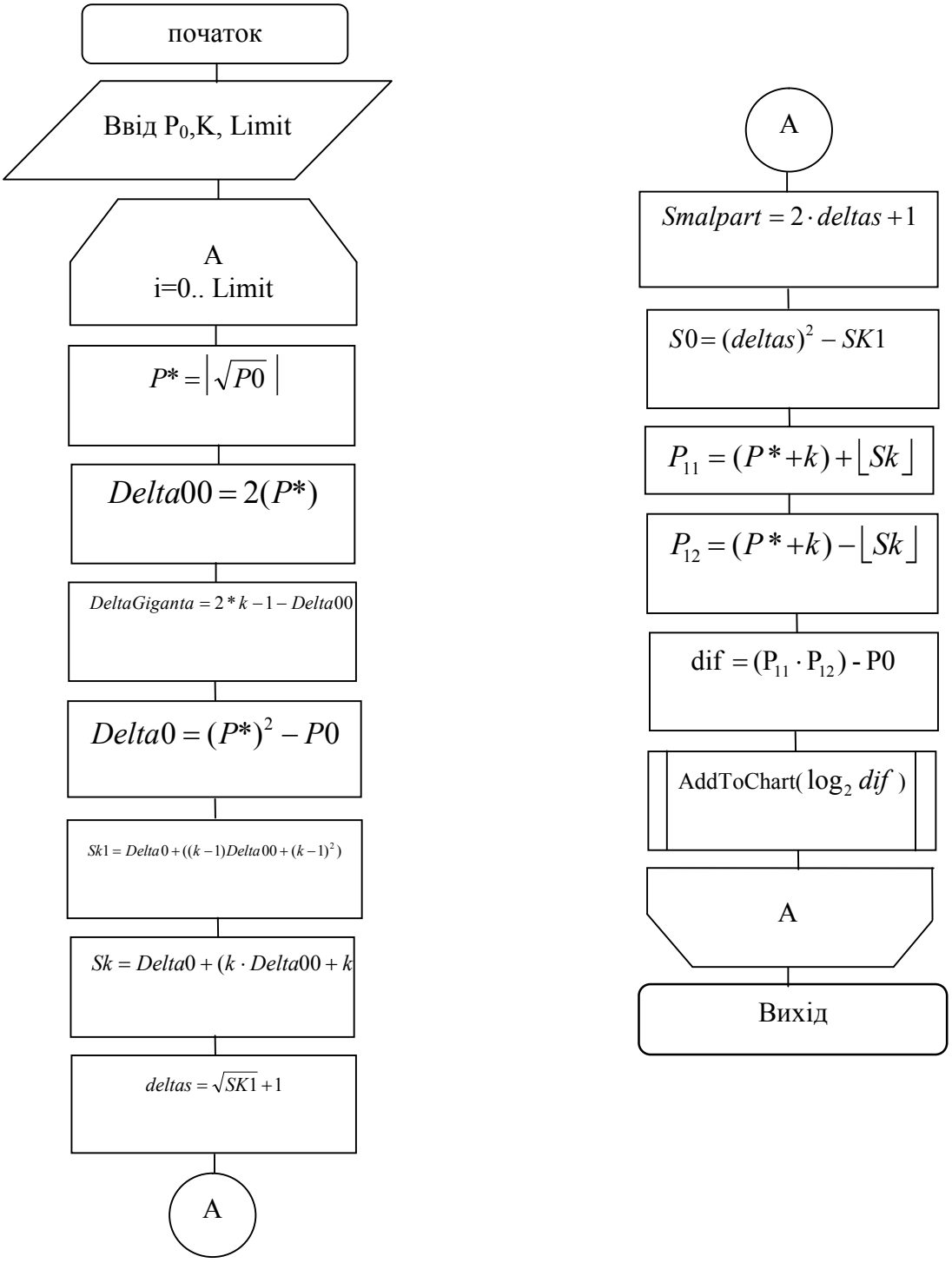
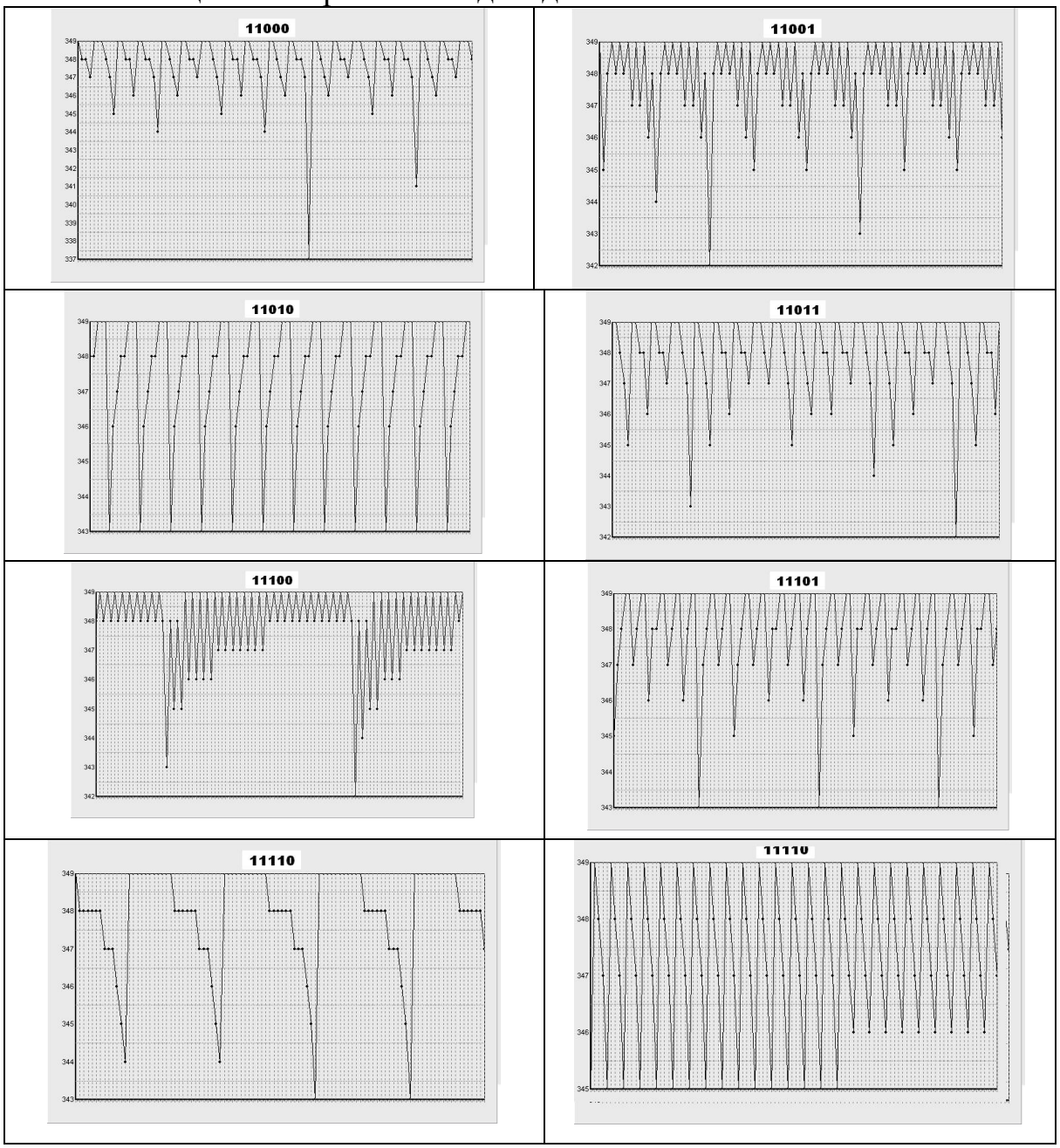


Рисунок 3.17 - Блок-схема алгоритму побудови фракталів

Проведені дослідження показують, що при аналізі 8-ми та 6-ти старших бітів значення k у діапазонах 1000000001 – 1111111111 та 10000001

– 11111111, 100001 – 111111 моделі - фрактали повторюються з деякими зміщеннями та варіаціями екстремумів (таблиця 3.9). Подібні дослідження проводились з метою виявити залежність старших розрядів підібраних множників та двійково-логарифмічної функції різниці пропонованого добутку та числа, що факторизується.

Таблиця 3.9 – Фрактали моделі для RSA 704



Таким чином, більш глибоке дослідження характеристик фракталів є перспективним для розвитку і вдосконалення методів факторизації БРЧ.

Обчислювальну складність задачі факторизації можна значно знизити, якщо врахувати, що відомі БРЧ, які використовуються при шифруванні інформації, побудовані на основі, так званих, «сильних» простих чисел, які у двійкових кодах мають обмежене число блоків нулів та одиниць. Числа P_0, які при цьому утворюються, також характеризуються цією властивістю. Дослідження з числом k показали, що вони теж можуть бути віднесені до складу сильних чисел.

Причому встановлено, що якщо P_c парне, то $\dot{\Delta}$ - непарне, тобто S_k – це квадрат непарного числа, а якщо $Pc = \widetilde{P}c + k$ - непарне, то $\dot{\Delta}$ - парне і S_k - квадрат парного числа. Ця властивість дозволяє розпочати процеси сканування різними парними та непарними k, які можна представити у вигляді графа (рисунок 3.18).

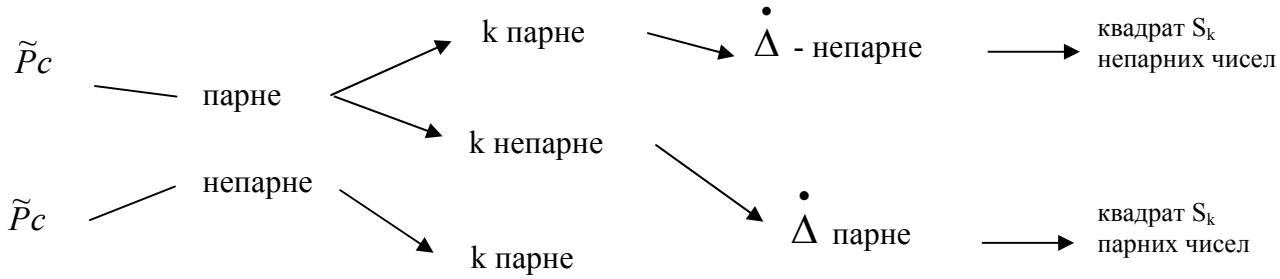


Рисунок 3.18 - Граф – модель визначення парності сканування числа *k*

Проведені дослідження показують, що зображення моделей фракталів в околі рішення задачі факторизації мають чітку структуру та дозволяють з допомогою маніпуляцій старших розрядів множників виявити деяку послідовність старших розрядів розв’язку. Встановлені верхні розряди розв’язків задачі факторизації призводять до зменшення необхідної кількості ітерацій для Ферма-подібних алгоритмів факторизації на декілька порядків.

ВИСНОВКИ ДО РОЗДІЛУ 3

1. Розроблено метод факторизації, що базується на зменшенні розрядності операндів та дозволяє виявити розрядності множників, що дозволило визначити двійкову розрядність числа кроків процесу факторизації згідно удосконаленого алгоритму Ферма.

2. Розроблений метод локалізації зони розв’язку задачі факторизації для БРЧ за рахунок симетричності екстремумів залишкової функції, що, в свою чергу, забезпечило зменшення експоненційної складової процесу факторизації на декілька порядків.

3. Розроблено удосконалений метод факторизації на основі використання квадратичних лишків, що характеризується нижчою обчислювальною складністю в порівнянні з існуючими.

4. Розроблено методи для графічної ідентифікації моделей фракталів в околі рішення розв’язку задач факторизації та моделей матричного представлення добутків БРПЧ у базисі Радемахера, що дозволяють графічно представити характеристики функцій двійкового логарифму БРЧ при рішенні задач теорії чисел.

РОЗДІЛ 4
ПРОГРАМНИХ ТА АПАРАТНИХ ЗАСОБІВ ОПРАЦЮВАННЯ БАГАТОРОЗРЯДНИХ ЧИСЕЛ У БАЗИСАХ РАДЕМАХЕРА ТА ХААРА – КРЕСТЕНСОНА

4.1 Проектування програмних рішень опрацювання багаторозрядних чисел

Для реалізації поставленого завдання було обрано мову програмування С++, середовище програмування С++ Buider 6.0.

Перевагами мови програмування С++ є гнучкість використання, інкапсуляція, поліморфізм, структурованість. Великою перевагою є тонкість програмного коду, що дозволяє зменшити затрати процесорного часу. Велику роль при обранні мови програмування відіграла наявність зовнішніх модулів, що забезпечують виконання операцій на великими числами.

Після запуску додатку користувачеві необхідно обрати і ввести в наступному числовому полі кількість простих модулів, які будуть брати участь в обчисленнях. Для БРЧ це може значно скоротити обчислення, а для малого може мати надлишковість, якщо їх добуток буде більшим, ніж саме число для факторизації. Додаток реалізує метод, описаний в [111].

Для факторизації числа його необхідно ввести в текстове поле та натиснути кнопку “Удосконалений алгоритм”, яка знаходиться у першому полі. При обчисленні відображається службова інформація, що використовується при пошуку наступного цілого квадрата:

- P0 3811 – саме число для факторизації;
- sqrtP0 62 – заокруглений корінь;
- sqrtP0diference 33 - різниця між наступним цілим квадратом та числом для факторизації;
- endstep 125 – службова змінна, що відображає, на якому кроці закінчуються обчислення, адже квадрат числа складається з суми непарних чисел;
- firststep 125 - крок, з якого починають обчислення;

- 00000000100000000000000000000000000001000000000000

00000000000000000000000000000001000000000000000000000 – ключ, за допомогою якого уникають надлишковості перевірок на цілісність кореня, він застосовується циклічно і його розмір залежить від кількості та розміру простих модулів, що обрані для обчислень;

- 103 – розв'язок задачі факторизації.

Після запуску програмного продукту для факторизації багаторозрядних чисел на екрані буде зображена форма, представлена на рисунку 4.1.

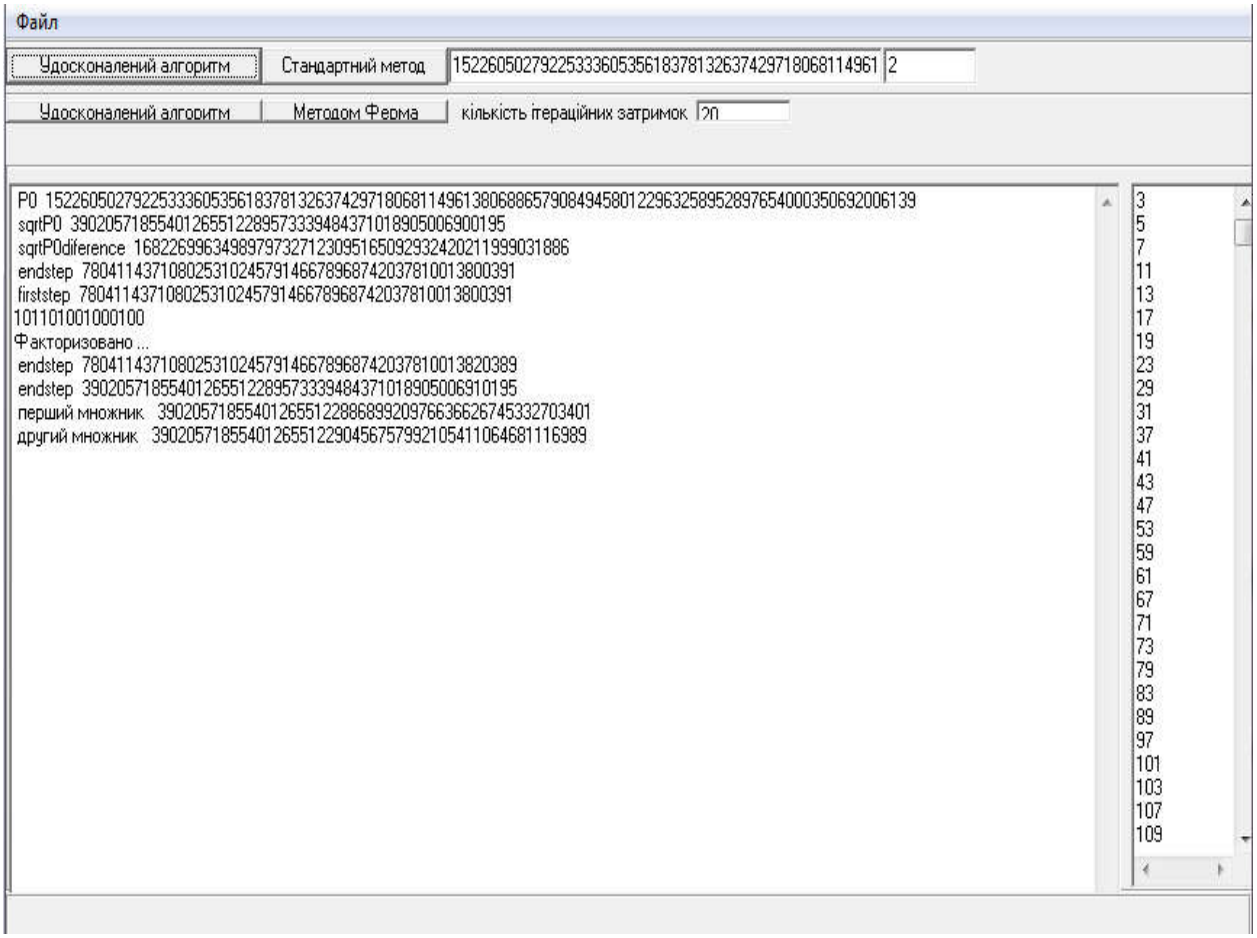


Рисунок 4.1 – Головне вікно додатку факторизації БРЧ

Додаток забезпечує виконання факторизації з допомогою реалізованого стандартного методу Ферма та удосконаленого алгоритму факторизації. Код програмного продукту наведений в додатку К.

Для порівняння методів факторизації реалізовано ітераційну затримку для виявлення переваг удосконаленого алгоритму факторизації. Додаток

забезпечує можливість внесення простих модулів для генерації ключа факторизації, що виключає виконання операції визначення квадратного кореня.

Також існує можливість вибрати лише декілька перших модулів, в прикладі наведено використання лише перших двох простих модулів.

Додаток інформує про перший крок до цілого квадрату та завершальний, з якого обчислюється значення чисел, що факторизуються. В інформаційному полі наводиться інформація про генерований ключ факторизації.

Розроблений також спеціальний програмний продукт, головне вікно якого представлене на рисунку 4.2, який дозволяє генерувати ключі факторизації для удосконаленого алгоритму факторизації на основі використання властивостей квадратичних лишків, досліджених в [29].

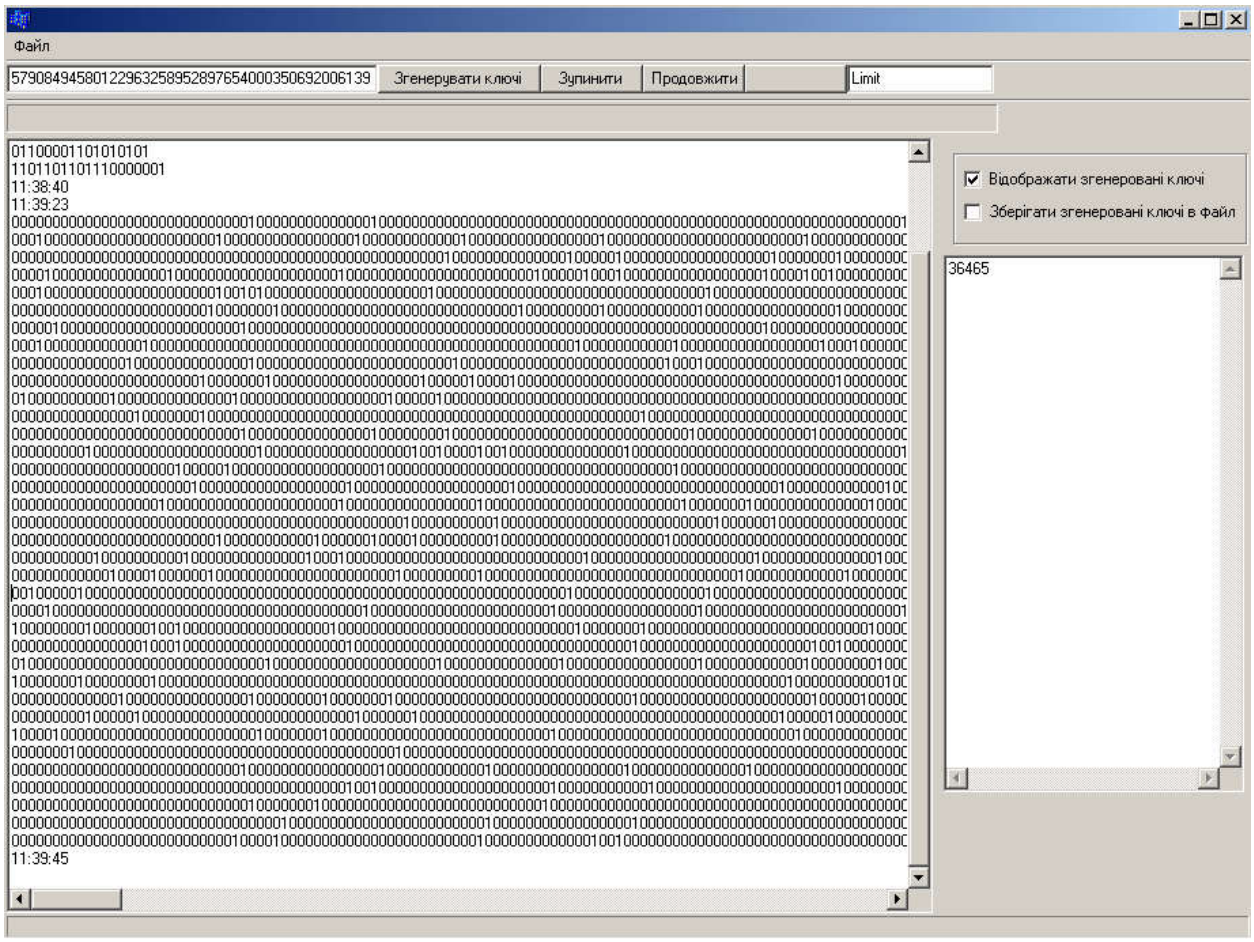


Рисунок 4.2 –Генерування ключа факторизації згідно модуля для БРЧ

Для кожного з модулів, введених у відповідному полі, генерується ключ факторизації та зберігається за необхідності у файл. У наведеному прикладі, що на рисунку 4.2, згенеровано ключ розміром 36465 біт: 1680 – одиниць та 34785 нулів. Результати проведених тестувань наведені в таблиці 4.1.

Таблиця 4.1 - Порівняння часових характеристик розробленого та існуючих методів

Розрядність	Факторизація методом Ферма, хвилин	Факторизація розробленими методом	
		Генерація ключа, хвилин	Факторизація, хвилин
32	62	3	50
64	9504	3	8420
128	-	3	101509

Аналіз отриманих результатів експериментального дослідження доводить ефективність розробленого методу факторизації [111] та достовірність отриманих аналітичних результатів. Тестування проводились для 100 випадково генерованих чисел різними алгоритмами.

Кожна одиниця показує ітерацію алгоритму факторизації, яка може задовільнити рівняння Ферма для заданого модуля, тобто вимагає перевірки на квадратичну цілісність, а кожен 0 - ітерацію, яка не потребує перевірки і є апріорі невірною. Набір декількох великих модулів зможе забезпечити уникнення необхідності пошуку квадратного кореня.

Для моделювання та дослідження розробленого методу пошуку залишку БРЧ [33] розроблений додаток, що дозволяє дослідити часові характеристики виконання стандартного методу та розробленого. Головне вікно додатку зображене на рисунку 4.3.

В додатку реалізована ітераційна затримка для проведення

експериментів при дослідженні часових характеристик, оскільки процесорний час між потоками розподіляється нерівномірно і він є необхідним для проведення цілого ряду експериментів.

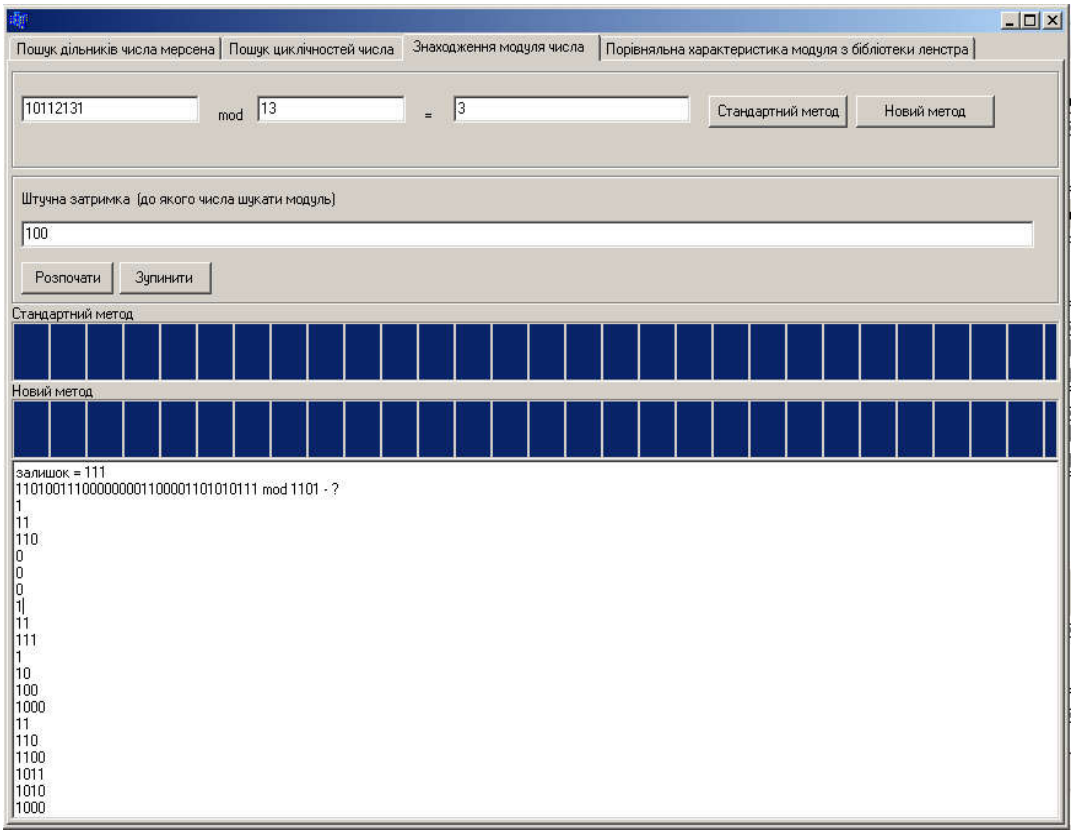


Рисунок 4.3 – Знаходження залишку багаторозрядного числа

Також в додатку відображаються контрольні дані для експериментальної перевірки кожного з етапів алгоритму. Реалізовано виконання знаходження залишку розробленим в 2.1. методом пошуку залишку.

Для виконання базових операцій з багаторозрядними числами розроблено додаток, представлений на рисунку 4.4, який забезпечує виконання основних операцій:

– додавання;
– віднімання;
– ділення;
– множення ;
– знаходження залишку;

– корінь квадратний;

– знаходження символу Якобі ;

– переведення з двійкової системи в десяткову і навпаки.

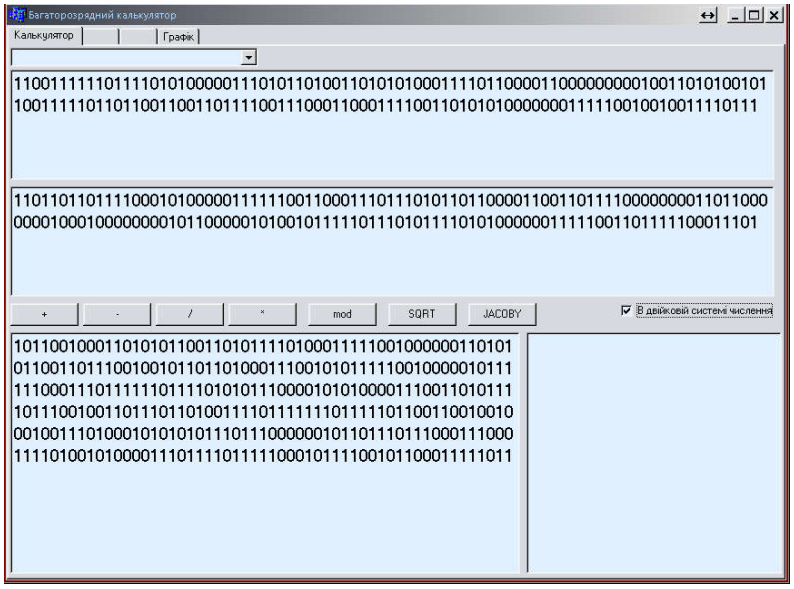


Рисунок 4.4 – Обчислювальні засоби для реалізації елементарних операцій над БРЧ

Наступні дві вкладки використовуються для ведення службової інформації. Вкладка «Графік» дозволяє відобразити матричне представлення добутку. Приклад графічного представлення багаторозрядного добутку розрядністю 330 біт, зображено на рисунку 4.5.

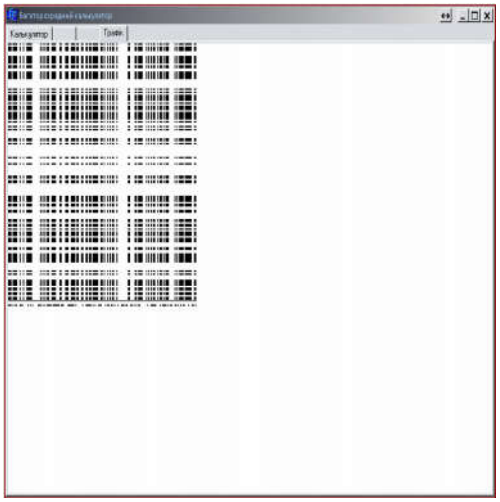


Рисунок 4.5 – Матричне представлення добутку БРЧ

Для дослідження методу перевірки *n* - розрядного числа на простоту розроблено додаток, представлений на рисунку 4.6. Додаток дозволяє експериментально дослідити метод перевірки *n* - розрядного числа на простоту в порівнянні з решетом числового поля та експериментально звірити результати. Для цього генерується послідовність простих чисел згідно алгоритму решета числового поля та тестується розробленим методом перевірки на простоту. Код програмного продукту представлений в додатку И.

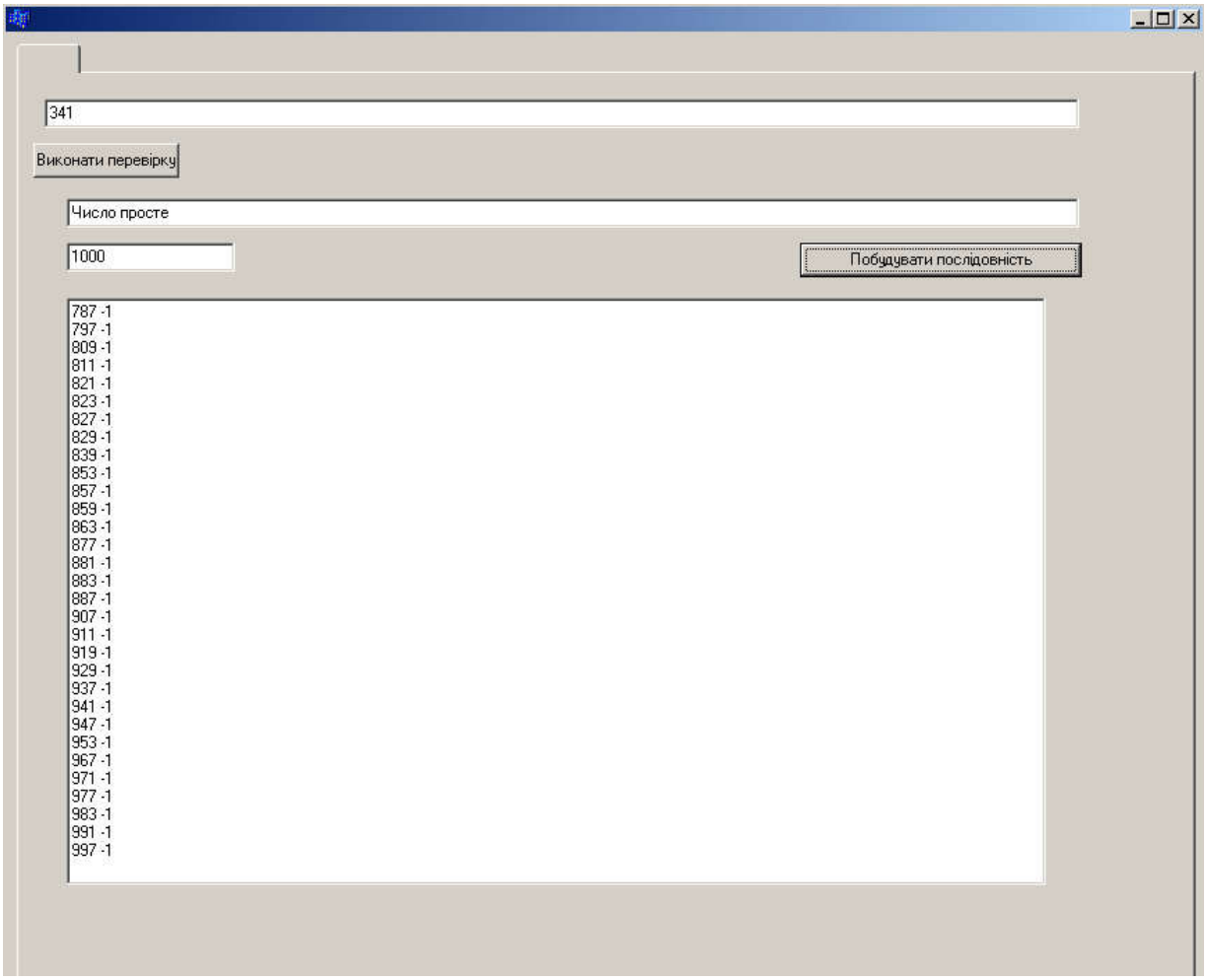


Рисунок 4.6 – Перевірка n- розрядних чисел на простоту

Для дослідження розробленого алгоритму факторизації БРЧ розроблено додаток, представлений на рисунку 4.7. Аналітичні результати, отримані в результаті проведених експериментів, наведені в [30].

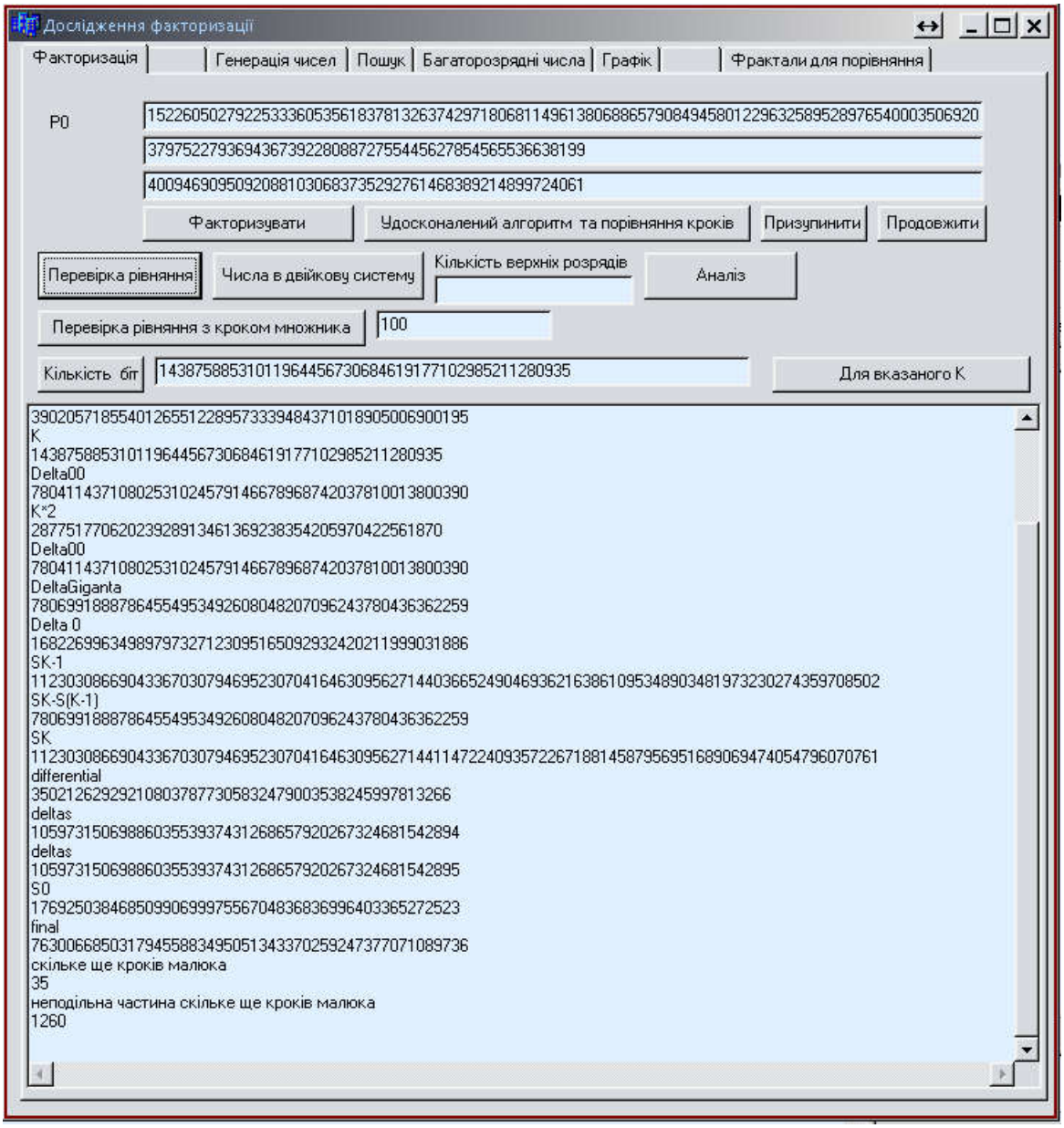


Рисунок 4.7 – Дослідження процесу факторизації

Додаток дозволяє проводити дослідження БРЧ, що описані в підрозділах 3.1, 3.2, та являє собою інструментальний засіб з відповідними можливостями: перевірка рівняння, що задовільняє теорему Ферма; відображення багаторозрядних кодів у двійковій та десятковій системах; дослідження порозрядної зміни змінних; графічне дослідження розв'язку задачі факторизації; кількості кроків між квадратами, фракталів в околі рішення задачі факторизації, підбір множників для заданого добутку та кількості ітерацій –К.

Форма містить декілька вкладок, що розділяють функції згідно призначення та етапів дослідження процесу факторизації.

Основні з них наведені на рисунках 4.7 та 4.8. На формі, представленій на рисунку 4.9, відображено дані та їх контрольні значення при виконанні процесу факторизації.

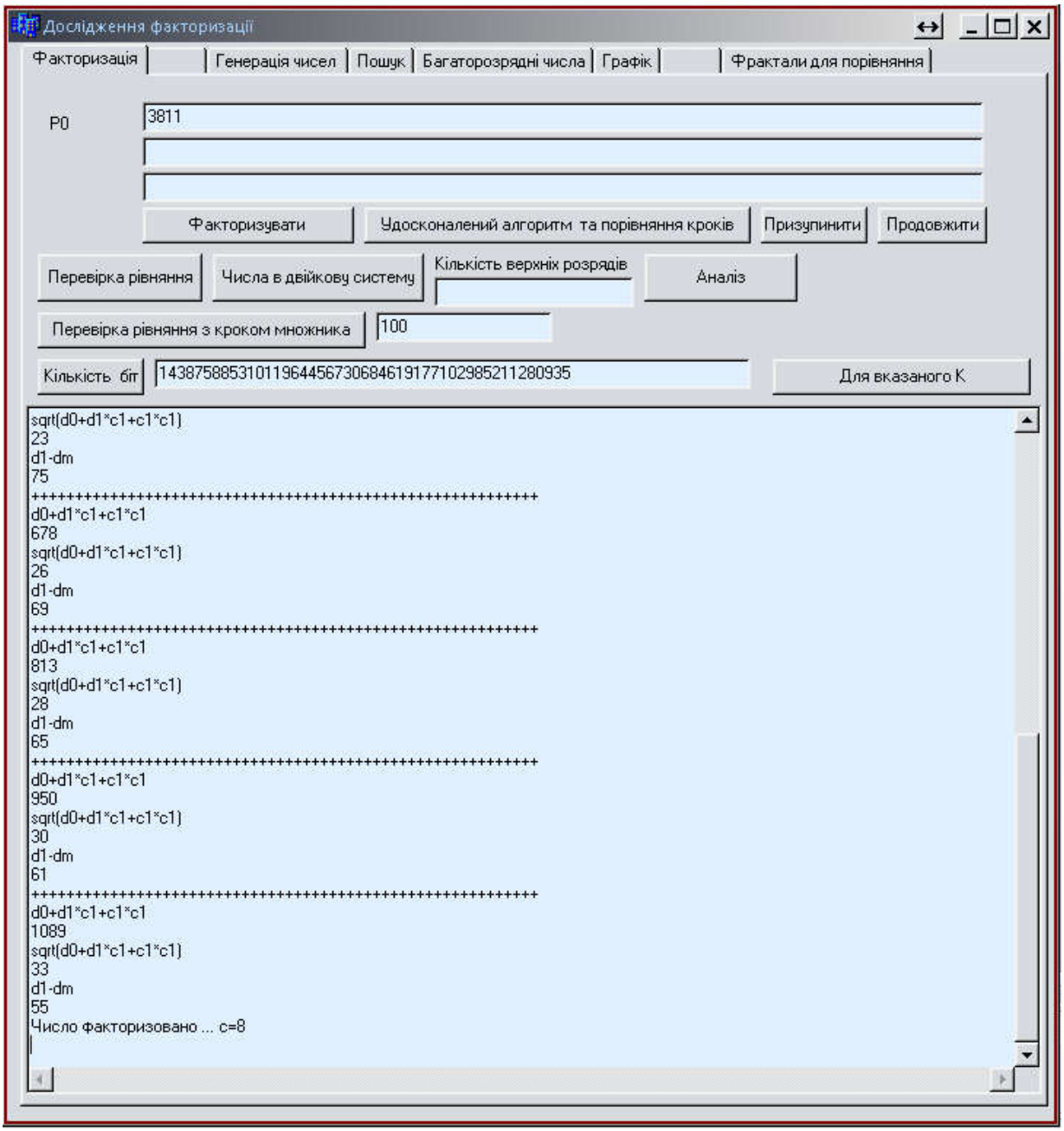


Рисунок 4.8 – Дослідження процесу факторизації. Перевірка змінних та кроку

На рисунку 4.9 зображено графік, побудований в результаті виконання процесу факторизації удосконаленим методом. На графіку показано дві послідовності, що відображають значення, які перевіряються на існування цілого квадрату та кроку факторизації.

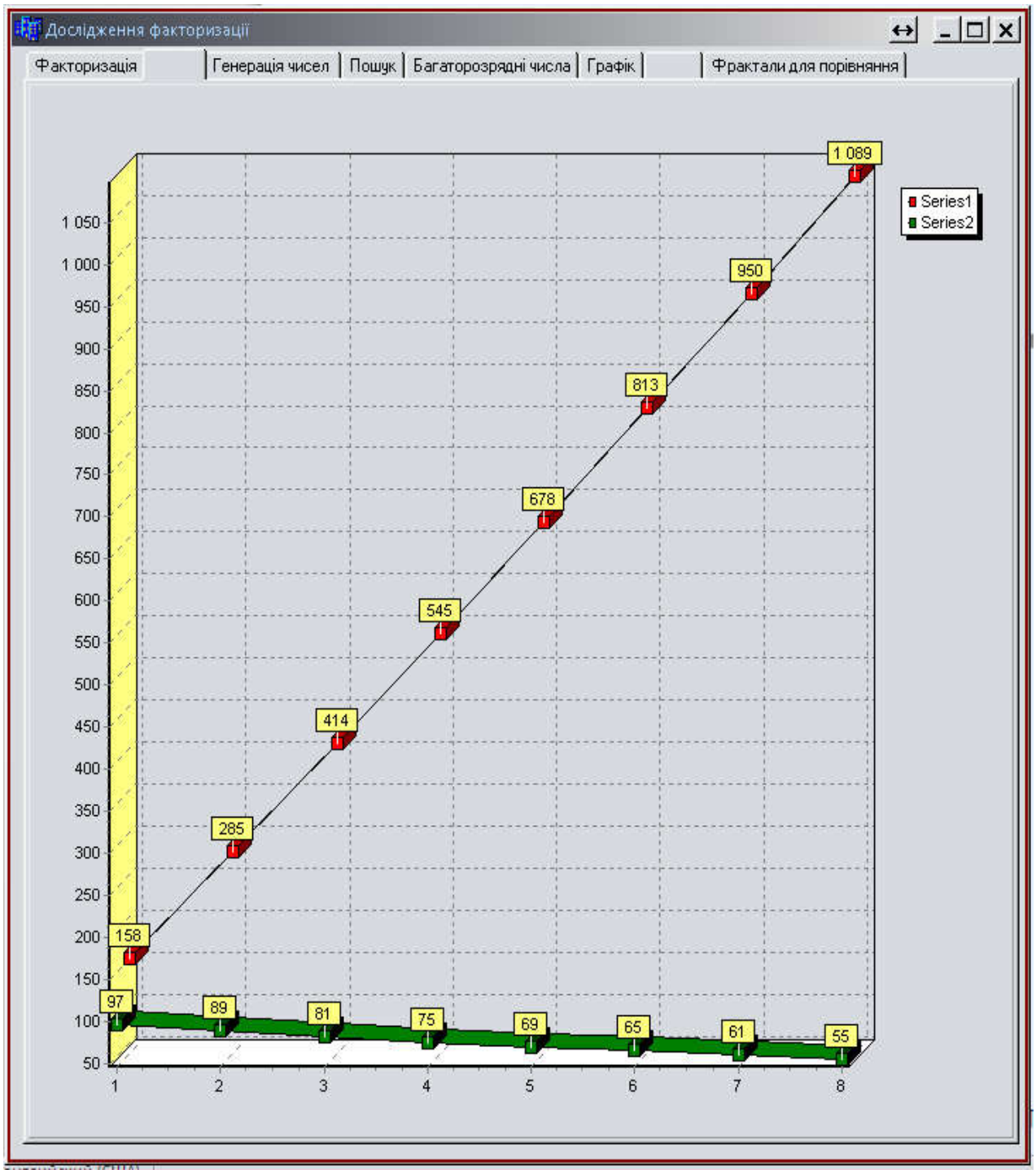


Рисунок 4.9 – Графічне представлення шкали квадратів та кроків факторизації удосконаленим методом

Наступна вкладка ілюструє процес генерування БРЧ (рисунок 4.10) з врахуванням вказаних параметрів, генерування другого множника у відповідності до першого та заданої кількості ітерацій, які будуть задовільняти рівняння Ферма. Форма забезпечує можливість встановлення

діапазону коливання псевдовипадкових чисел та розрядність генерованих БРЧ.

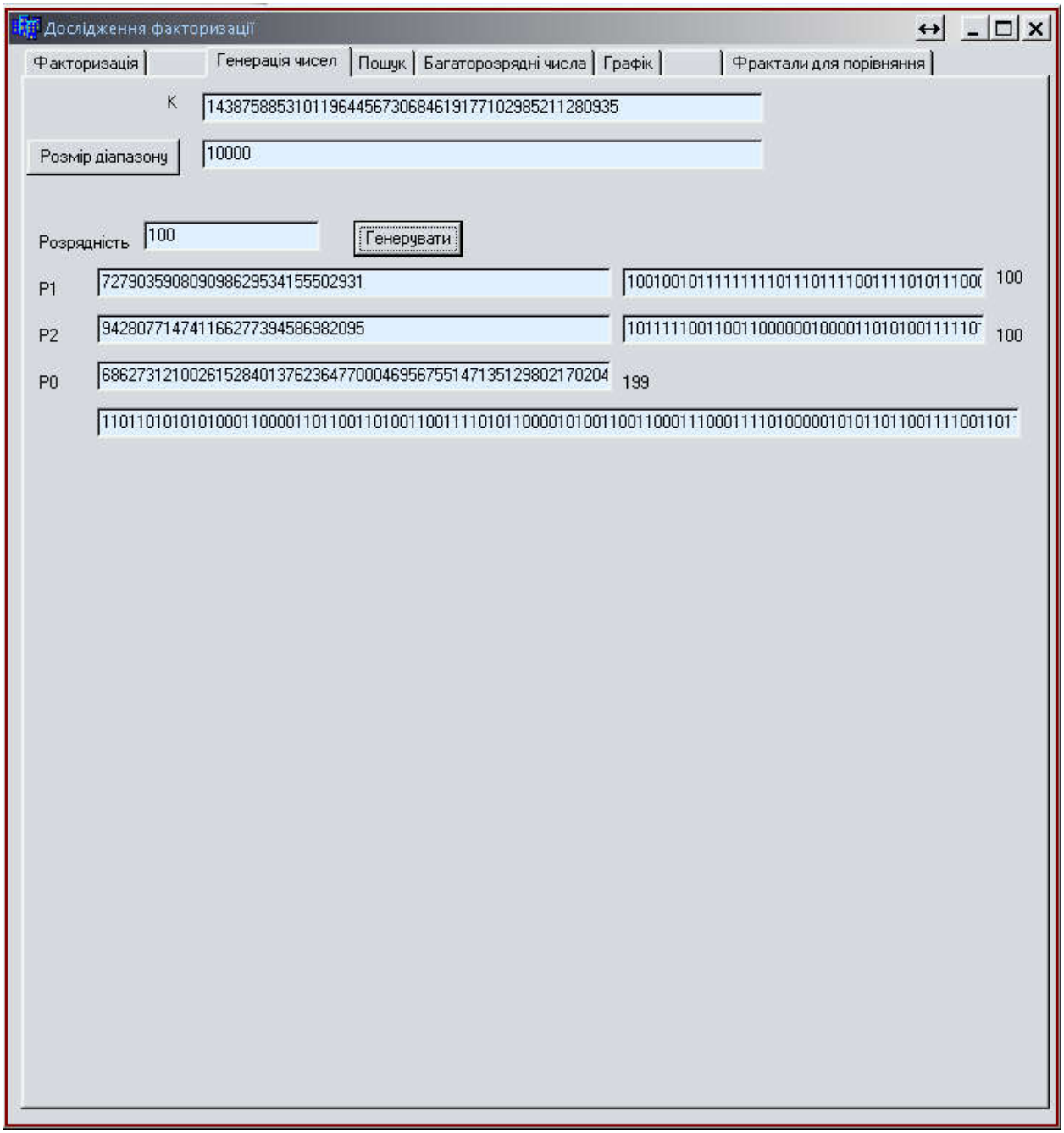


Рисунок 4.10 – Генерування випадковим чином БРЧ заданої розрядності

Дослідження зміни розрядностей змінної STEP, зміни залишкової функції, порозрядного підбору K здійснюється з допомогою вкладки, представленої на рисунку 4.11. Додаток дозволяє не лише порозрядну зміну кількості ітерацій, а й прямий перебір з вказаним кроком.

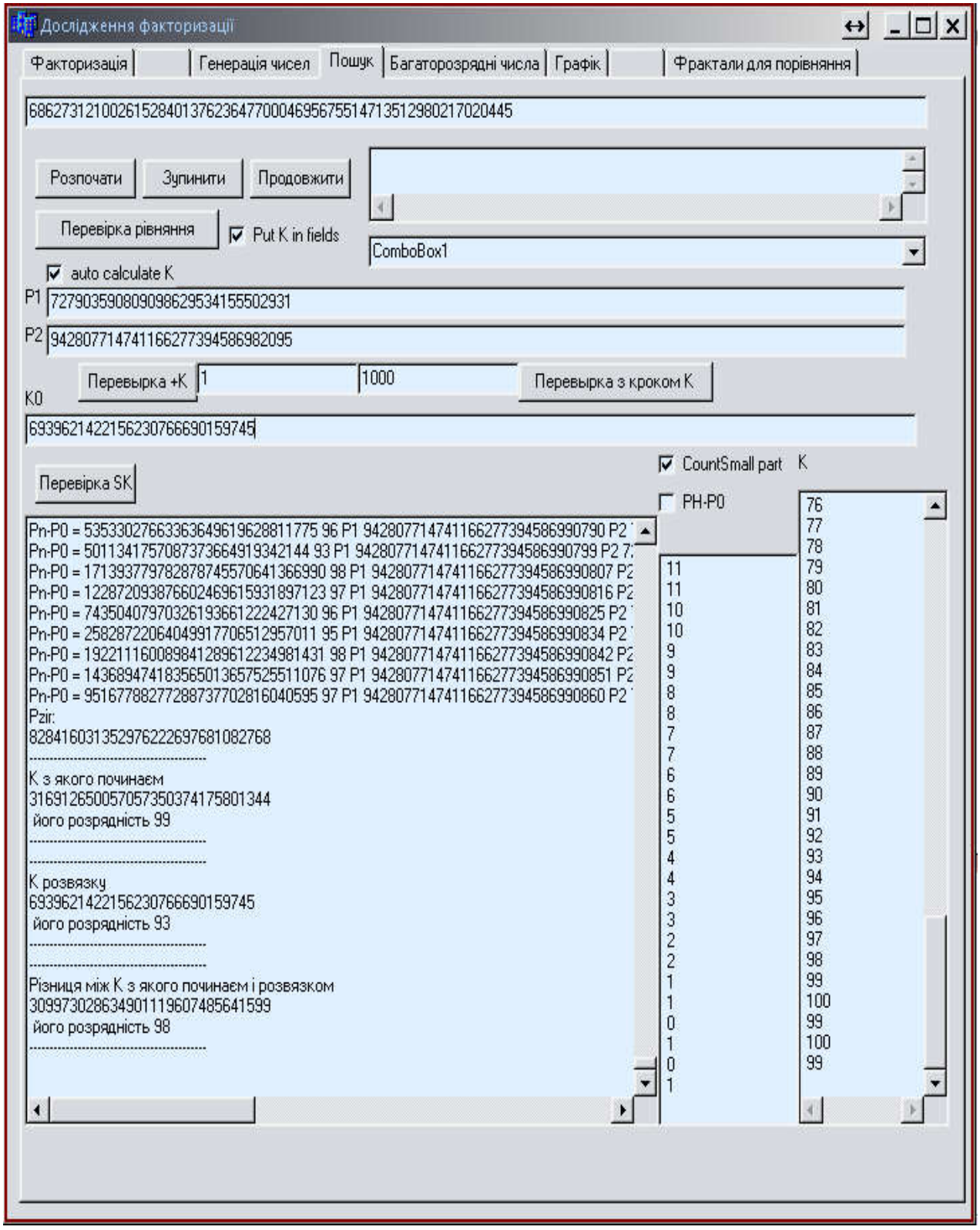


Рисунок 4.11 – Дослідження змінних в процесі факторизації, згідно удосконаленого алгоритму факторизації

Додаток дозволяє дослідити розрядності добутку при пропонованому К та добутку, введеного користувачем, провести як порозрядні, так і лінійні дослідження розрядності кількості ітерацій, як показано на рисунку 4.12.

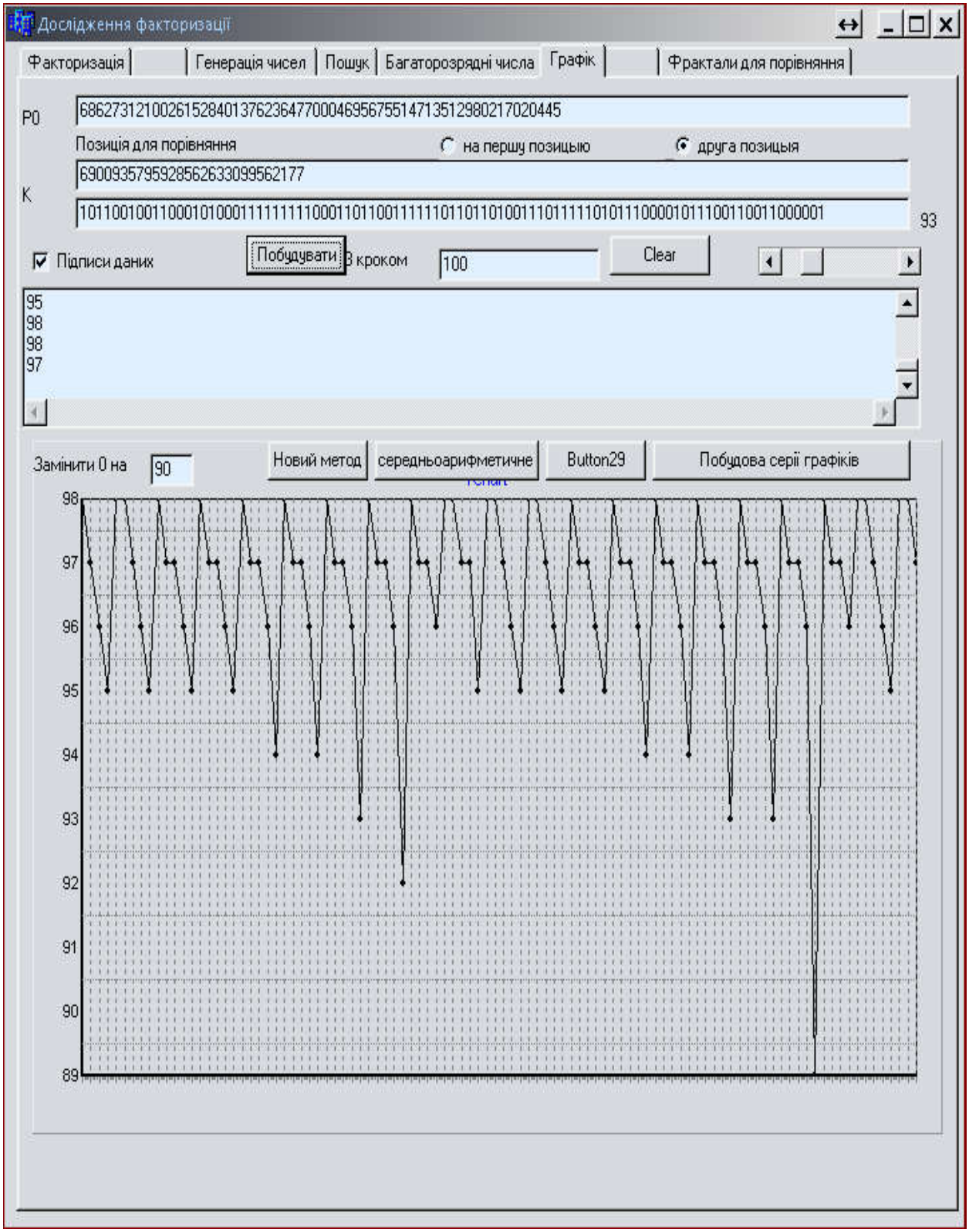


Рисунок 4.12 – Побудова графічного представлення задачі факторизації

Графіки кількості розрядів різниць пропонованого та введеного добутку наведені на рисунку 4.13. Вони дозволяють представити дані для порівняння фракталів в околі рішення задачі факторизації.

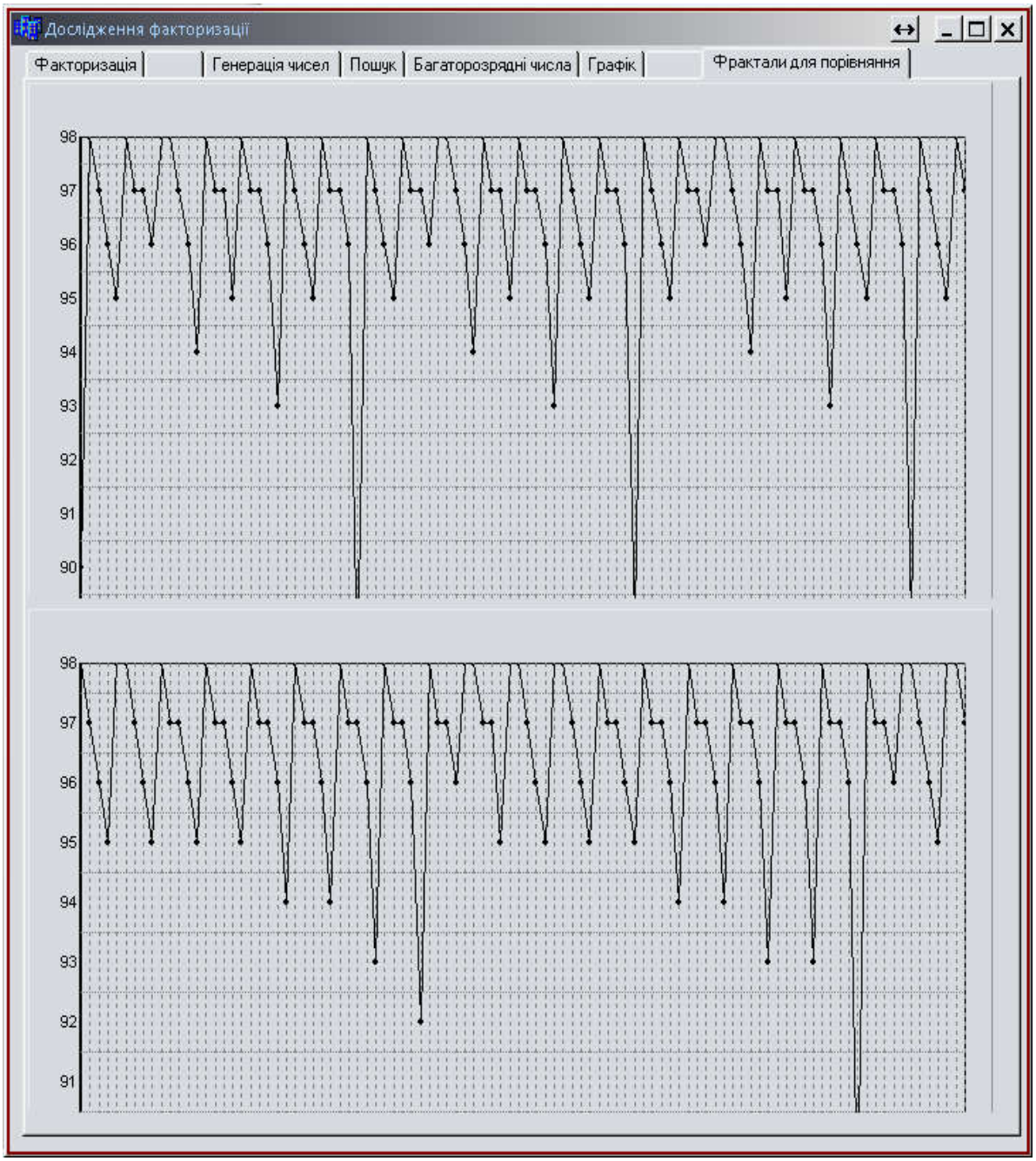


Рисунок 4.13 – Порівняння моделей фракталів задачі факторизації

При проектуванні додатку були використані компоненти наступних типів:

- TpageControl (Набір сторінок) являє собою складені одна на іншу сторінки з доступом до кожної з них та містить свій набір елементів управління, який здійснюється через вкладки з певними назвами, які можна вписати. Даний елемент управління зручний тим, що дозволяє ефективно використовувати обмежений простір екрана, створюючи ефект книги, яку можна розкрити на будь-якій сторінці;

- TtabSheet представляє собою окрему закладку об'єкта TPageControl. В C ++ Builder компонент TPageControl зберігає індексований масив закладок в свою властивість Pages;

- Tbutton – кнопка;

- поля Memo можна лише частково розглядати як класичний список, що складається з незалежних рядків. З одного боку, властивість Text являє вміст цього об'єкта як один довгий символьний рядок та містить керуючі символи ("перехід рядка", "Повернення каретки", горизонтальна табуляція). Ручне або автоматичне включення таких символів в текст дозволяє відображати довгий рядок у вигляді багаторядкового документа. З іншого боку, об'єкт Memo володіє властивістю Lines, що забезпечує доступ до кожного рядка по її індексу;

- TstatusBar – компонент, зручний для відображення різної службової та налагоджувальної інформації, яка дозволяє вести візуальний контроль за діями додатка. З допомогою StatusBar можна поділити вікно на будь-яку кількість окремих міні-статусних вікон.

Порівняння швидкодії виконання операцій множення, знаходження залишку БРЧ засобами, реалізованими бібліотекою Ленстра та на основі розроблених методів, наведено в таблиці 4.2.

Таблиця 4.2 – Порівняння отриманих експериментальних часових характеристик

Розрядність	Стандартні засоби		Розроблені методи	
	Множення, мс.	Знаходження залишку, мс.	Множення, мс.	Знаходження залишку, мс.
32	0,12	0,4	0,2	0,4
64	0,24	0,26	0,25	0,24
128	0,78	0,5	0,6	0,4
256	1,32	4,1	1,1	2,3
512	2,1	1,3	1,9	1,1

Експериментальні часові результати були отримані в результаті знаходження середньоарифметичного 100 тестувань випадковим чином згенерованих операндів.

Для роботи із змінними великого розміру була використана спеціалізована бібліотека А. Ленстра, яка складається з Lip.h та Lip.c.

Сама бібліотека є оптимізованою для опрацювання БРЧ в системах опрацювання інформаційних потоків та містить в собі наступні функції, що були використані в проектованому додатку: zsadd, zadd, zsub, zsmul, zsq, zsqin, zsmod, zmod, zlshift, zrshift, zbit, zgetbits, zsetbit, ziszero, z2log.

Для отримання розкладу числа на дільники була створена функція, код якої представлений в додатку Ж.

4.2 Реалізація апаратних компонентів виконання операцій піднесення до квадрату за модулем в теоретико-числовому базисі Хаара-Крестенсона

Квадратор відноситься до засобів обчислювальної техніки і може бути використаний при розробці дискретних пристроїв для задач статистичного аналізу та розробці високопродуктивних компонентів проблемно-орієнтованих спецпроцесорів у різних ТЧБ. Розробка високопродуктивного квадратора обумовлена його використанням, як компонента пристрою факторизації, описаного в підрозділі 4.5.

Відомий пристрій – квадратор [2], який містить вхідну шину, елемент затримки, лічильник-регістр, виходи якого порозрядно, через логічні ключі, підключені до накопичувача.

Недоліком такого пристрою є низька швидкодія, яка обумовлена наявністю елемента затримки, що потребує двохразового виконання операції додавання n-розрядних двійкових кодів ТЧБ Радемахера у 2n-розрядному накопичувачі з наскрізними переносами та двохразового запису кодів суми у регістрі пам’яті накопичувача.

Серед відомих засобів слід виділити число-імпульсний перемножуючий пристрій [3], який містить вхідну шину, лічильник, виходи якого порозрядно

через логічні ключі підключені до накопичувача.

Недоліком пристрою [3], який при однаковому числі імпульсів на вхідних шинах виконує обчислення їх квадрату, є низька швидкодія та висока структурна складність. Низька швидкодія відомого пристрою обумовлена тим, що обчислювальні операції у пристрої виконуються у двійковій системі числення ТЧБ Радемахера, що приводить до одноразового виконання наскрізних переносів у суматорі накопичувача та запису двійкових кодів у його регістрі пам'яті.

Згідно структури відомого пристрою, його часова складність та швидкодія, яка визначається сумарною затримкою сигналів після кожного вхідного імпульсу у послідовно з'єднаних компонентах пристрою, визначається згідно виразу :

$$\tau_н = (\tau_л + \tau_к + \tau_р + \tau_с), \qquad (4.1)$$

де $\tau_л = 2\upsilon$ - швидкодія переключення JK-тригера синхронного двійкового лічильника;

$\tau_к = 2\upsilon$ - швидкодія переключення логічних елементів логічних ключів, які складаються з двох послідовно включених логічних елементів «І», «АБО»;

$\tau_р = 2n \cdot \tau_с$, - швидкодія переключення D-тригера регістра накопичуючого суматора, n – число розрядів лічильника, в якому формується двійковий код вхідного унітарного коду з числом 2^n імпульсів;

$\tau_к = (2 \div 4)\upsilon$ - часова затримка сигналів накопичуючого багаторозрядного суматора у залежності від схеми його мікроелектронної реалізації на вентилях ПЛІС (υ - швидкодія переключення вентиля).

Наприклад, мінімальна часова складність та машинна швидкодія відомого пристрою буде рівна:

$$n = 8: \quad \tau_п = 2 + 2 + 2 + 16 \cdot 2 + 2 = 40\upsilon;$$

$$n = 16: \ \tau_n = 2 + 2 + 2 + 32 \cdot 2 + 2 = 72\upsilon .$$

Висока структурна складність відомого пристрою [3], яка ускладнює його синтез та реалізацію на мікроелектронному кристалі, обумовлена тим, що такий пристрій має нерегулярну структуру і містить різнотипні компоненти: лічильник на Т- або JK-тригерах, логічні ключі на елементах «АБО» та «І», накопичувач на основі повних однорозрядних суматорів на елементах «І», «І-НЕ», «АБО» та «Виключаюче АБО», регістр пам'яті накопичувача на D-тригерах.

Неоднорідність структури пристрою [3] обумовлена тим, що даний пристрій при максимальному числі імпульсів вхідного унітарного коду 2^n має різну розрядність лічильника (n) та накопичуючого суматора (2n).

Обмежені функціональні можливості відомого пристрою обумовлені тим, що вихідним кодом квадрату вхідного числа імпульсів є двійковий код системи числення базису Радемахера. Це не дозволяє його використати для подальшого швидкодіючого опрацювання даних у системі числення залишкових класів базису Хаара-Крестенсона.

В основу розробки пристрою поставлена задача вдосконалення, підвищення швидкодії, регулярності структури та розширення функціональних можливостей квадратора шляхом додаткового представлення числа імпульсів вхідного унітарного коду у модульній системі числення залишкових класів та додаткового введення логічного модуля рандомізації, шифратора та вихідної шини коду СЗК базису Крестенсона, що дозволяє додатково отримати код квадрату у базисі Радемахера-Крестенсона за 3 мікротакти та двійковий код квадрату у базисі Радемахера за 10 мікротактів після кожного вхідного імпульсу.

Ефективність пристрою досягається шляхом представлення вхідного числа імпульсів у модульному коді Хаара-Крестенсона, з його структури вилучений компонент з найнижчою швидкодією – накопичувач та найвищою структурною складністю – двійковий суматор з наскрізними переносами. Реалізація лічильника та накопичуючого суматора виконується на D-

тригерах, а логічний ключ містить тільки логічний елемент «АБО». Дослідження властивостей залишків квадратів для побудови пристрою наведені в [29].

На рисунку 4.14 показана структурна схема модульного лічильника системи числення залишкових класів ТЧБ Хаара-Крестенсона: Start – початкова установка D-тригерів модульного лічильника у стан «0», крім нульового тригера, який встановлюється в стан «1»;

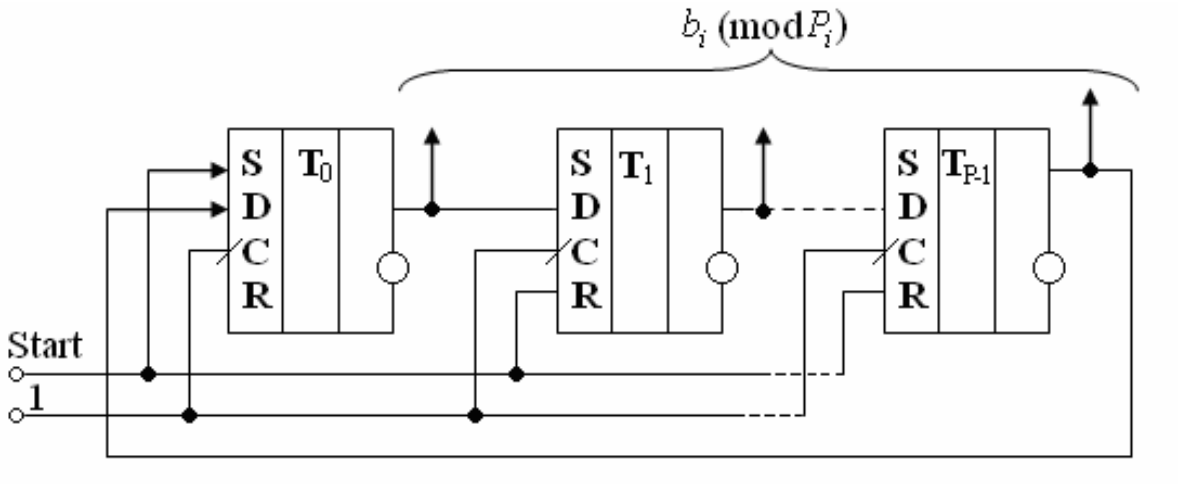


Рисунок 4.14 – Структурна схема модульного лічильника системи числення залишкових класів ТЧБ Хаара-Крестенсона

Поставлена задача вирішується завдяки тому, що квадратор містить вхідну шину, лічильник, логічні елементи та регістр пам’яті. У розробленому пристрої на відміну від відомого [3] додатково введені розрядно-позиційні лічильники ТЧБ Хаара-Крестенсона, входи яких з’єднані з вхідною шиною, а виходи - з входами додатково введеного логічного модуля рандомізації, виходи якого з’єднані з першими додатково введеними виходами пристрою кодів квадратів у ТЧБ Хаара-Крестенсона і першими інформаційними входами регістра пам’яті, другий вхід якого з’єднаний з вхідною шиною, а виходи з’єднані з входами додатково введеного шифратора, виходи якого є другими виходами пристрою у вигляді двійкових кодів квадратів числа вхідних імпульсів у ТЧБ Радемахера.

Перед початком кожного циклу роботи розробленого квадратора всі D-

тригери модульних лічильників пристрою окремою мікрокомандою скидаються в «0», крім нульового тригера, який встановлюється в стан «1» .

На рисунку 4.15 подано приклад реалізації формування коду квадрата числа на виходах логічного модуля рандомізації у базисі Хаара – Крестенсона (P=11).

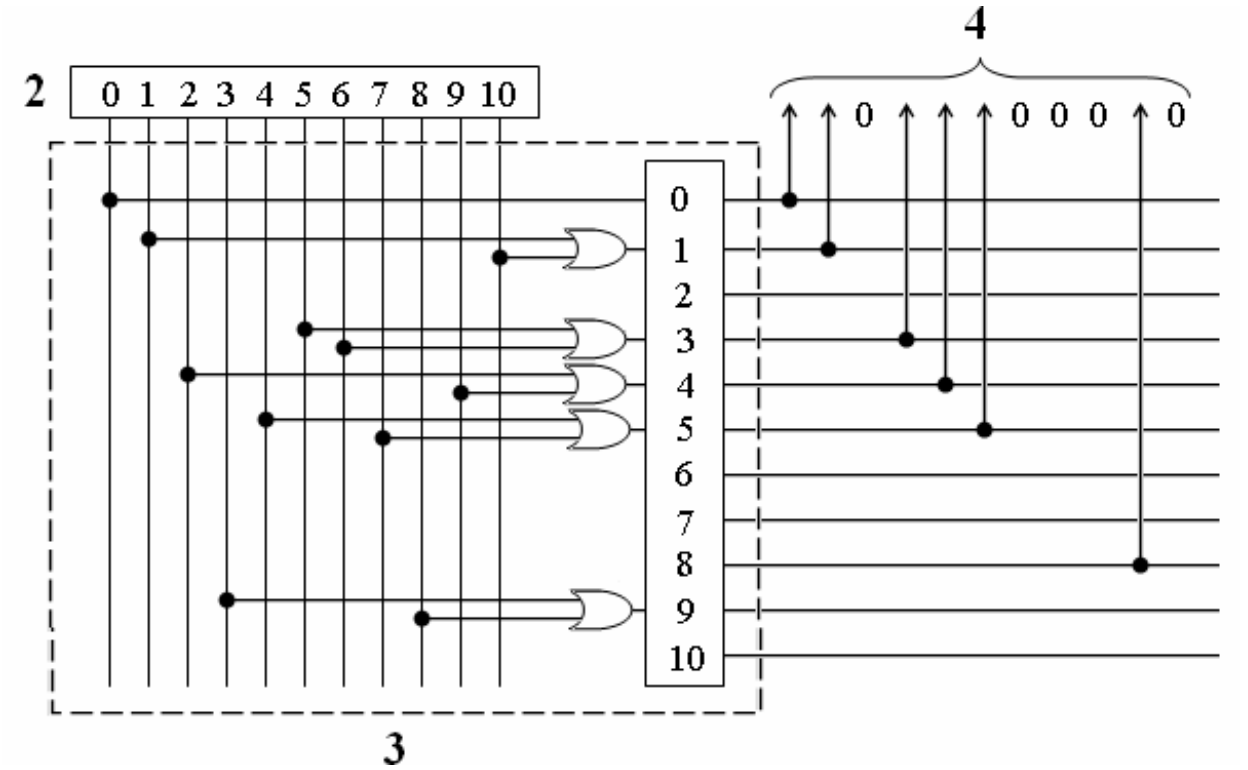


Рисунок 4.15 – Приклад реалізації формування коду квадрата числа на виходах логічного модуля рандомізації у базисі Хаара – Крестенсона (P=11)

При поступленні кожного імпульсу унітарного коду (рисунок 4.16) числа на вхідну шину (1) у модульних лічильниках (2) накопичуються коди залишків числа імпульсів у СЗК базису Хаара-Крестенсона, які поступають на входи відповідних логічних модулів рандомізації (3), на виходах яких формується код квадрата числа вхідних імпульсів у ТЧБ Хаара-Крестенсона, який поступає на першу вихідну шину (4) та інформаційні входи регістра пам’яті (5), запис в який синхронізується імпульсами вхідної шини. З виходів регістра пам’яті отримані розрядно-позиційні коди ТЧБ Хаара-Крестенсона дешифруються шифратором (6), на виходах якого формуються двійкові коди квадратів вхідного числа імпульсів у ТЧБ Радемахера, що є другою вихідною

шиною пристрою (7).

На рисунку 4.16 представлена структурна схема пристрою, де 1 – вхідна шина; 2 – модульний лічильник системи числення залишкових класів ТЧБ Хаара-Крестенсона; 3 – логічний модуль рандомізації; 4 –вихідна шина.

Приклад реалізації формування коду квадрата числа на виходах логічного модуля рандомізації (3) для P=11 наведений у таблиці 4.3.

Таблиця 4.3 - Приклад реалізації формування коду квадрата

$b_i \times b_i = d_i (\mathrm{mod}\ 11)$	$b_i \times b_i = d_i (\mathrm{mod}\ 7)$	$b_i \times b_i = d_i (\mathrm{mod}\ 5)$
0 x 0 = 0	0 x 0 = 0	0 x 0 = 0
1 x 1 = 1	1 x 1 = 1	1 x 1 = 1
2 x 2 = 4	2 x 2 = 4	2 x 2 = 4
3 x 3 = 9	3 x 3 = 2	3 x 3 = 4
4 x 4 = 5	4 x 4 = 4	4 x 4 = 1
5 x 5 = 3	5 x 5 = 4	
6 x 6 = 3	6 x 6 = 1	
7 x 7 = 5		
8 x 8 = 9		
9 x 9 = 4		
10 x 10 = 1		

Згідно системи числення залишкових класів, для однозначного представлення вхідного числа імпульсів унітарного коду 2^n повинна виконуватися умова, що добуток взаємно простих модулів P_i має бути рівний або більший 2^n, що відповідає умові: сума двійкових розрядностей модулів P_i повинна бути на 1-2 розряди більша по відношенню до розрядності двійкового числа, яке підноситься до квадрату і представляє число імпульсів

унітарного коду N у двійковій системі числення, де $n = \hat{E}[\log_2 N]$ є знак цілочисельної функції з округленням до більшого цілого.

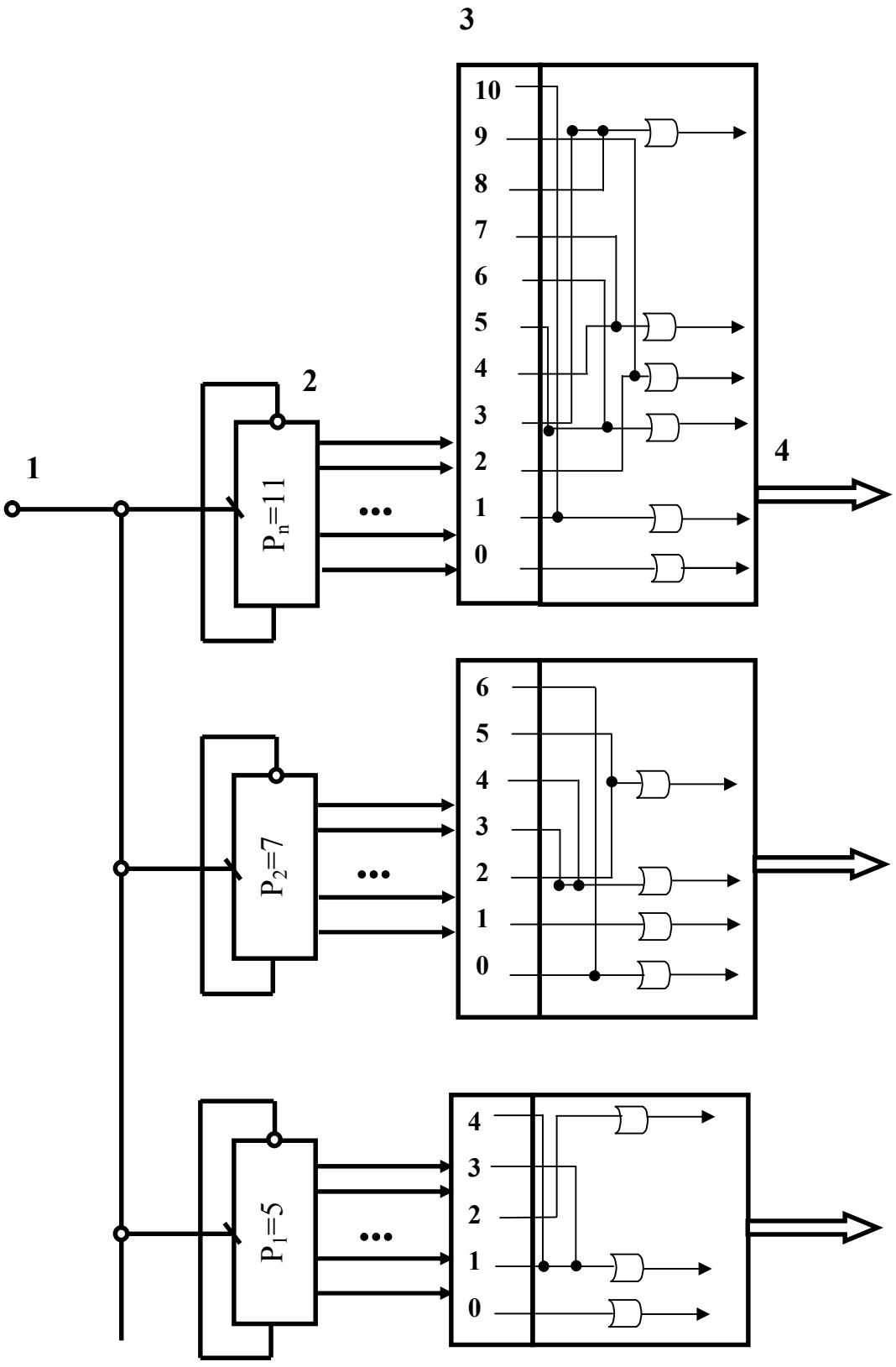


Рисунок 4.16 – Структурна схема пристрою піднесення до квадрата в базисі Хаара –Крестенсона

При проектуванні квадратора на ПЛІС використовується логічний модуль з повним числом логічних елементів «АБО», які синтезуються в якості відповідних утиліт по кожному модулю P_i.

У таблиці 4.4 наведено приклад формування квадрату в СЗК при кількості модулів k=4.

Таблиця 4.4 – Формування квадрату в СЗК при кількості модулів k=4

		P_1	P_2	P_3	P_4	2n+1
		m_1	m_2	m_3	m_4	
N	100	9	11	13	15	16
n	7	4	4	4	4	
N	128	9	11	13	15	16
n	7	4	4	4	4	
N	256	9	11	13	17	17
n	8	4	4	4	5	
N	65536	129	131	137	263	33
n	16	8	8	8	9	

При іншій кількості модулів СЗК можуть застосовуватися інші набори, які відповідають вказаним умовам. При піднесенні до квадрату 512-розрядних двійкових чисел потрібно 101 десятибітний модуль P_i. При цьому швидкодія піднесення чисел до квадрату у базисі Хаара-Крестенсона не залежить від розрядності і в запропонованому пристрої виконується за $\tau = 3\upsilon$.

Розрахунок апаратної складності квадратора, реалізованого у базисі Хаара – Крестенсона, виконується для процесорів різної розрядності.

Пристрій характеризується підвищеною на 1-2 порядки швидкодією по відношенню до відомих рішень, а також більш високою регулярністю структури за рахунок реалізації модульних синхронних лічильників на D-тригерах та логічного модуля рандомізації на елементах «АБО».

4.3 Розробка алгоритму та реалізація програми компактного кодування багаторозрядних простих чисел

Згідно схеми, представленої на рисунку 4.17, для кодування та

збереження простих чисел розроблено програмний продукт, який дозволяє зберігати послідовність БРПЧ шляхом порозрядного розбиття та отримувати із збереженого бінарного файлу будь-яку кількість попередньо збережених БРЧ.

На рисунку 4.17 зображена схема порозрядного розбиття на групи послідовності БРПЧ, яка є адаптивною і дозволяє вибрати довільне двійкове закінчення згенерованої двійкової послідовності БРПЧ.

Обраний варіант розбиття ефективний для послідовності до 32 розрядів, для розрядностей 512-1024 біт він ефективно буде зберігати групу розрядів по два байти [32].

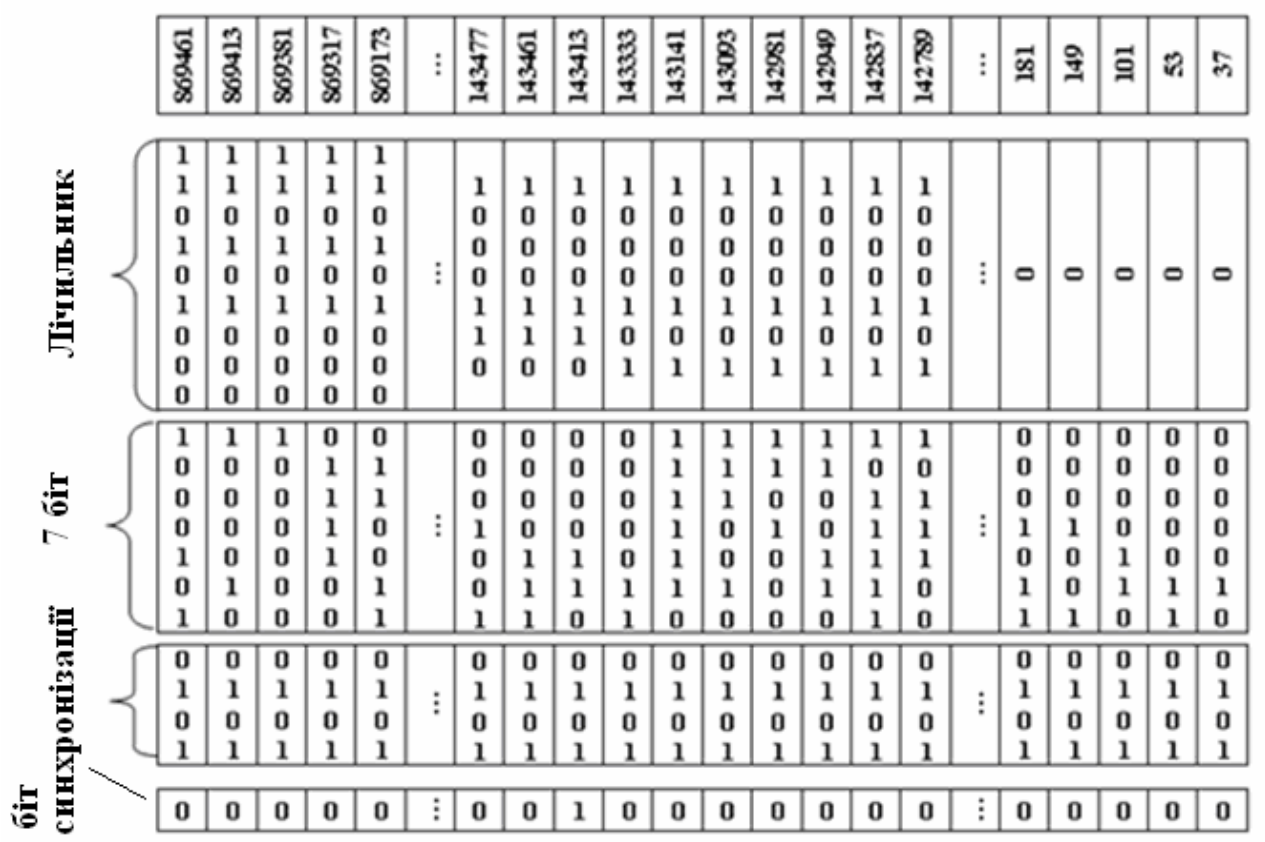


Рисунок 4.17 – Схема розбиття числа на групи розрядів

Розроблений додаток для генерування та кодування простих БРЧ забезпечує максимально простий інтерфейс, головне вікно якого зображене на рисунку 4.18.

Даний додаток дозволяє обрати двійкове закінчення для вибірки,

кодування і збереження послідовної множини простих чисел, причому розмірність двійкового закінчення не обмежена [32].

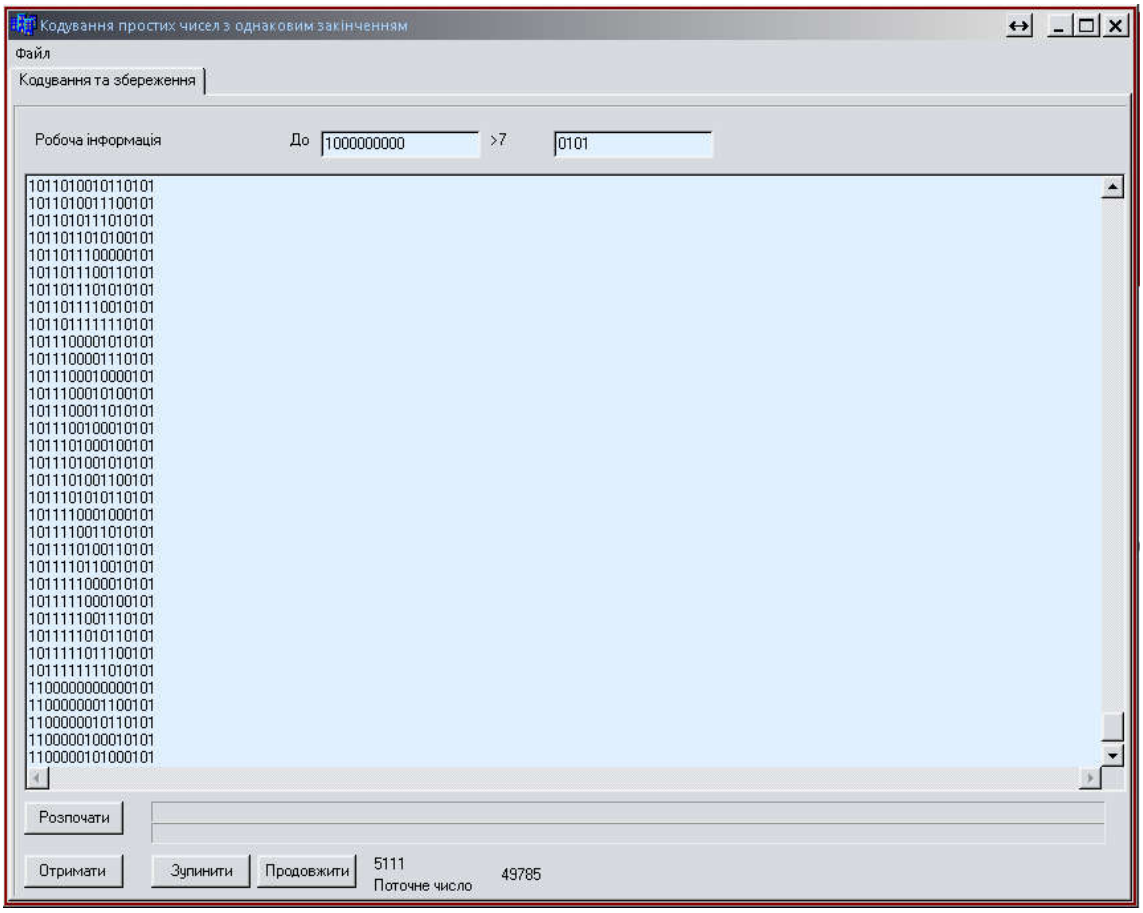


Рисунок 4.18 – Кодування простих чисел з однаковим закінченням

При обчисленні простих чисел створюється декілька потоків, що пришвидшує процес обчислення та уникає зависань. У файл записується 7-бітний код та біт синхронізації, що свідчить про наростання лічильника. Таким чином, додаток дозволяє уникнути зберігання надлишкового двійкового коду, зберігаючи лише 1 байт інформації для кожного числа. При декодуванні підраховується кількість бітів синхронізації, їх сума утворює верхню частину числа. Конкатенація отриманої суми, семи бітів та обраних нижніх розрядів для вибірки з послідовності простих чисел утворює код простого числа.

При генеруванні послідовності простих БРЧ в операційній системі створюється обчислювальний потік, що забезпечує паралельне опрацювання даних та дозволяє роботу з додатком навіть під час проведення генерування

БРПЧ, а також паралельне зберігання групи розрядів в файл на диск.

Для декодування розроблено додаток, головна форма якого зображена на рисунку 4.19.

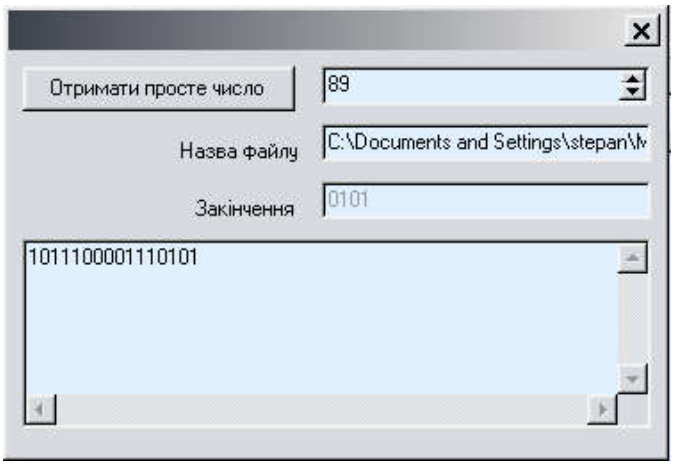


Рисунок 4.19 – Отримання коду простого числа з однаковим закінченням

Для отримання простого числа необхідно внести назву файлу, номер простого числа та двійкове закінчення простого числа. При вибірці додаток послідовно зчитує записи з двійкового файлу, формуючи верхню частину двійкового числа за рахунок підрахунку кількості бітів синхронізації.

В результаті проведених експериментальних досліджень було отримано часові характеристики програмного продукту, які представлені в таблиці 4.5.

Таблиця 4.5 – Експериментальні часові характеристики роботи додатку

Розрядність	Час, год.
32	16
64	72
128	240

Чисельні експерименти показують, що при генеруванні та кодуванні послідовності простих чисел до 32 розрядів основну часову затримку складала операція збереження групи розрядів, після 32 розрядів часові затримки пов’язані з генерування БПЧ [32].

4.4 Пристрій компактного кодування багаторозрядних простих чисел

На основі запропонованого у підрозділі 2.5 методу компактного кодування [34] розроблена структурна схема пристрою зберігання та генерування БПЧ, яка показана на рисунку 4.20.

Пристрій містить: 1- вхідна інтерфейсна шина; 2 – адресна шина; 3 – шина стартового коду старших розрядів простого числа; 4 – вихідна шина коду БПЧ; +1 – інкрементна одиниця накопичуючого суматора; 2^0-2^{14} – код БПЧ; БІ – блок ініціалізації; ГІ – генератор імпульсів; ПЗП – постійний запам'ятовуючий пристрій; РГ – регістр пам'яті; ∑ - багаторозрядний паралельний суматор; 2 – вихідна шина.

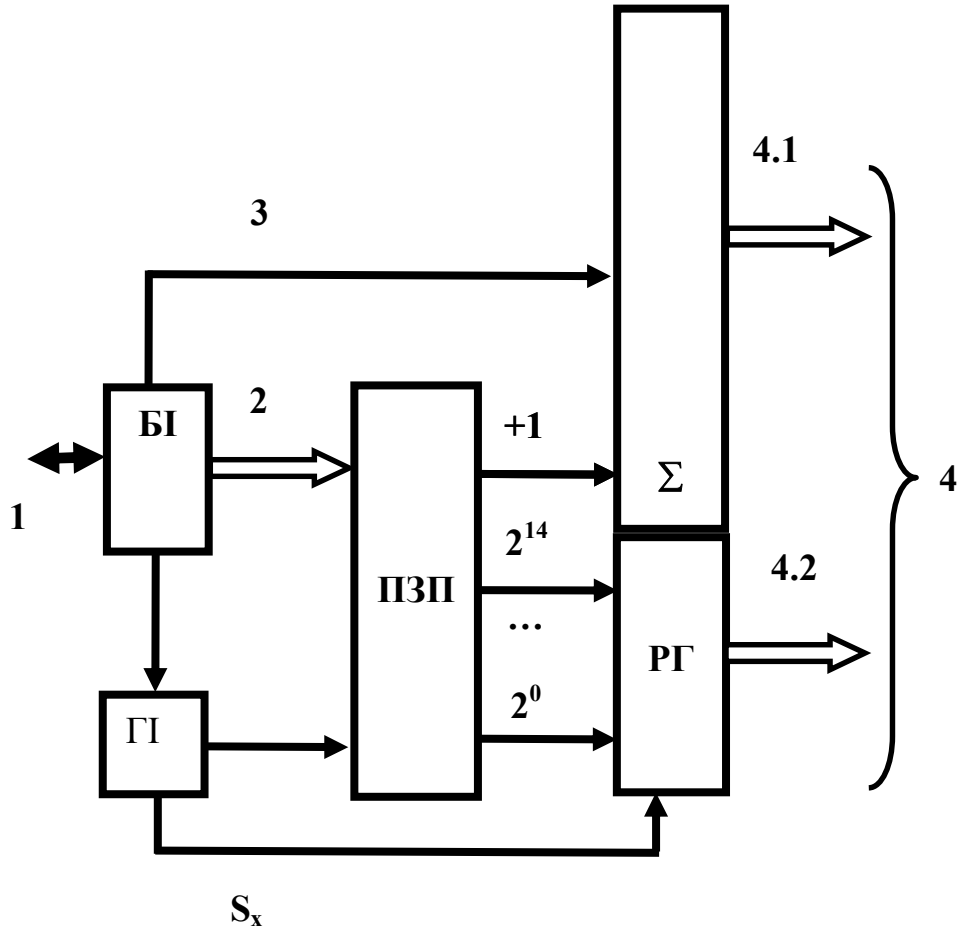


Рисунок 4.20 – Структурна схема пристрою компактного кодування та генерування БПЧ

В основу пристрою покладено процес зберігання в ПЗП 15 молодших

розрядів кодів БПЧ та одного розряду – біту синхронізації, на основі якого відбувається інкрементне нарощення старших бітів у суматорі ∑, починаючи з 16-го розряду числа у суматорі. В структурі пристрою функціональні модулі виконують наступні операції: БІ – блок ініціалізації, оснащений інтерфейсною шиною 1, реалізує інформаційний зв'язок з зовнішнім комп’ютерним пристроєм і виконує стартові функції: записує в ПЗП стартовий код 15 біт молодших розрядів БПЧ, а в суматор - багаторозрядний стартовий код старших розрядів БПЧ і запускає генератор імпульсів ГІ.

В процесі генерування імпульсів відбувається інкрементне генерування адресів ПЗП, що забезпечує генерування кодів молодших розрядів БПЧ. У момент появи біта синхронізації на виході ПЗП відбувається інкрементне нарощення кодів суматора. Перший вихід генератора інкрементує адресацію ПЗП, а другий вихід генератора реалізує запис інформації в регістр та запис стартового коду суматора. На вихідній шині 4 формується послідовність багаторозрядних кодів БПЧ.

В якості ПЗП використовуються кристали флеш-пам’яті з відповідною адресною розрядністю. При цьому, враховуючи, що для запису 15 молодших розрядів БПЧ і біта синхронізації необхідно 2 байти, незалежно від розрядності БПЧ, що забезпечує необхідну компактність зберігання великого об’єму кодів.

Наприклад, при об’ємі флеш-пам’яті 32 ГБ число компактно закодованих та генерованих 32-бітних простих чисел відповідно складатиме $64 \cdot 10^9$ розрядів кодів чисел.

В залежності від розрядності стартового коду розрядність генерованих БПЧ може довільно зростати. У результаті досягається зменшення об’єму кодів для зберігання БПЧ відповідно в межах 32 розрядів у два рази, для 256 розрядів в 16 разів, а при 1024 розряди - в 64 рази[34].

У таблиці 4.6 приведено характеристики та об’єми рекомендованої флеш-пам’яті.

Таблиця 4.6 – Рекомендовані характеристики флеш-пам’яті, необхідної при реалізації пристрою кодування

Модель пристрою	Характеристики (об’єм / швидкість)
Silicon Power Marvel M01	32 Гб; 5 Гбіт/с.
Transcend JetFlash 350	32 Гб; 5 Гбіт/с.
Kingston DataTraveler SE9 G2	64 Гб; 15 МБ/с.
Kingston DataTraveler 101 G2	128 Гб; 5 МБ/с.

На рисунках 4.21, 4.22, показані структури мікроелектронних компонентів пристрою компактного кодування та генерування БПЧ.

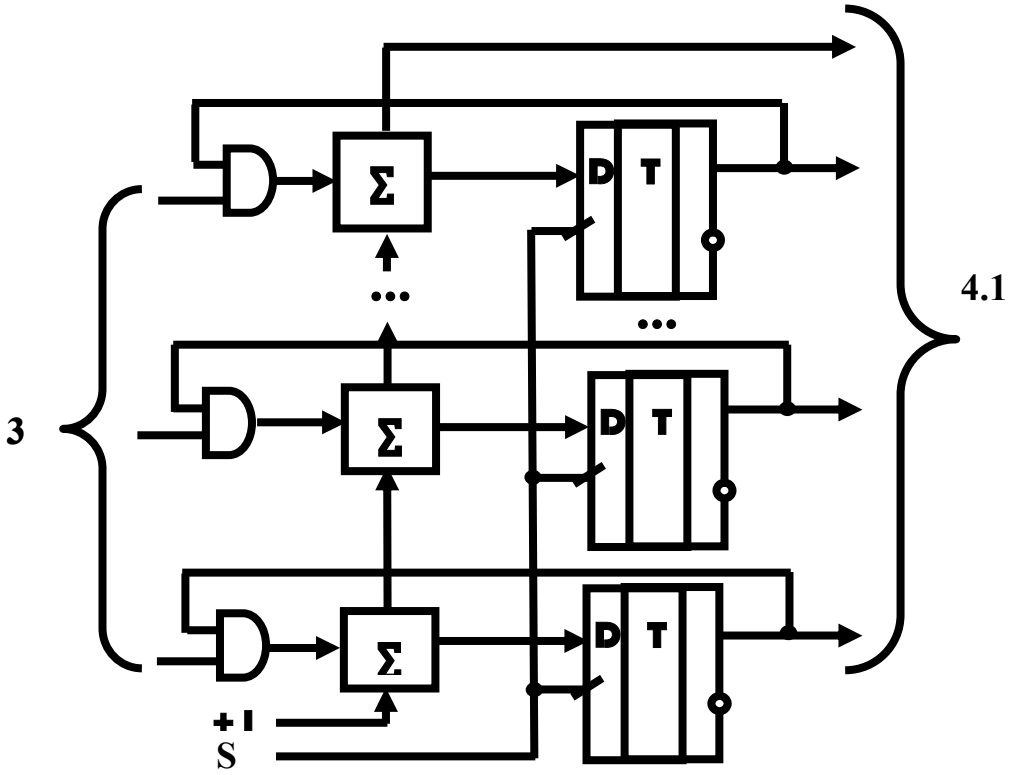


Рисунок 4.21 - Структурна схема накопичуючого суматора з паралельним 3 та інкрементним +1 входами - формувача старших розрядів БПЧ

Швидкодія структури неповного суматора, представленого на рисунку 4.22, визначається часом переключення одного логічного елемента, тобто складає 1υ.

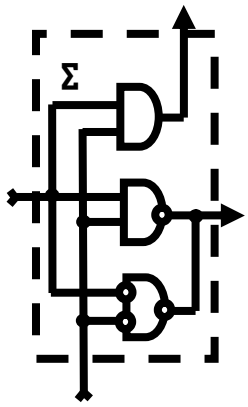


Рисунок 4.22 - Структура неповного суматора на логічних елементах І, І-НЕ та І-НЕ з інверсними входами

Така реалізація компонента багаторозрядного суматора забезпечує підвищення швидкодії наскрізних переносів у 3 – 5 разів у порівнянні з відомими схемами, які побудовані на логічних елементах І, АБО, НЕ, а також на елементах типу XOR, які в своїй структурі містять п’ять логічних елементів, з яких три з’єднані послідовно.

Розрахунок апаратної складності запропонованих компонентів пристрою компактного кодування та генерування БПЧ у залежності від розрядності розраховується згідно виразу:

$$A = A_{РГ} + A_{\Sigma}$$

$$A_{РГ} = 15 \cdot A_{\mathrm{DT}} = 15 \cdot 2 = 30\upsilon,$$

де υ - вентилі, які реалізуються на ПЛІС.

$A_{\Sigma} = ЛЕ + \mathrm{A}_S + A_{DT}$, де $\mathrm{A}_S = 3ЛЕ$ - число логічних елементів однорозрядного неповного суматора, звідки $A_{\Sigma} = \mathrm{n}(ЛЕ + 3ЛЕ + 2ЛЕ) = 6\mathrm{n}\upsilon$. На рисунку показано характеристики апаратної складності базових компонентів пристрою та зменшення об’єму використовуваної пам’яті при зростанні розрядності БПЧ.

З рисунку видно, що зменшення апаратної складності порівняно з відомими структурами складає півтора-два рази, а об’єм пам’яті при

зростанні розрядності БПЧ в межах 32-1024 відповідно зменшується в 2 – 64 рази.

На рисунку 4.23 показано характеристики часової складності пристрою, які розраховуються на основі параметрів накопичуючого суматора, який має більше число послідовно з'єднаних елементів $\tau_{РГ} = 2\upsilon$, а $\tau_{\Sigma} = 3n$ нс.

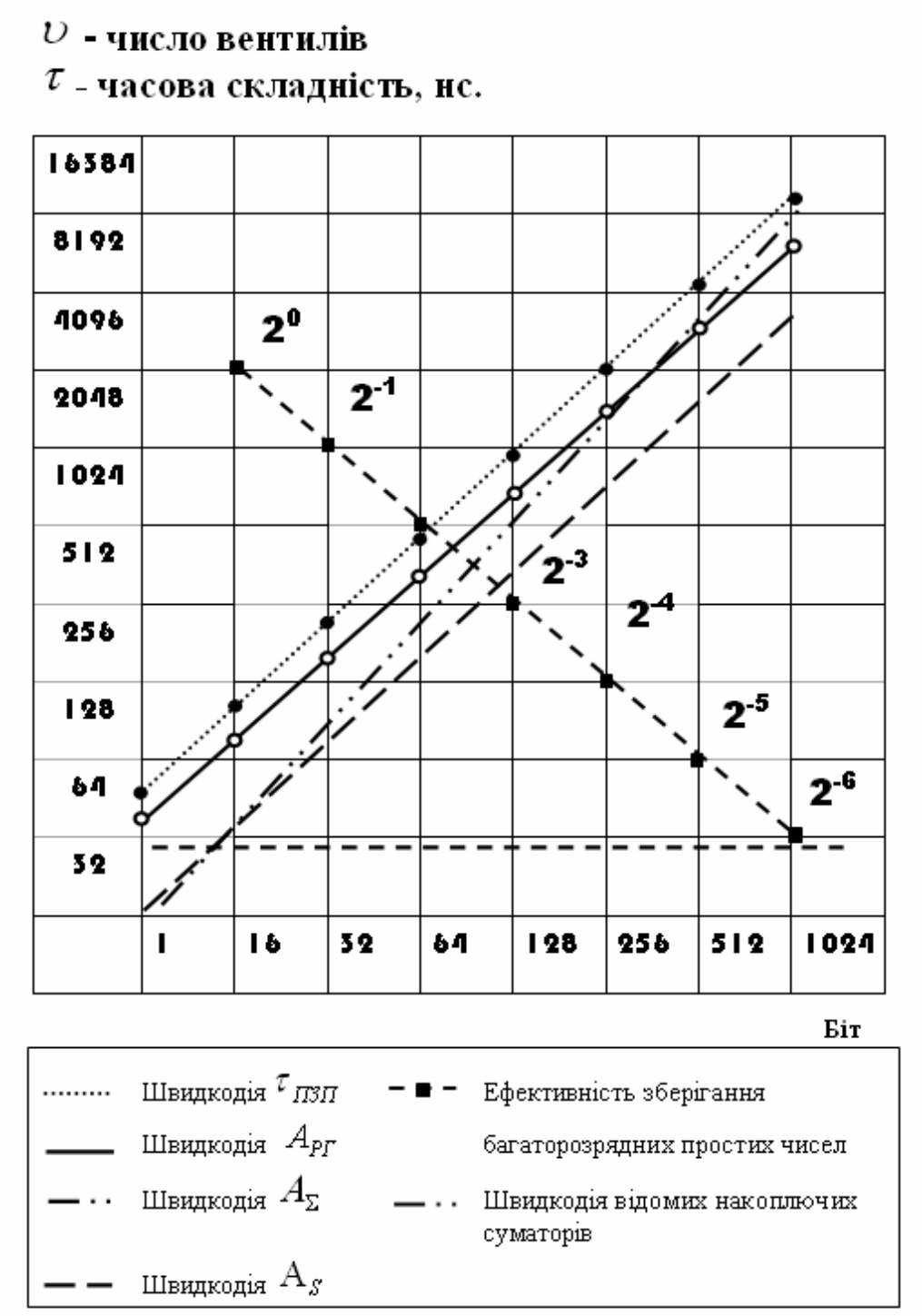


Рисунок 4.23 - Характеристики часової складності пристрою кодування багаторозрядних чисел та його структурних елементів

Це призводить до зменшення часової складності в більш, ніж два рази в порівнянні з відомими реалізаціями накопичуючих суматорів, часова складність яких складає $\tau_{\Sigma} = 7n$,. Тобто швидкодія розробленого схемотехнічного рішення перевищую аналоги в два рази.

Суттєве підвищення швидкодії пристрою компактного кодування та генерування БПЧ може бути досягнуте застосуванням в якості компонента накопичуючого суматора запропонованого в роботі [68] швидкодіючого багаторозрядного суматора, структура якого показана на рисунку 4.24. При цьому часова складність складає $log_2 n$. В той же час, як видно з рисунку 4.23, його апаратна складність, практично, на порядок вища в порівнянні з розробленим, що може обмежувати доцільність практичного застосування подібного елемента в розробленому пристрої.

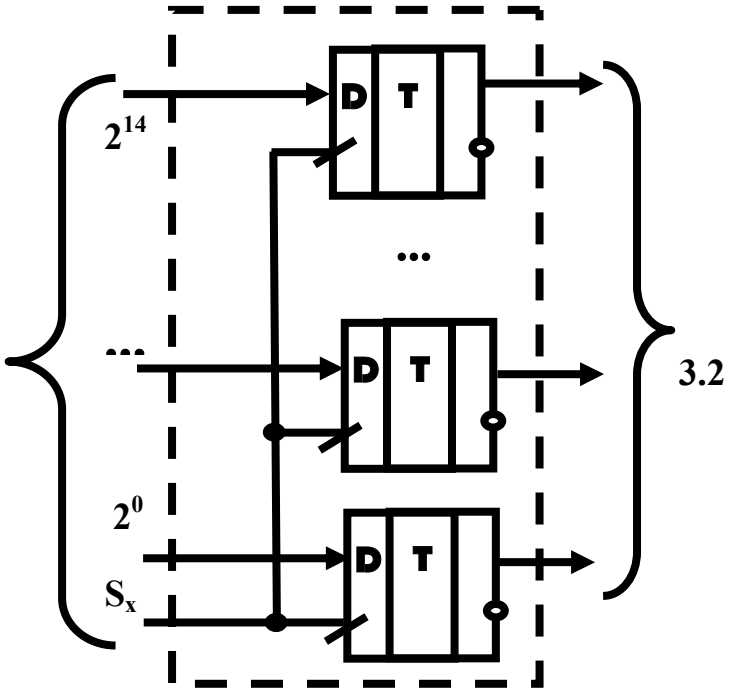


Рисунок 4.24 – Структура регістра пам’яті молодших розрядів БПЧ на D-тригерах

Обчислимо ефективність пристрою згідно суми ефективності обчислювальних елементів: $\tau = \tau_{ГІ} + \tau_{ПЗП} + \tau_{\Sigma}$. Кількість логічних елементів

генератора імпульсів $\tau_{ГІ} = 2\upsilon$; регістр пам'яті містить $\tau_{РП} < \tau_{\sum}$; постійно запам'ятовуючий пристрій $\tau_{ПЗП} = 10\upsilon$; суматор - $\tau_{\sum} = 4\upsilon$.

Таким чином, швидкодія пристрою генерування та кодування БПЧ складає $\tau = 16\upsilon$, що свідчить про високу ефективність розробленого пристрою опрацювання БПЧ.

4.5 Апаратна реалізація методу факторизації багаторозрядних чисел

Найбільш близьким за технічною суттю до розробленого пристрою є метод факторизації БРЧ, що ґрунтується на представленні чисел у позиційній двійковій системі числення ТЧБ Радемахера [28].

Метод полягає у тому, що число *m* представляється k-розрядним двійковим числом, з якого визначається ціла частина від квадратного кореня з *n*, $m = \lfloor \sqrt{n} \rfloor$ для різних значень *x=1,2,…,* послідовно багатократно визначається значення згідно виразу $q(x) = (m + x)^2 - n$ до тих пір, поки отримане значення не буде рівне повному квадрату у вигляді цілого числа згідно двійкової арифметики. Недоліком такого способу є велика обчислювальна складність, яка обумовлена виконанням великого числа обчислювально-складних операцій у позиційній двійковій системі числення над БРЧ, які включають операції піднесення до степеня, додавання, віднімання та добування квадратного кореня числа, розрядність якого експоненційно зростає, починаючи з розрядності числа *n.* Недоліком також є низька швидкодія реалізації способу факторизації БРЧ, обумовлена наявністю наскрізних переносів при виконанні операцій додавання, множення, піднесення до квадрату та віднімання, які в найбільшій мірі знижують швидкодію реалізації способу факторизації БРЧ [28].

Розглянемо приклад факторизації БРЧ у двійковій системі числення над десятковими числами, заданими у прикладі: *n* = 1000101011111100011000101111100010010100010010110000100010101011110001010101011101011101010001000101 (k=102). Для визначення

51-розрядних двійкових чисел p= 1000110000010100000111101101001110 1101010110000111 та q= 1111111000000000110001100001001111111111 01011010011 необхідно виконати операцію визначення кореня квадратного $m = \lfloor \sqrt{n} \rfloor$ = 1011110010100000101011010000010101010111100111000000, визначити цілу частину 1011110010100000101011010000010101 01011110011100000 та виконати x=1000011010011100010101101110100 0010101101001100 ітерацій для визначення числа q(x). На завершальному кроці ітерації отримується число $(m+x)^2 - n =$ 1001011110101001000101001001001101110110011000110010000010110111001101111110000110111111100011110010000 .

Поставлена задача розробки нового способу факторизації БРЧ у системі числення залишкових класів ТЧБ Крестенсона, що дозволяє зменшити обчислювальну складність та підвищити швидкодію процесу факторизації шляхом вилучення операцій добування квадратного кореня при виконанні багатократних ітерацій, а також зменшити розрядність чисел, над якими виконуються обчислювальні операції. Це приводить до підвищення швидкодії реалізації способу факторизації БРЧ у порівнянні з відомим способом на два-три порядки, оскільки операції віднімання, множення та піднесення до степеня виконуються паралельним способом без наскрізних переносів згідно арифметики системи числення залишкових класів ТЧБ Крестенсона [88].

Пристрій працює наступним чином: з відомого числа *n*, представленого в двійковій системі числення, визначається корінь квадратний, який округлюється до більшого цілого у вигляді P_c^*. Отримане число представляється залишками $b_1,b_2,b_i,\ldots,b_k]$, *i=1,2,3...k* у системі взаємно простих модулів $[p_1,p_2,p_i,\ldots,p_k]$ згідно виразу $b_i = \text{res } P_c^* \bmod p_i$. Визначається квадрат $S_0 = (P_c^*)^2 = (S_1,S_2,S_i,S_k)$, *i=1,2,3...k* згідно модульної арифметики СЗК $S_i = \text{res } (b_i \times b_i) \bmod p_i$, додатково визначається різниця між значеннями $\Delta_0 = S_0 - n$ згідно модульної арифметики СЗК $\Delta_{0i} = \text{res } S_{0i} - b_i$, визначається крок приростів квадратів, починаючи з S_0 згідно виразу $2 \cdot P_c^* + 1$, ітераційно

виконується визначення значень S_x у модульній арифметиці СЗК згідно виразу $S_x = 2 \cdot P_c^* x + x^2 + \Delta_0$. Запам'ятовуються у стекову пам'ять отримані значення S_x, S_{x-1}, S_{x-2} ,…,S_{x-z}, записуються у стекову пам'ять $P_c^* + x$, які є коренями квадратними з S_x , у кожній ітерації порівнюються значення S_x з усіма значеннями S_{x-z}, які містяться у стековій памяті, а при співпаданні кодів $S_{x.i}$= $S_{x-z,i}$ у СЗК факторизовано числа *p,q* визначаються згідно виразів $p = P_c^* + x - \sqrt{S_{x-z,i}}$, $q = \sqrt{S_{x-z,i}} + P_c^* + x$ представлених в [111].

Розглянемо простий приклад. Нехай *n=93=pq*, де *p=3*, *q=31*. На рисунку 4.25 показано граф розв'язання запропонованого способу реалізації, де $\sqrt{n} \approx 9{,}6436507$, тобто наближення до більшого цілого $\lfloor \sqrt{n} \rfloor = 10$, таким чином найближчий старший квадрат $10^2 = 100$, звідки $\Delta_0 = 100 - 93 = 7$, таким чином стартове значення $S_0 = 7$, а крок зростання квадратів дорівнює $2 \cdot 10 + 1 = 21$, наступні кроки збільшуються на 2. Запам'ятовуються стартовий корінь квадратний та його квадрат, який відповідно на кожній ітерації буде змінюватися з кроком 21, 23, 2 5,… в стековій пам'яті, на кожній ітерації відбувається асоціативне порівняння чисел в СЗК S_x з поточними квадратами, що містяться у асоціативній стековій пам'яті.

Розглянемо виконання процесу факторизації числа n на прикладі кодів СЗК. Вибираємо систему взаємно простих модулів залишкових класів, розрядність добутку яких перевищує розрядність *n*.

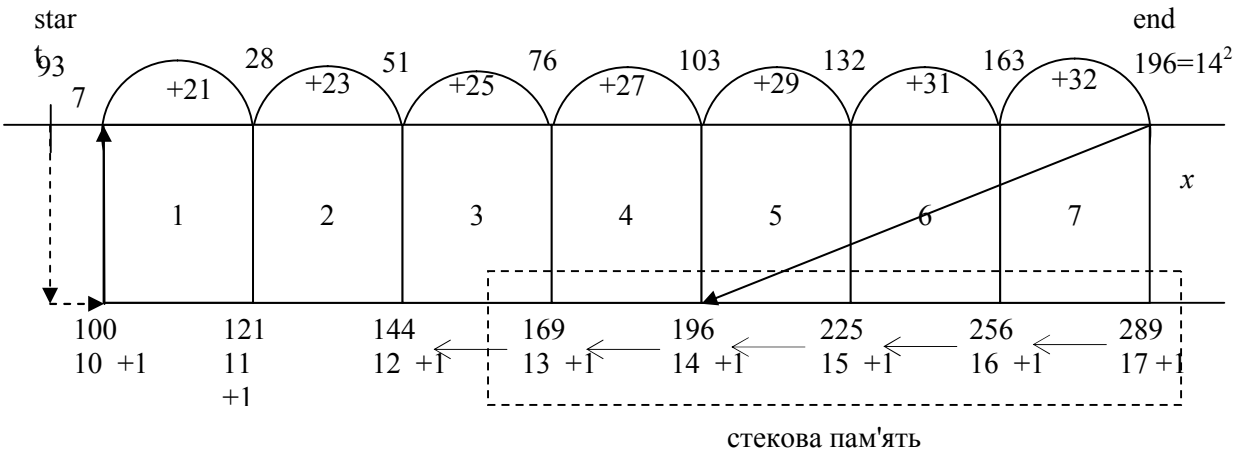


Рисунок 4.25 - Процес факторизації n – розрядного числа

При співпадінні отриманого значення із значенням, що у стековій пам'яті, завершується процес факторизації числа *n,* як показано стрілкою на рисунку 4.25.

Нехай p_1=5, p_2=7, p_3=11 - обрана система взаємно простих модулів. Розрядність добутку обраних модулів (8 біт) задовільняє умові і перевищує розрядність *n=93.*

Представляємо Δ_0 в системі залишкових класів:

Δ_0=7 → res(mod 5) =2 ; res(mod 7)=0 ; res(mod 11)=7 ;

$2 \cdot P_c^* + 1$ =21 → res(mod 5) =1 ; res(mod 7)=0 ; res(mod 11)=10 ;

10 → res(mod 5) =0; res(mod 7)=3; res(mod 11)=10;

100 → res(mod 5) =0; res(mod 7)=2; res(mod 11)=1.

Виконуємо операції запропонованого способу факторизації, використовуючи обрану систему взаємно простих модулів (p_1=5, p_2=7, p_3=11) у СЗК, як показано в таблиці 4.7.

Таблиця 4.7 - Метод факторизації використанням системи взаємно простих модулів(p_1=5, p_2=7, p_3=11)

x	0	1	2	3	4	5	6	7
P_1=5	2	3	1	1	3	2	3	1
	1	3	3	0	2	4	1	2
	0	1	2	3	4	0	1	2
	0	1	4	4	1	0	1	4
P_2=7	0	0	2	6	5	6	2	0
	0	2	2	4	6	1	3	4
	3	4	5	6	0	1	2	3
	2	2	4	1	0	1	4	2
P_3=11	7	6	7	10	4	0	9	9
	10	1	1	3	5	7	9	10
	10	0	1	2	3	4	5	6
	1	0	1	4	9	5	3	3
	7	28	51	76	103	132	163	196
	21	23	23	25	27	29	31	17
	10	11	12	13	14	15	16	17
	100	121	144	169	196	225	256	289

Уведення виконання операції факторизації БРЧ на основі представлення чисел у системі числення залишкових класів дозволяє на 3 - 4 порядки підвищити швидкодію реалізації способу за рахунок виконання модульних операцій піднесення до квадрату, множення, додавання та порівняння на основі модульної арифметики, які виконуються паралельно по кожному модулю за 2 - 4 мікротакти, оскільки не містять наскрізних переносів і їх швидкодія не залежить від розрядності чисел, які факторизуються. Функціональна схема пристрою наведена на рисунку 4.26.

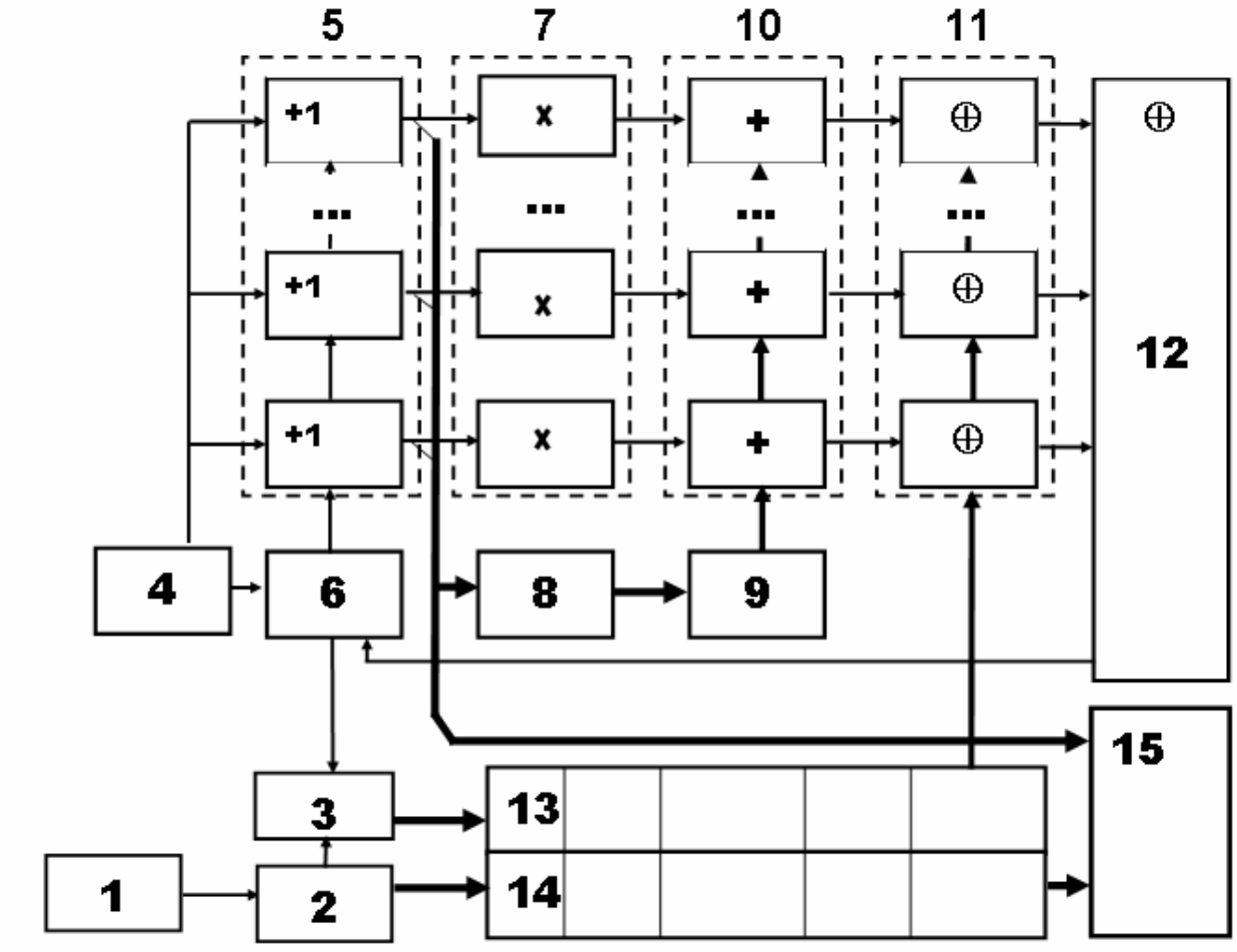


Рисунок 4.26 – Схема реалізації способу факторизації в СЗК

Пристрій складається з таких компонентів: 1 – блок вводу двійкового коду числа, що факторизується; 2 - блок визначення двійкового значення квадратного кореня числа n з заокругленням до більшого цілого, перетворення його у СЗК та ітераційного формування інкрементно

зростаючих кодів; 3 - блок здійснює модульне піднесення числа у системі числення залишкових класів до квадрату та формування їх інкрементної послідовності у кожному циклі ітерації; 4 – стартовий блок початку процесу факторизації; 5 – лічильник у базисі Хаара-Крестенсона; 6 - тактовий генератор; 7 - блок формування кодів квадратів числа ітерацій у СЗК; 8 - блок перемноження; 9 - перший блок додавання; 10 - другий блок додавання; 11 - блок порівняння кодів; 12 - блок визначення рівності кодів; 13 - стекова пам'ять асоціативна квадратів; 14 – стекова пам'ять коренів квадратних квадратів у двійковій системі числення; 15 - блок реєстрації та перетворення кодів СЗК у двійкову систему числення.

Досягнуте підвищення швидкодії запропонованого способу факторизації [111] дозволяє спростити розв’язання ряду фундаментальних задач теорії чисел, зокрема: розкладу числа на прості множники, визначення символів Якобі, генерація БРПЧ, повний розклад числа на множники.

Процес факторизації згідно запропонованого способу включає наступні етапи: число, що факторизується, з блоку 1 поступає на вхід блоку 2, який здійснює у двійковій системі числення визначення кореня квадратного, округлюється до більшого цілого, перетворюється в СЗК і з першого виходу поступає в стекову пам'ять 14, на виході якої у блоці 15 формується двійковий код завершення процесу факторизації. В блоці 3 здійснюється формування квадратів чисел у СЗК, що записуються у асоціативну стекову пам'ять 13. Початок процесу факторизації здійснює блок 4, який записує стартовий код числа ітерацій в блок 5 та виконує запуск тактового генератора 6, під дією тактових сигналів генератора 6 здійснюється інкрементне формування та запис інкрементно наростаючих кодів СЗК у стекову асоціативну пам'ять 13, 14, а також інкрементне кодування числа ітерацій у блоці 5 та їх реєстрація у блоці 15. При цьому синхронно у блоці 7 виконується піднесення числа ітерацій до квадрату, а у блоці 8 - перемноження числа ітерацій на постійний код $2P_c$. В блоці 9 відбувається додавання отриманих результатів з кодом Δ_0. Далі отримана сума додається до квадратів чисел, які формуються у блоці 10. В 11 відбувається порівняння

отриманої суми блоку 10 з усіма кодами асоціативної пам'яті 13 і у випадку співпадіння одного з кодів, що визначається блоком 12, вихідний сигнал, якого зупиняє тактовий генератор 6. При цьому на виході 15 реєструються коди числа ітерацій та кореня квадратного асоціативної пам'яті 14, що є завершенням процесу факторизації. Код реалізації пристрою для факторизації багаторозрядного числа на ПЛІС наведений в додатку Н, а його технологічна схема - в додатку П.

Основна апаратна складність спецпроцесора факторизації розраховується згідно схеми, представленої на рисунку 4.26, складається з суми апаратної складності його компонентів:

$A_5+A_6+A_7+A_8+A_9+A_{10}+A_{11}+A_{12}+A_{13}+A_{14}+A_{15}$.

Розрахунок виконано для 32-бітного спецпроцесора:

$A_5=(P_1+P_2+ P_3 +P_4 +P_5 +P_6 +P_7)2=(17 +19+23+25+27+29+31)2=257$ *v;*

$A_6= 4v$;

$A_7=(P_1+P_2+ P_3 +P_4 +P_5 +P_6 +P_7)/2=(17 +19+23+25+27+29+31)/2=86$ *v;*

$A_8=(P_1+P_2+ P_3 +P_4 +P_5 +P_6 +P_7)=171$;

$A_9=(P_1+P_2+ P_3 +P_4 +P_5 +P_6 +P_7)=171$;

$A_{10}=(P_1^2+P_2^2+P_3^2+P_4^2+P_5^2+P_6^2+P_7^2)+(P_1+P_2+P_3+P_4+P_5+P_6+P_7)= 4506$ *v;*

$A_{11}=(P_1+P_2+ P_3 +P_4 +P_5 +P_6 +P_7)\ 320=(17 +19+23+25+27+29+31)\ 320= 54720$ *v;*

$A_{12} = \mathrm{Log}_2(P_1+P_2+ P_3 +P_4 +P_5 +P_6 +P_7)=8$ *v;*

$A_{13}=64 \cdot A_7 \cdot 2= 11008$ *v;*

$A_{14}=(16+15+13+11+7) \cdot 2 \cdot 64= 7936$ *v.*

Блоки А1, А4, А15 є інтерфейсними і з'єднані з комп'ютером.

Розрахунок часової складності спецпроцесора факторизації за одну ітерацію роботи алгоритму визначається сумарним числом найбільшого числа послідовно з'єднаних модулів згідно виразу: $\tau_5+\tau_7+\tau_{10}+\tau_{11}+\tau_{12}$, де $\tau_5=2\upsilon$; $\tau_7=1\ \upsilon$; $\tau_{10}=2\upsilon$; $\tau_{11}=3\upsilon$; $\tau_{12}=2\upsilon$. Тобто число мікротактів виконання однієї ітерації факторизації числа розробленим процесором у базисі Хаара-Крестенсона не перевищує 10 υ. При організації пара-фазних виходів з матрично-модульного суматора 10 швидкодія модуля 11 може бути

реалізована за один мікротакт, тому тривалість одного циклу ітерацій буде становити 8υ. При швидкодії елементої бази ПЛІС υ=1 $_{нс}$ тривалість ітерації не перевищує 10 мікросекунд. За одну годину пристрій виконає приблизно 360000000000 ітерацій. При збільшенні числа розрядів процесора до 512-1024 швидкодія не змінюється за рахунок модульної арифметики. Таким чином, сумарна кількість вентилів 32-бітного спецпроцесора факторизації БРЧ складає 78867 *v*.

ВИСНОВКИ ДО РОЗДІЛУ 4

1. На базі C++ реалізовано основні компоненти, які відповідають теоретично розрахованим параметрам і підтверджують правильність та результативність запропонованого наукового підходу по вдосконаленню методів та алгоритмів опрацювання багаторозрядних інформаційних кодів. На основі розроблених теоретичних положень, результатів досліджень та моделювання, розроблено пакет прикладних програм, який переданий для реалізації на низових рівнях спеціалізованих РКС ВАТ ТКБР «Стріла» та «Інтеграл».

2. Розроблено та реалізовано апаратні компоненти виконання операцій піднесення до квадрату за модулем в ТЧБ Хаара-Крестенсона, що дозволяють за 2 мікротакти виконати піднесення до квадрату.

3. Розроблено та реалізовано алгоритм компактного кодування та генерування БРПЧ, що дозволяє на 75% відсотків зменшити використання пам’яті для зберігання послідовності простих чисел.

4. На основі запропонованих методів обчислень у базисах Радемахера - Крестенсона та алгоритмів опрацювання БРЧ розроблено пристрій компактного кодування та генерування БПЧ, який призводить до зменшення об’єму необхідної пам’яті на один-два порядки при зростанні розрядності генерованих простих чисел.

5. На основі запропонованих методів факторизації на основі ТЧБ Радемахера-Крестенсона розроблено спецпроцесор факторизації БРЧ, який характеризується високою швидкодією та може бути використаний при реалізації компонентів перетворення досконалої та модифікованої форм СЗК.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

У дисертаційній роботі розв’язано науково-практичну задачу розробки методів та обчислювальних засобів рішення задач теорії чисел у базисах Радемахера - Крестенсона:

1. Проаналізовані і досліджені архітектури спецпроцесорів та методи опрацювання багаторозрядних чисел для задач теорії чисел, способи кодування даних в комп’ютерних системах на основі різних теоретико - числових базисів та існуючі алгоритми факторизації. Обґрунтовано перспективи застосування різних теоретико - числових базисів при розв’язанні задач теорії чисел та виконана постановка завдання досліджень.

2. Отримано аналітичні вирази характеристик складності формування й опрацювання багаторозрядних чисел в задачах теорії чисел, перевірки на простоту, модулярного множення, визначення квадратичних лишків, які склали теоретичну основу спрощення часових характеристик компонентів методу факторизації та зменшення часової складності.

3. Розроблено метод компактного кодування масивів багаторозрядних простих чисел шляхом зберігання молодших двійкових розрядів та інкрементного нарощення кодів старших розрядів, який призводить до зменшення об’єму необхідної пам’яті на один-два порядки при зростанні розрядності генерованих простих чисел.

4. Розроблено метод факторизації багаторозрядних чисел шляхом вдосконалення та спрощення алгоритму Ферма та виконання модульних обчислювальних операцій у базисі Радемахера-Крестенсона, що дозволило, у порівнянні з відомими методами, зменшити обчислювальну складність з $n \cdot log_2^2 n$ до $n \cdot log_2 n$.

5. Розроблено метод визначення околу рішення задачі факторизації для багаторозрядних чисел врахуванням симетричності екстремумів залишкової функції, графічної ідентифікації моделей фракталів та матричного представлення добутків багаторозрядних чисел у базисі Радемахера, що дозволило підвищити швидкодію алгоритму більш, ніж в два рази.

6. Розроблено високопродуктивні методи визначення квадратичних лишків та векторно - модульного множення багаторозрядних чисел у базисі Радемахера - Крестенсона, які, порівняно з відомими, забезпечують підвищення швидкодії обчислень на один – два порядки.

7. Розроблено функціональні та структурні рішення пристроїв компактного кодування багаторозрядних простих чисел, квадратора та спецпроцесора факторизації багаторозрядних чисел у базисі Радемахера та Хаара - Крестенсона, які, в порівнянні з існуючими, забезпечують зменшення об’єму кодів пам’яті в 4-16 разів при розрядностях 32-128 та підвищують швидкодію квадратора та спецпроцесора факторизації на 2-3 порядки.

8. Розроблено програмне забезпечення комп’ютерного моделювання та рішення задач теорії чисел в базисі Радемахера - Крестенсона на основі застосування спеціалізованої бібліотеки А. Ленстра, об’єктно – орієнтованої мови С++, засобів VHDL для проектування на ПЛІС, що підтверджують збігання з теоретичними розрахунками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. А. С. №1282160. Многоканальное устройство для вычисления структурной функции / Я.М. Николайчук – Бюлетень №1.-1987.
2. А. С. СССР № 475619. Квадратор /Грибок Н.И., Обуханич Р.-А.В. - Бюллетень № 24.-1975.
3. А. С. СССР № 754414. Числоимпульсное множительное устройство / Я.М. Николайчук– Бюллетень № 29.- 1980.
4. А. С. СРСР №337784, кл. G 06 F 15/34. Бюлетень № 15. Опубліковано 05.05.1972.
5. А. С. СРСР №840924, кл. G 06 F 15/36. Бюлетень №23. Опубліковано 23.06.1981.
6. Айерлэнд К. Классическое введение в современную теорию чисел / К.Айерлэнд, М.Роузен. – М.: Мир,1987. – 416с.
7. Акушский И.Я. Машинная арифметика в остаточных класах / И.Я. Акушский, Д.И. Юдицкий – М: Сов.радио, 1968. – 440 с.
8. Аулов І. Ф. Порівняльний аналіз криптографічних бібліотек з відкритим кодом та рекомендації з їх використання / І.Ф. Аулов, Ю.І. Горбенко // Прикладная радиоэлектроника. - 2012. - Т. 11, № 2. - С. 220-224.
9. Архітектура системи Motorola Canopy [Електронний ресурс] – Режим доступу URL: http://rtm.ru/canopy.html. - Назва з титул. екрану.
10. Бекчанова Ш.Б. Синтез быстродействующих спецпроцессоров Хаара на основе матричной диаграммы / Ш.Б. Бекчанова, Х.Н. Зайнидинов – Тезисы докл. НТК «Молодёжь в развитии науки и техники». - Ташкент, 2002. – 78с.
11. Бекчанова Ш.Б. Алгоритмы и структуры на основе быстрых преобразований Хаара / Ш.Б. Бекчанова, Х.Н. Зайниддинов // Техника юлдузлари. – Ташкент, 2002. –№4. - С. 45-54.
12. Бекчанова Ш.Б. Принципы построения высокопроизводительных вычислительных структур / Ш.Б. Бекчанова, Х.Н. Зайнидинов // Тезисы докладов НТК «Мафкуравий жараёнлар ва Узбекистонда фанлар ривожининг

долзарб муаммолари», Андижон. - 2002. – С. 441.

13. Бернард С. Цифровая связь. Теоретические основы и практичиское приминение / С. Бернард. – М.: Издательский дом «Вильямс», 2003. – 1104 с.

14. Болтков А.П., Червяков Н.И., Хлевнов С.Н. Устройство для преобразования чисел из позиционной системы счисления в систему остаточных классов// А.С. СССР №1008729. Бюллетень №12. – 1983.

15. Білан СМ., Білан М.М., Білан А.М., Білан С.С. Генератор псевдовипадкових бітових послідовностей на основі клітинних автоматів.- Патент України на корисну модель № 93427, Бюл. № 18 від 25.09.14 р.

16. Бухштаб А.А. Теория чисел / А.А.Бухштаб. – М.: Просвещение, 1966. – 384с.

17. Виноградов И.М. Основы теории чисел / И.М. Виноградов. – М.: Наука, 1981. – 176с.

18. Грибанов Ю.И. Автоматические цифровые корреляторы./ Ю.И. Грибанов, Г.П. Веселова, В.Н Андреев. – М.: Энергия, 1971. – 240с.

19. Дивак М.П. Удосконалений метод допустимого оцінювання параметрів інтервальних динамічних моделей / М. П. Дивак, П. Г. Стахів, І. Я. Каліщук // Відбір та обробка інформації. – 2007. – Вип 26 (102) – С. 27–35.

20. Дывак Н.П. Структурная идентификация интервальных моделей статических систем / Н.П. Дывак, Манжула В.И. // Проблемы управления и информатики. – 2008. – №2. – С. 105–116.

21. Дудикевич В.Б. Число-імпульсні функціональні перетворювачі з імпульсними зворотними зв’язками: Монографія / В.Б. Дудикевич, В.М. Максимович, Л.В.Мороз / Львів: Видавництво Львівської політехніки, 2011. — 244 с.

22. Задірака В.К. Методи захисту фінансової інформації: Навчальний посібник / В.К. Задірака, О С.Олексюк. - Тернопіль: Збруч, 2000. - 460с.

23. Задірака В.К. Комп’ютерна арифметика багаторозрядних чисел: Наукове видання / В.К. Задірака, О.С. Олексюк. – Київ. –2003. – 264 с.

24. Задірака В.К. Комп’ютерна криптологія: Підручник

/ В.К. Задірака, О.С. Олексюк. – Київ, 2002. – 504 с.

25. Залмазон Л.А. Преобразование Фурье, Уолша, Хаара и их применение в управлении, связи и других областях/Л.А. Залмазон. –М.: Наука, 1989. – 496 с.

26. Задирака В. К. Облачные вычисления в криптографии и стеганографии / В. К. Задирака, А. М. Кудин // Кибернетика и системный анализ. - 2013. - Т. 49, № 4. - С. 113-119.

27. Зотов В.Ю. Проектирование встраиваемых микропроцессорных систем на основе ПЛИС фирмы XILINX / В.Ю. Зотов. - М: Горячая Линия. – Телеком, –2006. –522с.

28. Ишмухаметов Ш.Т. Методы факторизации натуральных чисел: учебное пособие / Ш.Т. Ишмухаметов.– Казань: Казан. ун. 2011.– 190 с.

29. Івасьєв С.В. Вдосконалений алгоритм пошуку символів Якобі / С.В. Івасьєв, І.З. Якименко, М.М. Касянчук // Оптико-електронні інформаційно-енергетичні технології. - Том 29, № 1. - 2015. – с. 45-50.

30. Івасьєв С.В. Збіжність екстремумів залишкової функції в околі розв’язку задачі факторизації/ С.В.Івасьєв, Я.М. Николайчук, І.З.Якименко, І.Р.Колісник // Вісник Хмельницького національного університету. Технічні науки. – 2015, №4. - С.157-164.

31. Івасьєв С.В. Матричний метод факторизації великорозрядних чисел / С.В. Івасьєв // Поступ в науку. Збірник праць Бучацького інституту менеджменту і аудиту – Бучач. – 2012. - №8. – С. 92-95.

32. Івасьєв С.В. Метод зберігання простих великорозрядних чисел у базисі Радемахера / С.В. Івасьєв, М.М. Касянчук, І.З. Якименко // Праці міжнародної молодіжної математичної школи “Питання оптимізації обчислень (ПОО-XXXVII)” Київ: Інститут кібернетики імені В.М. Глушкова НАН України, 2013. – С. 142-144.

33. Івасьєв С.В. Метод знаходження залишків велико-розрядних чисел в базисі Радемахера / С.В. Івасьєв, О.І. Волинський // Поступ в науку. Збірник наукових праць Бучацького інституту менеджменту і аудиту . – Бучач. – 2011. – №7. Т1. –С.88-91.

34. Івасьєв С.В. Метод організації компактної бібліотеки простих чисел великої розрядності / С.В. Івасьєв //Збірник матеріалів міжнародної наукової координаційної наради «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління» (ICSM) – Тернопіль, 2014. – С. 86-89.

35. Івасьєв С.В. Метод факторизації велико-розрядних чисел в базисі Радемахера / С.В. Івасьєв // Вісник національного університету "Львівська політехніка" "Комп'ютерні системи та мережі". – Львів. – 2012. - С. 118–126.

36. Івасьєв С.В. Метод факторизації чисел великої розрядності на основі ТЧБ Радемахера-Крестенсона / С.В. Івасьєв, І.З. Якименко, В.І. Назаров // Сучасні комп'ютерні інформаційні технології: Матеріали V Всеукраїнської школи-семінару молодих вчених і студентів. - 2015. – С.184.

37. Івасьєв С.В. Методи знаходження залишків велико-розрядних чисел Мерсена в базисі Радемахера / С.В. Івасьєв, В.І. Пашко // Матеріали II всеукраїнської науково-практичної конференції молодих учених і студентів «Інформаційні технології в освіті, техніці та промисловості». м. Івано-Франківськ. – 2015. - С.184-186.

38. Касянчук М.М. Аналітичний пошук модулів досконалої форми системи залишкових класів та їх застосування в китайській теоремі про залишки./ М.М. Касянчук, І.З. Якименко, І.Р. Паздрій, Я.М. Николайчук // Вісник Хмельницького національного університету. Технічні науки. - №1(221). – Хмельницький, 2015.– С. 170-176.

39. Касянчук М.М. Концепція теоретичних положень досконалої форми перетворення Крестенсона та його практичне застосування / М.М. Касянчук // Міжнародний науково-технічний журнал «Оптико-електронні інформаційно-енергетичні технології». - №2 (20). – 2010. – с.43-47.

40. Касянчук М.М. Теорія та оптимізація алгоритмів опрацювання великорозрядних чисел у базисі Крестенсона / М.М. Касянчук, І.З. Якименко, С.В. Івасьєв // Праці міжнародної молодіжної математичної школи "Питання оптимізації обчислень (ПОО-XXXVII)". - Київ: Інститут кібернетики імені

В.М. Глушкова НАН України, 2011. - С. 67-68.

41. Корнейчук В.И. Основы компьютерной арифметики / В.И. Корнейчук, В.П. Тарасенко. – К.: Вища школа. –2003. - С. 34-56.

42. Куприянов М. Коммуникационные контроллеры фирмы Motorola / М. Куприянов, О. Мартынов, Д. Панфилов. - СПб.: БХВ.-Петербург, 2001.- 560 с.

43. Лабунец В.Г. Теоретико-числовые преобразования над полями алгебраических чисел / В.Г. Лабунец. – Свердловск: УПИ, 1981. - С. 44-54.

44. М. Касянчук. Векторно-модульний метод множення багаторозрядних чисел в базисі Радемахера-Крестенсона// М. Касянчук, І.З. Якименко, Я.М. Николайчук, С.В. Івасьєв/ Матеріали Міжнародної конференції “Захист інформації і безпека інформаційних систем - 2014”. -5 – 6 червня 2014, Львів, Україна. – С. 53-54.

45. Касянчук М.М. Теоретичні основи аналітики та алгоритми оптимізації обчислень простих чисел./ М.М. Касянчук, І.З. Якименко, О.І. Волинський, С.В. Івасьєв // Поступ в науку. Збірник наукових праць Бучацького інституту менеджменту і аудиту // Матеріали Міжна-родної проблемно-наукової міжгалузевої конференції «Інформаційні проблеми комп’ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління (ПНМК-2010) ».-В.6, т.1.- С.33-36.

46. Касянчук М.М. Векторно-модульний метод модулярного множення / М.М. Касянчук, І.З. Якименко, Л.М. Тимошенко, Я.М. Николайчук, С.В. Івасьєв // Матеріали Міжнародної науково-практичної конференції «Сучасні інформаційні та електронні технології». - Одеса, 26 — 30 травня 2014 р. – С. 152-152.

47. Майоров С.А. Принципы организации цифровых машин / С.А. Майоров, Г.И. Новиков. – Л.: Машиностроение, –1974. – 306 с.

48. Малашевич Б. М. Неизвестные модулярные суперЭВМ / Б. М.Малашевич // PC WEEK/RE, 2005. - № 9. - С. 44-45.

49. Мельник А. О. Програмовані процесори обробки сигналів / А.О.Мельник. – Львів: Вид-тво Національного університету "Львівська

політехніка", 2000. –55 с.

50. Мельник А.О. Архітектура комп’ютера: Наукове видання / А.О. Мельник. – Луцьк: Волинська обласна друкарня, 2008. – 470с.

51. Мельник А.О. Спеціалізовані комп’ютерні системи реального часу / А.О. Мельник // – Львів: Державний університет “Львівська політехніка”, 1996. – 54 с.

52. Мандрона М.Н. Исследование статистических характеристик модифицированных генераторов Фибоначчи / М.Н.Мандрона, В.Н.Максымович // Проблемы управления и информатики. Международный научно-технический журнал. НАН Украины. – 2014. - № 6. — С. 83-87.

53. Николайчук Я. М. Ефективний метод модулярного множення в теоретико-числовому базисі Радемахера–Крестенсона / Я.М. Николайчук, М.М. Касянчук, І.З. Якименко, С.В. Івасьєв // Вісник Національного університету "Львівська політехніка". Комп’ютерні системи та мережі. - 2014. - № 806. - С. 195-199.

54. Николайчук Я.М. Дослідження системних характеристик двомірних кодів з особливими кореляційними властивостями / Я.М. Николайчук, О.М. Заставний // Вісник технологічного університету Поділля –Хмельницький, 2004. – №2. – С. 82-90.

55. Николайчук Я.М. Коди поля Галуа: теорія та застосування / Я.М. Николайчук // Монографія - Тернопіль: ТзОВ «Терн граф» - 2012. – 576с.

56. Николайчук Я.М. Метод збереження простих великорозрядних чисел у базисі Радемахера / Я.М. Николайчук, І.З. Якименко, М.М. Касянчук, С.В. Івасьєв // Праці міжнародної молодіжної математичної школи “Питання оптимізації обчислень (ПОО-XXXVII)”. Київ: Інститут кібернетики імені В.М. Глушкова НАН України. - 2015. –С. 159-161.

57. Николайчук Я.М. Метод факторизації багаторозрядних чисел та дослідження в околі розв'язання задач / Я.М. Николайчук, С.В. Івасьєв // Матеріали XIV Міжнародного наукового семінару “Сучасні проблеми інформатики в управлінні, економіці та освіті”, Київ – оз. Світязь, 29 червня

– 3 липня 2015 року. – С.83-88.

58. Николайчук Я.М. Методы цифровой обработки шумоподобных сигналов на основе кодовых систем / Я.М. Николайчук, Б.М. Шевчук – Киев, Сб. тр. ИКАН УССР, 1988.

59. Николайчук Я.М. Проблеми реорганізації структури процесорів у різних теоретико-числових базисах / Я.М. Николайчук // Збірник матеріалів міжнародної наукової координаційної наради «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління» (ICSM-2014). - Тернопіль,2014.- С.110-114.

60. Николайчук Я.М. Проектування спеціалізованих ком'ютерних систем / Я.М. Николайчук, Н.Я. Возна, І.Р. Пітух. - Т.: Терно-граф, 2010. - 392 с.

61. Николайчук Я.М. Теоретико-числові базиси Крестенсона та Галуа – фундаментальна основа оптимізації опрацювання велико розрядних чисел / Я.М. Николайчук // Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2011. - №7.-С.114-122.

62. Николайчук Я.М. Теорія джерел інформації / Я.М. Николайчук – Тернопіль: ТзОВ „Терно–граф", 2010. – 536 с.

63. Николайчук Я.М. Теорія цифрових перетворень мультибазисного супершвидкодіючого процесора /Я.М. Николайчук// Искусственный интеллект. – 2008. – №4. – С. 387-394.

64. Николайчук Я.М. Фундаментальні засади теорії факторизації багаторозрядних чисел на основі фракталів зображень в околі рішення / Я.М. Николайчук, С.В. Івасьєв // Збірник матеріалів міжнародної наукової координаційної наради «Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління»(ICSM). – Тернопіль, 2014. – С. 116-120.

65. Николайчук Я.М. Фундаментальні основи теорії чисел та оптимізації обчислень в базисах Крестенсона і Галуа / Я.М. Николайчук // Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2010 - №6.-С.20-22.

66. Николайчук Я.М. Просторові моделі та процесори перетворення інформації у базисі Крестенсона та Галуа / Я.М.Николайчук, Г.Я. Ширмовський, А.М. Лазорів // Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2011 - №7.-С.184-186.

67. Николайчук Я.М., Албанський І.Б. Багатоканальний цифровий корелятор // Патент на корисну модель № 73320.– 25.09.2012р.

68. Николайчук Я.М. Спецпроцесори кореляційного опрацювання даних у ТЧБ Радемахера з порозрядним сумуванням результатів обчислень / Я.М.Николайчук, І.Б. Албанський, В.В. Кирилюк // Сучасні комп'ютерні інформаційні технології: Матеріали ІІІ Всеукраїнської школи-семінару молодих вчених і студентів АСІТ'2013.- Тернопіль: ТНЕУ, 2013.- с.105-106.

69. Николайчук Я.М. Розробка кореляційних спецпроцесорів у базисі Хаара-Крестенсона / Я.М.Николайчук, І.Б.Албанський, О.І. Волинський // Збірник наукових праць. Хмельницький: Тріада-М. – 2008. - С.270.

70. Николайчук Я.М., Албанський І.Б., Волинський О.І. Цифровий автокорелятор // Патент на корисну модель № 76622.– 10.01.2013р.

71. Николайчук Я.М., Волинський О.І. Пристрій для перетворення чисел з позиційної системи в систему залишкових класів // Патент на корисну модель № 76623 МПК G06F5/02 Опубл. 10.01.2013 Бюл.№1.

72. Николайчук Я.М., Волинський О.І. Спосіб визначення залишку двійкового числа // Патент на корисну модель № 74576.– 12.11.2012р.

73. Николайчук Я.М. Теоретичні основи побудови та структура спец процесорів в базисі Крестенсона / Я.М.Николайчук, О.І.Волинський, С.В.Кулина // Вісник Хмельницького національного університету. - Хмельницький.- 2007.- №3.- Т1.- С.85-90.

74. Николайчук Я.М., Воронич А.Р., Погонець І.О. Пристрій для визначення автокореляційної міри ентропії // Патент на корисну модель № 68044.– 12.03.2012р.

75. Николайчук Я.М., Воронич А.Р., Погонець І.О. Пристрій для визначення автокореляційної міри ентропії // Патент на корисну модель № 58743.– 26.04.2011р.

76. Николайчук Я.М. Дослідження архітектури комп'ютерів: принципи побудови процесорів на основі вертикально-інформаційної технології / Я.М. Николайчук, П.В. Гуменний // Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2009.- №5. - С.69-74.

77. Николайчук Я.М. Реалізація операції додавання на основі вертикально-інформаційної технології у базисі Галуа / Я.М. Николайчук, П.В. Гуменний // Збірник наукових праць. Хмельницький: Тріада-М. – 2008. - С.274.

78. Николайчук Я.Н. Теоретические основы, методы и процесоры преобразования информации в кодах поля Галуа на базе вертикально-информационной технологии / Я.М. Николайчук, П.В. Гуменный // Кибернетика и системный анализ. - Международный научно-теоретический журнал института кибернетики им. В.М. Глушкова НАН Украины.- Том 50. - №3. - 2014.- с.17-26.

79. Николайчук Я.М. Структура та функції кореляційного спецпроцесора для ідентифікації та реєстрації гармонічних сигналів / Я.М.Николайчук, Т.О.Заведюк // Праці міжнародної наукової конференції "Питання оптимізації обчислень (ПОО-XL)" Київ: Інститут кібернетики імені В.М. Глушкова НАН України, 2013. - С.190-191.

80. Николайчук Я.М., Заведюк Т.О., Воронич А.Р., Албанський І.Б. Багатоканальний пристрій для обчислення знакової функції // Патент на корисну модель № 70338.– 11.06.2012р.

81. Николайчук Я.М. Теоретичні основи та високопродуктивний алгоритм обчислення мультистепеневої функції в базисі Крестенсона / Я.М. Николайчук, В.К. Задірака, М.М. Касянчук // Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2010.- №6.-С.30-32.

82. Николайчук Я.М. Теоретичні засади та принципи побудови арифметико-логічного пристрою на основі вертикально-інформаційної технології / Я.М. Николайчук, О.М. Заставний, П.В. Гуменний // Вісник Хмельницького національного університету.- Хмельницький, 2012.- №2(187).-С.190-196.

83. Николайчук Я.М. Теоретичні основи виконання модулярних операцій множення та експоненціювання в теоретико-числовому базисі Крестенсона-Радемахера / Я.М. Николайчук, М.М. Касянчук, І.З. Якименко, Т.М. Долинюк // Науковий журнал «Інформатика та математичні методи в моделюванні» - №2. – 2011. – с. 123–130.

84. Николайчук Я.М. Теорія алгоритмів пошуку найбільшого спільного дільника у базисі Крестенсона / Я.М. Николайчук, М.М. Касянчук, І.З. Якименко // Науковий журнал «Вісник ТНТУ ім.. І.Пулюя».- Тернопіль.- 2011.- Т. 16. - №1– С.154-161.

85. Николайчук Я.М. Теорія алгоритмів перетворень китайської теореми про залишки в матрично-розмежованому базисі Радемахера-Крестенсона / Я.М. Николайчук, М.М. Касянчук, І.З. Якименко // Вісник НУ «Львівська політехніка» Комп'ютерні системи та мережі.- Львів.-2010.- №688.- С.118-124.

86. Николайчук Я.М. Теоретические основы аналитического вычисления коэффициентов базисных чисел преобразования Крестенсона / Я.М. Николайчук, М.Н.Касянчук, И.З. Якименко // Кибернетика и системный анализ. — 2014. — Том 50, № 5. — С. 3–8.

87. Николайчук Я.М. Теоретико-числові басиси та їх застосування для побудови багаторозрядних процесорів для шифрування інформації / Я.М.Николайчук, Б.Б. Круліковський // Стратегічні рішення інформаційного розвитку економіки, суспільства та бізнесу: тези доповідей ІІІ Міжнародної науково-практичної конференції науковців.-Рівне: НУВГП, 2014р. – С.72-73.

88. Николайчук Я.М. Теоретичні основи та критерії оцінки структурної складності обчислювальних компонентів процесорів багаторозрядної арифметики / Я.М. Николайчук, Б.Б. Круліковський, Н.Я. Возна // Стратегічні рішення інформаційного розвитку економіки, суспільства та бізнесу: тези доповідей ІІІ Міжнародної науково-практичної конференції науковців.-Рівне: НУВГП, 2014р. – С.65-67.

89. Николайчук Я.М. Теорія та принципи побудови спец процесора на основі базисів Радемахера, Крестенсона, Галуа / Я.М. Николайчук,

Н.Д. Круцкевич, О.М.Заставний, Р.І. Король // Контроль і управління в складних системах.(КУСС-2003). Тези доповідей сьомої міжнародної науково-технічної конференції. м. Вінниця, 8-11жовтня 2003.- Вінниця:"УНІВЕРСУМ-Вінниця", 2003.-с.70.

90. Николайчук Я.М. Матричні системи числення / Я.М. Николайчук, Н.Д. Круцкевич // Вісник Хмельницького національного університету.- Хмельницький.- 2007.- №3.- Т1.- С.62-64.

91. Николайчук Я.М. Метод та алгоритм множення у двовимірній системі числення матричного Радемахера / Я.М. Николайчук, О.Д. Круцкевич, Н.Д. Круцкевич, О.М. Заставний // Міжнародний науково-технічний журнал "Оптико-електронні інформаційно-енергетичні технології".- 2010.- №1(19).- С.80-83.

92. Николайчук Я.М. Теорія та техніка високопродуктивних мультибазисних процесорів / Я.М. Николайчук, О.Д. Круцкевич, С.В. Кулина, О.І. Волинський // Праці міжнародного симпозіуму "Питання оптимізації обчислень (ПОО-XXXV)". – Київ: Інститут кібернетики імені В.М.Глушкова НАН України, 2009. – Т.2. – С.165-169.

93. Николайчук Я.М. Теорія побудови та компоненти швидкодіючих процесорів на основі досконалої та розмежованої форм системи залишкових класів / Я.М. Николайчук, С.В.Кулина, О.І.Волинський // Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2008. - №4. Т1 С.31-35.

94. Николайчук Я.М. Коди поля Галуа та їх застосування в перетворювачах форм інформації / Я.М.Николайчук, Я.Б. Кусик // Тезисы докладов 7-го симпозиума Проблемы создания преобразователей формы информации - Киев: ИКАН Украины. - 1992.

95. Николайчук Я.М., Піх В.Я., Кімак В.Л., Круліковський Б.Б. Пристрій для обчислення спектрального косинусного перетворення в залишкових класах // Патент на корисну модель № 03633 – 17.04.2015 р.

96. Николайчук Я.М. Теоретичні засади побудови процесорів спектрального аналізу сигналів у різних теоретико-числових базисах /

Я.М. Николайчук, В.Я. Піх, В.С. Павлюкович // Збірник наукових праць Бучацького інституту менеджменту і аудиту. – Бучач. – 2011 - №7.- С.123-127.

97. Николайчук Я.М. Теорія та процесори визначення інформаційної міри ентропії на основі кореляційних функцій / Я.М. Николайчук, І.О. Погонець, А.Р. Воронич, І.Б. Албанський // Науковий вісник Чернівецького університету Комп'ютерні системи та компоненти. - 2011.- Т2.- №2.-С.37-44.

98. Николайчук Я.М. Дослідження алгоритмів захисту інформації в комп'ютерних системах / Я.М. Николайчук, І.З. Якименко, Л.О. Дубчак // Збірник матеріалів проблемно-наукової міжгалузевої конференції «Юриспруденція та проблеми інформаційного суспільства» (ЮПІС-2011)- Ів.Франківськ, 2011.- С.140-143.

99. Новиков Л.Г. Счётчики импульсов с коэффициентами счёта, управляемыми с помощью двоичного кода / Л.Г. Новиков, И.Т. Шурыгин // Приборы и системы управления. - № 6, 1972. –С.30-31.

100. Орнатский П.П. Теоретические основы информационно-измерительной техники / П.П. Орнатский. – К.: Вища школа. – 1983. – 455с.

101. Офіційний сайт.Intel microcontrollers [Електронний ресурс] – Режим доступу: URL: http://www. Intel.com/design/embrocontrol/index.htm.

102. Палагин А.В. Реконфигурируемые структуры на ПЛИС / А.В. Палагин, В.Н. Опанасенко, В.Г. Сахарин // УсиМ. – 2000. – № 3. – С. 33-43.

103. Палагин А.В. Опыт разработки микропроцессорных распределенных систем реального времени./ А.В. Палагин, Я.Н. Николайчук// – Киев: Знание, – 1988. – 19 с.

104. А. С. №13726221 СССР, МКИ Н03 М1/38. "Аналого-цифровой преобразователь"/Я.Н. Николайчук. – Опубл. 07.02.88, Бюл. №5.

105. А. С. №1462477 СССР, МКИ Н03 М1/38. "Аналого-цифровой преобразователь"/ Я.Н. Николайчук – Опубл. 28.02.89, Бюл. №8.

106. Петришин Л.Б. Николайчук Я.Н., Ищеряков С.М., Цифровая

обработка сигналов на основе преобразования кодов поля Галуа / Л.Б. Петришин, Я.Н. Николайчук, С.М. Ищеряков // Методы и микроэлектронные средства цифровой обработки и преобразования сигналов.- Рига: ИЭВТ АН Латвии. - 1989.- С.130 - 132.

107. Петришин Л.Б. Теоретичні основи перетворення форми та цифрової обробки інформації в базисі Галуа / Л.Б. Петришин // - Київ.: ІЗіМН МОУ, - 1997. - 237 с.

108. Петришин Л.Б. Теоретичні основи перетворення форми та цифрової обробки інформації в базисі Галуа: Навч. посібник / Л.Б. Петришин // - Київ.: ІЗіМН МОУ, 1997. - 237 с.

109. Пилипенко І.А. Дослідження методів стиснення інформації. / І.А. Пилипенко // Науковий вісник інституту менеджменту та економіки «Галицька академія» – 2006. – №2(10). – С. 78-82.

110. Романец Ю.В. Защита информации в компьютерных системах и сетях / Под ред. В.Ф.Шаньгина. / Романец Ю.В., Тимофеев П.А., Шаньгин В.П. – М.: Радио и связь, 1999. – 328с.

111. Ивасьев С.В. Метод факторизации многоразрядных чисел на основе свойств квадратичности вычетов в системе остаточных классов / С.В. Ивасьев, Я.Н. Николайчук, И.З. Якименко, М.Н. Касянчук // Вестник Брестского государственного технического университета. – 2015. – № 5(95): Физика, математика, информатика. – С. 45–45.

112. Садыхов Р.Х. Методы и средства обработки сигналов в дискретных базисах / Р.Х. Садыхов, П.М. Чеголин, В.П. Шмерко. – Мн.: Наука и техника, 1987. - 296 с.

113. Самофалов К.Г. Цифровые ЭВМ / К.Г. Самофалов, В.И. Корнейчук, В.П. Тарасенко// – СПб.: Вища школа – К. – 2000. – 528с.

114. Синьков М.В. Непозиционные представления в многомерных числовых системах / М.В. Синьков, Н.М. Губарени - Киев: Наукова думка, 1979. - 137 с.

115. Стешенко В.Б ПЛИС фирмы «Altera» / В.Б. Стешенко // М., «Додека». –2002. –575с.

116. Столлингс В. Передача данных / В. Столлингс – 4-е изд. – СПб.: Питер, 2004. – 750 с.

117. Столлингс В. Современные компьютерные сети / В. Столлингс – СПб.: Питер, 2003. – 783 с.

118. Таненбаум Э. Компьютерные сети / Э. Таненбаум – 4-е изд. – Питер, 2003. – 992 с.

119. Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС XILINX с применением языка VHDL / И.Е. Тарасов // М.: Горячая линия. – Телеком, –2005. – 782с.

120. Тимошенко Л.М. Алгоритми факторизащї для криптоаналізу асиметричних криптосистем / Л.М. Тимошенко, К.В. Вербик, Я.М. Николайчук, С.В. Івасьєв // Інформатика та математичні методи в моделюванні. – Одеса 2014.- № 4(4). – С. 342-349.

121. Тимошенко Л.М. Удосконалений метод Ферма факторизації чисел / Л.М. Тимошенко, К.В. Вербик, С.В. Івасьєв // Зб.мат. пробл-наук. міжн. конф. ПНМК-2014. Львів.- 2014. - С.348-350.

122. Тимошенко Л.М. Удосконалення алгоритму факторизації для криптографічних систем захисту інформації / Л.М. Тимошенко, К.В. Вербик, С.В. Івасьєв // Сучасна спеціальна техніка. – 2014. –№ 3(38). – С 56-59.

123. Кох Х. Алгебраическая теория чисел / Х. Кох. — М.: ВИНИТИ, 1990. - 301 с.

124. Хетагуров Я.А., Руднев Ю.П. Повышение надежности цифровых устройств методами избыточного кодирования / Я.А. Хетагуров, Ю.П. Руднев. - М.: Энергия, 1974. – 272с.

125. Червяков Н.И. Нейрокомпьютеры в остаточных класах / Н.И. Червяков, П.А. Сахнюк, А.В. Шапошников, А.Н. Макоха - М.: Радиотехника, 2003. - 272 с.

126. Червяков Н.И. Преобразователь десятичного кода в код системы остаточных классов// А.С. СССР №374595 Бюллетень №15. – 1973.

127. Червяков Н.И. Устройство для преобразования чисел из десятичной системы счисления в систему остаточных классов// А.С. СССР

№377767 Бюллетень №18. – 1973.

128. Шевчук Б. М. Технологія багатофункціональної обробки і передачі інформації в моніторингових мережах. / Б.М. Шевчук, В.К. Задірака, Л.О. Гнатів, С.В. Фраєр // НАН України, Ін-т кібернетики ім. В. М. Глушкова. – К. : Наук. Думка, 2010. – 371 с.

129. Ширмовська Н.Г. Оптимальна дискретизація заданих кореляційною функцією сигналів / Н.Г. Ширмовська // Методи та прилади контролю якості.- Івано-Франківськ.- 2009.- №22.- С.107-111.

130. Манин Ю.И. Введение в теорию чисел / Ю. И. Манин, А.А. Панчишкин. — М.: ВИНИТИ, 1990. — Т. 49. — 341 с.

131. Яцків Н.Г. Методи стиснення в багатоканальних системах на основі кодів Галуа / Н.Г. Яцків, Я.М. Николайчук // Вісник національного університету «Львівська політехніка». Радіоелектроніка та телекомунікації. – 2002. - №443. – С. 135-138.

132. Яцків Н.Г., Спецпроцесори обробки даних на основі перетворення Крестенсона – Галуа. / Н.Г. Яцків, Р.І. Король, В.В. Яцків, Т.Г. Федчишин // Вісник Технологічного університету Поділля. – 2003. -ТІ, №3. – С. 105-108.

133. Siewobr H. Application of Residue Number System to Advance Encryption Standard Algorithm / H. Siewobr, K. Gbolagade // International Journal of Computational Intelligence and Information Security. - Vol. 2, No. 7, July, 2011. - P. 66-72.

134. Mahyar H. Reliable and High-Speed KASUMI Block Cipher by Residue Number System Code / H. Mahyar // World Applied Sciences Journal. - Vol.17 (9). – 2012. - P. 1149-1158.

135. Teli I. Residue number systems in optical computing / I. Teli // International Journal of Electronics and Communication Enginnering. - Vol. 2, №1. – 2012. - P. 27-29.

136. Gbolagade K. A. Residue-to-Decimal Converters for Moduli Sets with Common Factors / K. A. Gbolagade, S. D. Cotofana // Proceedings of 52nd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS-2009). -

Cancun, Mexico. – 2009. - p. 624-627.

137. Gbolagade K.A. A Memoryless MRC Technique for RNS-to-Binary Conversion Using The Moduli Set (2^n,2^n-1, 2^{n-1}-1) / K.A. Gbolagade // International Journal of Soft Computing. - Vol. 4(3). - 2009. - P. 127-130.

138. Gbolagade K.A. MRC Technique for RNS to Decimal Conversion Using the Moduli Set {2^n +2, 2^n + 1, 2^n} / K.A.Gbolagade, S. D. Cotofana // Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing. -Veldhoven, The Netherlands. – 2008. - P. 318-321.

139. Gbolagade K.A. Residue Number System Operands to Decimal Conversion for 3-Moduli Sets, / K.A.Gbolagade, S. D. Cotofana // Proceedings of 51st IEEE Midwest Symposium on Circuits and Systems (MWSCAS-08). - Knoxville, USA. – 2008. - P. 791-794.

140. Kozaczko D. Vector Module Exponential in the Remaining Classes System/ D.Kozaczko, I.Yakymenko, M. Kasianchuk, S.Ivasiev // Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS–2015) – Warsaw, Poland. – V.1. – September, 2015. – P.161–163.

141. Yang L. L. A residue number system based parallel communication scheme using orthogonal signaling: Part II—Multipath fading channels / L. L. Yang, L. Hanzo // IEEE Trans. Veh. Technol. – 2002. - Vol. 51. – P. 1541-1553.

142. Yang L. L. Minimum-distance decoding of redundant residue number system codes / L. L. Yang, L. Hanzo // Proc. IEEE ICC '2001. – Helsinki (Finland) – 2001. - P. 2975-2979.

143. Yang L. L. Performance analysis of coded M-array orthogonal signaling using errors-and-erasures decoding over frequency-selective fading channels / L. L. Yang, L. Hanzo // IEEE J. Select. Areas Community. – 2001.- Vol. 19. - P. 211-221.

144. Lakhani G. Some Fast Residual Arithmetic Adders / G. Lakhani // International Journal of Electronics. - 1994. - P. 225-240.

145. Lenstra H. W. Divisors in residue classes / H. W. Lenstra // Math. Comput.- 1984. -Vol. 42. - N 165. - P.331-340.

146. R. T. Gregory and D. W. Matula, "Base Conversion in Residue Number System", Third Symp. Computational Arithmetic, pp. 117-125, 1975.

147. Abdallah M. On MultiModuli residue number systems with moduli of forms ra, rb-1, rc+1 / M. Abdallah and A. Skavantzos // IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-I: REGULAR PAPERS, VOL. 52, NO. 7 , 2005.

148. Hosseinzadeh M. Design Residue Number System Circuits in Current mode / M. Hosseinzadeh, K. Navi and S. Timarchi // 14th Iranian Conference of Electrical Engineering, 2006. – P. 234 – 236/

149. Hosseinzadeh M. New Design of 4-3 Compressor / M. Hosseinzadeh, K. Navi and S. Timarchi // 11th International CSI Computer Conference of Iran, 2006. – P. 101-103.

150. Hosseinzadeh M. A. Novel Multiple Valued Logic OHRNS Modulo rn Adder Circuit," / M. A. Hosseinzadeh, S. J. Jassbi and k. Navi // International Journal of Electronics, Circuits and Systems. - Vol. 1, No. 4. – 2007. - P. 245-249.

151. Kasianchuk M. Efficient methods for modular multiplication through the use of Rademacher - Krestenson TNB/ M. Kasianchuk, I. Yakymenko, Ya. Nykolaychuk, S. Ivasiev// Proceedingsofthe XI–th International Conference "Modern Problems of RadioEngineering, Telecommunications and ComputerScience" (TCSET–2014).–L'viv–Slavske.– 2014. – P.93-94.

152. Kasyanchuk M. Matrix Algorithms of Processing of the Information Flow in Computer Systems Based on Theoretical and Numerical Krestenson's Basis / M. Kasyanchuk, I. Yakymenko, Ya. Nykolaychuk // Proceedings of the X–th International Conference "Modern Problems of Radio Engineering, Telecommunications and Computer Science" (TCSET–2010).–L'viv–Slavske.– 2010. – P.241.

153. Montgomery P. L. Modular multiplication without trial division / P. L. Montgomery// Math. Computation. - Vol. 44. - 1985. - P. 519 – 521.

154. A. Huang, "The Implementation of a Residue Arithmetic Unit via Optical and Other Physical Devices", Proc. Int\'l Optical Computing Conf., P. 14-18, 1975

155. Kasyanchuk M. Fundamental Backgrounds of the Discrete Logarithms

Theory in the Rademacher-Krestensons Basis/ M.Kasyanchuk, S. Ivasiev, I. Pazdriy, R. Trembach, I. Yakymenko // Proceedings of the XI-th International conference “Modern Problems of Radio Engineering, Telecommunications and Computer Science” (TCSET-2012). – Lviv-Slavsk. – P.93

156. Szabo N. Residue arithmetic and its applications to computer technology / N. Szabo and R. Tanaka. - New York: McGraw Hill. – 1967. - 236 p.

157. H. L. Garner, "The Residue Number System", IRE Trans. Electronic Computers, vol. EL-8, no. 6, P. 140-147, 1959.

158. Omondi B. Residue Number Systems: Theory and Implementation / B.Omondi, Premkumar S. - World Scientific Publishing Co. Pte. Ltd. - 2007. - 296 p.

159. Patterson D. Computer Architecture. A Quantitative Approach / D. Patterson, J. Hennessy // Morgan Kaufmann Publishers, Inc. –1996.

160. Mohammad S. Improved One-hot 2^n -modulo Multiplier in RNS / S. Mohammad, H. Toofani, M. Mahmoodi, M. Mohammadizadeh // Journal of Automation & Systems Engineering. – Vol. 5, №4. - 2011. – P. 160-164.

161. Moharrami S. The Application of the Residue Number System in Digital Image Processing: Propose a Scheme of Filtering in Spatial Domain. Research / S. Moharrami, D. Kheirandish // Journal of Applied Sciences. – Vol. 7. – 2012. – P. 286-292.

162. Stallings W. Computer Organization and Architecture / W. Stallings// 5^{th} ed., New York, NY: Macmillan Publishing Company Stallings. –2000. – 329 p.

163. Belan S. Use of Cellular Automata to Create an Artificial System of Image Classification and Recognition / S. Belan, N. Belan. - Springer-Verlag Berlin Heidelberg. – 2012. - 493 p.

164. Bilan S. Models and hardware implementation of methods of Pre-processing Images based on the Cellular Automata / Stepan Bilan // Advances in Image and Video Processing. - Vol 2, No 5. – 2014. - P. 76-90.

165. Tomczak T. Hierarchical residue number systems with small moduli and simple converters / T. Tomczak // International Journal of Applied Mathematics and Computer Science. - Volume 21, Issue 1. – 2011. – P. 173–192.

166. Tanenbaum A. Structured Computer Organization / A. Tanenbaum//

4th ed. Upper Saddle River, NJ: Prentice Hall –1999.

167. Houk T. L. Method And Apparatus For Pipelined Detection Of Overflow In ResidueArithmetic Multiplication / T. L. Houk. - Boeing Company, Seattle, Wash. Appl. - №4. - 1990. – 234 p.

168. Theodore L. Residue Addition Overflow Detection Processor / L.Theodore. - Boeing Company, Seattle, Wash.Appl. - №4. - 1989. – 334 p.

169. Chren W. A. Delay Power Product Simulation Results For One-Hot Residue Number System Arithmetic Circuits / W. A. Chren, Jr., C. H. Brogdon and D. Andrevska // IEEE. - 1997. - P. 544-547.

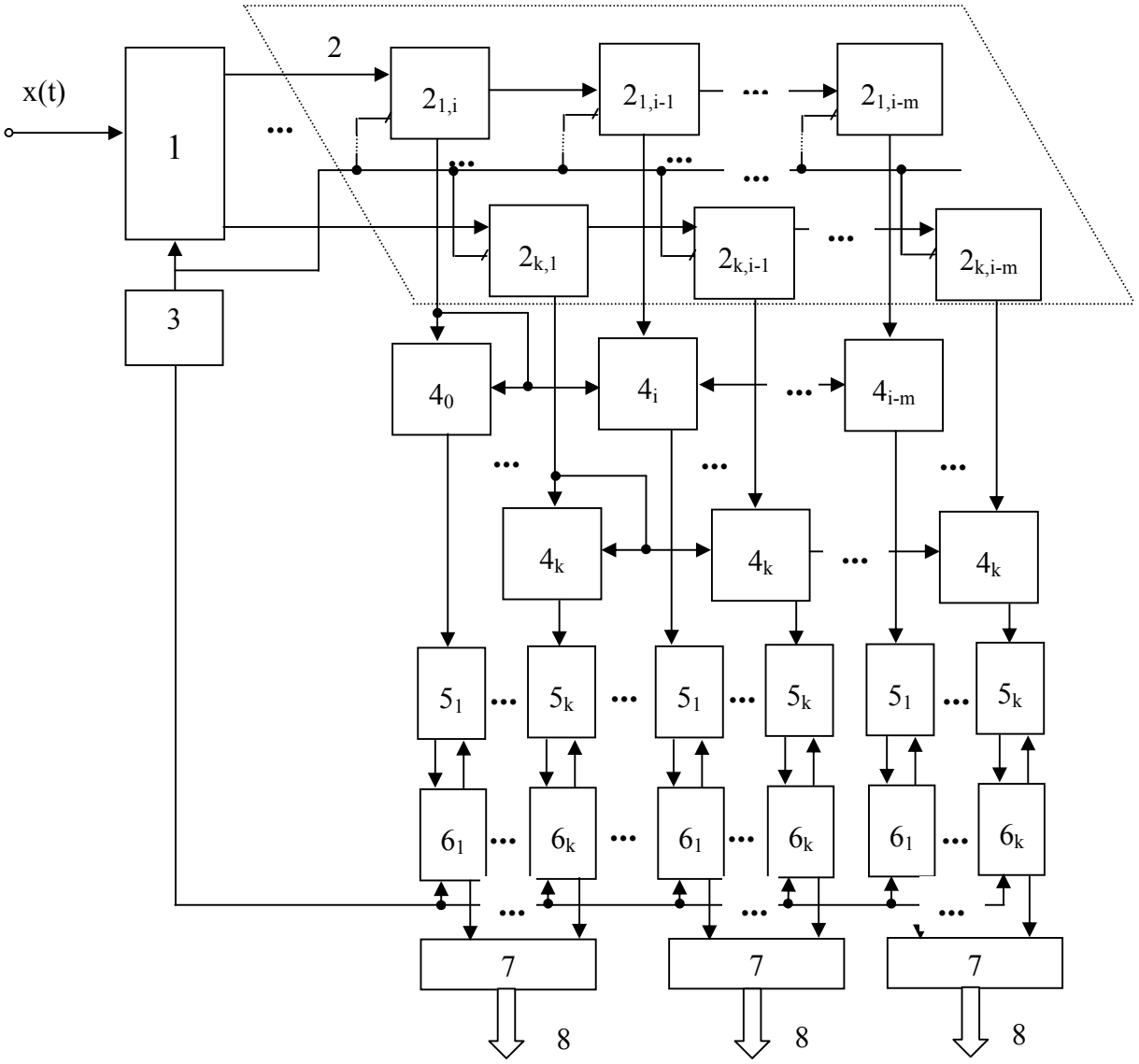
170. Nykolaychuk Ya. Fundamental theoretical and algorithmic principles of the appli-ed tasks decision of theory of numbers and construction of the high-performance spe-cial processors on their basis/ Ya. Nykolaychuk, S. Ivasiev, I. Yakymenko, M. Kasjanchuk // XI International Conference “The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM-2011)”, 23-25 February, 2011, Polyana-Svalyava (Zakarpattya), Ukraine. – P. 168-169.

171. Nykolaychuk Ya. Effective Method of Modular Multiplication in the Theoretical and Numerical Basis of Rademacher-Krestenson’s / Ya. Nykolaychuk, M. Kasianchuk, I. Yakymenko, S. Ivasev // Proceedings of the International Conference TCSET’2014. — Lviv, 2014. — P. 369.

172. Nykolaychuk Ya. Rademacher-Krestenson’s method of between-bases transformations in designing processors / Ya. Nykolaychuk, O. Volynskyy, A. Borovyi // Proceedings of the 6th International Conference “Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications”. – Prague, Czech Republic. –2011. – P. 310-313.

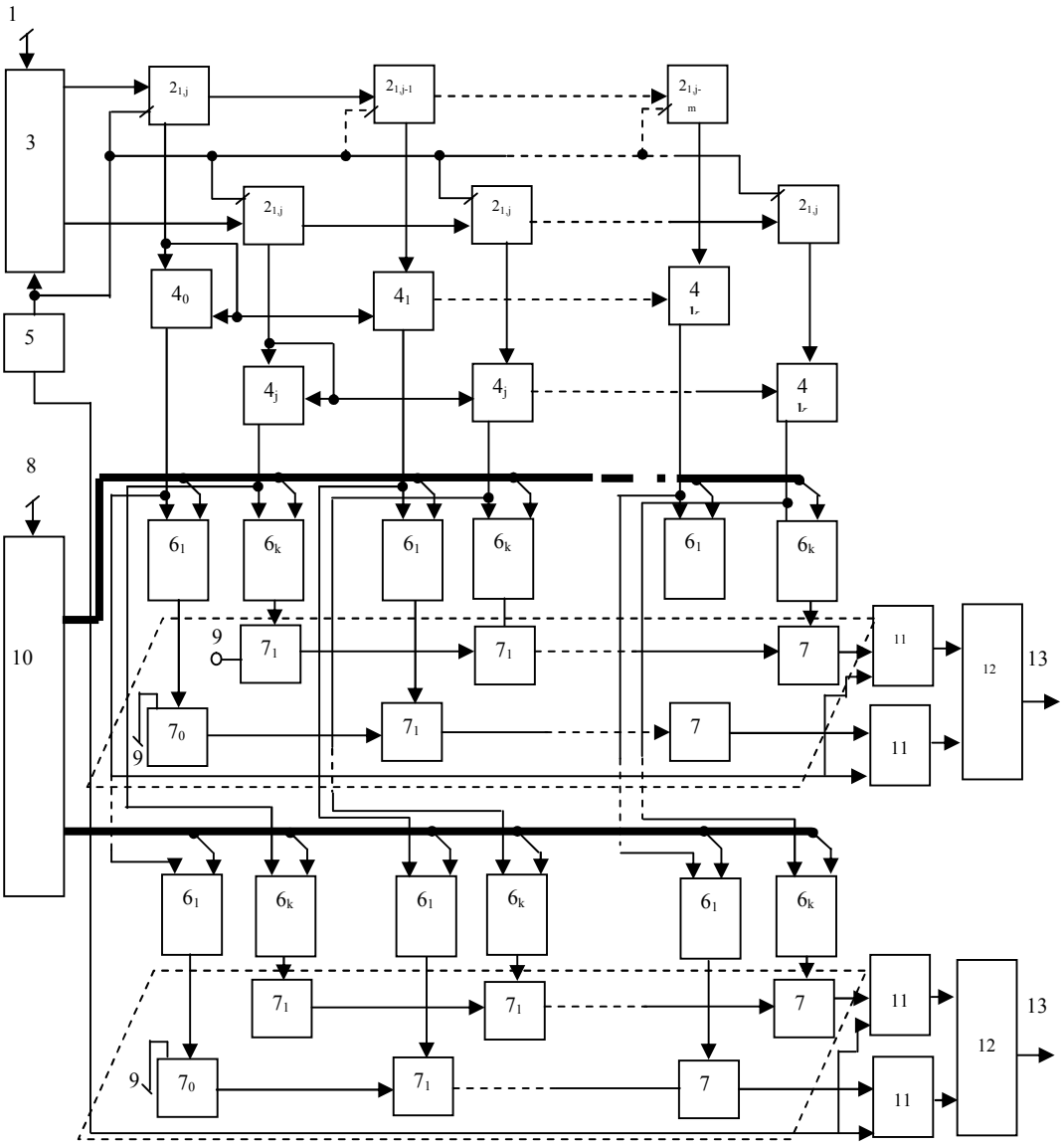
Додаток А

Структура цифрового автокорелятора



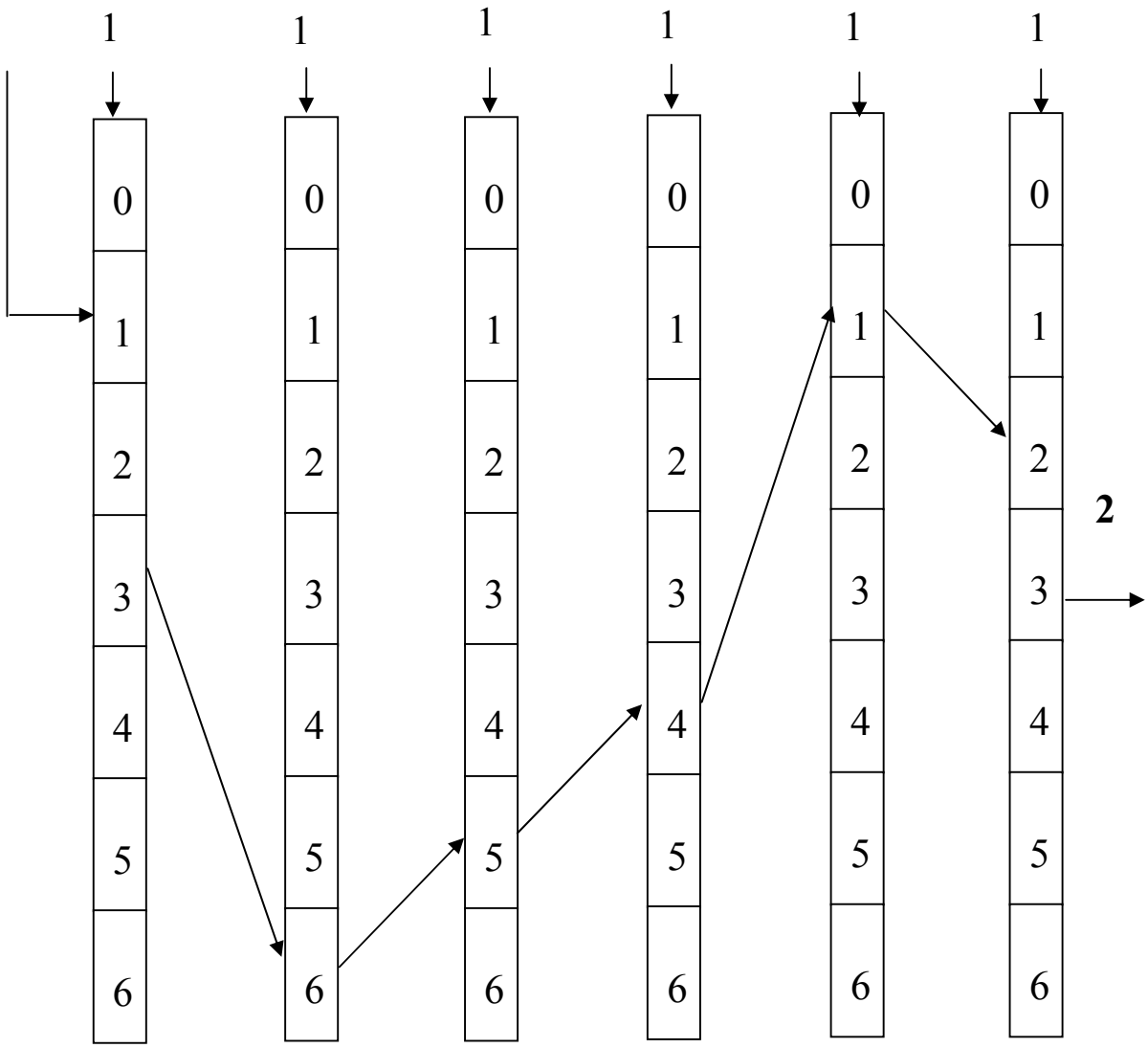
Додаток Б

Схема пристрою спектрального – косинусного перетворення



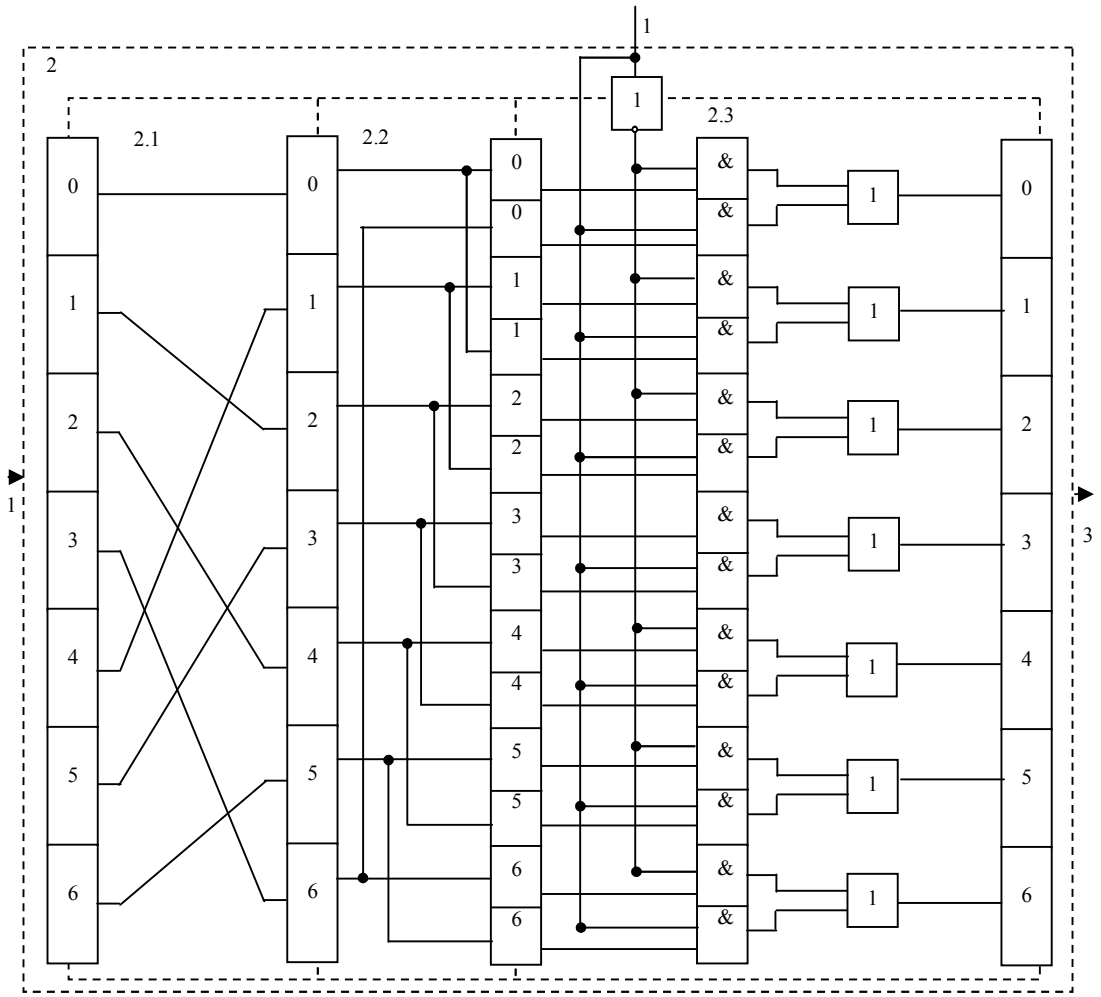
Додаток В

Функціональна схема пристрою перетворення чисел з позиційної системи в систему залишкових класів



Додаток Г

Структурна схема однорозрядного рандомізатора – мультиплексора



Додаток Д

Теоретичні засади пошуку символів Якобі та Лежандра

Нехай p – просте, a – ціле число. Символ Лежандра $\left(\frac{a}{p}\right)$ визначається так:

$$\left(\frac{a}{p}\right) = \begin{cases} 0, \text{якщо } p \text{ ділиться на } a; \\ 1, \text{якщо } a \in Q_p; \\ -1, \text{якщо } a \in \overline{Q_p}. \end{cases}$$

Згідно теореми доведеної в роботі [24] число a, яке не ділиться на непарне просте p, є квадратичним лишком за модулем p тоді і тільки тоді, коли $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$, і квадратичним нелишком тоді і тільки тоді коли $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$.

За теоремою Ферма [25, 27] та запропонованого в роботі вдосконаленого алгоритму $a^{p-1} \equiv 1 \pmod{p}$ при НСД(a, p) = 1 та НСД(2, p) = 1. Або:

$$\left(a^{\frac{p-1}{2}} + 1\right) * \left(a^{\frac{p-1}{2}} - 1\right) \equiv 0 \bmod p.$$

Звідси вираз в одній із дужок ділиться на p. Обидві дужки не можуть ділитися на p, оскільки тоді на p ділилася б і їх різниця, яка дорівнює 2, а за умовою теореми p – непарне просте число. Якщо a є квадратичним лишком, то $a = x^2 \pmod{p}$ для деякого такого x, що НСД(x, p) = 1. Маємо: $a^{\frac{p-1}{2}} \equiv \left(x^2\right)^{\frac{p-1}{2}} \equiv x^{p-1} \equiv 1 \bmod p$, тобто $a^{\frac{p-1}{2}} \equiv 1 \bmod p$ або $a^{\frac{p-1}{2}} - 1$ ділиться на р. Якщо a є квадратичним нелишком, то $a^{\frac{p-1}{2}} - 1$ не ділиться на p, звідки

$a^{\frac{p-1}{2}} + 1$ повинно ділитися на p, або $a^{\frac{p-1}{2}} \equiv -1 \bmod p$.

Тоді $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod p$, якщо число a є квадратичним лишком за модулем p, то за означенням символа Лежандра $\left(\frac{a}{p}\right) = 1$, а за критерієм Ейлера $a^{\frac{p-1}{2}} \pmod p \equiv 1$. Відповідно, якщо число a є квадратичним нелишком за модулем p, то $\left(\frac{a}{p}\right) = -1$ і $a^{\frac{p-1}{2}} \pmod p \equiv -1$, звідки і випливають наступні властивості символа Лежандра [24, 25]:

1. $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod p$. Вказана властивість є наслідком критерію Ейлера.

Зокрема $\left(\frac{1}{p}\right) = 1$ та $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$.

Отже $-1 \in Q_p$ якщо $p \equiv 1 \pmod 4$ та $-1 \in \overline{Q}_p$ якщо $p \equiv 3 \pmod 4$.

2. $\left(\frac{a*b}{p}\right) = \left(\frac{a}{p}\right)*\left(\frac{b}{p}\right)$. Властивість випливає з послідовності очевидних порівнянь:

$$\left(\frac{a*b}{p}\right) \equiv (ab)^{\frac{p-1}{2}} \equiv a^{\frac{p-1}{2}} * b^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right)*\left(\frac{b}{p}\right) \pmod p.$$

Зокрема, якщо $a \in Z_p^*$, то $\left(\frac{a^2}{p}\right) = 1$ та $\left(\frac{a^2 b}{p}\right) = \left(\frac{b}{p}\right)$.

3. Якщо $a \equiv b \pmod p$, то $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$. Властивість випливає з того, що числа одного класу є одночасно або квадратичними лишками, або нелишками. Виходячи з цієї властивості:

$$\left(\frac{a}{p}\right) = \left(\frac{a+pt}{p}\right), t \in Z.$$

4. $\left(\frac{1}{p}\right) = 1$. Одиниця є квадратичним лишком для довільного непарного простого p. Ця властивість випливає з того, що порівняння $x^2 \equiv 1 \pmod{p}$ завжди має розв'язки $x = \pm 1 \pmod{p}$.

5. $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$.

Якщо p = 8k ± 1, то $\frac{p^2-1}{8} = \frac{(8k \pm 1)^2 - 1}{8} = \frac{64k^2 \pm 16k}{8} = 8k^2 \pm 2k$ – парне число.

Якщо $p = 8k \pm 3$, то $\frac{p^2-1}{8} = \frac{(8k \pm 3)^2 - 1}{8} = \frac{64k^2 \pm 48k + 8}{8} = 8k^2 \pm 6k + 1$ – непарне число.

Отже $\left(\frac{2}{p}\right)$=1, якщо $p \equiv 1$ або 7 (mod 8) та $\left(\frac{2}{p}\right)$= -1, якщо $p \equiv 3$ або 5 (mod 8).

6. Закон взаємності непарних простих чисел. Якщо p – просте непарне число, відмінне від q, то

$$\left(\frac{p}{q}\right) * \left(\frac{q}{p}\right) = (-1)^{\frac{(p-1)(q-1)}{4}}.$$

Помноживши цю рівність на $\left(\frac{p}{q}\right)$, отримаємо: $\left(\frac{q}{p}\right) = (-1)^{\frac{(p-1)(q-1)}{4}} * \left(\frac{p}{q}\right)$. Якщо виконується хоча б одна з рівностей $p \pmod 4 \equiv 1$ чи $q \pmod 4 \equiv 1$, то $\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right)$, інакше $\left(\frac{p}{q}\right) = -\left(\frac{q}{p}\right)$.

Символ Якобі є узагальненням символу Лежандра[24] на випадок, коли n є непарним, але не обовя'язково простим.

У випадку коли n – непарне ціле число, n ≥ 3 і відомо, що $n = p_1^{k_1} p_2^{k_2} \ldots p_t^{k_t}$, де p_i – прості числа. Символ Якобі $\left(\frac{a}{n}\right)$ визначається так:

$$\left(\frac{a}{n}\right)=\left(\frac{a}{p_1}\right)^{k_1}\left(\frac{a}{p_2}\right)^{k_2}\dots\left(\frac{a}{p_t}\right)^{k_t}.$$

Якщо n просте, то символ Якобі стає символом Лежандра.

Властивості символа Якобі:

1. $\left(\frac{a}{n}\right)$ може приймати одне з трьох значень: -1, 0 чи 1. При цьому $\left(\frac{a}{n}\right) = 0$ тоді і тільки тоді коли НСД(a, n) $\neq$ 1.

2. $\left(\frac{a*b}{n}\right) = \left(\frac{a}{n}\right)*\left(\frac{b}{n}\right)$. Якщо $a \in Z_n^*$, то $\left(\frac{a^2}{n}\right) = 1$.

3. $\left(\frac{a}{m*n}\right) = \left(\frac{a}{m}\right)*\left(\frac{a}{n}\right)$.

4. Якщо $a \equiv$ b (mod n), то $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$.

5. $\left(\frac{1}{n}\right)= 1$.

6. $\left(\frac{-1}{n}\right)=(-1)^{\frac{n-1}{2}}$. Отже $\left(\frac{-1}{n}\right)= 1$, якщо $n \equiv 1$ (mod 4) та $\left(\frac{-1}{n}\right)=$ -1, якщо $n \equiv 3$ (mod 4).

7. $\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}}$.

Отже $\left(\frac{2}{n}\right)=1$, якщо $p \equiv 1$ або 7 (mod 8) та $\left(\frac{2}{n}\right)=$ -1, якщо $p \equiv 3$ або 5 (mod 8).

8. $\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)(-1)^{\frac{(m-1)(n-1)}{4}}$.

З властивостей символу Якобі випливає, що якщо n непарне, а число a подати у вигляді $a = 2^k a_1$, де a_1 – непарне число, то:

$$\left(\frac{a}{n}\right) = \left(\frac{2^k}{n}\right)\left(\frac{a_1}{n}\right) = \left(\frac{2^k}{n}\right)\left(\frac{n \bmod a_1}{a_1}\right)(-1)^{\frac{(a_1-1)(n-1)}{4}}$$

Ця формула дає можливість обчислити значення символа Якобі не маючи розкладу числа *n* на прості множники.

На відміну від символу Лежандра, символ Якобі $\left(\frac{a}{n}\right)$ не визначає, чи є число *a* квадратичним лишком за модулем *n*. Справді, якщо $a \in Q_n$, то $\left(\frac{a}{n}\right) = 1$, але з того що $\left(\frac{a}{n}\right) = 1$ не випливає $a \in Q_n$.

Додаток Е

Аналіз методів факторизації та їх обчислювальної складності

1) Повний перебір можливих дільників.

Найпростішим алгоритмом факторизації є повний перебір можливих дільників. Кількість можливих варіантів при використанні такого методу дорівнює *(n-1)/2*, де *n* – добуток чисел для факторизації.

Обчислювальна складність повного перебору можливих дільників дорівнює $O(N^{1/2})$, де $N=log_2 n$. Недоліком такого методу факторизації є висока обчислювальна складність, яка базується на складності виконання операцій ділення, та велика кількість ітерацій за умови, якщо дільники однакової розрядності [28].

2) Дослідження методу факторизації Ферма.

Відомий метод факторизації багаторозрядних чисел [28], що ґрунтується на представленні відомого числа n, яке є добутком двох шуканих БРПЧ $n = p \cdot q$, що задовольняє співвідношення згідно теореми Ферма:

$$n = A^2 - B^2,$$

де $p = A + B,\ q = A - B$.

Суть методу полягає в тому, що визначається ціла частина від квадратного кореня з *n*, $m = \lfloor \sqrt{n} \rfloor$. Для різних чисел *x=1,2,…* послідовно багатократно визначається значення згідно виразу $q(x) = (m + x)^2 - n$ до тих пір, поки отримане значення не буде рівне повному квадрату у вигляді цілого числа згідно десяткової арифметики.

Таким чином, в алгоритмі факторизації Ферма на кожній ітерації 1 раз виконується операція добування квадратного кореня з БРЧ *n* та виділення цілої частини *m,* а в кожній ітерації - додавання $m + x$ та піднесення до

квадрату, віднімання від отриманого результату числа *n,* визначення квадратного кореня з отриманого результату та перевірки отриманого результату на цілочисельну повноту.

Суттєвим недоліком такого методу є значна обчислювальна складність, яка обумовлена виконанням великого числа обчислювально-складних операцій у позиційній системі числення над БРЧ, які включають операції піднесення до степеня та добування квадратного кореня над числами, розрядність яких експоненційно зростає, починаючи з розрядності числа *n*.

При реалізації обчислювальних операцій факторизації методом Ферма на основі використання двійкової позиційної системи числення виникає висока кількість наскрізних переносів, що приводить до низької швидкодії обчислювальних засобів. Особливо це стосується опрацювання БРЧ при виконанні операцій додавання, множення, піднесення до квадрату, добування квадратного кореня, перевірки отриманого результату на цілочисельність та віднімання згідно двійково-десяткової та двійкової арифметик [35].

Наприклад, при факторизації 31-розрядного десяткового числа *n* = 2752899074514048103193505027141 для визначення 16-розрядних десяткових чисел p= 1232144355415431 та q= 2234234213235411 необхідно виконати операцію визначення кореня квадратного $m = \lfloor \sqrt{n} \rfloor$ = 1659186268781792, визначити цілу частину 1659186268781792 та виконати x=74003015543628 ітерацій для визначення числа q(x).

На завершальному кроці визначається число $(m+x)^2 - n$ = 3003945095300461569704458176400, корінь квадратний з якого є повним квадратом, тобто $m + x = 1733189284325420$.

В [28] розглянуто алгоритм Ферма (додаток Е) для факторизації БРЧ, який характеризуються експоненційною обчислювальною складністю *O(exp(N))* у зв'язку з експоненційним зростанням квадратів, які повинні задовільнити співвідношення $n = A^2 - B^2$.

3) (p — 1)-метод Полларда.

В праці Ішмухаметова Ш.Т. [28] проведено дослідження методу

Полларда, який базується на властивостях малої теореми Ферма, згідно якої для кожного $a,\ 1 \le a < p,$ виконується умова $a^{p-1} \equiv 1(\mathrm{mod}\, p)$ (Додаток Е).

Недоліком даного методу є необхідність зберігати велику кількість попередніх значень x_j, що задовольняють умови $(x_j - x_i) \equiv 0(\mathrm{mod}\, p)$, $(f(x_j) - f(x_i)) \equiv 0(\mathrm{mod}\, p)$, з яких отримується рішення задачі факторизації. Тому обчислювальна складність методу складає $O(N^{1/4})$.

4) (p+1)- метод Вільямса

Метод Вільямса базується на рекурентній послідовності Люка-Лемєра (Lucas) [28] u_n,, яка визначається співвідношеннями $u_0=0, u_1=u, u_{n+1}=P \cdot u_n - Q \cdot u_{n-1,}$ де $P,\ Q$ фіксовані цілі числа. Даний метод представлений в додатку Е і схожий на $(p - 1)$ - метод Полларда, основною ідеєю якого є припущення гладкості числа $p+1,$ де p — простий дільник факторизовуваного числа n.

В результаті досліджень методів факторизації побудовано графік для порівняння обчислювальних складностей досліджуваних алгоритмів, представлений на рисунку 1.

Особливістю алгоритму Вільямса є застосування мультиплікативних степеневих функцій, які характеризуються особливою складністю обчислень та різким зростанням розрядів обчислювальних чисел. Крім того, даний метод передбачає зберігання та генерування великих масивів БРПЧ, а також передбачає достатньо складного обчислення операції НСД. Граничною ефективністю методу Віллямса є обчислювальна складність $O(N^{1/4})$.

Аналіз показує, що метод Ферма характеризується експоненціальною складністю, але є алгоритмічно найпростіший і характеризується формалізацією границь кількості ітерацій, піддається розпаралеленню, не є рекурентним та ймовірнісним, характеризується обмеженим числом типів операцій (додавання, множення, експоненціювання, порівняння).

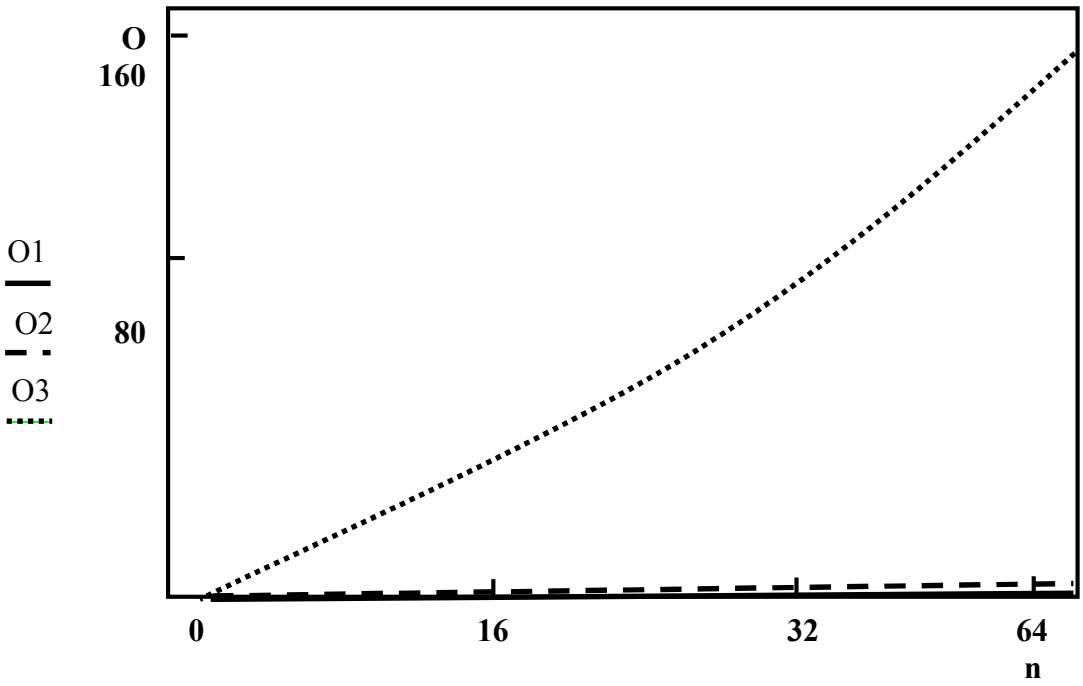


Рисунок 1 – Графіки обчислювальної складності алгоритмів факторизації: прямого перебору(O2), Поларда та Вільямса(O1), Ферма(O3).

Аналіз алгоритму факторизації БРЧ методами Поларда та Вільямса дозволяє встановити наступні їх характеристики обчислювальної, часової та прогнозно апаратної складності:

- методи Поларда та Вільямса в окремих випадках, коли факторизується добуток чисел різної розрядності, є більш ефективними у порівнянні з методом Ферма;

- розроблені на основі методу Поларда модифіковані алгоритми є ймовірнісними і не гарантують надійного кінцевого результату факторизації чисел;

- в даних методах не досліджені можливості глибокого розпаралелення обчислювальних операцій для підвищення швидкодії алгоритмів;

- не досліджені можливості застосування систем числення різних ТЧБ та відповідної модульної арифметики, в той час, коли в алгоритмі присутнє велике число модульних операцій.

При дослідження методу факторизації Ферма виявлено наступні переваги, що дозволяють підвищити його швидкодію та покращити

характеристики ефективності:

- метод дозволяє розпаралелення обчислювального процесу;

- Ферма-подібні методи відзначаються ефективністю при умові рівності множників;

- значні затрати пам’яті при факторизації БРЧ, що використовуються на опрацювання квадратів більших 1024 біт можна уникнути використовуючи властивості квадратичних лишків досліджені 2 пункті 2.4;

- експоненційну обчислювальну складність, яка виникає в процесі факторизації, можна спростити з допомогою використання СЗК ТЧБ Крестенсона.

Тому такі характеристики обґрунтовують перспективу його вдосконалення з використанням сучасних програмно-апаратних засобів та спецпроцесорів, у тому числі швидкодіючої модульної арифметики Хаара та Крестенсона.

Додаток Ж

Лістинг бібліотеки для опрацювання багаторозрядних чисел

```
//---------------------------------------------------------------------------


#pragma hdrstop

#include "bitarray.h"
//#include "unit3.h"
//#include "stdlib.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)


//++++++++++++++++++++++++++
long bitarray::addbit(long bit){
 if (bit<array_size){
 int i;
 for (i=bit;i<number_size&&mainarray[i]==true;i++)mainarray[i]=false;
 if (i<number_size)mainarray[i]=true ;

  if (i==number_size){mainarray[i]=true;number_size++;};


  if (i>number_size){
   for (i=number_size;i<bit;i++){mainarray[i]=false;};
   mainarray[bit]=true;
   number_size=bit+1;
  };
  };
  return 1;
};
//++++++++++++++++++++++++++
long bitarray::addbitarray(bitarray &number){
 for (int i=0;i<number.number_size;i++){
  if (number.mainarray[i])addbit(i);
 };
return 1;
};
//++++++++++++++++++++++++++
long bitarray::calculate_cikle(verylong a,TEdit *edit){
verylong res=0;
```

```
zsetbit(&res, 0);
long i;
bool cikle_smaller=false;
for (i=0;i<2147483645&&!cikle_smaller;i++)
 {
 //edit->Text=IntToStr(i);
  z2mul(res,&res);
 zmod(res,a,&res);
cikle_smaller=(zscompare(res, 1)==0);
 };

 //if (zscompare(res, 1)==0&&!cikle_smaller) {return true;} else {return
false;};
zfree(&res);
 return i;
};
//++++++++++++++++++++++++++
long bitarray::calculate_cikle(bitarray &number){
 if (number.mainarray[0]){
  bitarray part(0,number.number_size+2);
  part.number_size=number.number_size+1;

  part.mainarray[part.number_size-1]=true;
  part.minus(number);
  number.cikle=number.number_size;

   while (!part.is_one){
   part.mul2();
   if (part.isbigger(number))part.minus(number);
   number.cikle++;
   };
 };
return number.cikle;
};

//++++++++++++++++++++++++++
long bitarray::calculate_cikle(bitarray &number,TProgressBar
*cikleprogres){
 if (number.mainarray[0]){
 cikleprogres->Max=number.to_long();
  bitarray part(0,number.number_size+2);
  part.number_size=number.number_size+1;

  part.mainarray[part.number_size-1]=true;
  part.minus(number);
```

```
  number.cikle=number.number_size;

   while (!part.is_one){
   part.mul2();
   if (part.isbigger(number))part.minus(number);
   number.cikle++;
   cikleprogres->Position=number.cikle;
   };
 };
return number.cikle;
};
//++++++++++++++++++++++++++
bool bitarray::calculate_cikle(bitarray &number,long cikle_){
 if (number.mainarray[0]){
 // if (bitarray::cikleprogres)bitarray::cikleprogres->Max=cikle_;

  bitarray part(0,number.number_size+2);
  part.number_size=number.number_size+1;

  part.mainarray[part.number_size-1]=true;
  part.minus(number);
  number.cikle=number.number_size;

   while (!part.is_one&&number.cikle<=cikle_){
   part.mul2();
   if (part.isbigger(number))part.minus(number);
   number.cikle++;
   //if (bitarray::cikleprogres)bitarray::cikleprogres->Position=number.cikle;
   };
 };
 if (number.cikle==cikle_){ return true;} else {return false;};
//return number.cikle;
};
//++++++++++++++++++++++++++
bool bitarray::calculate_cikle(bitarray &number,long cikle_,TProgressBar
*cikleprogres,TMemo *memo){
 if (number.mainarray[0]){
  //if (cikleprogres)cikleprogres->Max=cikle_;

  bitarray part(0,number.number_size+2);
  part.number_size=number.number_size+1;

  part.mainarray[part.number_size-1]=true;
  part.minus(number);
  number.cikle=number.number_size;
```

```
   while (!part.is_one&&number.cikle<=cikle_){
   part.mul2();
   if (part.isbigger(number))part.minus(number);
   number.cikle++;
  // memo->Lines->Add(IntToStr(number.cikle));
//   if (cikleprogres)cikleprogres->Position=number.cikle;
   };
 };
// memo->Lines->Add(IntToStr(number.cikle));
 if (number.cikle==cikle_){ return true;} else {return false;};
//return number.cikle;
};
//++++++++++++++++++++++++++
bitarray::bitarray(){
randomize();

};
//++++++++++++++++++++++++++
long bitarray::to_long(){//перевірено
long sum=0;
 if (number_size<=32){
  for (int i=0;i<number_size;i++)if (mainarray[i]) sum=sum+pow(2,i);
 };
return sum;
};
//++++++++++++++++++++++++++
int bitarray::nullarray(){
for (int i=0;i<array_size;i++)mainarray[i]=false;
return 1;
};
//++++++++++++++++++++++++++
int bitarray::minusbit(long bitnumber){ //перевірено
 //if (bitnumber=>0&&bitnumber<=number_size+count_of_smaller_bit0){
  if
((bitnumber>=0&&bitnumber<number_size)&&count_of_smaller_bit0==0)
{
   int i,a=0;
   if (mainarray[bitnumber]==true){
    mainarray[bitnumber]=false;
   }else {
   for
(i=bitnumber;i<number_size&&mainarray[i]==false;i++)mainarray[i]=true;
   mainarray[i]=false;
   };
```

```
   for (i=0;i<number_size;i++)
   if (mainarray[i])a=i;
   number_size=a+1;
  };
  if (number_size==1&&mainarray[0]==true)is_one=true;
 //};
 return 1;
};
//++++++++++++++++++++++++++
int bitarray::mul2(){ //перевірено
 if (bitarray::to_long()!=0){
 if (number_size<array_size){
  for (int i=number_size;i>=0;i--){
   if (mainarray[i]){
    mainarray[i]=false;
    mainarray[i+1]=true;
   };
  };
  number_size++;
 }; };
 //count_of_smaller_bit0++;
 return 1;
};
//++++++++++++++++++++++++++
bitarray::bitarray(long num){//перевірено
randomize();
 array_size=32;
 cikle=0;
 count_of_smaller_bit0=0;
 mainarray=new bool [array_size];
 nullarray();
 int i;
 if (num==1){is_one=true;}else{is_one=false;};
 for (i=0; num; num>>=1, i++){
 mainarray[i]=(bool((num&1)?(1):(0)));
 };
 number_size=i;
};
//++++++++++++++++++++++++++
bitarray::bitarray(AnsiString s){
randomize();
// в процесі розробки !!!!!!!!!!!!
};
//++++++++++++++++++++++++++
```

```
bitarray::bitarray(long num,long reserve){
randomize();
 cikle=0;
 array_size=reserve;
 count_of_smaller_bit0=0;
 mainarray=new bool [array_size];
 nullarray();
 int i;
 if (num==1){is_one=true;}else{is_one=false;};
 for (i=0; num; num>>=1, i++){
  mainarray[i]=(bool((num&1)?(1):(0)));
 };
 number_size=i;
};
//++++++++++++++++++++++++++
bitarray::bitarray(AnsiString s,long reserve){
randomize();
// в процесі розробки !!!!!!!!!!!!
};
//++++++++++++++++++++++++++
bitarray::bitarray(vector<bool>& temp,long size,long reserve){
randomize();
 int i;
 cikle=0;
 count_of_smaller_bit0=0;
 array_size=reserve;
 mainarray=new bool [array_size];
 nullarray();
// if(size<=temp.size()&&size<=reserve){
  for(i=0;i<size;i++) mainarray[i]=temp[i];
  //}
 // else {
  ///  в процесі розробки !!!!!!!!!!!!
 // };
 if (number_size==1&&mainarray[0]==true)is_one=true;
};
//++++++++++++++++++++++++++
AnsiString bitarray::ToString(){
//перевірено
 AnsiString s;
 count_of_smaller_bit0=0;
 for(int i=number_size-1;i>=0;i--){
  if (mainarray[i]){s=s+"1";} else {s=s+"0";};
 };
return s;
```

```
};

//++++++++++++++++++++++++
int bitarray::minus(bitarray &vidyemnyk){//перевірено
 if (isbigger(vidyemnyk)){

  for (int i=vidyemnyk.number_size-1;i>=0;i--){
   if (vidyemnyk.mainarray[i]==true)bitarray::minusbit(i);
    };
 };
return 1;
};
//++++++++++++++++++++++++


bool bitarray::isbigger(bitarray &another){//перевірено
 bool is=false;
 if (another.number_size<number_size+count_of_smaller_bit0) is=true;
 if (another.number_size>number_size+count_of_smaller_bit0) is=false;
 if (another.number_size==number_size+count_of_smaller_bit0){
  int i;
  for (i=another.number_size-1;i>0&&mainarray[i]==another.mainarray[i];i-
-);
  if (mainarray[i]){is=true;} else {is=false;};
 };
 return is;

// return true;
};
//++++++++++++++++++++++++
void bitarray::calculate_part(bitarray &number){
 if (number.mainarray[0]){
  bitarray sum(number.to_long()*2);
  bitarray part(number.to_long()*2,ceil(number.to_long()/2)+2);
  part.mainarray[0]=true;
  //if (partprogres)partprogres->Max =ceil(number.to_long()/2)+2;
bool cikle_=calculate_cikle(part,number.to_long());
 while(!cikle_&&part.number_size<part.array_size){
 part.addbitarray(sum);
 cikle_=calculate_cikle(part,number.to_long());
 //  if (partprogres)partprogres->Position=part.number_size;
 };
 if (cikle_){
  number.mainarray=part.mainarray;
  number.array_size=part.array_size;
```

```
  number.count_of_smaller_bit0=part.count_of_smaller_bit0;
  number.number_size=part.number_size;
  part.mainarray=0;

  part.mainarray=0;
   };
 };
};
//++++++++++++++++++++++++
void bitarray::calculate_part(bitarray &number,TProgressBar
*cikleprogres,TProgressBar *partprogres,TMemo *memo){
 if (number.mainarray[0]){
  bitarray sum(number.to_long()*2);
  bitarray part(number.to_long()*2,ceil(number.to_long()/2)+2);
  part.mainarray[0]=true;
  long n=number.to_long();
  if (partprogres)partprogres->Max =ceil(number.to_long()/2)+2;
bool cikle_=calculate_cikle(part,n,cikleprogres, memo);
 while(!cikle_&&part.number_size<part.array_size){
 //sum.mul2();
 part.addbitarray(sum);
 //part.mainarray[0]=false;
 //part.mul2();
 //part.mainarray[0]=true;
 cikle_=calculate_cikle(part,n);
   if (partprogres)partprogres->Position=part.number_size;
 };
 if (cikle_){
  number.mainarray=part.mainarray;
  number.array_size=part.array_size;
  number.count_of_smaller_bit0=part.count_of_smaller_bit0;
  number.number_size=part.number_size;
  part.mainarray=0;

 // part.mainarray=0;
   };
 };
};
//++++++++++++++++++++++++
long bitarray::mod(bitarray &number,bitarray &modul){

bitarray res(1,number.array_size);

for (int i=number.number_size-2;i>=0;i--){
res.mul2();
```

```
if (number.mainarray[i])res.mainarray[0]=true;
if (res.isbigger(modul))res.minus(modul);

};
return res.to_long();
return 0;
};
//++++++++++++++++++++++++++
long bitarray::modpresent(bitarray &number,bitarray &modul,TMemo
*memo){
memo->Lines->Add(number.ToString()+" mod "+modul.ToString()+" - ?");
bitarray res(1,number.array_size);
memo->Lines->Add("1");
for (int i=number.number_size-2;i>=0;i--){
res.mul2();
if (number.mainarray[i])res.mainarray[0]=true;
if (res.isbigger(modul))res.minus(modul);
memo->Lines->Add(res.ToString());
};

memo->Lines->Add("залишок = "+res.ToString());
return res.to_long();

};
//++++++++++++++++++++++++++
double bitarray::mod_time (bitarray &number,bitarray &modul){//повертає
залишок в число і час за який йоло було знайдено
TDateTime DateTime = Time();
bitarray res(1,number.array_size);
for (int i=number.number_size-2;i>=0;i--){
res.mul2();
if (number.mainarray[i])res.mainarray[0]=true;
if (res.isbigger(modul))res.minus(modul);
};
//number.mainarray=res.mainarray;
//number.array_size=res.array_size;
//number.count_of_smaller_bit0=res.count_of_smaller_bit0;
//number.number_size=res.number_size;
//res.mainarray=0;
return Time().Val-DateTime.Val;
};
//++++++++++++++++++++++++++
void bitarray::random_(long size){
//delete[] mainarray;
array_size=size+1;
```

```
number_size=size;
mainarray=new bool [array_size];
cikle=0;

for (int i=1;i<number_size-1;i++){
//randomize();
if (random(2)==1){mainarray[i]=true;}else {mainarray[i]=false;};};
mainarray[number_size-1]=true;
mainarray[0]=true;
mainarray[size-1]=true;
};
//++++++++++++++++++++++++++
void bitarray::randomnumber(bitarray &number,long countof_bit){

 number.newsize(countof_bit+1);
 number.number_size=countof_bit;

 for (int i=0;i<number.number_size-1;i++){
  if
(random(2)==0){number.mainarray[i]=false;}else{number.mainarray[i]=tru
e;};
 };
 number.mainarray[countof_bit-1]=true;
 number.mainarray[0]=true;
};
//++++++++++++++++++++++++++
void bitarray::newsize(long numbersize){
delete[] mainarray;
 cikle=0;
 array_size=numbersize;
 count_of_smaller_bit0=0;
 mainarray=new bool [array_size];
 nullarray();
 int i;
 mainarray[0]=false;
 number_size=1;
};
//++++++++++++++++++++++++++
void bitarray::from_bitarray_to_verylong(bitarray &number,verylong *a){
  char *str;
 zsetlength(a,number.number_size ,str);
 for(long i=0;i<number.number_size;i++)
  {
   if (number.mainarray[i]==true){
   zsetbit(a, i);
```

```
   }else{
   //if (zbit(a,i))zsetbit(&a, i);
   };
  // zadd( b,  c, &c);
//   z2mul(b,&b);
  // zcopy(d, &c);

  };
};

//++++++++++++++++++++++++++
AnsiString bitarray::from_verylong_string_binary(verylong a){
 // z2mul(a,&a);  z2mul(a,&a);              // не працює правильно
 long number_weight=z2log(a);
 AnsiString s="";
 for (int i=number_weight-1;i>=0;i--)
 if (zbit(a, i)==1){s=s+'1';}
 else{s=s+'0';};
 //if (zbit(a, number_weight-1)==1)s=s+'1';
 return s;
/* char *str;
str= new char [32];
for (int i=0;i<32;i++)str[i]='0';
zswrite(str,a);
s=str;
 return s;   */
};
//++++++++++++++++++++++++++

AnsiString bitarray::from_bitarray_to_verylong_(bitarray
&number,verylong a){
AnsiString s;
//verylong c=0;
//verylong b=0;
//zsetbit(&b, 0);
  //zcopy();
  char *str;
 zsetlength(&a,number.number_size ,str);
 for(long i=0;i<number.number_size;i++)
  {
   if (number.mainarray[i]==true){
   zsetbit(&a, i);
   }else{
 //  if (zbit(a,i))zsetbit(&a, i);
   };
```

```
  // zadd( b,  c, &c);
//   z2mul(b,&b);
  // zcopy(d, &c);

  };

 // zcopy(c, a);
 //a=&c;


//str= new char [32];
//for (int i=0;i<32;i++)str[i]='0';
//zswrite(str,a);
//s=str;
//return s;
};
//++++++++++++++++++++++++++
bool bitarray::have_that_cikle(verylong a,long cikle){
verylong res=0;
zsetbit(&res, 0);
long i;
bool cikle_smaller=false;
for (i=0;i<cikle&&!cikle_smaller;i++)
 {
  z2mul(res,&res);
 zmod(res,a,&res);
 if (i<cikle-1) cikle_smaller=(zscompare(res, 1)==0);
 };
if (zscompare(res, 1)==0&&!cikle_smaller) {return true;} else {return
false;};
zfree(&res);
};
//++++++++++++++++++++++++++
bool bitarray::find_mersene_part(verylong *a,long cikle){
verylong doublesum=0;
verylong divisor=0;
bitarray number(cikle);
from_bitarray_to_verylong(number,&doublesum);
z2mul(doublesum,&doublesum);
zsetbit(&divisor, 0);

long bit_count_limit=ceil(cikle/2);
bool limit_out=false;
bool part_finded=false;
```

```
 while (!limit_out){
  zadd(divisor, doublesum, &divisor);
  if
(have_that_cikle(divisor,cikle)){limit_out=true;part_finded=true;}else{limit
_out=z2log(divisor)>=bit_count_limit;};
 };
zswap(a, &divisor);
zfree(&doublesum);
zfree(&divisor);

if(part_finded){return true;}else{return false;};
//from_bitarray_to_verylong(Divisor,cikle);


};

//++++++++++++++++++
bool bitarray::find_mersene_part_new_incpuding_prime_3(long
cikle,TMemo *memo){
//додано врахування кроку 2c відносно деяких простих модулів
// тестова функція з урахуванням декількох модулів
verylong doublesum=0;
verylong divisor=0;
bitarray number(cikle);
from_bitarray_to_verylong(number,&doublesum);
z2mul(doublesum,&doublesum);
//long mod2c_2=zsmod(doublesum, 2);

long mod2c_3=zsmod(doublesum, 3);
long mod2c_5=zsmod(doublesum, 5);
long mod2c_7=zsmod(doublesum, 7);
long mod2c_11=zsmod(doublesum, 11);
long mod2c_13=zsmod(doublesum, 13);
long mod2c_17=zsmod(doublesum, 17);
long mod2c_19=zsmod(doublesum, 19);
long mod2c_23=zsmod(doublesum, 23);
long mod2c_29=zsmod(doublesum, 29);
long mod2c_31=zsmod(doublesum, 31);
long mod2c_37=zsmod(doublesum, 37);
long mod2c_41=zsmod(doublesum, 41);
long mod2c_43=zsmod(doublesum, 43);
long mod2c_47=zsmod(doublesum, 47);
long mod2c_53=zsmod(doublesum, 53);
long mod2c_59=zsmod(doublesum, 59);
long mod2c_61=zsmod(doublesum, 61);
```

```
long mod2c_67=zsmod(doublesum, 67);
long mod2c_71=zsmod(doublesum, 71);
long mod2c_73=zsmod(doublesum, 73);
long mod2c_79=zsmod(doublesum, 79);
long mod2c_83=zsmod(doublesum, 83);
long mod2c_89=zsmod(doublesum, 89);
long mod2c_97=zsmod(doublesum, 97);

long mod2c_101=zsmod(doublesum, 101);
long mod2c_103=zsmod(doublesum, 103);
long mod2c_107=zsmod(doublesum, 107);
long mod2c_109=zsmod(doublesum, 109);
long mod2c_113=zsmod(doublesum, 113);
long mod2c_127=zsmod(doublesum, 127);
long mod2c_131=zsmod(doublesum, 131);
long mod2c_137=zsmod(doublesum, 137);
long mod2c_139=zsmod(doublesum, 139);
long mod2c_149=zsmod(doublesum, 149);

long part_=1;

zsetbit(&divisor, 0);

long bit_count_limit=ceil(cikle/2);
bool limit_out=false;
bool part_finded=false;
bool have;
bool next=true;
long roz;
long ost;
zsmul(doublesum,part_, &divisor);
zsetbit(&divisor, 0);
if (have_that_cikle(divisor,cikle)){limit_out=true;part_finded=true;};
 while (!limit_out){
  next=true;
  while (next){
  next=false;
  //if (mod2c_3==part_%3){part_++;next=true;};
  if (((mod2c_3*(part_%3))%3)==2){part_++;next=true;};

  if (((mod2c_5*(part_)%5)%5)==4){part_++;next=true;};
  if (((mod2c_7*(part_)%7)%7)==6){part_++;next=true;};

//if (((mod2c_7*(part_)%7)%7)==2){part_++;next=true;};
//if (((mod2c_7*(part_)%7)%7)==3){part_++;next=true;};
```

```
 if (((mod2c_11*(part_)%11)%11)==10){part_++;next=true;};
 if (((mod2c_13*(part_)%13)%13)==12){part_++;next=true;};
 if (((mod2c_17*(part_)%17)%17)==16){part_++;next=true;};
 if (((mod2c_19*(part_)%19)%19)==18){part_++;next=true;};
 if (((mod2c_23*(part_)%23)%23)==22){part_++;next=true;};
 if (((mod2c_29*(part_)%29)%29)==28){part_++;next=true;};
 if (((mod2c_31*(part_)%31)%31)==30){part_++;next=true;};
 if (((mod2c_37*(part_)%37)%37)==36){part_++;next=true;};
 if (((mod2c_41*(part_)%41)%41)==40){part_++;next=true;};
 if (((mod2c_43*(part_)%43)%43)==42){part_++;next=true;};
 if (((mod2c_47*(part_)%47)%47)==46){part_++;next=true;};
 if (((mod2c_53*(part_)%53)%53)==52){part_++;next=true;};
 if (((mod2c_59*(part_)%59)%59)==58){part_++;next=true;};
 if (((mod2c_61*(part_)%61)%61)==60){part_++;next=true;};
 if (((mod2c_67*(part_)%67)%67)==66){part_++;next=true;};
 if (((mod2c_71*(part_)%71)%71)==70){part_++;next=true;};
 if (((mod2c_73*(part_)%73)%73)==72){part_++;next=true;};
 if (((mod2c_79*(part_)%79)%79)==78){part_++;next=true;};
 if (((mod2c_83*(part_)%83)%83)==82){part_++;next=true;};
 if (((mod2c_89*(part_)%89)%89)==88){part_++;next=true;};
 if (((mod2c_97*(part_)%97)%97)==96){part_++;next=true;};

 if (((mod2c_101*(part_)%101)%101)==100){part_++;next=true;};
 if (((mod2c_103*(part_)%103)%103)==102){part_++;next=true;};
 if (((mod2c_107*(part_)%107)%107)==106){part_++;next=true;};
 if (((mod2c_109*(part_)%109)%109)==108){part_++;next=true;};
 if (((mod2c_113*(part_)%113)%113)==112){part_++;next=true;};
 if (((mod2c_127*(part_)%127)%127)==126){part_++;next=true;};
 if (((mod2c_131*(part_)%131)%131)==130){part_++;next=true;};
 if (((mod2c_137*(part_)%137)%137)==136){part_++;next=true;};
 if (((mod2c_139*(part_)%139)%139)==138){part_++;next=true;};
 if (((mod2c_149*(part_)%149)%149)==148){part_++;next=true;};

 };
 zsmul(doublesum,part_, &divisor);
 zsetbit(&divisor, 0);

 part_++;
// memo->Lines->Add(IntToStr(part_-1));
 if (have_that_cikle(divisor,cikle)){limit_out=true;
      long number_weight=z2log(divisor);
      AnsiString s="";
      for (int i=number_weight-1;i>=0;i--)
      if (zbit(divisor, i)==1){s=s+"1";}
```

```
        else{
        s=s+"0";};
        memo->Lines->Add(s);

  part_finded=true;}else{
  roz=z2log(divisor);
  limit_out=roz>=bit_count_limit;};
 };
zfree(&doublesum);
zfree(&divisor);


if(part_finded){return true;}else{return false;};



};

//++++++++++++++++++++++++
bool bitarray::find_mersene_part(long cikle,TProgressBar
*partprogres,TProgressBar *cikleprogres)
{

verylong doublesum=0;
verylong divisor=0;
bitarray number(cikle);
from_bitarray_to_verylong(number,&doublesum);
z2mul(doublesum,&doublesum);
zsetbit(&divisor, 0);
long position=0;
long bit_count_limit=ceil(cikle/2);
bool limit_out=false;
bool part_finded=false;
partprogres->Max=bit_count_limit;
 while (!limit_out){

 partprogres->Position=position;
  zadd(divisor, doublesum, &divisor);
  if (have_that_cikle(divisor,cikle)){limit_out=true;part_finded=true;}else{
  position=z2log(divisor);
  limit_out=position>=bit_count_limit;};
 };
//zswap(a, &divisor);
zfree(&doublesum);
zfree(&divisor);
```

```
if(part_finded){return true;}else{return false;};
//from_bitarray_to_verylong(Divisor,cikle);

};
//++++++++++++++++++++++++
bool bitarray::have_that_cikle(verylong a,long cikle,TProgressBar
*progres){
progres->Max=cikle;
verylong res=0;
zsetbit(&res, 0);
long i;
bool cikle_smaller=false;
for (i=0;i<cikle&&!cikle_smaller;i++)
 {
  z2mul(res,&res);
 zmod(res,a,&res);
 if (i<cikle-1) cikle_smaller=(zscompare(res, 1)==0);
 progres->Position=i;
 };
if (zscompare(res, 1)==0&&!cikle_smaller) {return true;} else {return
false;};
zfree(&res);

};

//**********************************************************************
***************

 bool bitarray::save_to_file(AnsiString filename,verylong a){
  long number_weight=z2log(a);
  char s;
  FILE *filetosave;
 filetosave=fopen(filename.c_str(),"w");
 for (int i=0;i<=number_weight-1;i++){
 if (zbit(a, i)==1){s='1'; }
 else{s='0';};
  fputc((int)s,filetosave); };
 fclose(filetosave);
 return true;

 };

//**********************************************************************
***************
```

```
bool bitarray::load_from_file(AnsiString filename, verylong *a){
char *str;

char s;
long i=0;
FILE *filetosave;
filetosave=fopen(filename.c_str(),"r");

while(!feof(filetosave)){
s=char(fgetc(filetosave));

if (s=='1') {
zsetlength(a,i+1,str);
zsetbit(a, i);
};
i++;
};
fclose(filetosave);
};
```

Додаток И

Лістинг додатку «Імовірнісний тест простоти»

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "lip.c"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button1Click(TObject *Sender)
{
verylong a=0;
verylong b=0;
verylong c=0;
verylong d=0;
zsetbit(&c,0);
zsetbit(&a,1);

//zsread(Edit1->Text.c_str(),&a);
zsread(Edit1->Text.c_str(),&b);
zsread(Edit1->Text.c_str(),&d);
bool is=true;
zsub(b,c,&b);
while (is){
zsub(b,c,&b);
is=!ziszero(b);
zlshift(a,1,&a);
zmod(a,d,&a);
   if (zscompare(a,1))is=true;
};
char buffer[1024];
```

```
zswrite(buffer, a);
Memo1->Lines->Add(IntToStr(zscompare(a,2)));
if (zscompare(a,2)){CheckBox1->Checked=!true;}else{CheckBox1-
>Checked=!false;};
}
//---------------------------------------------------------------------------
```

Додаток К

Лістинг додатку «Факторизація багаторозрядних чисел»

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
factorizing_key_generation *fp;
TForm1 *Form1;
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::N2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
/*
verylong all=0;
zsetbit(&all,0);
long nnull;
        for (int i=0; i <1024;i++){
//if (factorizing_key_generation::is_true_square_modul(all,&nnull)){
break;
//}else{
zsadd(all,nnull,&all);
Memo1->Lines->Add("null  - "+IntToStr(nnull));
};
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
```

```
 //Memo1->Lines->Add("*** "+IntToStr(max));
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Memo1->Lines->Add(TimeToStr(Time()));
fp=new factorizing_key_generation(false);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
fp->Suspend();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{
fp->Resume();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N3Click(TObject *Sender)
{
if (SaveDialog1->Execute())Memo1->Lines->SaveToFile(SaveDialog1->FileName);
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit2.h"
#include "Unit1.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------

//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall factorizing_key_generation::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------
AnsiString factorizing_key_generation::print(verylong p){
AnsiString s;
```

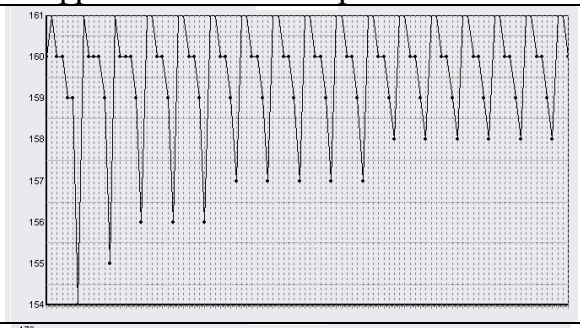
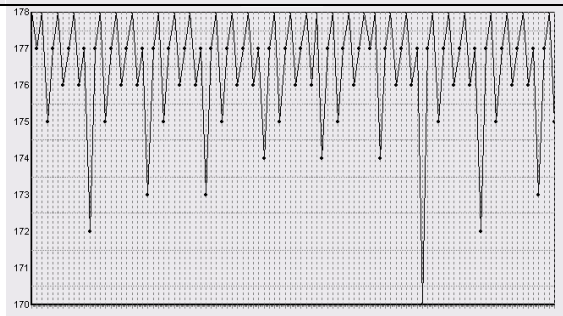
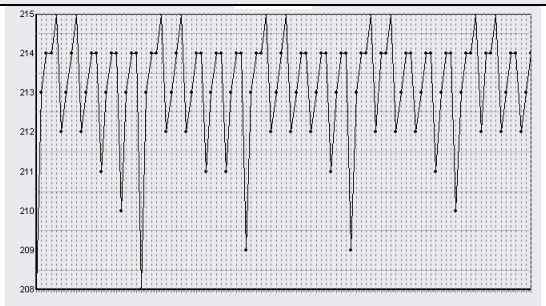
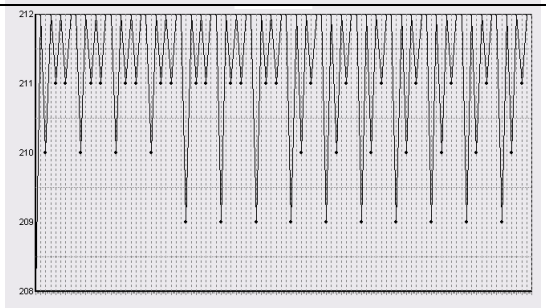
```
        char q[1024];
        zswrite(q,p);
        s=s+" ";
        s=s+q;
return s;
        };
//---------------------------------------------------------------------------
bool factorizing_key_generation::is_square_modulus(verylong part,long modul){
long m=1;
bool is=false;
for(long i=1;i<=modul/2;i++){
m=(i*i)%modul;
if (zscompare(part,m)==0){is=true;break;}
};
return is;

};
//---------------------------------------------------------------------------
bool factorizing_key_generation::is_square_modulus(long part,long modul){
long m=1;
bool is=false;
for(long i=1;i<=modul/2;i++){
m=(i*i)%modul;
if (part==m||part==0){is=true;break;}
};
return is;

};
//---------------------------------------------------------------------------
void factorizing_key_generation::frommemotovector(vector <long> &temp1,TMemo
*Memo1){
temp1.resize(Memo1->Lines->Count);
  for (int i=0;i<Memo1->Lines->Count;i++){
  temp1[i]=StrToInt(Memo1->Lines->Strings[i]);
  };
//  Form1->Memo1->Lines->Add(IntToStr(temp1[5]));
};

__fastcall factorizing_key_generation::factorizing_key_generation(bool
CreateSuspended)
        : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall factorizing_key_generation::Execute()
{
// Тут відбувається генерація ключів для факторизації по простих числах
ofstream key_file;
```

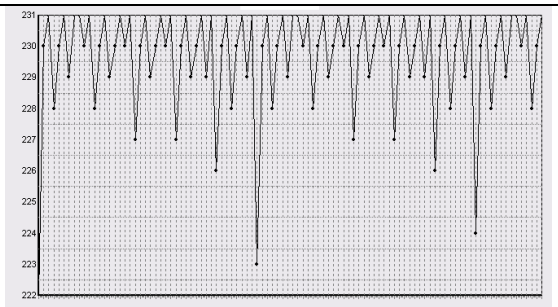
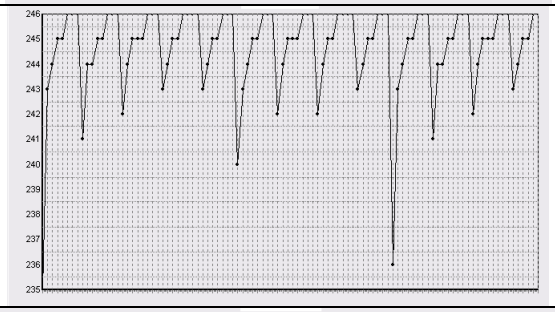
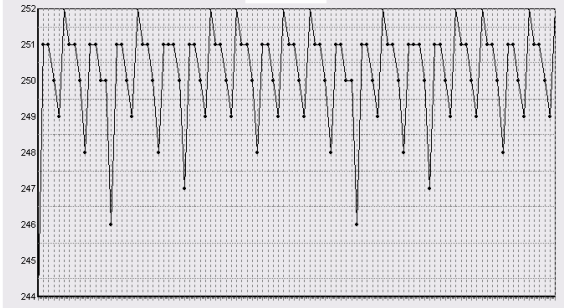
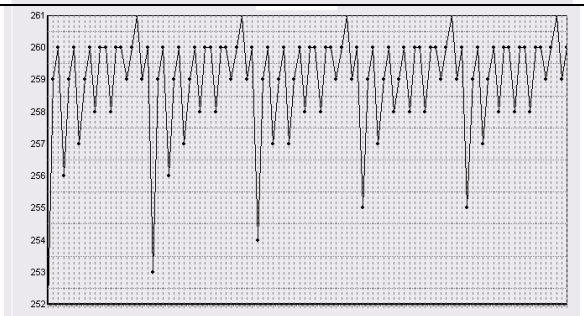
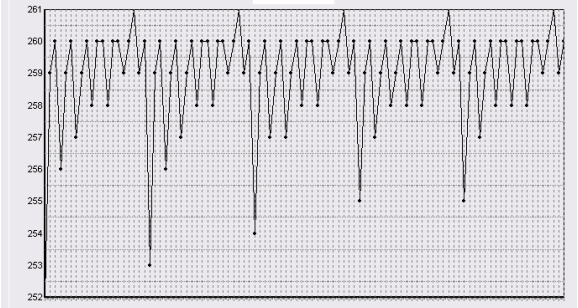
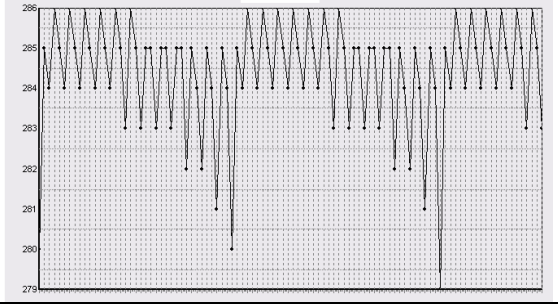
```
key_file.open("key.log",ios::out);   // файл куди будуть зберігатись кожен ключ
key_file<<(Form1->Edit1->Text.c_str())<<endl;
max=0;
verylong p=0;
verylong p1=0;
verylong p2=0;
verylong p3=0;
verylong one=0;
zsetbit(&one,0);
AnsiString s="";
long m=0; // результуючий залишок шуканої суми, яка повинна бути квадратом
long stepm; // це корынь квадратний помножений на два +1 , відстань до наступного
квадрату
zsread(Form1->Edit1->Text.c_str(),&p);
vector <long> temp;

frommemotovector(temp,Form1->Memo2);
zsqrt(p,&p1,&p2);
zsadd(p1,1,&p1);
zsq(p1,&p2);
zsub(p2,p,&p2);
zsmul(p1,2,&p1);
zsub(p1,one,&p1);
result.resize(temp.size());
result_long.resize(temp.size());
Form1->ProgressBar1->Max=temp.size();
for (int j=0; j<temp.size();j++){
     Form1->ProgressBar1->Position=j;
       m=zsmod(p2,temp[j]);
       stepm=zsmod(p1,temp[j]);
       s="";
       for(int i=0;i<temp[j];i++){
       stepm=(stepm+2);
       m=(m+stepm)%temp[j];
       result[j].resize(temp[j]);
       result_long[j].resize(temp[j]);
       result[j][i]=is_square_modulus(m,temp[j]);
      if (Form1->CheckBox1->Checked||Form1->CheckBox2->Checked) if
(result[j][i]){s=s+("1");}else{s=s+("0");};
//       if (Form1->CheckBox1->Checked) if (result[j][i]){s=s+("1");}else{s=s+("0");};
       };
// ця частина підраховує кількість нулів і зберігає в вектор
       for(int i=0;i<temp[j];i++){
       result_long[j][i]=0;
          for(int g=i;g<temp[j];g++){
          if (!result[j][g]){result_long[j][i]++;}else{break;};
          };
       };
```

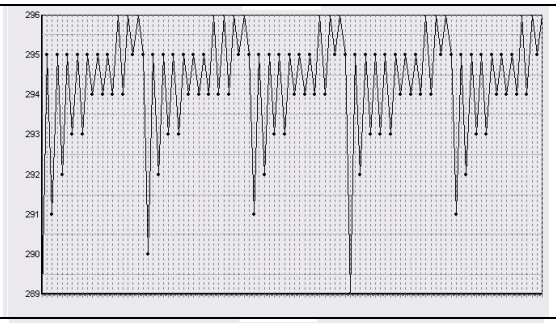
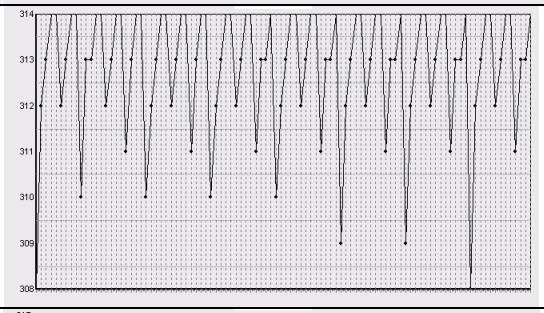
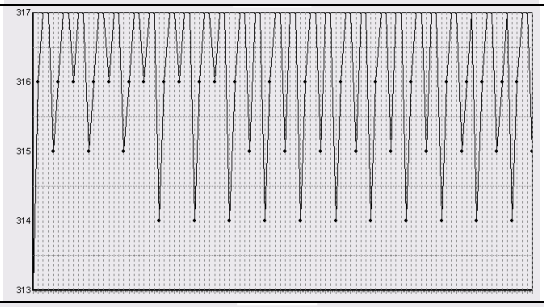
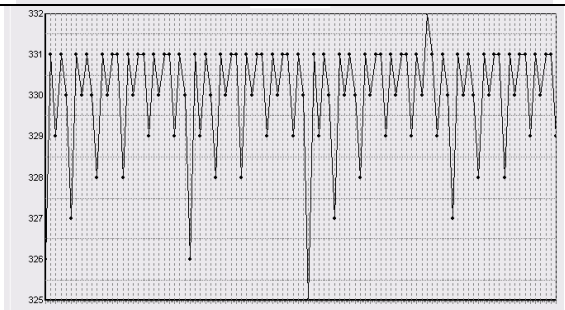
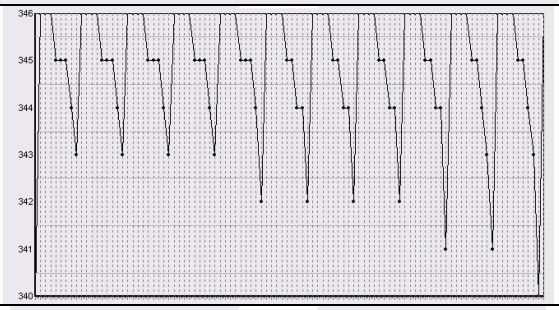
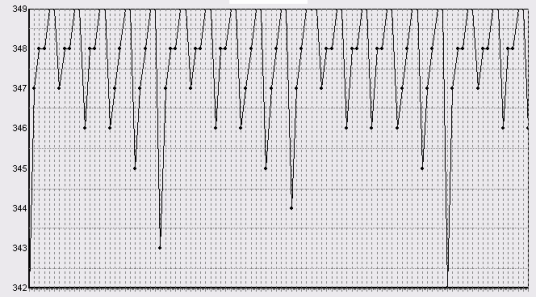
```
      if (Form1->CheckBox1->Checked){ Form1->Memo1->Lines->Add(s);};
      if (Form1->CheckBox2->Checked){key_file<<s.c_str()<<" "<<temp[j]<<endl;};
}
zfree(&p);
zfree(&p1);
zfree(&p2);
zfree(&p3);
zfree(&one);
//for (int j=0; j<temp.size();j++){
//  for(int i=0;i<temp[j];i++){
 // Memo1->Lines->Add(result_long[j][i]);
//  };
 // };
  Form1->Memo1->Lines->Add(TimeToStr(Time()));
key_file.close();
      //---- Place thread code here ----
}
//***************************************************************
// ця  процедура для перевірки в векторі ключів на наявність залишку квадрату
// вона повертає також скільки нулів до наступного квадрату
bool factorizing_key_generation::is_true_square_modul(verylong modul,long *nnull){
      bool is=true;
      long curent;
            for (long i=0;i<result.size();i++){
                  if (!result[i][zsmod(modul,result[i].size())]){
                  is=false;
                  *nnull=result_long[i][zsmod(modul,result[i].size())];
                  break;}
            if (max<i)max=i;
            };
      return is;
};
//***************************************************************
//---------------------------------------------------------------------
```

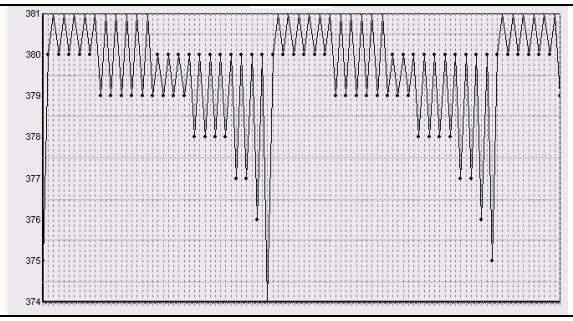
Додаток М

Моделі фракталів в околі рішень задачі факторизації

RSA	Біт	Моделі фракталів в околі рішень
100	330	
110	364	
120	397	
129	426	
130	430	

140	463	
150	496	
155	512	
160	530	
170	563	
576	576	

180	596	
190	629	
640	640	
200	663	
210	696	
704	704	

768	768		

Додаток Н
Реалізація пристрою факторизації на ПЛІС

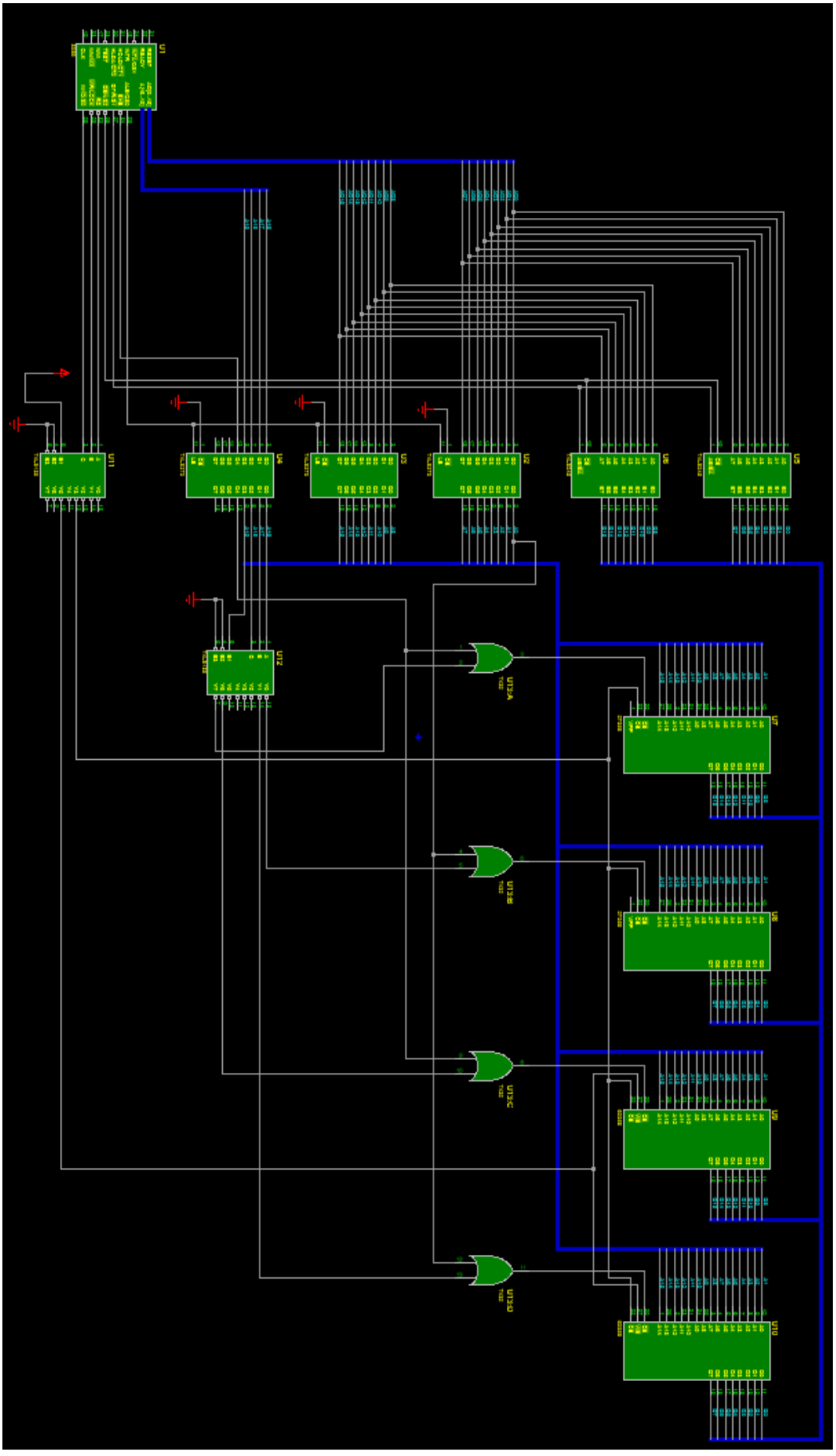
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_signed.all;
use ieee.numeric_std.all;
use ieee.numeric_bit.all;
use IEEE.std_logic_arith.all;
use IEEE.MATH_REAL.ALL;
use IEEE.MATH_complex.ALL;

entity comp is

        port (
        clk : in std_logic;
        p,m: in integer
        );

end comp;

architecture arch of comp is
signal p1 : integer :=0;
signal p2 : integer :=0;
signal pN : integer:=0;
signal sN : integer:=0;
signal q : integer:=0;
signal isi : std_logic_vector(0 to 36);
signal k : integer :=0;
signal ex : real := 1.0;
signal s : real := 1.0;
signal c : integer := 1;

        begin
 read: process(clk)
        begin
  if clk'event and clk='1' then
        while q < 2 loop
        if sqrt(real(p)) mod 1.0 > 0.5 then

   p1<= (integer(sqrt(real(p)))) + 1 ;
   if p1/=0 then
        p2 <= ((p1*p1)-p);
```

```
if p2 /=0 then
pN <= (2*p1) - 1;
sN <= p2 mod m;
end if;
      if pN/=0 then
 pN<= pN mod m;
 sN<= (pN+sN) mod m;
      q<=q+1;

      if q = 1   then


 if sN = 0 then
      isi(k)<='1';
      k<=k+1;
 elsif sN = 1 then
      isi(k)<='1';
      k<=k+1;
 elsif sN = 4 then
      isi(k)<='1';
      k<=k+1;
 elsif sN = 9 then
      isi(k)<='1';
      k<=k+1;
 else
      isi(k)<='0';
      k<=k+1;
 end if;
 sN<=sN+0;
      q<=q+2;
 end if;
      end if;
end if;
      else
      p1<= (integer(sqrt(real(p)))) + 2 ;
if p1/=0 then
      p2 <= ((p1*p1)-p);
 if p2 /=0 then
pN <= (2*p1) - 1;
sN <= p2 mod m;
end if;
      if pN/=0 then
 pN<= pN mod m;
 sN<= (pN+sN) mod m;
      q<=q+1;
```

```
      if q = 1   then


 if sN = 0 then
      isi(k)<='1';
      k<=k+1;
 elsif sN = 1 then
      isi(k)<='1';
      k<=k+1;
 elsif sN = 4 then
      isi(k)<='1';
      k<=k+1;
 elsif sN = 9 then
      isi(k)<='1';
      k<=k+1;
 else
      isi(k)<='0';
      k<=k+1;
 end if;
 sN<=sN+0;
      q<=q+2;
 end if;
      end if;
end if;

      end if;
        exit when q<2;
      end loop;
      while q = 3 loop
if k = m+1 then
      q<=q+1;
else

      while k/=m+1 loop
 pN<=(pN+2)mod m;
       sN<=(sN+pN+2)mod m;
 if sN = 0 then
      isi(k)<='1';
      k<=k+1;
 elsif sN = 1 then
      isi(k)<='1';
      k<=k+1;
 elsif sN = 4 then
      isi(k)<='1';
```

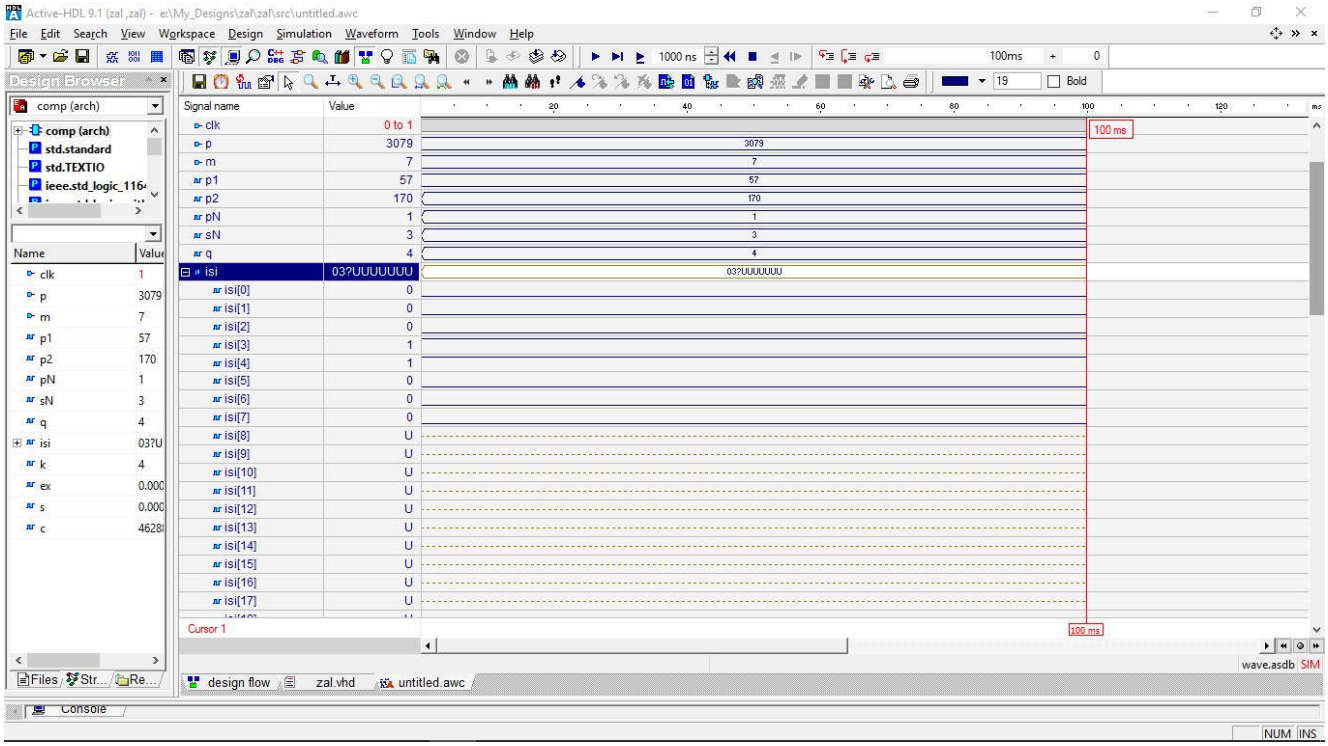
```
        k<=k+1;
    elsif sN = 9 then
        isi(k)<='1';
        k<=k+1;
    else
        isi(k)<='0';
        k<=k+1;
    end if;
    exit when k/=m+1;
    end loop;
   end if;
   exit when q = 3;
        end loop;
        while q = 4 loop
        while s /= 0.0  loop
      if k = 0 then
        k<=((k+1) mod (m+1));
        else
       if isi(k mod (m+1))='0' then
        k<=((k+1) mod (m+1));
        c<=(c+1);
       else
        ex<= sqrt(real(c*(2*(p1-1)+c)+p2)) ;
        s<= (sqrt(real(c*(2*(p1-1)+c)+p2)) mod 1.0);
        k<=((k+1) mod (m+1));
        c<=(c+1);
                end if;
      end if;
        exit when s/=0.0;
        end loop;
        exit when q = 4;
        end loop;

   end if;
 end process;
end arch;
```

Додаток П
Технологічна схема пристрою факторизації



Додаток Р

Результати тестування пристрою факторизації



Додаток Т
Акти впровадження результатів дисертаційної роботи

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ
«ТЕРНОПІЛЬСЬКЕ КОНСТРУКТОРСЬКЕ БЮРО РАДІОЗВ`ЯЗКУ «СТРІЛА»

46020, м. Тернопіль, вул 15 Квітня, 6, тел/факс – 28-75-00, 28-72-00, tkbr_strila@ukr.net, tkbr_strila@mail.ru
р/р 26009308101054 в ТВБВ 10019/08 Філія Тернопільського обласного управління Ощадбанку,
МФО 338545, код 14042350
№ 275 від 30.09.2015 р.

ЗАТВЕРДЖУЮ
Директор ТОВ ТКБР «Стріла»
_____ Рафалюк О.О.
"__" _____ 2016 р.

АКТ
про впровадження результатів дисертаційної роботи аспіранта кафедри спеціалізованих комп'ютерних систем Тернопільського національного економічного університету Івасьєва Степана Володимировича «Методи та обчислювальні засоби рішення задач теорії чисел в базисах Радемахера - Крестенсона»

Даний акт складений про те, що результати дисертаційної роботи Івасьєва С.В. «Методи та обчислювальні засоби рішення задач теорії чисел в базисах Радемахера - Крестенсона» передані для впровадження ТОВ ТКБР «Стріла», що включають:

1. Високопродуктивні алгоритми опрацювання інформаційних потоків: знаходження залишку, факторизації багаторозрядних чисел, генерування та компактного зберігання простих чисел з метою захисту передачі інформації по радоканалах зв'язку.
2. Схемотехнічні рішення спецпроцесорів розробленого способу факторизації багаторозрядних чисел на основі удосконаленого алгоритму Ферма, що реалізується у системі числення залишкових класів теоретико-числового базису Крестенсона та забезпечує суттєве зменшення обчислювальної складності при опрацюванні чисел розрядністю 512-1024 біт з метою вибору оптимально стійкого ключа.

Розроблені програмні та схемотехнічні рішення будуть впроваджені для забезпечення необхідного рівня захисту при передачі керуючої інформації каналами зв'язку при виконанні робіт по захисту передачі інформації у системах контролю керування електричних підстанцій.

Головний інженер Піскун С.О.

Товариство з обмеженою відповідальністю
" Інтеграл "

Україна, 46010, м. Тернопіль
вул. Текстильна, 28
ЗКПО 22604135 тел. 43-32-73

р\р 26001060181824 в
Тернопільській Філії ПАТ КБ
"ПриватБанк" МФО 338783

АКТ

про впровадження результатів дисертаційної роботи аспіранта кафедри спеціалізованих комп'ютерних систем Тернопільського національного економічного університету Івасьєва Степана Володимировича «Методи та обчислювальні засоби рішення задач теорії чисел в базисах Радемахера - Крестенсона»

Даний акт складений про те, що результати дисертаційної роботи Івасьєва С.В. «Методи та обчислювальні засоби рішення задач теорії чисел в базисах Радемахера - Крестенсона» передані для впровадження ТОВ «ІНТЕГРАЛ», що включають:

Високопродуктивні алгоритми опрацювання багаторозрядних чисел, а саме алгоритми: знаходження залишку, факторизації багаторозрядних чисел, генерування та компактного зберігання простих чисел для підбору взаємно простих модулів модифікованої та досконалої форми залишкових класів.

Схемотехнічні рішення спецпроцесорів кодування та компактного зберігання послідовності простих чисел, піднесення до квадрату, факторизації багаторозрядних чисел при виконанні пристроїв підбору взаємно простих модулів системи числення залишкових класів в реальному часі.

Передані для впровадження схемотехнічні рішення використані для забезпечення необхідного рівня захисту у спеціалізованих комп'ютерних системах генерування цифрового підпису та систем обробки цифрових даних реального часу.

Директор О. С. Колос

«ЗАТВЕРДЖУЮ»
Першому проректору
Тернопільського національного
економічного університету
Шинкарику М.І.
_____
«____»_____2016 р.

АКТ
про впровадження результатів дисертаційної роботи
Івасьєва Степана Володимировича
«Методи та обчислювальні засоби рішення задач теорії чисел в базисах Радемахера - Крестенсона»

Комісія у складі декана факультету комп'ютерних інформаційних технологій Тернопільського національного економічного університету (ТНЕУ), д.т.н., проф. Дивака Миколи Петровича (голова комісії), завідувача кафедри комп'ютерної інженерії ТНЕУ, д.т.н., проф. Березького Олега Миколайовича (член комісії), к.т.н., доц. кафедри комп'ютерної інженерії ТНЕУ Якименка Ігора Зіновійовича (член комісії) підтверджує, що результати кандидатської дисертації Івасьєва Степана Володимировича впроваджені і використовуються в навчальному процесі на кафедрі комп'ютерної інженерії Тернопільського національного економічного університету при вивчені дисциплін: «Захист інформації в комп'ютерних системах» та «Комп'ютерна криптографія» для студентів спеціальності 8.05010201 «Комп'ютерні системи та мережі», а саме:

- метод перевірки n-розрядного числа на простоту;
- метод виявлення квадратичних лишків багато розрядних чисел;
- метод факторизації на основі використання квадратичних лишків та системи залишкових класів;
- вдосконалений спосіб зменшення розрядності представлення двійкових чисел в алгоритмі Ферма;
- векторно-модульний алгоритм модулярного множення на основі обчислювальних операцій в теоретико-числовому базисі Радемахера – Крестенсона.

Декан факультету комп'ютерних інформаційних технологій ТНЕУ
д.т.н., проф. Дивак М.П.

завідувач кафедри комп'ютерної інженерії ТНЕУ,
д.т.н., проф. Березький О.М.

доцент кафедри кафедри комп'ютерної інженерії ТНЕУ,
к.т.н., доц. Якименко І.З.

«ЗАТВЕРДЖУЮ»
Проректор з науково-педагогічної
роботи Тернопільського
національного економічного
університету
проф. Задорожній З.-М.В.
_____
«____»_____2016 р.

АКТ
про впровадження результатів дисертаційної роботи викладача
кафедри комп'ютерної інженерії
Івасьєва Степана Володимировича «Методи та обчислювальні засоби рішення задач теорії чисел в базисах Радемахера - Крестенсона» у науково-дослідній роботі на тему:
«Опрацювання багаторозрядних чисел в системі залишкових класів»

Комісія у складі завідувача кафедри спеціалізованих комп'ютерних систем Тернопільського національного економічного університету (ТНЕУ), наукового керівника науково-дослідної роботи, д.т.н., проф. Николайчука Я.М., к.т.н. доц. Сегін А.І. та начальника науково-дослідний інституту інноваційного розвитку (НДІ ІРД) ТНЕУ д.т.н., доцента Длугопольського О. В., створена для приймання роботи, виконаної в рамках тематичного плану науково-дослідних робіт ТНЕУ на 2015 р. на тему "Розробка теоретичних засад методів формування та цифрового опрацювання даних у розподілених спеціалізованих комп'ютерних системах" (державний реєстраційний номер 0112U008458), встановила:

1. Розроблені ефективні методи пошуку залишків багаторозрядних чисел в теоретико-числовому базисі Радемахера, перевірки багаторозрядних чисел на простоту, визначення квадратичних лишків на основі використання властивостей системи залишкових класів, на відміну від існуючих, дозволили зменшити часову складність виконання відповідних операцій на 1-2 порядки.

2.Розроблені високопродуктивні програмні та апаратні засоби, які реалізовують розроблені методи.

Завідувач кафедри спеціалізованих
комп'ютерних систем ТНЕУ,
науковий керівник
науково-дослідної роботи,
д.т.н., професор Николайчук Я.М.

доцент кафедри кафедри спеціалізованих
комп'ютерних систем ТНЕУ,
к.т.н., доц. Сегін А.І.

Директор НДІ ІРД
д.е.н., доцент Длугопольський О. В.

«ЗАТВЕРДЖУЮ»
Першому проректору
Тернопільського національного
економічного університету
Шинкарику М.І.
_____
«____»_____ 2016 р.

АКТ
про впровадження результатів дисертаційної роботи
Івасьєва Степана Володимировича
«Методи та обчислювальні засоби рішення задач теорії чисел в базисах Радемахера - Крестенсона»

Комісія у складі декана факультету комп'ютерних інформаційних технологій Тернопільського національного економічного університету (ТНЕУ), д.т.н., проф. Дивака Миколи Петровича (голова комісії), завідувача кафедри спеціалізованих комп'ютерних систем ТНЕУ, д.т.н., проф. Николайчука Ярослава Миколайовича (член комісії), к.т.н., доц. кафедри спеціалізованих комп'ютерних систем ТНЕУ Сегіна Андрія Ігоровичв (член комісії) підтверджує, що результати кандидатської дисертації Івасьєва Степана Володимировича впроваджені і використовуються в навчальному процесі на кафедрі спеціалізованих комп'ютерних систем Тернопільського національного економічного університету при вивчені дисциплін: «Спецпроцесори в різних теоретико-числових базисах» та «Програмне забезпечення СКС» для студентів спеціальності 8.05010203 «Спеціалізовані комп'ютерні системи», а саме:

- метод факторизації великорозрядних чисел у базисі Радемахера та Хаара-Крестенсона;
- метод компактного кодування та генерування значних за об'ємом масивів великорозрядних простих чисел;
- метод представлення розв'язку задачі факторизації добутку багаторозрядних простих чисел в околі рішення моделями фракталів та кодових матриць у теоретико-числовому базисі Радемахера;
- вдосконалений спосіб зменшення розрядності представлення двійкових чисел в алгоритмі Ферма;
- векторно-модульний алгоритм модулярного множення на основі обчислювальних операцій в теоретико-числовому базисі Радемахера – Крестенсона.

Декан факультету комп'ютерних інформаційних технологій ТНЕУ д.т.н., проф. Дивак М.П.

завідувач кафедри спеціалізованих комп'ютерних систем ТНЕУ, д.т.н., проф. Николайчук Я.М.

доцент кафедри спеціалізованих комп'ютерних систем ТНЕУ, к.т.н., доц. Сегін А.І.