Міністерство освіти і науки України
Харківський національний університет імені В.Н. Каразіна
Факультет комп’ютерних наук
Спеціальність 125 «Кібербезпека»
Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»
В.о. зав. кафедрою БІСТ
Ольга МЕЛКОЗЬОРОВА
_____
« » 2023 р.

Пояснювальна записка
до кваліфікаційної роботи магістра
на тему: «Реалізація криптоалгоритму RSA на базі спеціалізованого засобу обробки даних у системі залишкових класів»

оцінка « »
Голова ЕК
Олександр ЛЕМЕШКО _____

Керівник: д.т.н., професор
Краснобаєв В.А.
Рецензент: д.т.н., професор
Толстолузька О.Г.
Виконавець: студент групи КБ-61
Кінчик А.С.

Харків – 2023

РЕФЕРАТ

Пояснювальна записка містить 78 сторінок, 16 рисунків, 4 таблиці, 1 додаток, 6 джерел.

Метою дипломної роботи є розробка спеціалізованого засобу обробки даних у системі залишкових класів для реалізації алгоритму RSA.

Актуальність даної розробки обумовлена необхідністю створення ефективних реалізацій складних криптографічних алгоритмів, таких як RSA, для систем з обмеженими ресурсами. Використання можливостей системи залишкових класів дозволить значно підвищити продуктивність виконання алгоритму RSA на цільових платформах.

Розроблений спеціалізований засіб має реалізувати всі необхідні компоненти для ефективних обчислень в системі залишкових класів, а саме: систему модулів, спеціалізовані регістри та арифметичне ядро.

Кінцевою метою дипломної роботи є практична реалізація запропонованого засобу обробки даних з алгоритмом RSA та експериментальне підтвердження досягнення суттєвого приросту продуктивності в порівнянні з існуючими рішеннями.

Результатами проведеної роботи є розробка спеціалізованого засобу обробки даних на основі системи залишкових класів для оптимізованої реалізації алгоритму RSA. Цей засіб включає систему модулів, спеціалізовані регістри та арифметичне ядро. Розробка HDL-моделі системи обробки інформації в системі залишкових класів. Модель протестована та підтвердила працездатність. Вибір мікросхеми EPM3064ATC100-4 та аналіз її параметрів для практичної реалізації запропонованого рішення.

Ключові слова: КРИПТОГРАФІЯ, АЛГОРИТМ RSA, СИСТЕМА ЗАЛИШКОВИХ КЛАСІВ, СПЕЦІАЛІЗОВАНИЙ ЗАСІБ ОБРОБКИ ДАНИХ, HDL-МОДЕЛЬ, МІКРОСХЕМА, МОДЕЛЮВАННЯ.

ABSTRACT

The explanatory note contains 78 pages, 16 figures, 4 tables, 1 appendix, 6 sources.

The purpose of the thesis is to develop a specialized data processing tool in a residue number system for implementing the RSA algorithm.

The relevance of this development is due to the need to create effective implementations of complex cryptographic algorithms, such as RSA, for resource-constrained systems. Using the capabilities of the residue number system will significantly improve the performance of the RSA algorithm on target platforms.

The developed specialized tool must implement all the necessary components for efficient computations in the residue number system, namely: a modulus system, specialized registers and an arithmetic unit.

The ultimate goal of the thesis is the practical implementation of the proposed data processing tool with the RSA algorithm and experimental confirmation of achieving a significant performance gain compared to existing solutions.

The results of the work are the development of a specialized data processing tool based on the residue number system for an optimized implementation of the RSA algorithm. This tool includes a modulus system, specialized registers and an arithmetic unit. Development of an HDL model of residue number system information processing system. The model has been tested and confirmed to be functional. Selection of the EPM3064ATC100-4 chip and analysis of its parameters for the practical implementation of the proposed solution.

Key words: CRYPTOGRAPHY, RSA ALGORITHM, RESIDUE NUMBER SYSTEM, SPECIALIZED DATA PROCESSING TOOL, HDL MODEL, MICROCHIP, MODELING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА СИМВОЛІВ

RSA	-	Rivest–Shamir–Adleman. Це криптографічний алгоритм з відкритим ключем, який широко використовується для шифрування даних, електронного підпису та інших криптографічних застосувань.
СЗК	-	Система залишкових класів
AES	-	Advanced Encryption Standard. Це симетричний алгоритм, тобто для шифрування та розшифрування даних використовується один і той самий ключ.
ПЗК	-	Повна система залишкових класів
ПСЧ	-	Позиційні системи числення
ПЗП	-	Програмована логічна матриця
HDL	-	Hardware Description Language. Це мови опису апаратури, які використовуються для проектування та моделювання електронних систем.
СП	-	Спеціалізований процесор
ПЛІС	-	Програмована логічна інтегральна схема.
ASIC	-	Application-Specific Integrated Circuit. Це інтегральна схема спеціального призначення.
FPGA	-	Field-Programmable Gate Array. Це програмована користувачем вентильна матриця

ВСТУП

Криптографія є важливою галуззю інформатики, яка займається розробкою та дослідженням методів захисту інформації від несанкціонованого доступу. Одним з найпопулярніших криптографічних алгоритмів є алгоритм RSA, який використовується для шифрування та дешифрування даних.

Алгоритм RSA є криптографічно стійким, але він вимагає значних обчислювальних ресурсів для своєї реалізації. Це обмежує його використання в системах з обмеженими ресурсами, таких як смарт-карти та мобільні пристрої.

У сучасному світі все більше інформації обробляється на мобільних пристроях та інших системах з обмеженими ресурсами. Це вимагає розробки нових ефективних методів реалізації криптографічних алгоритмів, таких як алгоритм RSA.

Метою дослідження є розробка спеціалізованого засобу обробки даних у системі залишкових класів для реалізації алгоритму RSA. Криптографія є важливою галуззю інформатики, що охоплює розробку та аналіз методів захисту інформації від несанкціонованого доступу. Алгоритм RSA широко застосовується для шифрування даних, однак водночас є обчислювально вимогливим. Це обмежує його впровадження в системах з обмеженими ресурсами, наприклад смарт-картах та мобільних пристроях. Саме тому актуальною задачею є створення оптимізованих реалізацій RSA.

У роботі пропонується використати можливості системи залишкових класів для прискорення обчислень з великими числами, необхідних в алгоритмі RSA. Це досягається завдяки особливостям представлення та операцій над даними в цій системі. Розроблений спеціалізований засіб обробки включає систему модулів, регістри та високопродуктивне арифметичне ядро на основі залишкових класів.

Об'єктом досліджень обрано алгоритм RSA, а предметом - сам засіб обробки даних на базі системи залишкових класів. В роботі представлено HDL-модель такого засобу, а також виконано аналіз придатної мікросхеми для його практичної реалізації.

В роботі використовується комплекс методів дослідження. Зокрема, аналітичний метод застосовується при вивченні теоретичних засад криптографії та алгоритму RSA. Також аналізуються можливості системи залишкових класів для оптимізації обчислень. Метод моделювання використовується при розробці HDL-моделі спеціалізованого засобу цифрової обробки на основі системи залишкових класів. Експериментальний метод застосовується для оцінювання ефективності запропонованого рішення на основі результатів моделювання та аналізу обраної мікросхеми.

Результати досліджень мають як теоретичне, так і практичне значення. В теоретичному плані проведено ґрунтовний аналіз можливостей оптимізації алгоритмів криптографії за рахунок використання системи залишкових класів. Практичним результатом є розробка спеціалізованого засобу цифрової обробки та його HDL-модель для ефективної реалізації алгоритму RSA. Ця розробка може лягти в основу створення високопродуктивних криптографічних систем.

Для вирішення поставлених завдань будуть використовуватися наступні методи:

- Аналітичний метод для дослідження теоретичних основ криптографії, алгоритму RSA та системи залишкових класів.
- Метод програмування для розробки спеціалізованого засобу обробки даних у системі залишкових класів.
- Метод експериментального дослідження для оцінки ефективності реалізації алгоритму RSA на основі розробленого засобу.

Запропонований підхід дає змогу оптимізувати один з найпоширеніших алгоритмів криптографії - RSA. Його широке застосування для шифрування та електронного цифрового підпису даних у сферах електронної комерції,

безпеки державних інформаційних систем та багатьох інших вимагає ефективних рішень.

Проте складність обчислень у алгоритмі RSA з використанням великих чисел накладає суттєві обмеження на його апаратну реалізацію. Саме тут надзвичайно корисними є властивості системи залишкових класів, які полягають у можливості компактного представлення даних та високопаралельних обчислень.

Розроблений спеціалізований процесор на основі системи залишкових класів дозволяє ефективно реалізувати усі складові алгоритму RSA, зокрема, генерацію ключів, операції піднесення до степеня за модулем, модульного множення та інші. Гнучка структура процесора надає можливості його адаптації до різних практичних задач захисту інформації.

У наступних розділах дипломної роботи будуть розглянуті теоретичні основи криптографії та алгоритму RSA, можливості застосування системи залишкових класів для оптимізації обчислень, а також практичні аспекти розробки спеціалізованого процесора на основі цієї системи.

1 ТЕОРЕТИЧНІ ОСНОВИ

1.1 Криптографія та її основи

Криптографія має довгу історію, яка сягає ще до стародавніх часів. Одним з найвідоміших прикладів криптографії є шифр Цезаря, який використовувався в Стародавньому Римі.

У середні віки криптографія використовувалася для захисту секретних повідомлень від ворогів. Одним з найвідоміших прикладів криптографії середньовіччя є шифр Віжинера, який був розроблений у XVI столітті.

У XIX столітті криптографія стала більш складною, завдяки появі нових математичних методів. Одним з найважливіших винаходів XIX століття в галузі криптографії є алгоритм RSA, який був розроблений у 1977 році.

Сучасна криптографія використовує найсучасніші математичні методи для захисту інформації від несанкціонованого доступу. Криптографічні алгоритми широко використовуються в різних сферах, таких як електронна комерція, телекомунікації та державна безпека.

Криптографія - наука про розробку та дослідження методів захисту інформації від несанкціонованого доступу. Криптографія включає в себе два основних напрями:

- Криптографія - це наука про розробку і дослідження методів шифрування та дешифрування даних.
- Криптологія - це наука про дослідження методів розшифрування зашифрованих даних, тобто про протидію криптографії.

Основними завданнями криптографії є забезпечення наступних властивостей інформації:

- Конфіденційність - інформація доступна тільки авторизованим користувачам.
- Цілісність - інформація не може бути змінена або знищена без відома авторизованих користувачів.

- Аутентифікація - авторизовані користувачі можуть бути ідентифіковані.

Криптографічний алгоритм - це алгоритм, який перетворює дані з відкритого тексту в зашифрований текст і навпаки. Криптографічні алгоритми можна класифікувати за різними ознаками. Найбільш поширена класифікація за видом ключа:

- Симетричні криптографічні алгоритми використовують один і той же ключ для шифрування і дешифрування даних.
- Асиметричні криптографічні алгоритми використовують два ключі: відкритий ключ для шифрування даних і закритий ключ для дешифрування даних.

Ключ шифрування - це секретна інформація, яка використовується для шифрування і дешифрування даних. Без знання ключа шифрування зашифровані дані неможливо дешифрувати.

Вони можуть бути різних типів. Наприклад, асиметричні ключі шифрування, які використовуються в алгоритмах асиметричного шифрування, складаються з відкритого ключа і секретного ключа. Відкритий ключ може бути опублікований, а секретний ключ повинен зберігатися в секреті.

Симетричні ключі шифрування, які використовуються в алгоритмах симетричного шифрування, складаються з одного ключа. Цей ключ повинен бути відомий обом сторонам, які обмінюються зашифрованими даними.

Симетричні криптографічні алгоритми є більш швидкими, ніж асиметричні, але вони мають ряд недоліків, таких як необхідність розподілу ключа між користувачами.

Асиметричні криптографічні алгоритми не вимагають розподілу ключа між користувачами, що робить їх більш безпечними, ніж симетричні. Однак вони є більш повільними.

Симетричні криптографічні алгоритми є більш швидкими, ніж асиметричні, тому що вони використовують один ключ для шифрування і

дешифрування. Асиметричні криптографічні алгоритми використовують два ключі, що вимагає додаткових обчислень.

Одним з основних недоліків симетричних криптографічних алгоритмів є необхідність розподілу ключа між користувачами. Це може бути складним або небезпечним, якщо користувачі не довіряють один одному.

Іншими недоліками симетричних криптографічних алгоритмів є:

- Необхідність зберігання ключа в секреті. Якщо ключ буде загублений або скомпрометований, то будь-хто, хто його знає, зможе дешифрувати зашифровані дані.
- Обмеження довжини ключа. Для забезпечення достатнього рівня безпеки симетричні криптографічні алгоритми вимагають використання ключів довжиною не менше 128 біт. Це може призвести до зниження продуктивності, особливо на пристроях з обмеженими ресурсами.

Однією з сучасних тенденцій в криптографії є використання штучного інтелекту. Штучний інтелект може використовуватися для розробки нових криптографічних алгоритмів і для захисту криптографічних систем від атак.

Також є використання хмарних обчислень. Хмарні обчислення можуть використовуватися для розміщення криптографічних систем, що може підвищити їх безпеку [1-6].

1.2 Алгоритм RSA

Алгоритм RSA (Rivest-Shamir-Adleman) - це асиметричний алгоритм шифрування, який був розроблений в 1977 році Роном Рівестом, Аді Шаміром і Леонардо Адлеманом. Алгоритм RSA є одним з найпопулярніших алгоритмів шифрування, який використовується в таких сферах, як електронна комерція, телекомунікації та державна безпека.

Алгоритм RSA заснований на неможливості факторизації великих чисел. На практиці, це означає, що якщо у вас є два великі числа, які є множниками один одного, то дуже важко знайти ці числа, якщо ви не знаєте один з них.

Алгоритми факторизації, в залежності від складності, можна розбити на дві групи (рис. 1.1). Однією з найважливіших нерозв'язаних задач сучасної теорії чисел є питання про існування поліноміального алгоритму факторизації на класичних комп'ютерах. Наразі відомі дві основні групи алгоритмів для факторизації великих чисел: експоненціальні та субекспоненціальні.

Експоненціальні алгоритми мають складність, котра зростає експоненціально зі збільшенням довжини вхідного параметра – довжини самого числа у двійковому поданні. Для опису складності таких алгоритмів використовують О-нотацію, яка дозволяє враховувати лише найбільш значущі частини функції складності, ігноруючи другорядні деталі. Субекспоненціальні ж алгоритми мають складність, що перевищує поліноміальну, але є меншою за експоненціальну. Їх складність описують за допомогою L-нотації.

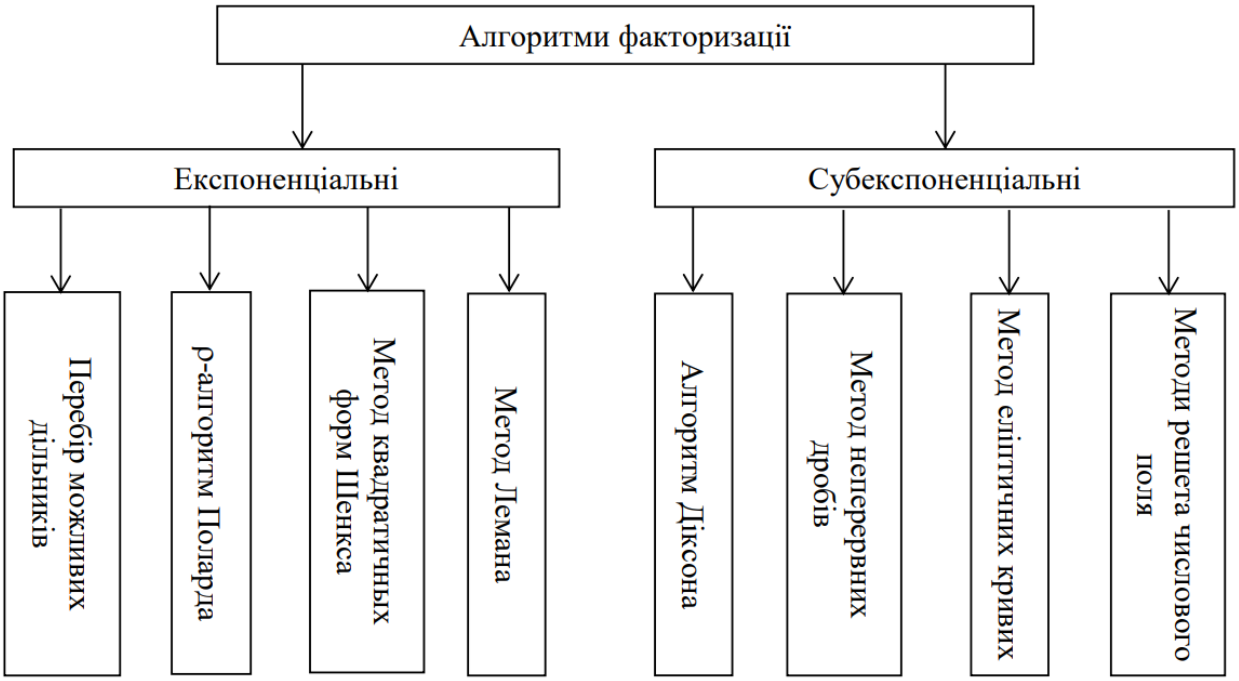

Рис. 1.1 – Класифікація алгоритмів факторизації

Обидві групи методів є досить трудомісткими, тож при факторизації чисел великої довжини потребують значних обчислювальних ресурсів. Проте, теоретичне обґрунтування складності таких обчислень, тобто існування нижніх границь складності, поки що не доведено. Відтак, питання про можливість створення поліноміального алгоритму факторизації залишається

відкритим. Його розв'язання матиме велике значення для розвитку сучасної теорії чисел.

Ця проблема є актуальною з огляду на широке застосування криптосистем з відкритим ключем, що ґрунтуються на складності розкладання великих чисел на множники. Зокрема, популярний стандарт шифрування RSA використовує добуток двох великих простих чисел як відкритий ключ. Якби існував ефективний поліноміальний алгоритм факторизації, він дозволив би швидко знаходити ці множники та ламати шифрування RSA. Тож створення такого алгоритму матиме суттєві наслідки для криптографії та кібербезпеки.

Алгоритм RSA працює наступним чином:

- Відправник генерує пару ключів: відкритий ключ і секретний ключ.
- Відправник публікує відкритий ключ.
- Одержувач зберігає секретний ключ.
- Відправник шифрує повідомлення за допомогою відкритого ключа одержувача.
- Одержувач дешифрує повідомлення за допомогою секретного ключа.

Алгоритм RSA складається з наступних етапів:

1) Генерація ключів: Відправник генерує пару ключів: відкритий ключ і секретний ключ. Відкритий ключ публікується, а секретний ключ зберігається в таємниці.
2) Шифрування: Відправник шифрує повідомлення за допомогою відкритого ключа одержувача.
3) Дешифрування: Одержувач дешифрує повідомлення за допомогою секретного ключа.

На першому етапі відправник генерує пару ключів: відкритий ключ і секретний ключ. Відкритий ключ публікується, а секретний ключ зберігається в таємниці.

Відкритий ключ складається з двох чисел: n і e. Число n є добутком двох великих простих чисел p і q. Число e є випадковим числом, яке є відносно першим до $(p-1)(q-1)$.

Секретний ключ складається з двох чисел: p і q, а також d. Число d є таким, що ed ділиться на $(p-1)(q-1)$.

На другому етапі відправник шифрує повідомлення за допомогою відкритого ключа одержувача.

Для шифрування повідомлення відправник використовує алгоритм RSA, щоб помножити повідомлення на e. Отримане число є зашифрованим повідомленням.

На третьому етапі одержувач дешифрує повідомлення за допомогою секретного ключа.

Для дешифрування повідомлення одержувач використовує алгоритм RSA, щоб помножити зашифроване повідомлення на d. Отримане число є дешифрованим повідомленням.

Безпека алгоритму RSA заснована на тому, що дуже важко факторизувати великі числа. На сьогоднішній день невідомо жодного алгоритму, який міг би ефективно факторизувати числа розміром більше 2000 цифр.

Факторизація чисел - це процес розкладання числа на його прості множники. Наприклад, число 12 можна факторизувати як $2 \times 2 \times 3$.

Алгоритм RSA використовує факторизацію чисел для шифрування і дешифрування даних. Відправник шифрує повідомлення, помножуючи його на відкритий ключ одержувача. Одержувач дешифрує повідомлення, помножуючи зашифроване повідомлення на секретний ключ.

Безпека алгоритму RSA полягає в тому, що для розшифрування повідомлення без секретного ключа необхідно спочатку факторизувати відкритий ключ. Це дуже складне завдання, оскільки відкритий ключ являє собою добуток двох великих простих чисел[16-21].

Існують деякі атаки на алгоритм RSA, які можуть бути ефективними при використанні великої кількості обчислювальних ресурсів. Ці атаки є теоретичними, але вони є потенційною загрозою для безпеки алгоритму RSA.

Для підвищення безпеки алгоритму RSA використовуються ключі великого розміру. Чим більше розмір ключів, тим складніше їх факторизувати, і тим безпечнішим є алгоритм RSA. На сьогоднішній день найбільш поширеними розмірами ключів RSA є 1024, 2048 і 4096 біт.

Крім того, для підвищення безпеки алгоритму RSA використовуються різні методи захисту від атак. До них відносяться:

- Захист ключів: Ключі алгоритму RSA повинні бути захищені від несанкціонованого доступу.
- Захист даних: Дані, які шифруються за допомогою алгоритму RSA, також повинні бути захищені від несанкціонованого доступу.
- Захист від атак: Для захисту від атак на алгоритм RSA використовуються різні методи, такі як контроль доступу і виявлення атак.

Алгоритм RSA має ряд переваг, таких як:

- Безпека: Є дуже безпечним, оскільки його безпека заснована на неможливості факторизації великих чисел.
- Швидкість: Є відносно швидким, як для шифрування, так і для дешифрування.
- Ефективність: Є ефективним у використанні обчислювальних ресурсів.

Недоліки:

- Обмін ключами: Для використання алгоритму необхідно обмінюватися ключами між відправником і одержувачем. Це може бути складно або небезпечно, якщо користувачі не довіряють один одному.
- Довжина ключів: Довжина ключів повинна бути досить великою, щоб забезпечити достатній рівень безпеки. Це може призвести до

зниження продуктивності, особливо на пристроях з обмеженими ресурсами.

Широко використовується в таких сферах, як:

- Електронна комерція: Використовується для захисту платежів і особистих даних в електронній комерції. Наприклад, при оплаті товарів або послуг в Інтернеті дані про платіж шифруються за допомогою алгоритму RSA, щоб запобігти їх перехопленню і крадіжці.
- Телекомунікації: Використовується для захисту даних в телекомунікаціях, таких як електронна пошта і VPN. Наприклад, повідомлення електронної пошти можуть бути зашифровані за допомогою алгоритму RSA, щоб запобігти їх перехопленню і читанню сторонніми особами.
- Державна безпека: Використовується для захисту можуть бути зашифровані за допомогою алгоритму RSA, щоб запобігти їх потраплянню в руки ворога.

Алгоритм RSA є одним з найпопулярніших і найбезпечніших алгоритмів шифрування. Він використовується в багатьох сферах, де важливо захистити інформацію від несанкціонованого доступу. Однак, має деякі недоліки, які можна покращити за рахунок використання нових методів і технологій[21-24].

1.3 Підвищення продуктивності алгоритму RSA за рахунок апаратної реалізації

Алгоритм RSA є одним з найбільш поширених асиметричних криптографічних алгоритмів, що використовується для шифрування та цифрового підпису. Проте обчислювальна складність цього алгоритму є досить значною. Тому актуальною задачею є підвищення продуктивності RSA за рахунок оптимізації його програмної та апаратної реалізації.

Одним з ефективних підходів до прискорення RSA є використання спеціалізованих процесорів та сопроцесорів, що реалізовують ключові операції алгоритму на апаратному рівні. Зокрема, найбільш

ресурсовитратними в RSA є операції піднесення до степеня за модулем, модульне множення та модульне зведення. Їх апаратна реалізація дозволяє суттєво прискорити роботу алгоритму порівняно із програмною реалізацією на універсальних процесорах.

Існують різні підходи до побудови спеціалізованих процесорів RSA, зокрема на основі застосування програмованих користувачем вентильних матриць, що надають гнучкість у налаштуванні архітектури. Вибір оптимальної архітектури та її параметрів потребує ретельного аналізу особливостей конкретної задачі шифрування чи підпису.

Ефективна апаратна реалізація RSA в спеціалізованих процесорах дозволяє досягти приросту продуктивності на 1-2 порядки порівняно з оптимізованою програмною реалізацією. Це робить такі процесори надзвичайно привабливими для застосування в системах, що потребують високої швидкодії та пропускної спроможності операцій шифрування і цифрового підпису.

Одним з перспективних напрямків подальшого підвищення продуктивності апаратної реалізації RSA є використання системи залишкових класів для представлення даних і виконання обчислень. На відміну від традиційного двійкового подання, система залишкових класів дозволяє компактніше представляти великі числа, що істотно зменшує апаратні витрати.

Операції з великими числами в системі залишкових класів можуть ефективно реалізовуватися на основі операцій над залишками. Це спрощує апаратну реалізацію та підвищує швидкодію обчислень. Завдяки цьому стає можливим побудувати високопродуктивний спеціалізований процесор RSA безпосередньо в системі залишкових класів.

Актуальним завданням є дослідження особливостей архітектур таких процесорів, розробка їх структурних та функціональних схем, а також HDL-моделей для наступної апаратної реалізації. Важливу роль відіграє вибір

оптимального базового модуля та ступеня розрядності системи залишкових класів з огляду на параметри конкретної задачі шифрування.

Реалізація спеціалізованого процесора RSA в системі залишкових класів на ПЛІС дозволить досягти максимально можливої продуктивності цього алгоритму в апаратному виконанні. Переваги такого підходу особливо яскраво проявляться при побудові високопродуктивних систем шифрування та електронного цифрового підпису, що актуально для багатьох галузей.

Прискорення обчислень в алгоритмі RSA за рахунок апаратної реалізації відкриває нові можливості для побудови високоефективних систем захисту інформації. Зокрема, актуальним є використання таких рішень в сучасних мережах 5G, де шифрування трафіку відіграє ключову роль.

Спеціалізовані процесорні рішення дозволяють розвантажити центральний процесор мережевого обладнання від ресурсовитратних криптографічних обчислень. Це підвищує пропускну здатність і скорочує затримки при обробці великих потоків даних. Особливо ефективним є спільне використання універсальних і спеціалізованих процесорів у гібридних архітектурах.

Створення процесорів RSA з апаратною підтримкою стандартів шифрування, що застосовуються в мережах 5G - таких як 5G NR Encryption Algorithm забезпечить максимальну продуктивність на ключових алгоритмах без втрати гнучкості та можливості оновлення. Актуальна розробка спеціалізованих чипів чи систем на кристалі, що інтегрують на одному кристалі процесорні ядра і апаратні прискорювачі RSA та інших алгоритмів. Такі рішення стануть основою побудови високоефективного мережевого устаткування нового покоління для систем зв'язку, Інтернету речей тощо на базі 5G.

Системи обробки транзакцій з використанням технології блокчейн. Забезпечення високої продуктивності операцій шифрування та електронного цифрового підпису є критично важливим для масштабованих розподілених реєстрів, що можуть обробляти сотні тисяч транзакцій за секунду.

Інтеграція спеціалізованих процесорів RSA безпосередньо в архітектуру вузлів блокчейну дозволяє розвантажити центральний процесор та оптимізувати виконання консенсусних алгоритмів. За рахунок локального прискорення криптографії може бути досягнуте суттєве підвищення загальної продуктивності та масштабованості системи.

Реалізація спеціалізованих чипів, до складу яких входитимуть як процесорні ядра загального призначення, так і високопродуктивні апаратні прискорювачі для операцій шифрування та цифрового підпису. Такі чипи стануть основою для створення нового покоління вузлів блокчейн-інфраструктури як для публічних, так і для приватних розподілених реєстрів.

Апаратна реалізація алгоритмів шифрування набуває все більшого значення в контексті розвитку квантових обчислень. Оскільки більшість існуючих криптосистем не є стійкими до атак з використанням потужних квантових комп'ютерів, гостро постає питання модернізації механізмів захисту інформації.

Впровадження квантовостійких криптоалгоритмів, таких як схеми шифрування McEliece та NTRUEncrypt. Їх реалізація вимагає ефективних обчислень з великими цілими числами, що може бути забезпечено саме апаратними засобами[11-17].

Розробка спеціалізованих процесорів чи сопроцесорів для прискорення операцій з великими числами стане важливим фактором масштабування та практичного застосування перспективних квантовостійких криптосистем. Це особливо важливо для впровадження таких рішень в інфраструктуру хмарних сервісів та центрів обробки даних.

Розвиток апаратно реалізованих криптосистем є однією з ключових складових побудови мереж і сервісів з квантовим захистом інформації в недалекому майбутньому. Від ефективності цих рішень залежатиме стійкість систем до загроз з боку квантових технологій.

Апаратна реалізація алгоритмів асиметричної криптографії, зокрема RSA, відкриває нові горизонти у сфері захищених вбудованих систем та

Інтернету речей. Ресурсна обмеженість сенсорних вузлів та мікроконтролерів у цих системах ускладнює реалізацію складних криптопротоколів програмними методами.

Використання компактних та енергоефективних апаратних IP-блоків дає змогу інтегрувати модулі шифрування, аутентифікації та цифрового підпису безпосередньо до складу мікросхем кінцевих пристроїв Інтернету речей. Це істотно розширює функціональність та підвищує захищеність таких систем без збільшення вартості та енергоспоживання.

Створення гібридних CPU + GPU чипів, які поєднують в собі процесорне ядро загального призначення та спеціалізований сопроцесор шифрування. Така архітектура оптимально поєднує гнучкість та продуктивність, що ідеально підходить для широкого кола IoT застосувань - від промислових контролерів до медичних пристроїв. Захист інформації в таких системах набуває все більш вирішальне значення.

Апаратна реалізація криптоалгоритмів набуває стрімкого розвитку в контексті розгортання хмарних сервісів та інфраструктури. Хмарні провайдери потребують надзвичайно високої продуктивності операцій шифрування та розшифрування для забезпечення безпеки даних величезної кількості користувачів в мульти-тенантних середовищах.

Використання високопродуктивних апаратних прискорювачів дає змогу розвантажити серверні процесори від навантаження при обробці HTTPS трафіку, шифруванні баз даних, реалізації VPN тунелів тощо. При цьому досягається значне підвищення пропускної спроможності хмарної інфраструктури. Впровадження спеціалізованих чипів та плат розширення для прискорення RSA, AES, Elliptic Curve криптографії на рівні окремих серверів та мережевих елементів хмари дозволить забезпечити гарантований рівень криптографічного захисту на апаратному рівні для задоволення найсуворіших вимог щодо безпеки даних корпоративних клієнтів хмарних провайдерів. Впровадження апаратно прискорених криптоалгоритмів та процесорів у хмарній інфраструктурі вимагає комплексного підходу, що поєднує як

розробку спеціалізованих мікросхем, так і інтеграцію з хмарними платформами[12-19].

З боку розробників мікросхем актуальним є створення гнучких і масштабованих архітектур процесорів RSA та інших алгоритмів з підтримкою провідних API і стеків програмування. Це спростить їх інтеграцію і використання в хмарних додатках.

Провайдери хмарних сервісів, в свою чергу, повинні адаптувати свої платформи для ефективного розпаралелювання криптозадач і виконання їх на гетерогенних обчислювальних системах (CPU + FPGA + GPU + крипто-співпроцесори). Для цього знадобиться модернізація ПЗ стеку – гіпервізорів, ОС, диспетчерів задач тощо.

Спільними зусиллями розробників можна досягти синергетичного ефекту, коли продуктивність апаратних прискорювачів криптографії розкривається на повну потужність завдяки інтелектуальним механізмам планування та розподілу навантаження в хмарних дата-центрах нового покоління. Це забезпечить якісно новий рівень швидкодії і масштабованості криптографічного захисту даних в хмарі.

Використання апаратної реалізації RSA та інших асиметричних криптоалгоритмів є системи штучного інтелекту. Нейромережі, що навчаються на конфіденційних наборах даних, потребують надійного захисту як самих даних, так і моделей та результатів навчання.

Вбудовані спеціалізовані співпроцесори шифрування дозволять реалізувати ефективне homomorphic encryption – технологію, котра уможливлює виконання обчислень над зашифрованими даними без попереднього розшифрування. Це критично важливо для забезпечення конфіденційності в системах машинного навчання.

Апаратні засоби криптографії можуть застосовуватись в AI-чипах та інтелектуальних сенсорах для захисту зібраних ними даних та моделей. Гнучке поєднання на одному кристалі процесорних ядер, нейромережевих і

криптографічних прискорювачів стане запорукою побудови надійних та продуктивних систем штучного інтелекту майбутнього.

Подальше підвищення продуктивності апаратної реалізації асиметричних криптоалгоритмів пов'язане з розвитком технологій систем на кристалі. Інтеграція на одному чипі спеціалізованих процесорів RSA та інших алгоритмів з процесорними ядрами загального призначення, локальною пам'яттю та інтерфейсами дозволить створювати комплексні рішення для систем захисту інформації.

Використання 3D-стекінг технологій, що дають змогу об'єднувати кілька кристалів в одному корпусі з високошвидкісним міжчиповим з'єднанням. Це уможливить побудову ієрархічних архітектур, де різні функціональні блоки реалізуються на окремих логічних рівнях. Зокрема, можливою є реалізація протоколів TLS, IPsec, SSH тощо шляхом розподілу відповідних операцій між процесорними, криптографічними і мережевими ядрами в стекованій 3D-системі на кристалі. Така архітектура забезпечить максимально можливу продуктивність криптографічних протоколів за рахунок паралельності та локальності обчислень.

Нарощування продуктивності апаратної реалізації RSA та інших алгоритмів асиметричної криптографії пов’язане з розвитком технологій квантових обчислень. Хоча повноцінні квантові комп’ютери, здатні зламати RSA, ще далекі від практичної реалізації, апаратні рішення для квантової криптографії вже є перспективним напрямом досліджень[9-11].

Розробляються оптичні системи генерації та детектування фотонів для квантового розподілу ключів, а також спеціалізована електроніка для керування такими системами. Поєднання квантових генераторів випадкових чисел і апаратних криптопроцесорів дасть змогу реалізувати принципово новий рівень захисту конфіденційності та цілісності інформації.

Результати досліджень свідчать, що застосування апаратурних прискорювачів дозволяє підвищити продуктивність RSA в десятки і сотні разів порівняно із традиційною програмною реалізацією на універсальних

процесорах. Це робить апаратне прискорення невід'ємною складовою побудови високопродуктивних систем криптографічного захисту інформації в таких галузях, як телекомунікації, хмарні обчислення, блокчейн, Інтернет речей тощо.

Незважаючи на досягнуті успіхи, залишається низка актуальних завдань щодо подальшої оптимізації апаратної реалізації RSA. Передусім, потрібні додаткові дослідження щодо підвищення енергоефективності та зменшення апаратурних витрат спеціалізованих процесорів шляхом удосконалення їх архітектури. Адже для багатьох практичних застосувань в енергообмежених системах критично важливими є саме ці показники. Також, потребує вирішення проблема інтеграції апаратно прискорених криптопроцесорів в існуючі обчислювальні платформи та програмні стеки. Хоча окремі спеціалізовані чипи демонструють вражаючу продуктивність, їх ефективне застосування в реальних системах часто гальмується складнощами вбудовування та узгодження роботи з іншими компонентами. Тому важливою задачею є розробка уніфікованих інтерфейсів та API для прискорювачів RSA.

Незважаючи на досягнуті успіхи в галузі апаратного прискорення асиметричної криптографії, подальший розвиток цього напряму потребує комплексних зусиль, що поєднують дослідження в галузі мікроелектроніки, архітектури комп’ютерів, програмного забезпечення та кібербезпеки. Лише синергетична взаємодія фахівців в різних предметних областях дасть змогу в повній мірі реалізувати потенціал апаратної реалізації складних криптоалгоритмів для побудови високопродуктивних та надійних систем захисту інформації.

2 СИСТЕМА ЗАЛИШКОВИХ КЛАСІВ

2.1 Основи системи залишкових класів

Система залишкових класів (СЗК) - це множина, яка складається з усіх можливих залишків від ділення цілих чисел на деяке число n. Ця множина зазвичай позначається як Z_n.

Елементами СЗК є числа $0, 1, 2, \dots, n-1$. Кожен елемент СЗК можна представити у вигляді залишку від ділення цілого числа на n. Наприклад, якщо n=5, то елементи СЗК будуть наступними:

$$Z_5 = \{0,1,2,3,4\} \tag{2.1}$$

У СЗК можна виконувати такі дії:

- Додавання:

$$a + b \equiv (a + b)\ (mod\ n)\ \} \tag{2.2}$$

- Віднімання:

$$a - b \equiv (a - b)\ (mod\ n)\ \} \tag{2.3}$$

- Множення:

$$a \cdot b \equiv (a \cdot b)\ (mod\ n)\ \} \tag{2.4}$$

- Ділення:

$$a \div b \equiv a \cdot b - 1\ (mod\ n) \tag{2.6}$$

Нерівність в СЗК виконується, якщо різниця між двома числами ділиться на n. Наприклад, якщо n = 5, то наступні нерівності виконуються:

$$1 < 2\ (mod\ 5) \tag{2.7}$$

$$3 < 4\ (mod\ 5) \tag{2.8}$$

$$1 < 4\ (mod\ 5) \tag{2.9}$$

Усяка обчислювальна структура тісно пов'язана із системою числення, у якій вона працює. Під системою числення розуміється сукупність прийомів для позначення (або запису) чисел, чи точніше, спосіб кодування (або подання) елементів деякої кінцевої моделі дійсних чисел словами одного або більше алфавітів. Кодування являє собою ін’єктивне відображення діапазону

системи числення на декартовий добуток його алфавітів, тобто $F : D \rightarrow A$, де $A = A \times A2 \times \ldots \times A_j$, тобто відображення F елементу x $\in D$ ставить у відповідність кодове слово $(x_1, x_2, \ldots, x_j)$, де $x_i = A_i\ (i = 1, n) - i -$ а цифра, j – довжина коду. За допомогою зворотного відображення F^{-1}, яке називається декодуванням, також можна визначити систему числення.

Система залишкових класів є підходом до представлення та обробки даних, який використовує арифметику по модулю замість звичайної двійкової арифметики. В основі цього підходу лежить ідея розбиття множини цілих чисел на класи за залишком від ділення на деяке фіксоване число модуль m.

Кожне ціле число можна представити у вигляді $a = km + r$, де k - ціле число, r - залишок від ділення числа a на m. Число модуль m називається базою системи залишкових класів. У рамках системи залишкових класів з певним модулем m усі числа з однаковим залишком від ділення на цей модуль належать до одного класу. Таким чином, утворюється m класів залишків: $0, 1, 2, \ldots, m - 1$.

Обчислення в системі залишкових класів здійснюються шляхом виконання арифметичних операцій над представниками класів, в результаті чого отримуємо представники результуючих класів. Наприклад, множення двох чисел a та b дає число c з залишком r від ділення на модуль m, яке належить до класу з номером r. Отже, фактичні значення a та b при обчисленнях значення не мають - важливим є лише їх клас, тобто залишок від ділення на модуль[7-11].

У системи залишкових класів є можливість ефективної апаратної реалізації обчислень. Оскільки при обчисленнях оперують не великими цілими числами, а їх залишками від ділення на порівняно невеликий модуль, можна побудувати простіший і швидший арифметичний пристрій. Це дозволяє істотно прискорити виконання деяких обчислювальних алгоритмів, наприклад криптографічних перетворень.

Система залишкових класів - це ефективний підхід до організації обчислень, заснований на арифметиці по модулю та представленні чисел

класами залишків. Використання системи залишкових класів надає можливості для побудови високопродуктивних спеціалізованих обчислювальних пристроїв для реалізації складних алгоритмів, таких як алгоритми криптографічного перетворення даних.

Перевага використання системи залишкових класів для реалізації обчислень є можливість паралелізації. Адже оскільки кожен клас залишків є незалежним від інших, операції над різними класами можна виконувати паралельно без будь-якої синхронізації. Це відкриває широкі можливості для побудови високопродуктивних обчислювальних структур на основі системи залишкових класів з великим ступенем паралелізму.

Наприклад, для N класів залишків можна організувати N незалежних паралельних обчислювачів, кожен з яких здійснюватиме операції над даними свого класу. Такий підхід дозволяє масштабувати продуктивність системи практично лінійно зі збільшенням кількості паралельних обчислювачів. При цьому на продуктивність не впливатимуть затримки, пов'язані з синхронізацією та обміном даними, характерні для традиційних багатоядерних систем[15-20].

Практична реалізація такої масштабованої обчислювальної архітектури можлива з використанням, наприклад, ПЛІС як апаратної платформи. Сучасні ПЛІС містять тисячі логічних елементів і дозволяють ефективно реалізувати десятки і сотні ядер-обчислювачів, працюючих паралельно. Використання концепції системи залишкових класів разом з паралельною архітектурою на ПЛІС дає можливість побудувати високопродуктивну систему для прискорення складних обчислювальних задач, таких як операції криптографічного перетворення даних великого обсягу.

Зважаючи на особливості обчислень в системі залишкових класів, виникає питання організації вводу-виводу даних при побудові спеціалізованих обчислювачів. Адже зовнішні дані зазвичай подаються у звичайному двійковому представленні, тоді як обчислювач оперує класами залишків. В такому разі, необхідно здійснити перетворення даних.

Для вводу даних до системи залишкових класів використовується блок перетворення двійкових чисел у їх залишки по заданому модулю. Кожне вхідне число розбивається на дві частини - частку та залишок. Частка відкидається, а залишок і є представленням числа у відповідному класі залишків.

Обернене перетворення виконується при виведенні даних із системи. Тут залишок відновлюється до повного двійкового представлення шляхом домноження на модуль та додавання того самого залишку.

Для прискорення процесів перетворення даних можуть використовуватися спеціалізовані швидкодіючі модулі, реалізовані апаратно. Наприклад, на базі ПЛІС можна ефективно побудувати високопродуктивний перетворювач "двійкове число - клас залишку" завдяки наявності великої кількості логічних елементів та можливостей паралелізації.

Такий підхід дозволяє оптимально поєднати швидкодію системи залишкових класів та зручність подання зовнішніх даних у двійковому вигляді. Загалом архітектура спеціалізованого обчислювача на основі залишкових класів включає в себе три основних компоненти: вхідний перетворювач даних, високопродуктивний ядро-обчислювач та вихідний перетворювач.

Важливим при побудові спеціалізованих процесорів на основі системи залишкових класів є вибір оптимального значення модуля m. Цей модуль визначає кількість класів залишків, а отже й ступінь паралелізму та продуктивності системи.

З одного боку, великі значення m дозволяють організувати велику кількість паралельних обчислювачів за класами. Але з іншого боку, надмірно великий модуль ускладнює апаратну реалізацію окремих обчислювачів залишків, оскільки потрібно оперувати з великими числами.

Тому на практиці найчастіше для системи залишкових класів обирають модуль m розміром $8,\ 16, 32$ або 64 біти. Це забезпечує компроміс між кількістю класів (ступенем паралелізму) та складністю обчислювача залишку.

Наприклад, при $m = 32$ біти отримуємо $232 = 4294967296$ класів залишків. А обчислювач залишку оперує 32-бітовими даними, що є прийнятним для апаратної реалізації.

Використання змішаної системи залишкових класів з різними модулями al та bl на різних рівнях. Наприклад, загальна система може мати 232 класів (модуль $a = 32$ біти), а кожен окремий клас розбиватися додатково на 216 підкласів (модуль $b = 16$ біт).

Така ієрархічна організація дозволяє гнучко масштабувати ступінь паралелізму шляхом збільшення кількості підкласів. Водночас апаратна реалізація спрощується завдяки невеликому базовому модулю 16 біт для обчислювачів підкласів. Вибір оптимального модуля або системи модулів є важливою оптимізаційною задачею при проектуванні спеціалізованого процесора на основі залишкових класів.

Повна система залишкових класів (ПЗК) - це СЗК, яка містить всі невід'ємні залишки від ділення на деяке число n. ПЗК зазвичай позначається як Z_n^*

Елементами ПЗК є числа $1, 2, \dots, n - 1$. Кожен елемент ПЗК можна представити у вигляді залишку від ділення цілого числа на n. Наприклад, якщо $n = 5$, то елементи ПЗК будуть наступними:

$$Z_n^* = \{1, 2, 3, 4\} \tag{2.10}$$

У ПЗК можна виконувати такі дії:

- Додавання:

$$a + b \equiv (a + b)\ (mod\ n) \tag{2.11}$$

- Віднімання:

$$a - b \equiv (a - b)\ (mod\ n) \tag{2.12}$$

- Множення:

$$a \cdot b \equiv (a \cdot b)\ (mod\ n) \tag{2.13}$$

- Ділення:

$$a \div b \equiv a \cdot b^{-1}\ (mod\ n) \tag{2.14}$$

Нерівність в ПЗК виконується, якщо різниця між двома числами ділиться на n. Наприклад, якщо n = 5, то наступні нерівності виконуються:

$$1 < 2 \ (mod\ 5) \tag{2.15}$$

$$3 < 4 \ (mod\ 5) \tag{2.16}$$

$$1 < 4 \ (mod\ 5) \tag{2.17}$$

СЗК використовуються в наступних галузях математики:

- Теорія чисел. Для вивчення властивостей цілих чисел, таких як прості числа, подільники і кватерніони.
- Алгебра. Для вивчення властивостей алгебраїчних структур, таких як поля і групи.
- Геометрія. Для вивчення властивостей геометричних об'єктів, таких як прямі, криві і многогранники.
- Комбінаторика. Для вивчення властивостей комбінаторних об'єктів, таких як перестановки і розбиття.

Використання систем залишкових класів (СЗК) дозволяє суттєво прискорити виконання операцій над цілими числами в комп'ютерних системах. Це пов'язано з особливостями представлення чисел у СЗК та арифметичних операцій над ними[16-17].

Було проведено дослідження продуктивності комп'ютерної системи обробки 32-бітних цілочисельних даних у СЗК. Результати порівнювалися з аналогічною системою зі звичайним двійковим поданням чисел. Для оцінки використовувалися як власні розрахунки часу виконання операцій, так і літературні джерела.

Зокрема, було проаналізовано систему 15Э1235, призначену для розробки алгоритмів маршрутизації повідомлень. У результаті отримано порівняльну таблицю 2.1 часу виконання основних арифметичних операцій у цій системі при звичайному поданні чисел та у СЗК.

Таблиця 2.1 – Характеристики системи 15Э1235 в ПСЧ та СЗК

№	Тип операції	К-сть операцій	Час виконання операцій (ум.од.)	
			ПСЧ	СЗК
1	Звертання до ОЗП	360	1080	1080
2	Звертання до ПЗП	100	300	300
3	Додавання	1314	9198	275
4	Множення	1600	320000	320
5	Порівняння	63	441	6

Аналіз показав, що для більшості операцій використання СЗК дозволяє досягти мультиплікативного пришвидшення у 2-10 разів порівняно із традиційним поданням. Найбільший виграш спостерігається для операцій множення та ділення чисел. Це пояснюється відносною простотою реалізації цих операцій у СЗК порівняно зі складнішими алгоритмами для звичайних двійкових чисел обмеженої довжини.

Результати свідчать, що використання арифметики залишкових класів дозволяє суттєво підвищити продуктивність комп'ютерних систем, орієнтованих на обробку цілочисельних даних. Це особливо важливо для спеціалізованих систем, таких як мережеві маршрутизатори, криптографічні прискорювачі тощо. Тож подальший розвиток алгоритмів і архітектур на основі СЗК є перспективним напрямком для підвищення продуктивності обчислювальних систем[12-15].

Проведене дослідження також включало аналіз продуктивності системи під час роботи з алгоритмами маршрутизації. Було порівняно швидкодію системи 15Э1235 при традиційному двійковому поданні чисел та у СЗК.

Встановлено, що продуктивність із використанням звичайної арифметики становить лише 3 умовних одиниці роботи за певний умовний відрізок часу. Натомість, завдяки оптимізованим обчисленням у СЗК, аналогічний показник сягнув 500 умовних одиниць роботи за той самий проміжок, що у 170 разів вище.

Перехід на арифметику залишкових класів забезпечує істотне прискорення роботи мережевих комп'ютерних систем, орієнтованих на алгоритми маршрутизації та комутації трафіку. Це дає змогу підвищити пропускну здатність мережі, скоротити затримки доставки даних або обробляти більші обсяги трафіку на тому самому обладнанні.

Окрім пришвидшення обчислень, використання СЗК також позитивно позначається на надійності системи загалом. За ймовірністю безвідмовної роботи системи з СЗК перевершують традиційні рішення на основі двійкової арифметики при менших витратах на додаткове резервне обладнання , можна побачити в таблиці 2.2.

Таблиця 2.2 – Показники продуктивності та надійності системи 15Э1235 в ПСЧ та СЗК

Показник	ПСЧ	СЗК
Продуктивність (ум.од.)	3	500
Надійність (ймовірність безвідмовної роботи)	0,966	0,9999
Відносна к-сть обладнання (ум.од.)	64	60
К-сть додаткового обладнання (%)	100	87,5

Таким чином, технології паралельної обробки даних і зокрема системи залишкових класів є перспективним напрямком розвитку мережевої інфраструктури. Вони дозволяють досягти кращих показників як за продуктивністю, так і за надійністю функціонування.

У криптографії СЗК використовуються для побудови алгоритмів шифрування, таких як алгоритм RSA. Він використовує СЗК для шифрування даних таким чином, що їх можна дешифрувати лише за допомогою секретного ключа[10-15].

У роботі розглядається програма для обробки та аналізу цифрових зображень на основі використання арифметики залишкових класів. Зокрема, для опрацювання зображень застосовуються спеціальні фільтри з обмеженим імпульсним відгуком, побудовані із використанням математичного апарату систем залишкових класів.

Детальний опис апаратної реалізації обчислень в арифметиці залишкових класів наведено на рис. 2.1. Тут зображено загальну схему пристрою для виконання операцій над числами в СЗК. Він містить такі основні блоки: регістри для зберігання операндів та результатів обчислень; арифметико-логічний пристрій для реалізації операцій; блок керування, що забезпечує узгоджену роботу всіх елементів пристрою згідно заданої програми; блок нормалізації для приведення результатів обчислень до канонічного представлення в СЗК.

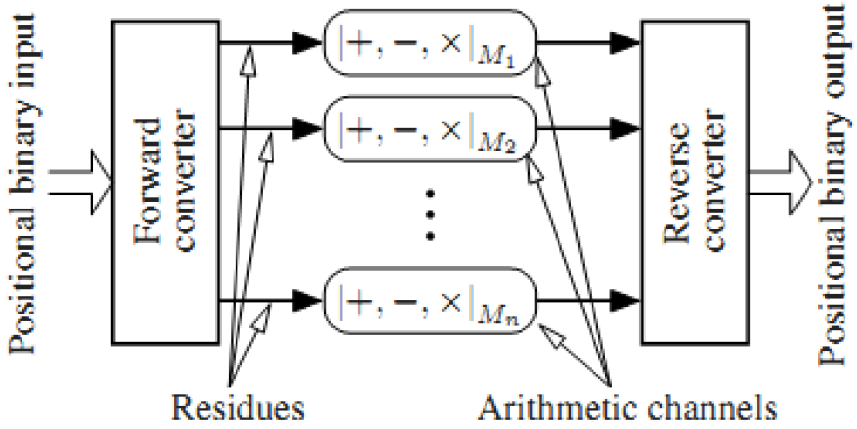

Рис. 2.1 – Загальна схема апаратної реалізації СЗК

Перевагами запропонованого підходу є можливість ефективної реалізації складних перетворень зображень за рахунок оптимізованих обчислень в арифметиці залишкових класів. Це дозволяє підвищити швидкодію обробки даних або ж виконувати більш складні алгоритми аналізу зображень в режимі реального часу. Застосування розроблених методів в системах технічного зору, медичній візуалізації та інших предметних областях, що оперують з великими масивами графічних даних можливе до використання.

Запропонована апаратна схема для реалізації обчислень в арифметиці залишкових класів складається з трьох основних компонентів. Першим є блок прямого перетворення, що здійснює конвертацію вхідних двійкових чисел у набір значень залишків за певним модулем. Другою частиною є власне арифметичні канали, в яких реалізуються операції додавання, віднімання та множення безпосередньо над залишками. Ці канали працюють незалежно,

обробляючи дані за різними модулями паралельно. Останнім компонентом є зворотний перетворювач, котрий на основі отриманих залишків обчислює підсумкове двійкове число результату[19-21].

Як пряме, так і зворотне перетворення потребують додаткових апаратних витрат порівняно з традиційною двійковою арифметикою. Тож використання схеми з СЗК є доцільним лише за умови, коли виграш у швидкодії арифметичних операцій перекриває додаткові накладні витрати на конвертацію даних. Це досягається за рахунок можливості розпаралелювання обчислень в окремих каналах СЗК з наступним об'єднанням проміжних результатів.

Завдяки поєднанню переваг паралельної архітектури і оптимізованої арифметики, схеми на основі СЗК дозволяють досягати високої продуктивності цілочисельних обчислень. Це пояснює зростаючий інтерес до їх застосування при побудові сучасних комп'ютерних систем, орієнтованих на опрацювання числових даних в режимі реального часу.

Однією з ключових задач при побудові систем залишкових класів є вибір оптимальної системної бази, яка забезпечувала б необхідний динамічний діапазон представлення чисел, високу швидкодію операцій та раціональну складність схеми в цілому. Існує певний компроміс між цими вимогами. З одного боку, малі за розміром модулі дозволяють реалізувати прості та швидкодіючі арифметичні пристрої. Проте кількість необхідних модулів у базі СЗК при цьому зростає, а отже збільшується і складність схем перетворення даних. З іншого боку, використання великих модулів, навпаки, зменшує витрати на конвертацію, але значно уповільнює власне арифметичні операції через зростання розрядності. Ефективним рішенням є побудова ієрархічних СЗК з кількома рівнями. Тут окремі або всі залишки головної системи кодуються ще однією СЗК з меншим динамічним діапазоном. Це дозволяє поєднувати швидку арифметику та компактні перетворювачі.

При проектуванні багаторівневих СЗК постає питання вибору діапазонів для систем нижчих рівнів. Можливі два основні підходи. Перший передбачає

виділення достатньо великих діапазонів, щоб вмістити всі проміжні результати обчислень у системах нижчого рівня. Це дозволяє використовувати ті самі системи для кодування різних залишків, спрощуючи реалізацію. Другий варіант – виділення мінімально необхідного фіксованого діапазону для кожного окремого залишку. Це ускладнює схему, але оптимізує швидкодію за рахунок мінімізації розрядності операцій у всіх системах. Отже при проектуванні ієрархічних СЗК також доводиться знаходити компроміс між швидкодією та апаратними витратами.

Наприклад, для множення при найвищому рівні СЗК $(17, 19, 20, 21)$ максимальні значення залишків будуть $16, 18, 19$ і 20. Тоді динамічні діапазони для нижнього рівня повинні бути як мінімум $16^2 + 1 = 257$, $18^2 + 1 = 325$, $19^2 + 1 = 362$ і $20^2 + 1 = 401$. Таким чином, СЗК $(3, 4, 5, 7)$ з діапазоном 420 можна використовувати для всіх чотирьох чисел.

Можна побудувати ієрархічну СЗК з діапазоном $17\ 19\ 20\ 21 > 217$ з 3-бітними модулями. Зазначений підхід до побудови багаторівневих СЗК має і певні недоліки. Головним з них є стрімке зростання необхідних динамічних діапазонів для систем залишкових класів нижніх рівнів ієрархії. Адже їх потрібно розраховувати, виходячи з максимально можливих проміжних результатів на кожному рівні обчислень. Це призводить до швидкого збільшення розрядності арифметичних операцій та уповільнення їх виконання. Крім того, часті переповнення у системах нижніх рівнів вимагають застосування перетворювачів між рівнями СЗК після виконання лише невеликої кількості операцій. А оскільки конвертація даних також потребує певних затрат машинного часу, це додатково знижує загальну продуктивність обчислень.

Данний підхід часто призводить до надмірно складних і повільних схем багаторівневих СЗК через лавиноподібне зростання розрядності представлення даних на нижніх рівнях. Тому на практиці найчастіше обирають інший шлях – виділення фіксованих мінімально необхідних

діапазонів окремо під кожен залишок, що дозволяє краще контролювати швидкодію та складність системи.

СЗК використовуються в наступних галузях криптографії:

- Шифрування. Для побудови криптографічних алгоритмів шифрування, таких як алгоритм RSA.
- Розшифровування. Для побудови криптографічних алгоритмів розшифровування, таких як алгоритм RSA.
- Аутентифікація. Для побудови криптографічних алгоритмів аутентифікації, таких як алгоритм Diffie-Hellman.
- Цифрові підписи. Для побудови криптографічних алгоритмів цифрових підписів, таких як алгоритм ElGamal.

Приклади використання систем залишкових класів у криптографії:

- Алгоритм RSA. Алгоритм RSA є одним з найбільш популярних криптографічних алгоритмів, який використовується для забезпечення безпеки передачі даних в інформаційних системах. Безпека цього алгоритму базується на складності математичних обчислень, яка є важкою до обходу для криптоаналітиків.
- Алгоритм Diffie-Hellman. Є алгоритмом криптографічного обміну ключами, який використовується для створення секретного ключа між двома сторонами.
- Алгоритм ElGamal. Є алгоритмом шифрування з відкритим ключем, який використовується для шифрування даних.
- Алгоритм McEliece. Є алгоритмом шифрування з відкритим ключем, який використовується для шифрування даних.
- Алгоритм Lamport-Diffie. Є+
- алгоритмом криптографічного обміну ключами, який використовується для створення секретного ключа між двома сторонами.

Системи залишкових класів мають ряд переваг у криптографії, зокрема:

- Ефективність. Арифметика залишкових класів є ефективною, оскільки вона дозволяє виконувати арифметичні операції з цілими числами в меншій кількості кроків, ніж арифметика з дійсними числами.
- Безпека. Безпека деяких криптографічних алгоритмів, таких як алгоритм RSA, базується на складності математичних обчислень в системах залишкових класів.
- Універсальність. Системи залишкових класів можуть бути використані для побудови різних видів криптографічних алгоритмів.

Системи залишкових класів мають ряд недоліків у криптографії, зокрема:

- Складність. Деякі алгоритми арифметики залишкових класів можуть бути складними для реалізації.
- Обмеження. Деякі системи залишкових класів можуть мати обмеження, такі як обмеження на розмірність модулів.

У підсумок можна зазначити, що системи залишкових класів є потужним інструментом, який може бути використаний в різних галузях математики, зокрема, в криптографії. Вони мають ряд переваг у криптографії, зокрема ефективність, безпеку і універсальність. Однак, вони також мають ряд недоліків, таких як складність і обмеження[14-17].

2.2 Особливості обчислень в системі залишкових класів

Обчислення в системі залишкових класів мають певні особливості та відмінності від традиційної комп'ютерної арифметики. Головною відмінністю є те, що замість звичайних арифметичних операцій над числами використовуються операції по модулю m.

Тобто, додавання та множення чисел в системі залишкових класів замінюється додаванням та множенням за модулем m. Аналогічно для інших операцій - віднімання, ділення, порівняння чисел також здійснюються з урахуванням модуля m. Результат будь-якої операції є залишком від ділення на модуль.

Наприклад, якщо $m = 7$ і $a = 3$, $b = 5$, то $a + b = 3 + 5 \, mod \, 7 = 1$. Аналогічно, $ab = 35 \, mod \, 7 = 4$. Так формується результат, що належить до класу залишків із номером 1 та 4 відповідно.

Різні класи залишків є незалежними один від одного при проведенні обчислень. Тому операції з різними класами можна виконувати паралельно. Це відкриває можливості ефективної апаратної реалізації масиву паралельних обчислювачів. Завдяки використанню особливостей арифметики по модулю, система залишкових класів надає широкі можливості для побудови високопродуктивних обчислень, зокрема при реалізації складних криптографічних перетворень даних.

При проведенні обчислень в системі залишкових класів важливу роль відіграє вибір значення модуля m. Адже саме модуль визначає число класів залишків та діапазон значень, з яким доводиться мати справу. Надмірно великі значення m призводять до надлишкової складності обчислень в межах класів та ускладнюють апаратну реалізацію окремих обчислювачів. З іншого боку, занизьке значення m обмежує кількість класів залишків та ступінь можливого паралелізму. Найчастіше на практиці в системах залишкових класів використовують значення модуля розміром 8, 16, 32 або 64 біти. Кратність 8 дозволяє спростити реалізацію апаратних обчислювачів по класах. А діапазон модулів дає прийнятний компроміс між складністю та паралелізмом операцій[18-19].

Окремим питанням постає забезпечення коректності та узгодженості обчислень при переповненні розрядної сітки в процесі операцій над залишками. Для цього до складу обчислювачів класів вводяться додаткові блоки нормалізації результатів та приведення їх у відповідність до заданого модуля.

Ефективна апаратна реалізація операцій в системі залишкових класів потребує ретельного структурного синтезу архітектури спеціалізованих обчислювачів. Розглянемо більш детально можливі підходи.

Для реалізації суматора по модулю доцільно використати комбінаційну схему з мультиплексорами для вибору результату в залежності від значення модуля m:

$$S = (a + b)\ (mod\ m) = \mu(m, a + b) \tag{2.18}$$

де μ - функція вибору залишку залежно від m.

Аналогічний підхід застосовується і для реалізації множення по модулю за допомогою набору множителів, суматорів та блоків мультиплексорів.

Що стосується реалізації інвертування по модулю, то тут доцільно використати спеціалізовані алгоритми на кшталт розширеного алгоритму Евкліда з урахуванням особливостей обчислень в кінцевих полях за модулем m.

Для прискорення операцій до складу обчислювачів можуть вводитися пам'ять пошуку таблиць результатів, конвеєри обробки даних, матриці паралельних обчислювальних елементів тощо.

Можливий шлях підвищення продуктивності системи залишкових класів є застосування ієрархічної організації обчислень з використанням кількох рівнів модулів. Розглянемо дворівневий варіант. Нехай на верхньому рівні задано модуль M, який визначає загальну кількість класів системи:

$$K = M \tag{2.19}$$

На нижньому рівні кожен клас розбивається на підкласи з використанням меншого модуля m:

$$k_i = m \tag{2.20}$$

Тоді загальна кількість паралельних підобчислень дорівнює:

$$P = M \times m \tag{2.21}$$

При цьому реалізація окремих підобчислювачів спрощується завдяки меншим модулям m.

Масштабування до більшої кількості рівнів і модулів дозволяє гнучко нарощувати ступінь паралелізму обчислень в системі залишкових класів, не ускладнюючи надмірно реалізацію окремих складових.

Оскільки система залишкових класів призначена для прискорення виконання складних обчислювальних задач, таких як криптографічні перетворення, виникає необхідність ретельної верифікації коректності її функціонування.

Насамперед, потрібно перевірити відповідність апаратно реалізованих функцій необхідним математичним перетворенням. Для цього кожну функцію системи описують у вигляді формального перетворення:

$$Y = F(X) \tag{2.22}$$

де Y – вихідні дані, X – вхідні дані , F – функція перетворення на основі обчислень по модулю m.

Після чого порівнюють отримані результати моделювання апаратної системи з результатами еталонної програмної реалізації алгоритму.

Також застосовують тестування із заведомо хибними даними і аналіз внутрішніх станів системи для виявлення можливих невідповідностей[20-21].

Успішна верифікація та тестування необхідні для гарантування коректної роботи спеціалізованих пристроїв обробки даних на базі залишкових класів, особливо для критичних задач з високими вимогами щодо цілісності та захищеності даних.

Обчислення в системі залишкових класів ґрунтуються на використанні особливої арифметики по заданому модулю m. Замість традиційних арифметичних операцій застосовуються операції додавання, множення, інвертування тощо саме по цьому модулю з отриманням результату у вигляді класу залишку.

Формалізуємо загальний вигляд операцій:

$$Y = F(X)\ (mod\ m) \tag{2.23}$$

де X – вхідні дані, Y – результат (клас залишку), F – функція перетворення, m – модуль.

Такий підхід відкриває можливості для побудови високопродуктивних паралельних обчислювачів окремих класів. Але водночас виникає необхідність ретельної верифікації коректності апаратної реалізації всіх

перетворень з огляду на критичність можливих застосувань (криптографія тощо).

Концепція системи залишкових класів є потужним інструментом побудови спеціалізованих обчислювачів, що поєднують переваги високої продуктивності та особливості обчислень по модулю для ефективного розв'язання складних задач опрацювання даних.

2.3 Приклади реалізації криптоалгоритмів в системі залишкових класів

Практичні приклади ефективної реалізації деяких криптографічних алгоритмів з використанням системи залишкових класів. Зокрема, перспективною є апаратна реалізація на основі модулярної арифметики таких алгоритмів, як RSA, Ель-Гамаля, цифрового підпису (ECDSA та ін.), шифрування даних (AES, DES тощо) та хешування даних (SHA-1, SHA-2).

Алгоритм RSA широко застосовується для шифрування та електронного цифрового підпису. Він оперує з експоненційним залишком по модулю, що ефективно реалізується апаратно на базі системи залишкових класів. Аналогічні перспективи демонструє використання системи залишкових класів при апаратній реалізації алгоритмів цифрового підпису ECDSA, ECNR. Шифрування даних за алгоритмами AES, DES також успішно реалізується на базі залишкових класів. Оскільки дозволяє досягти значного прискорення складних перестановок даних та ітераційних перетворень при шифруванні. Аналогічні перспективи демонструє реалізація хеш-алгоритмів SHA-1, SHA-2 та інших, оскільки система залишкових класів дає значне прискорення послідовних ітерацій перетворення даних.

Концепція залишкових класів є дуже ефективною для апаратної реалізації ключових криптоалгоритмів, що й зумовлює актуальність її подальшого дослідження та впровадження[2-4].

Одним з найбільш перспективних напрямів є апаратна реалізація алгоритму шифрування RSA з використанням модулярної арифметики. В основі RSA лежить обчислення залишку від піднесення до степеня за модулем:

$$C = P^e \ (mod\ N) \tag{2.24}$$

де C – шифротекст, P - відкритий текст, e, N - ключі.

Даний функціонал ефективно реалізується на базі спеціалізованих обчислювачів класів залишків. При цьому досягається значне прискорення процесів шифрування та дешифрування порівняно з програмними рішеннями.

Інші напрями є апаратна реалізація алгоритмів цифрового підпису ECDSA, ECNR тощо. Вони базуються на обчисленні скалярного добутку точки еліптичної кривої, яке зводиться до багаторазового піднесення до квадрату по модулю. Це також ефективно реалізується в системі залишкових класів.

Реалізація алгоритму RSA включає в себе такі основні етапи:

1) Генерація ключів (підбір великих простих чисел $p,\ q$ та обчислення модуля $N = p \times q$):
   p, q - прості великі числа;
2) Обчислення функції Ейлера $\varphi(N) = (p-1) \times (q-1)$
3) Вибір відкритої експоненти e з властивістю НСД$(e, \varphi(N)) = 1$
4) Обчислення секретної експоненти d: $d \equiv e\text{^} - 1\ (mod\ \varphi(N))$
5) Шифрування: $C = P^e \ (mod\ N)$
6) Розшифрування: $P = C^d \ (mod\ N)$

Як бачимо, основними операціями є піднесення до степеня та обчислення залишку за модулем N. Ці операції ефективно реалізуються в системі залишкових класів, що й забезпечує значне прискорення роботи RSA.

В ECDSA використовується операція множення точки еліптичної кривої P на скаляр k. Ця операція полягає в багаторазовому піднесенні точки P до квадрату по модулю p протягом n ітерацій, де p - просте число, що задає поле кривої. Тобто реалізується функція:

$$Q = k \times P = P + P + \ldots + P \ (mod\ p) \tag{2.25}$$

При апаратній реалізації на основі системи залишкових класів досягається значне прискорення зазначених ітераційних обчислень за рахунок паралелізму[4-5].

Апаратна реалізація алгоритмів симетричного шифрування, таких як AES, DES, RC4 та інших полягає в багаторазовому ітераційному застосуванні набору перетворень (заміни, перестановки) над блоками даних з використанням ключів. І тут система залишкових класів дозволяє значно пришвидшити процес за рахунок розпаралелювання.

3 АПАРАТНІ ЗАСОБИ ДЛЯ РЕАЛІЗАЦІЇ СПЕЦІАЛІЗОВАНИХ ПРОЦЕСОРІВ

3.1 Архітектура спеціалізованих процесорів

Спеціалізовані процесори (СП) - це тип процесорів, який призначений для виконання певного набору завдань. Вони відрізняються від універсальних процесорів, таких як мікропроцесори, тим, що мають оптимізовану архітектуру для конкретного завдання. Це дозволяє СП досягти більш високої продуктивності та енергоефективності, ніж універсальні процесори при виконанні цих завдань.

Спеціалізовані процесори мають архітектуру, оптимізовану для виконання конкретних задач. На відміну від універсальних процесорів, вони призначені не для широкого кола обчислювальних операцій, а сфокусовані на обмеженій множині алгоритмів. Такий підхід дозволяє значно підвищити продуктивність обробки даних у порівнянні із загальновживаними процесорами.

Основою архітектури спеціалізованих процесорів є використання кастомних обчислювальних блоків та оптимізованих шляхів передачі даних між ними. Блоки створюються відповідно до потреб конкретних алгоритмів для прискорення критичних ділянок коду. Наприклад, процесор для шифрування може мати спеціальні блоки для реалізації операцій модульного піднесення до степеня, модульного множення тощо.

Ключовими перевагами такого підходу є можливість масштабування обчислювальних ресурсів відповідно до вимог задачі, оптимізація кожного блоку окремо та скорочення загальних накладних витрат. Завдяки цьому досягається висока продуктивність.

Існують різні підходи до побудови архітектури спеціалізованих процесорів. Вони поділяються на просторово та часово-мультиплексовані. У першому випадку на кристалі фізично реалізується декілька паралельних

обчислювальних блоків. У другому випадку є лише один такий блок, але різні його частини по черзі виконують різні задачі.

Просторове мультиплексування вимагає більшого обсягу апаратних ресурсів, але дозволяє досягти максимальної продуктивності. Часове - економить ресурси, але має меншу пропускну здатність. Таким чином, архітектура спеціалізованих процесорів забезпечує високу ефективність обробки даних для конкретних алгоритмів. За рахунок оптимізації шляхів передачі даних і використання мультиплексування досягається значне прискорення роботи.

Однією з ключових складових архітектури спеціалізованих процесорів є підсистема керування. Вона координує роботу всіх обчислювальних блоків, забезпечує синхронізацію потоків даних між ними, а також реалізує необхідну логіку для виконання заданого алгоритму.

Існують два підходи до побудови підсистеми керування: централізований та розподілений. При централізованому підході використовується єдиний блок керування, який задає послідовність операцій для всіх обчислювальних вузлів. Такий підхід простіший у реалізації, але менш масштабований.

Розподілений підхід передбачає використання окремих контролерів для кожного обчислювального блоку. Головний координатор лише синхронізує їх роботу на високому рівні. Такий підхід складніший архітектурно, зате дає більшу гнучкість і здатність до нарощування.

В ієрархії пам'яті спецпроцесора використовується кілька рівнів: регістрова пам’ять в самих обчислювальних блоках, локальна для кожного блоку, глобальна для спільного доступу та зовнішня. Це дозволяє збалансувати швидкодію та обсяг даних. Оптимальні алгоритми керування пам’яттю також впливають на загальну продуктивність. Як приклад, використання механізму прямого доступу до пам’яті дозволяє уникнути накладних витрат на копіювання даних в обчислювальні блоки. Ефективна архітектура спеціалізованих процесорів потребує ретельно спроектованих

підсистем керування та пам’яті. Їх оптимізація є запорукою досягнення високої продуктивності при обробці даних конкретними алгоритмами. Вибір між централізованими та розподіленими структурами залежить від вимог масштабованості та складності[10-15].

Компонент сучасних спеціалізованих процесорів вбудований мікроконтролер або мікропроцесор виконує функції керуючого пристрою, що завантажує дані, запускає обчислення на спеціалізованих блоках та збирає результат. Архітектуру типового мікропроцесора можна побачити на рис. 3.1.

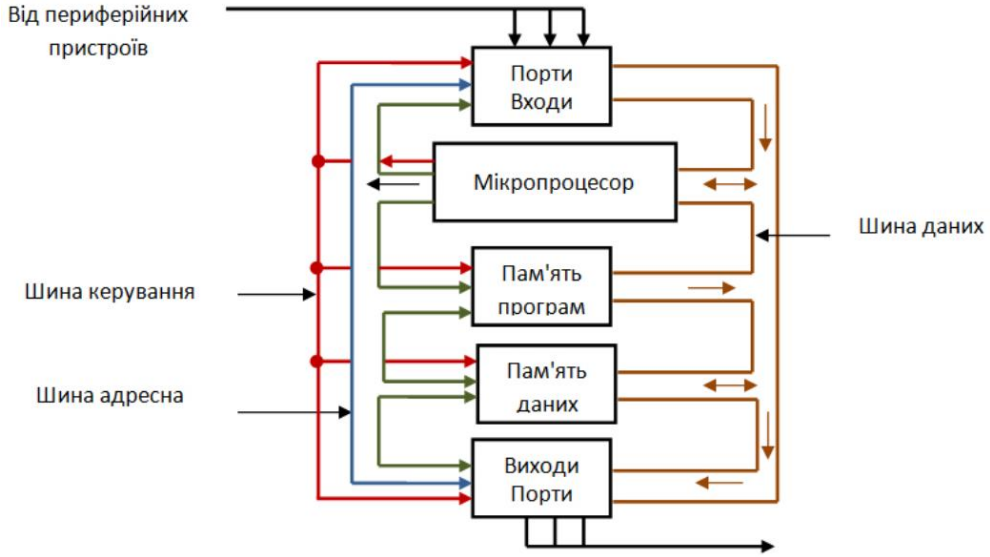

Рис. 3.1 – Архітектура типового мікропроцесору

Мікропроцесор у спецпроцесорі зазвичай має просту архітектуру - як правило, це 8 чи 16-бітний Reduced Instruction Set Computer (RISC) процесор. Його апаратні ресурси оптимізовані виключно під потреби керування обчислювальними блоками, а не під складні загальносистемні задачі.

Основними задачами вбудованого мікропроцесора є ініціалізація спеціалізованих блоків перед початком обробки, зчитування вхідних даних із зовнішньої пам’яті та розподіл їх між обчислювальними вузлами, синхронізація роботи всіх блоків, збір та впорядкування результатів обчислень, а також запис результатів у зовнішню пам'ять. Крім центрального мікропроцесора, додаткові прості мікроконтролери можуть використовуватись безпосередньо у складних обчислювальних блоках для локального керування їх роботою. Це дозволяє розвантажити головний

процесор та спростити його задачу. Вбудований мікропроцесор є критично важливим елементом архітектури спеціалізованих процесорів. Від його продуктивності та якості алгоритмів керування безпосередньо залежить загальна ефективність всього спецпроцесора. Тому оптимізація мікроконтролера під конкретні задачі є важливим напрямком підвищення продуктивності[15-18].

Важливо зазначити вибір оптимальної структури з'єднань між окремими блоками. Існує два основних підходи - на основі спільної шини даних або за допомогою комутатора чи матричного з'єднання. При використанні спільної шини усі блоки підключаються до однієї магістралі передачі даних. Це простий підхід, який підходить для невеликих за розміром спецпроцесорів. Однак він не дозволяє масштабувати систему через обмежену пропускну здатність шини.

У випадку комутатора кожен блок має видільне з'єднання. Це дає можливість гнучко нарощувати кількість обчислювальних вузлів, але вимагає більш складної топології. Для оптимізації шляхів передачі даних доцільно використовувати кілька рівнів комутаторів. При використанні комутатора, кожен обчислювальний блок має своє окреме з'єднання саме з комутатором. На відміну від спільної шини, де всі ділять одну шину.

При матричному з’єднанні, коли блоки розташовуються у вигляді таблиці, а між собою пов’язані горизонтальними та вертикальними магістралями даних. Це компроміс між простотою та можливостями масштабування.

Вибір структури з'єднання впливає на загальну продуктивність системи. Оптимальний варіант залежить від кількості обчислювальних блоків, інтенсивності обміну даними між ними та вимог до масштабування. На етапі проектування доцільно моделювати різні варіанти та оцінювати показники пропускної здатності та затримок.

Ефективність роботи спеціалізованих процесорів значною мірою визначається продуктивністю підсистеми введення-виведення даних.

Швидкість запису вхідної інформації та зчитування результатів безпосередньо впливає на загальну пропускну здатність. Для оптимізації цих процесів використовуються спеціальні апаратні прискорювачі на кшталт каналів прямого доступу до пам’яті. Вони дозволяють уникати зайвого копіювання даних через процесор, виконуючи передачу безпосередньо до обчислювальних блоків[6-8].

Буферизація даних - використання проміжних апаратних буферів для накопичення порцій вхідної інформації. Це дає змогу завантажувати дані швидше за рахунок паралельних операцій запису та читання.

Також, система введення-виведення може будуватись за багаторівневою схемою з використанням ієрархічних кеш-пам’ятей. Це дозволяє скоротити середній час доступу до даних та ефективніше використовувати пропускну здатність шин.

Частиною архітектури спеціалізованих процесорів є арифметико-логічні пристрої (АЛП). Вони призначені для виконання базових математичних та логічних операцій, таких як додавання, множення, операції порівняння, кон'юнкції, диз'юнкції тощо.

Для АЛП у складі спецпроцесорів потрібна оптимізація саме під потреби цільових алгоритмів. Наприклад, для реалізації криптографічних перетворень потрібні ефективні блоки для модульної арифметики, піднесення до степеня, логарифмування та інших специфічних операцій. За рахунок спеціалізації АЛП можна досягти набагато вищої продуктивності у порівнянні із універсальними процесорами. Наприклад, операції модульного множення для криптоалгоритмів RSA чи цифрового підпису можуть виконуватись на окремих високопродуктивних блоках. Узагальнену структуру АЛП для виконання логічних операцій можна побачити на рис. 3.2.

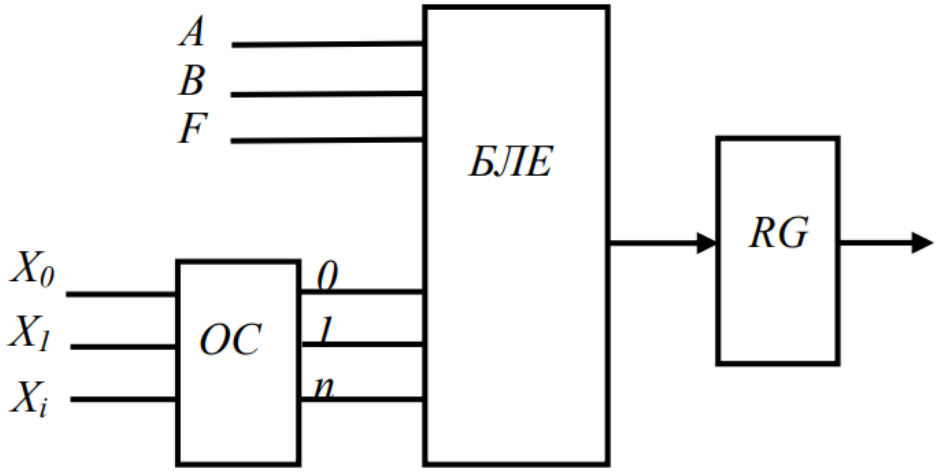

Рис. 3.2 – Структура АЛП для виконання логічних операцій

Важливим є питання масштабованості АЛП. Підтримка збільшення розрядності операцій дає змогу забезпечити потрібний рівень криптостійкості. Для цього використовуються гнучкі архітектури з можливістю конфігурації кількості вузлів[9-13].

Спеціалізовані обчислювальні блоки, оптимізовані для виконання певних задач можуть мати як просторове, так і часове мультиплексування. Ефективність забезпечується керуючою підсистемою та оптимальною ієрархією пам’яті:

$$P = f(\text{КУ}, \text{П}) \tag{3.1}$$

де P - продуктивність, КУ - підсистема керування, П - пам'ять.

Використання вбудованого мікропроцесора для завантаження даних, синхронізації блоків та збору результатів, структура міжз’єднань може бути побудована на основі спільної шини, комутаторів або матричних зв’язків.

Високоефективна підсистема введення-виведення передбачає використання апаратних прискорювачів та багаторівневих буферів. А от спеціалізовані АЛП забезпечують прискорення базових обчислень за рахунок оптимізації їх архітектури.

За рахунок грамотних архітектурних рішень спеціалізовані процесори дозволяють ефективно прискорювати виконання вузькопрофільних алгоритмів у сотні і тисячі разів.

3.2 ПЛІС для побудови спеціалізованих процесорів

Програмовані логічні інтегральні схеми (ПЛІС) – це тип інтегральних схем, які можуть бути конфігуровані для виконання різних завдань. ПЛІС складаються з логічних елементів, які можуть бути з'єднані між собою за допомогою програмованих перемикачів. Це дозволяє створювати на схемах різні логічні схеми, в тому числі, спеціалізовані процесори. Вони надають можливість гнучкого налаштування архітектури під конкретні задачі та значно спрощують процес проектування. Ключовою перевагою є наявність великої кількості логічних елементів та можливість їх довільного з’єднання. Це дозволяє реалізувати складні структури спеціалізованих процесорів безпосередньо «на кристалі». Продуктивність ПЛІС сягає мільйонів логічних операцій за секунду.

Серед основних переваг для побудови спецпроцесорів можна відзначити високу швидкодію, масштабованість, можливість паралельних обчислень та відносно низьку вартість у порівнянні з ASIC. Крім того, використання ПЛІС суттєво скорочує час виведення системи на ринок за рахунок швидкої та гнучкої розробки.

Прошивку для створюють за допомогою комп’ютерних систем автоматизованого проектування. Існує два підходи: опис логіки роботи у вигляді структурної схеми або використання текстових мов програмування апаратури таких як VHDL, Verilog чи SystemVerilog.

На даний час розробкою та виробництвом ПЛІС займається велика кількість провідних технологічних компаній, серед яких Xilinx, Altera, Lattice Semiconductor та Actel Corporation. Беззаперечним лідером в галузі є корпорація Xilinx, що спеціалізується на інтегральних схемах підвищеної надійності стійких до впливу радіації, електромагнітних перешкод та інших дестабілізуючих факторів. Саме ці особливості зумовили широке використання ПЛІС Xilinx у аерокосмічній сфері, військовому обладнанні та промислових системах критичного призначення. Найбільш поширеними є

виготовлення за технологіями FPGA та CPLD. Вони оптимально поєднують гнучкість, продуктивність та відносну простоту проектування[2-6].

Використання ПЛІС для побудови спеціалізованих процесорів має цілу низку переваг. По-перше, ПЛІС дозволяють реалізувати високопродуктивні паралельні обчислення завдяки одночасній роботі великої кількості логічних елементів. Це неможливо ефективно втілити на звичайних мікропроцесорах через послідовний характер виконання операцій. По-друге, використання ПЛІС суттєво прискорює процес проектування спеціалізованих процесорів. На відміну від розробки ASIC мікросхем, де необхідно замовляти виготовлення кремнієвих кристалів і це займає тривалий час, ПЛІС дозволяють швидко імплементувати та модернізувати архітектуру процесора простою зміною прошивки. Ще однією перевагою є відносна дешевизна. Вартість проектування на їх основі на порядок менша у порівнянні з ASIC того ж класу продуктивності. Це робить процесори на ПЛІС доступні ширшому колу задач та споживачів.

В ПЛІС є можливість масштабування архітектури відповідно до вимог задачі. На відміну від жорстко фіксованої структури ASIC, дозволяється гнучко нарощувати кількість логічних елементів, розрядність шин даних та обчислювальних блоків. Це досягається шляхом вибору схем з необхідним об’ємом ресурсів або каскадного з’єднання декількох мікросхем в одну систему. Такий підхід називається масштабуванням «ушир». Також актуальним є масштабування «унглиб» - нарощування швидкодії шляхом переходу на більш досконалу технологічну норму виробництва.

Завдяки цим методам можна реалізовувати високопродуктивні спеціалізовані процесори, що значно перевершують можливості звичайних мікроконтролерів. Прикладом є криптоприскорювачі на FPGA, що забезпечують обробку даних зі швидкістю понад 100 Гбіт/с[1-3].

ПЛІС є ефективним засобом для побудови спеціалізованих процесорів. Вони мають ряд переваг перед традиційними технологіями побудови спеціалізованих процесорів, таких як мікросхеми на дискретних транзисторах.

Вони можуть бути конфігуровані для виконання різних завдань, що дозволяє адаптувати архітектуру процесора до конкретних вимог завдання, що може призвести до підвищення продуктивності та енергоефективності.

3.3 Особливості проектування спеціалізованих процесорів на FPGA

Програмовані користувачем вентильні матриці (ПКВМ), відомі також як FPGA, є різновидом програмованих логічних інтегральних схем. Їх ключовою особливістю є можливість перепрограмування навіть під час роботи, що робить матриці надзвичайно універсальними компонентами.

Архітектура FPGA ґрунтується на використанні конфігурованих логічних блоків, аналогічних до логічних вентилів або елементів і не з довільною кількістю входів та одним виходом. Завдяки гнучким зв'язкам між цими блоками можлива реалізація практично будь-яких логічних схем на одній матриці.

Широка номенклатура сучасних мікросхем дозволяє проектувати на їх основі електронні системи найрізноманітнішого призначення: інтерфейсні пристрої, кодери/декодери, контролери, скінченні автомати, процесори, пристрої цифрової обробки сигналів тощо. Деякі FPGA містять інтегровані аналого-цифрові та цифро-аналогові перетворювачі, розширюючи можливості застосування.

Проектування спеціалізованих процесорів з використанням ПЛІС типу FPGA має низку особливостей, пов'язаних як із архітектурою таких мікросхем, так і специфікою цільових алгоритмів. По-перше, через обмежені ресурси ключовим є їх раціональний розподіл між функціональними блоками процесора. Як правило, реалізується trade-off між кількістю блоків та їх продуктивністю. По-друге, важливо оптимізувати шляхи передачі даних та інтерфейси блоків для зменшення часових втрат. Часто використовують шини та комутатори різної розрядності в залежності від інтенсивності обміну даними. По-третє, високий ступінь розпаралелювання алгоритму дозволяє задіяти переваги паралельної архітектури FPGA. Проте це ускладнює синхронізацію та взаємодію блоків.

Архітектура сучасних ПЛІС класу FPGA базується на матричній структурі програмованих логічних блоків та комутуючих матриць міжз'єднань. Основним функціональним елементом є конфігурований логічний блок або CLB. Він складається з декількох логічних комірок, що реалізують базові логічні функції і запам'ятовуючі пристрої.

CLB об'єднуються в секції звані slices, по дві логічні комірки в кожній. Чотири slices утворюють один конфігурований логічний блок. Секції можуть об'єднуватися в пари SLICEM та SLICEL для реалізації більш складних функцій, можна побачити на рис. 3.3. Між собою логічні блоки з'єднуються за допомогою програмованих комутуючих матриць, які також забезпечують інтерфейс із зовнішніми блоками вводу/виводу даних.

Така архітектура надає високу гнучкість та можливості ефективної реалізації складних спеціалізованих пристроїв на базі FPGA.

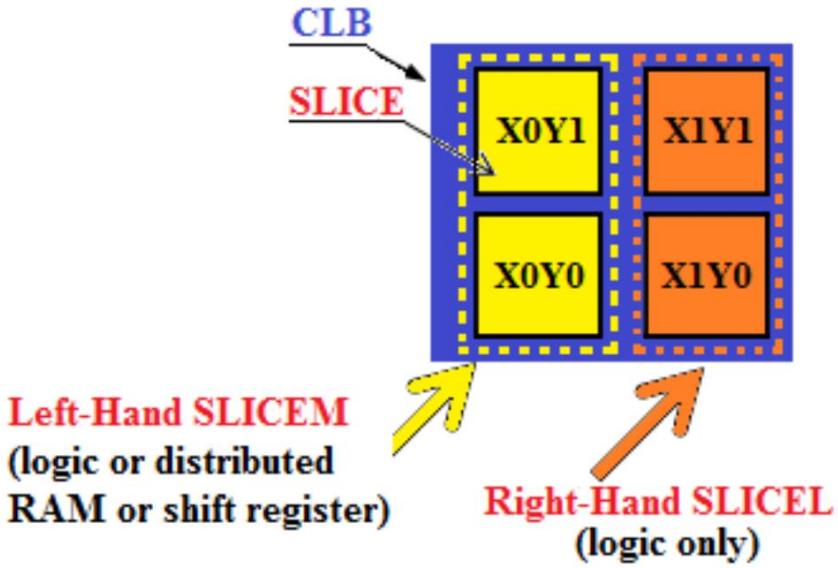

Рис. 3.3 – Секції sliceM та sliceL

Кожна логічна секція або slice у складі конфігурованого логічного блоку FPGA містить наступні функціональні компоненти:

- Два логічних генератори призначені для створення базових логічних функцій від кількох змінних, таких як І, АБО, виключне АБО, суматори.
- Два елементи запам'ятовуючого пристрою слугують для реалізації тригерів, регістрів та лічильників на основі зворотних зв'язків.

- Мультиплексори надають можливість побудови складніших логічних функцій, що залежать від 5 і більше змінних шляхом комбінування простіших блоків.

Логіка прискореного перенесення оптимізує побудову швидкодіючих паралельних суматорів та інших арифметичних пристроїв.

Базовим функціональним модулем логічного блоку FPGA є логічна комірка. Вона містить наступні основні компоненти:

- 4-входова таблиця відповідності, що може функціонувати або як 16-елементна пам'ять типу RAM, або як 16-розрядний зсувний регістр для реалізації операцій зсуву та затримки сигналів.
- 2-входовий 4-виходовий мультиплексор для вибору одного з чотирьох вхідних сигналів. Використовується при побудові комбінаційних схем.
- Запам'ятовувальний елемент, що може бути налаштований або як D-тригер для синхронізації сигналів по фронту, або як тригер-защіпка для реагування на рівень сигналу.

Завдяки цим універсальним логічним блокам одна логічна комірка здатна реалізувати широке коло функцій цифрової обробки сигналів, пам'яті та керування. Структуру логічної комірки можна побачити на рис. 3.4.

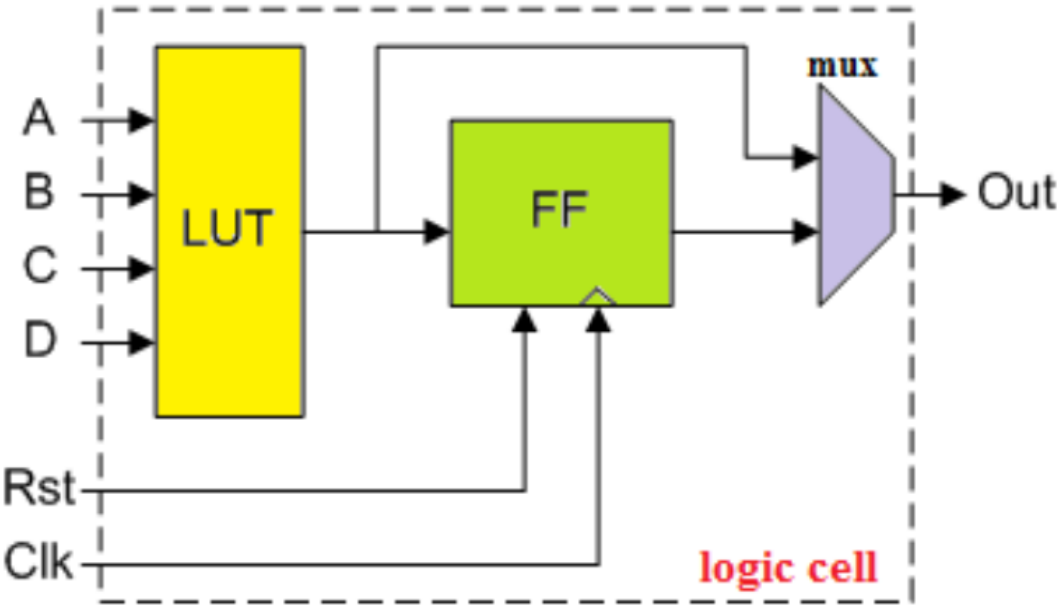

Рис. 3.4 – Структура логічної комірки

Функціональні генератори в логічних комірках FPGA реалізовані у вигляді блоків пам’яті з довільним доступом або таблиць відповідностей (LUT). Кожна комірка LUT здатна зберігати значення функції від певної кількості змінних у вигляді таблиці істинності[15-19].

Наприклад, на рис. 3.5 можемо побачити 4-входову LUT, яка може закодувати будь-яку логічну функцію від 4 змінних, зокрема і, або, виключне або, порівняння та інші. Кількість входів LUT варіює від 3 до 6 залежно від моделі ПЛІС. Збільшення розмірності LUT дозволяє будувати складні логічні функції без додаткових витрат.

Такі програмовані за допомогою таблиці істинності LUT є ефективним та гнучким способом реалізації широкого класу комбінаційних і послідовних логічних схем на базі FPGA.

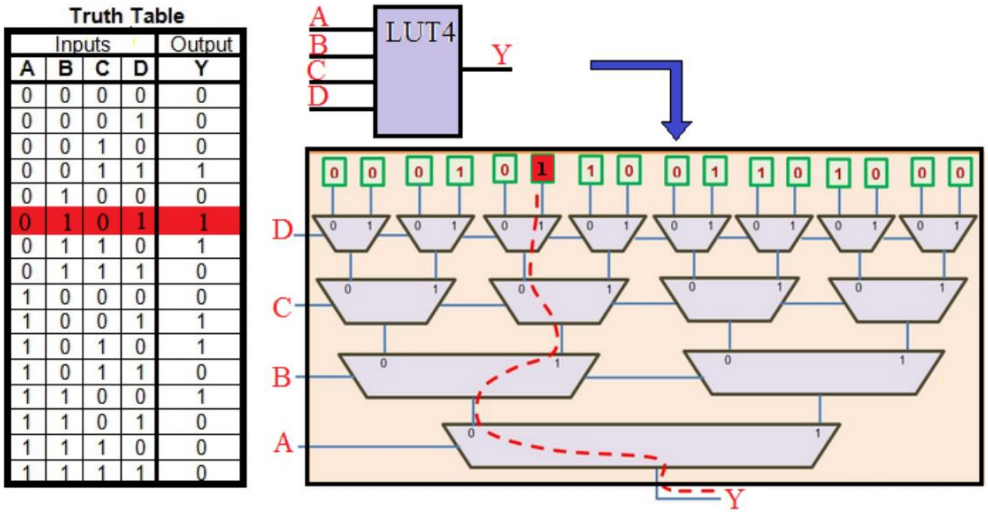

Рис. 3.5 – 4-входовий LUT, що реалізує таблицю істинності комбінаційної фукнції Y

Через обмежені апаратні ресурси FPGA головним завданням є їх оптимальний розподіл між функціональними вузлами процесора. Це вимагає знаходження компромісу між кількістю реалізованих блоків та їх продуктивністю. Важливим є проектування ефективної системи зв’язків та інтерфейсів між блоками за допомогою шин і комутаторів різної розрядності. Задля реалізації переваг паралельної обробки даних на FPGA доцільно максимально розпаралелити алгоритм. Водночас, це ускладнює синхронізацію.

4 МОДЕЛЮВАННЯ СПЕЦІАЛІЗОВАНОГО ПРОЦЕСОРУ В СИСТЕМІ ЗАЛИШКОВИХ КЛАСІВ

4.1 Синтез блоків спеціалізованого обчислювального інструменту в системі залишкових класів

Виходячи з основної властивості СОК, що полягає в паралельності виконання операцій, синтез блоків і вузлів СОІ в СЗК має ряд особливостей, які відрізняють його від синтезу традиційних обчислювальних систем.

1) Операційний пристрій СОІ в СЗК складається з окремо функціонуючих арифметичних пристроїв (суматори, віднімачі і помножувачі). Кожен арифметичний пристрій функціонує незалежно один від одного і паралельно в часі. Це означає, що при синтезі оперативного пристрою СОІ необхідно забезпечити паралельну роботу всіх арифметичних пристроїв. Для цього необхідно використовувати тактовий генератор, який забезпечує синхронізацію роботи всіх арифметичних пристроїв.

2) Арифметичний пристрій СОІ в СЗК складається з n послідовних обчислювачів, де n – кількість основ СОК. Кожен обчислювач працює в $k = \left\lceil \log_2 (m-1) \right\rceil + 1$ – розрядній сітці. Ця особливість синтезу арифметичних пристроїв СОІ в СЗК обумовлена тим, що арифметика СОК виконується в системі залишкових класів. У системі залишкових класів арифметичні операції реалізуються за допомогою послідовних перетворень, які виконуються в кожній із основ СОК. Кожен обчислювач арифметичного пристрою СОІ виконує арифметичні операції в одній із основ СОК. Розрядність кожного обчислювача визначається кількістю розрядів, необхідних для представлення чисел у відповідній основі. Таким чином, арифметичний пристрій СОІ виконує арифметичні операції за допомогою паралельного виконання послідовних перетворень в n різних основах СОК.

3) Малоразрядность залишків що представляють операнд у СЗК дозволяє реалізувати арифметичні операції як на базі малорозрядних

двійкових суматорів, так і на базі ПЗП, виконаних в табличному (матричному) варіанті. Дана обставина істотно впливає на час виконання арифметичних операцій в СОІ в СЗК. Малорозрядність залишків обумовлена тим, що в системі залишкових класів числа представляють у вигляді набору залишків від ділення на різні модулі. Кожен залишок може бути представлений у вигляді двійкового числа з невеликою кількістю розрядів. Це дозволяє реалізувати арифметику СОК за допомогою малорозрядних двійкових суматорів. Такий підхід забезпечує високу продуктивність арифметичних операцій, оскільки малорозрядні двійкові суматори мають високу швидкодію. Однак, використання малорозрядних двійкових суматорів може призвести до збільшення площі і споживання енергії арифметичних пристроїв СОІ. Для підвищення енергоефективності арифметичних пристроїв СОІ можна використовувати ПЗП, виконані в табличному (матричному) варіанті. Такий підхід дозволяє реалізувати арифметику СОК за допомогою невеликої кількості елементів ПЗП. Однак, використання ПЗП може призвести до зниження продуктивності арифметичних операцій, оскільки ПЗП мають більш низьку швидкодію, ніж малорозрядні двійкові суматори. Відповідно до цього, при виборі архітектури арифметичних пристроїв СОІ необхідно враховувати такі фактори, як продуктивність, енергоефективність, площа і споживання енергії.

4) Блокова побудова СОІ в СЗК дозволяє локалізувати помилки (або групи помилок, що не перевищують значення $k = \left\lceil \log_2 \left(m - 1 \right) \right\rceil + 1)$ у одному тракті обчислювача. Дана обставина дозволяє створити систему контролю і корекції помилок, не перериваючи рішення задачі (алгоритму), тобто проводити корекцію помилок в динаміці обчислювального процесу. При блочному побудові СОІ кожен обчислювач арифметичного пристрою реалізує одну із стадій арифметичної операції. Це дозволяє локалізувати помилки в одному тракті обчислювача.

4.2 Розробка HDL-моделі системи обробки інформації у системі залишкових класів

На рисунку 4.1 можемо побачити модель СОІ ТА у СЗК

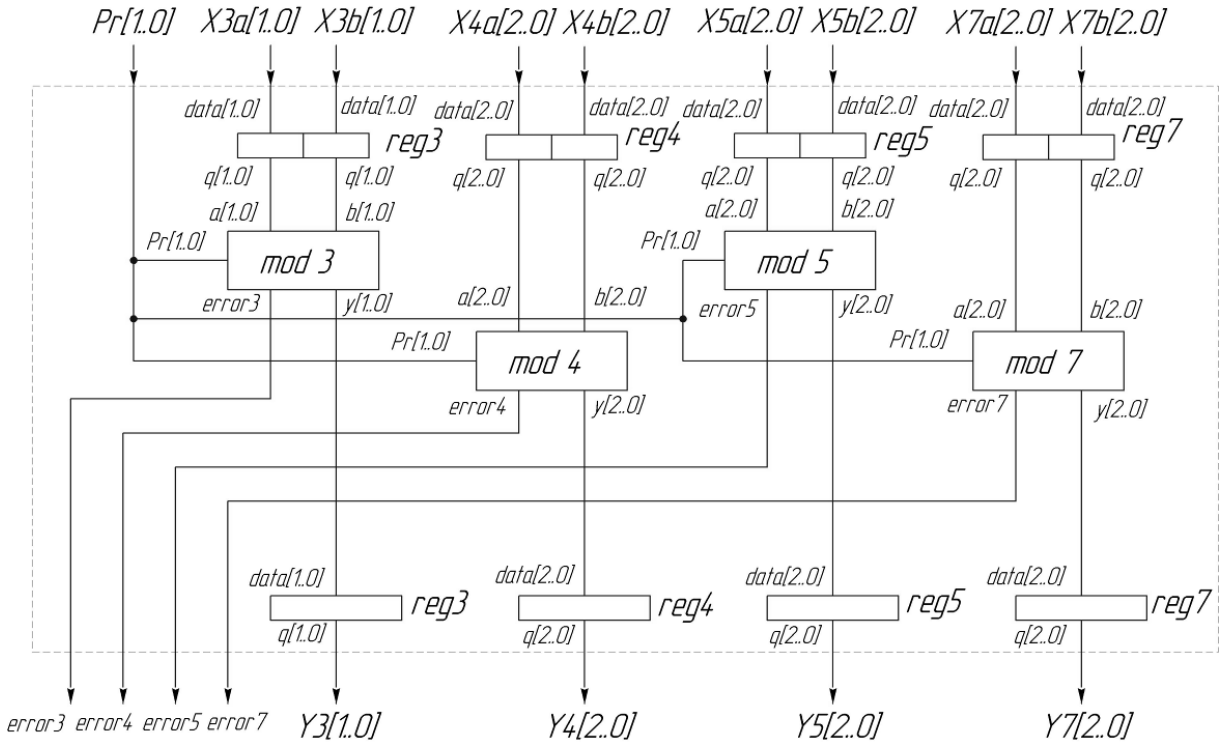

Рисунок 4.1 – HDL-модель СОІ ТА у СЗК

Розглянемо розроблені функціональні блоки HDL-модели СОІ у СЗК з зазначенням їх призначення, виконуваних функцій, і результатів моделювання їх роботи.

Функціональний блок *Reg_2* – блок вхідних і вихідних регістрів, який забезпечує зберігання вхідних і вихідних операндів. Містить вхід операнда data[1..0], вхід сигналу clock, що управляє, а також вихід q[1..0]. Опис блоку представлений нижче.

```
reg_2
INCLUDE "lpm_ff.inc";
SUBDESIGN reg_2
(
clock : INPUT;
data[1..0] : INPUT;
q[1..0] : OUTPUT;
)
VARIABLE
lpm_ff_component : lpm_ff WITH
(
LPM_WIDTH = 2,
LPM_FFTYPE = "DFF"
);
```

```
BEGIN
q[1..0] = lpm_ff_component.q[1..0];
lpm_ff_component.clock = clock;
lpm_ff_component.data[1..0] =
data[1..0];
END;
```

Результат моделювання роботи блоку показаний на рис. 4.2.

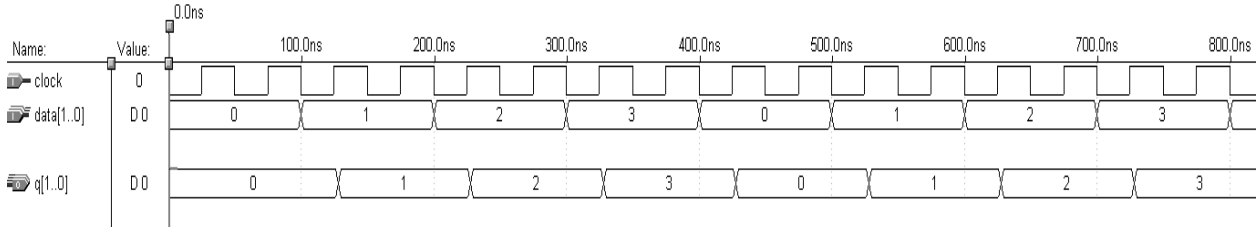

Рисунок 4.2 – Результат моделювання роботи блоку *Reg_2*

Функціональний блок *Reg_3* – блок вхідних і вихідних регістрів, який забезпечує зберігання вхідних і вихідних операндів. Містить вхід операнда data[2..0], вхід сигналу clock, що управляє, а також вихід q[2..0]. Опис блоку представлений нижче.

```
INCLUDE "lpm_ff.inc";
SUBDESIGN reg_3
(
clock : INPUT;
data[2..0] : INPUT;
q[2..0] : OUTPUT;
)
VARIABLE
lpm_ff_component : lpm_ff WITH
(
LPM_WIDTH = 3,
LPM_FFTYPE = "DFF"
);
BEGIN
q[2..0] = lpm_ff_component.q[2..0];
lpm_ff_component.clock = clock;
lpm_ff_component.data[2..0] =
data[2..0];
END;
```

Результат моделювання роботи блоку показаний на рис. 4.3.

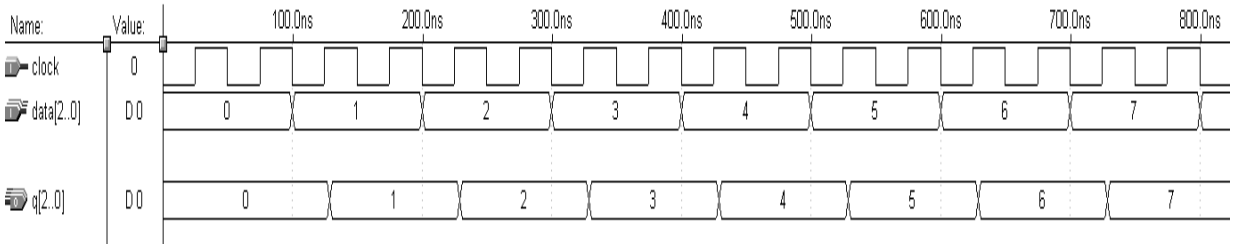

Рисунок 4.3 – Результат моделювання роботи блоку *Reg_3*

Функціональний блок *tab_mod_3* – блок комутаторів по модулю $m = 3$

призначений для виконання арифметичних операцій складання, віднімання і множення. Містить: входи операндів a[1..0] і b[1..0], вхід шини коди операції pr[1..0], а також виходи результату операції y[1..0] і сигналу помилки error3. Опис блоку представлений нижче.

```
SUBDESIGN tab_mod_3
(
a[1..0], b[1..0], pr[1..0] : INPUT;
y[1..0], error3: OUTPUT;
)
BEGIN
if pr[] == b"00" then %+%
TABLE
a[1..0], b[1..0]=>  y[1..0];
b"00", b"00"   =>  b"00";
 b"00", b"01"   =>  b"01";
b"00", b"10"   =>  b"10";
b"01", b"00"   =>  b"01";
b"01", b"01"   =>  b"10";
b"01", b"10"   =>  b"00";
b"10", b"00"   =>  b"10";
 b"10", b"01"   =>  b"00";
b"10", b"10"   =>  b"01";
END TABLE;
elsif pr[] == b"01" then %-%
TABLE
a[1..0], b[1..0] =>  y[1..0];
b"00", b"00"   =>  b"00";
b"00", b"01"   =>  b"10";
b"00", b"10"   =>  b"01";
b"01", b"00"   =>  b"01";
b"01", b"01"   =>  b"00";
b"01", b"10"   =>  b"10";
b"10", b"00"   =>  b"10";
b"10", b"01"   =>  b"01";
b"10", b"10"   =>  b"00";
END TABLE;
elsif pr[] == b"10" then %*%
TABLE
a[1..0], b[1..0] =>  y[1..0];
b"00", b"00"   =>  b"00";
 b"00", b"01"   =>  b"00";
b"00", b"10"   =>  b"00";
b"01", b"00"   =>  b"00";
b"01", b"01"   =>  b"01";
b"01", b"10"   =>  b"10";
b"10", b"00"   =>  b"00";
 b"10", b"01"   =>  b"10";
b"10", b"10"   =>  b"01";
END TABLE;
else  error3 = b"1";
end if;
END;
```

Результат моделювання роботи блоку показаний на рис. 4.4

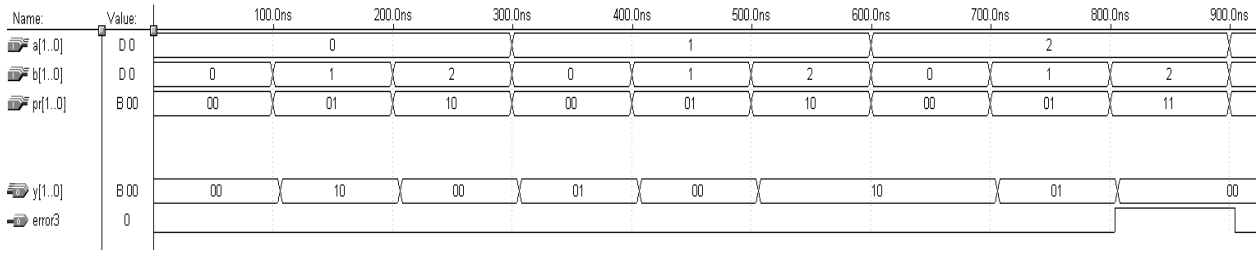

Рисунок 4.4 – Результат моделювання роботи блоку *tab_mod_3*

Функціональний блок *tab_mod_4* – блок комутаторів по модулю $m=4$ призначений для виконання арифметичних операцій складання, віднімання і множення. Містить: входи операндів a[1..0] і b[1..0], вхід шини коди операції pr[1..0], а також виходи результату операції y[1..0] і сигналу помилки error4. Опис блоку представлений нижче.

```
SUBDESIGN tab_mod_4
(
a[1..0], b[1..0], pr[1..0] : INPUT;
y[1..0], error4: OUTPUT;
)
BEGIN
if pr[] == b"00" then %+%
TABLE
a[1..0], b[1..0]=>  y[1..0];
b"00", b"00"   =>  b"00";
 b"00", b"01"   =>  b"01";
b"00", b"10"   =>  b"10";
b"00", b"11"   =>  b"11";
b"01", b"00"   =>  b"01";
b"01", b"01"   =>  b"10";
b"01", b"10"   =>  b"11";
b"01", b"11"   =>  b"00";
b"10", b"00"   =>  b"10";
 b"10", b"01"   =>  b"11";
b"10", b"10"   =>  b"00";
b"10", b"11"   =>  b"01";
b"11", b"00"   =>  b"11";
b"11", b"01"   =>  b"00";
b"11", b"10"   =>  b"01";
b"11", b"11"   =>  b"10";
END TABLE;
elsif pr[] == b"01" then %-%
TABLE
a[1..0], b[1..0] =>  y[1..0];
b"00", b"00"   =>  b"00";
 b"00", b"01"   =>  b"11";
b"00", b"10"   =>  b"10";
b"00", b"11"   =>  b"01";
b"01", b"00"   =>  b"01";
b"01", b"01"   =>  b"00";
b"01", b"10"   =>  b"11";
b"01", b"11"   =>  b"10";
b"10", b"00"   =>  b"10";
 b"10", b"01"   =>  b"01";
b"10", b"10"   =>  b"00";
b"10", b"11"   =>  b"11";
b"11", b"00"   =>  b"11";
```

```
b"11", b"01"   =>  b"10";
b"11", b"10"   =>  b"01";
b"11", b"11"   =>  b"00";
END TABLE;
elsif pr[] == b"10" then %*%
TABLE
a[1..0], b[1..0] =>  y[1..0];
b"00", b"00"   =>  b"00";
 b"00", b"01"   =>  b"00";
b"00", b"10"   =>  b"00";
b"00", b"11"   =>  b"00";
b"01", b"00"   =>  b"00";
b"01", b"01"   =>  b"01";
b"01", b"10"   =>  b"10";
b"01", b"11"   =>  b"11";
b"10", b"00"   =>  b"00";
 b"10", b"01"   =>  b"10";
b"10", b"10"   =>  b"00";
b"10", b"11"   =>  b"10";
b"11", b"00"   =>  b"00";
b"11", b"01"   =>  b"11";
b"11", b"10"   =>  b"10";
b"11", b"11"   =>  b"01";
END TABLE;
else  error4 = b"1";
end if;
END;
```

Результат моделювання роботи блоку показаний на рис. 4.5.

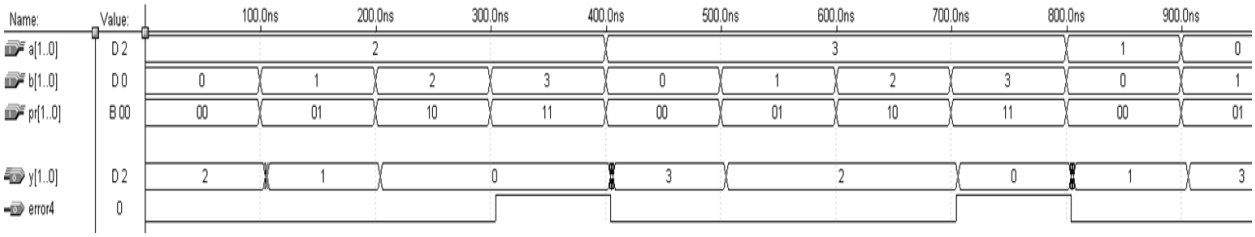

Рисунок 4.5 – Результат моделювання роботи блоку *tab_mod_4*

Функціональний блок *tab_mod_5* – блок комутаторів по модулю $m=5$ призначений для виконання арифметичних операцій складання, віднімання і множення. Містить: входи операндів a[2..0] і b[2..0], вхід шини коди операції pr[1..0], а також виходи результату операції y[2..0] і сигналу помилки error5. Опис блоку представлений нижче.

```
SUBDESIGN tab_mod_5
(
a[2..0], b[2..0], pr[1..0] : INPUT;
y[2..0], error5: OUTPUT;
)
BEGIN
if pr[] == b"00" then %+%
TABLE
a[2..0], b[2..0]=>  y[2..0];
b"000", b"000"   =>  b"000";
 b"000", b"001"   =>  b"001";
b"000", b"010"   =>  b"010";
```

```
b"000", b"011"   =>  b"011";
b"000", b"100"   =>  b"100";
b"001", b"000"   =>  b"001";
b"001", b"001"   =>  b"010";
b"001", b"010"   =>  b"011";
b"001", b"011"   =>  b"100";
 b"001", b"100"   =>  b"000";
b"010", b"000"   =>  b"010";
b"010", b"001"   =>  b"011";
b"010", b"010"   =>  b"100";
b"010", b"011"   =>  b"000";
b"010", b"100"   =>  b"001";
b"011", b"000"   =>  b"011";
b"011", b"001"   =>  b"100";
b"011", b"010"   =>  b"000";
b"011", b"011"   =>  b"001";
b"011", b"100"   =>  b"010";
b"100", b"000"   =>  b"100";
 b"100", b"001"   =>  b"000";
b"100", b"010"   =>  b"001";
b"100", b"011"   =>  b"010";
b"100", b"100"   =>  b"011";
END TABLE;
elsif pr[] == b"01" then %-%
TABLE
a[2..0], b[2..0] =>  y[2..0];
b"000", b"000"   =>  b"000";
 b"000", b"001"   =>  b"100";
b"000", b"010"   =>  b"011";
b"000", b"011"   =>  b"010";
b"000", b"100"   =>  b"001";
b"001", b"000"   =>  b"001";
b"001", b"001"   =>  b"000";
b"001", b"010"   =>  b"100";
b"001", b"011"   =>  b"011";
 b"001", b"100"   =>  b"010";
b"010", b"000"   =>  b"010";
b"010", b"001"   =>  b"001";
b"010", b"010"   =>  b"000";
b"010", b"011"   =>  b"100";
b"010", b"100"   =>  b"011";
b"011", b"000"   =>  b"011";
b"011", b"001"   =>  b"010";
b"011", b"010"   =>  b"001";
b"011", b"011"   =>  b"000";
b"011", b"100"   =>  b"100";
b"100", b"000"   =>  b"100";
 b"100", b"001"   =>  b"011";
b"100", b"010"   =>  b"010";
b"100", b"011"   =>  b"001";
b"100", b"100"   =>  b"000";
END TABLE;
elsif pr[] == b"10" then %*%
TABLE
a[2..0], b[2..0] =>  y[2..0];
b"000", b"000"   =>  b"000";
 b"000", b"001"   =>  b"000";
b"000", b"010"   =>  b"000";
b"000", b"011"   =>  b"000";
b"000", b"100"   =>  b"000";
b"001", b"000"   =>  b"000";
b"001", b"001"   =>  b"001";
```

```
b"001", b"010"   =>  b"010";
b"001", b"011"   =>  b"011";
 b"001", b"100"   =>  b"100";
b"010", b"000"   =>  b"000";
b"010", b"001"   =>  b"010";
b"010", b"010"   =>  b"100";
b"010", b"011"   =>  b"001";
b"010", b"100"   =>  b"011";
b"011", b"000"   =>  b"000";
b"011", b"001"   =>  b"011";
b"011", b"010"   =>  b"001";
b"011", b"011"   =>  b"100";
b"011", b"100"   =>  b"010";
b"100", b"000"   =>  b"000";
 b"100", b"001"   =>  b"100";
b"100", b"010"   =>  b"011";
b"100", b"011"   =>  b"010";
b"100", b"100"   =>  b"001";
END TABLE;
else  error5 = b"1";
end if;
END;
```

Результат моделювання роботи блоку показаний на рис. 4.6.

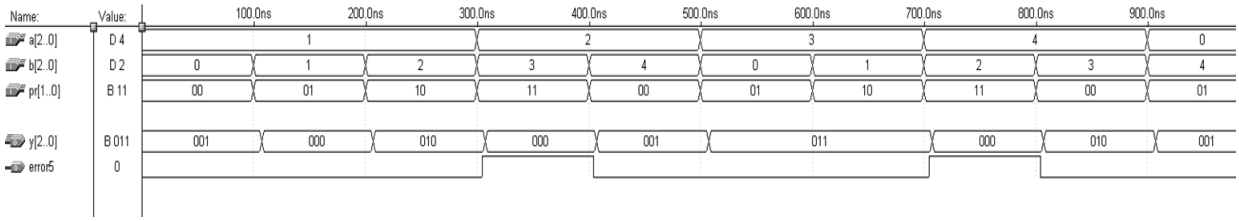

Рисунок 4.6 – Результат моделювання роботи блоку *tab_mod_5*

Функціональний блок *tab_mod_7* – блок комутаторів по модулю $m = 7$ призначений для виконання арифметичних операцій складання, віднімання і множення. Містить: входи операндів a[2..0] і b[2..0], вхід шини коди операції pr[1..0], а також виходи результату операції y[2..0] і сигналу помилки error5. Опис блоку представлений нижче.

```
SUBDESIGN tab_mod_7
(
a[2..0], b[2..0], pr[1..0] : INPUT;
y[2..0], error7: OUTPUT;
)
BEGIN
if pr[] == b"00" then %+%
TABLE
a[2..0], b[2..0]=>  y[2..0];
b"000", b"000"   =>  b"000";
 b"000", b"001"   =>  b"001";
b"000", b"010"   =>  b"010";
b"000", b"011"   =>  b"011";
b"000", b"100"   =>  b"100";
b"000", b"101"   =>  b"101";
b"000", b"110"   =>  b"110";
b"001", b"000"   =>  b"001";
b"001", b"001"   =>  b"010";
```

```
b"001", b"010"  =>  b"011";
b"001", b"011"  =>  b"100";
 b"001", b"100"  =>  b"101";
b"001", b"101"  =>  b"110";
b"001", b"110"  =>  b"000";
b"010", b"000"  =>  b"010";
b"010", b"001"  =>  b"011";
b"010", b"010"  =>  b"100";
b"010", b"011"  =>  b"101";
b"010", b"100"  =>  b"110";
b"010", b"101"  =>  b"000";
b"010", b"110"  =>  b"001";
b"011", b"000"  =>  b"011";
b"011", b"001"  =>  b"100";
b"011", b"010"  =>  b"101";
b"011", b"011"  =>  b"110";
b"011", b"100"  =>  b"000";
b"011", b"101"  =>  b"001";
b"011", b"110"  =>  b"010";
b"100", b"000"  =>  b"100";
 b"100", b"001"  =>  b"101";
b"100", b"010"  =>  b"110";
b"100", b"011"  =>  b"000";
b"100", b"100"  =>  b"001";
b"100", b"101"  =>  b"010";
b"100", b"110"  =>  b"011";
b"101", b"000"  =>  b"101";
 b"101", b"001"  =>  b"110";
b"101", b"010"  =>  b"000";
b"101", b"011"  =>  b"001";
b"101", b"100"  =>  b"010";
b"101", b"101"  =>  b"011";
b"101", b"110"  =>  b"100";
b"110", b"000"  =>  b"110";
 b"110", b"001"  =>  b"000";
b"110", b"010"  =>  b"001";
b"110", b"011"  =>  b"010";
b"110", b"100"  =>  b"011";
b"110", b"101"  =>  b"100";
b"110", b"110"  =>  b"101";
END TABLE;
elsif pr[] == b"01" then %-%
TABLE
a[2..0], b[2..0] =>  y[2..0];
b"000", b"000"  =>  b"000";
 b"000", b"001"  =>  b"110";
b"000", b"010"  =>  b"101";
b"000", b"011"  =>  b"100";
b"000", b"100"  =>  b"011";
b"000", b"101"  =>  b"010";
b"000", b"110"  =>  b"001";
b"001", b"000"  =>  b"001";
b"001", b"001"  =>  b"000";
b"001", b"010"  =>  b"110";
b"001", b"011"  =>  b"101";
 b"001", b"100"  =>  b"100";
b"001", b"101"  =>  b"011";
b"001", b"110"  =>  b"010";
b"010", b"000"  =>  b"010";
b"010", b"001"  =>  b"001";
b"010", b"010"  =>  b"000";
b"010", b"011"  =>  b"110";
```

```
b"010", b"100"   =>  b"101";
b"010", b"101"   =>  b"100";
b"010", b"110"   =>  b"011";
b"011", b"000"   =>  b"011";
b"011", b"001"   =>  b"010";
b"011", b"010"   =>  b"001";
b"011", b"011"   =>  b"000";
b"011", b"100"   =>  b"110";
b"011", b"101"   =>  b"101";
b"011", b"110"   =>  b"100";
b"100", b"000"   =>  b"100";
 b"100", b"001"   =>  b"011";
b"100", b"010"   =>  b"010";
b"100", b"011"   =>  b"010";
b"100", b"100"   =>  b"000";
b"100", b"101"   =>  b"110";
b"100", b"110"   =>  b"101";
b"101", b"000"   =>  b"101";
 b"101", b"001"   =>  b"100";
b"101", b"010"   =>  b"011";
b"101", b"011"   =>  b"010";
b"101", b"100"   =>  b"001";
b"101", b"101"   =>  b"000";
b"101", b"110"   =>  b"110";
b"110", b"000"   =>  b"110";
 b"110", b"001"   =>  b"101";
b"110", b"010"   =>  b"100";
b"110", b"011"   =>  b"011";
b"110", b"100"   =>  b"010";
b"110", b"101"   =>  b"001";
b"110", b"110"   =>  b"000";
END TABLE;
elsif pr[] == b"10" then %*%
TABLE
a[2..0], b[2..0] =>  y[2..0];
b"000", b"000"   =>  b"000";
 b"000", b"001"   =>  b"000";
b"000", b"010"   =>  b"000";
b"000", b"011"   =>  b"000";
b"000", b"100"   =>  b"000";
b"000", b"101"   =>  b"000";
b"000", b"110"   =>  b"000";
b"001", b"000"   =>  b"000";
b"001", b"001"   =>  b"001";
b"001", b"010"   =>  b"010";
b"001", b"011"   =>  b"011";
b"001", b"100"   =>  b"100";
b"001", b"101"   =>  b"101";
b"001", b"110"   =>  b"110";
b"010", b"000"   =>  b"000";
b"010", b"001"   =>  b"010";
b"010", b"010"   =>  b"100";
b"010", b"011"   =>  b"110";
b"010", b"100"   =>  b"001";
b"010", b"101"   =>  b"011";
b"010", b"110"   =>  b"101";
b"011", b"000"   =>  b"000";
b"011", b"001"   =>  b"011";
b"011", b"010"   =>  b"110";
b"011", b"011"   =>  b"010";
b"011", b"100"   =>  b"101";
b"011", b"101"   =>  b"001";
```

```
b"011", b"110"   =>  b"100";
b"100", b"000"   =>  b"000";
b"100", b"001"   =>  b"100";
b"100", b"010"   =>  b"001";
b"100", b"011"   =>  b"101";
b"100", b"100"   =>  b"010";
b"100", b"101"   =>  b"110";
b"100", b"110"   =>  b"011";
b"101", b"000"   =>  b"000";
 b"101", b"001"   =>  b"101";
b"101", b"010"   =>  b"011";
b"101", b"011"   =>  b"001";
b"101", b"100"   =>  b"110";
b"101", b"101"   =>  b"100";
b"101", b"110"   =>  b"101";
b"110", b"000"   =>  b"000";
b"110", b"001"   =>  b"110";
b"110", b"010"   =>  b"101";
b"110", b"011"   =>  b"100";
b"110", b"100"   =>  b"011";
b"110", b"101"   =>  b"010";
b"110", b"110"   =>  b"001";
END TABLE;
else  error7 = b"1";
end if;
END;
```

Результат моделювання роботи блоку показаний на рис. 4.7.

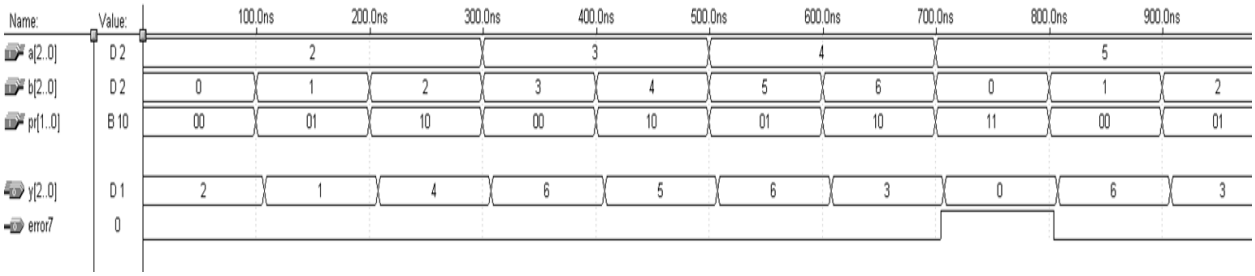

Рисунок 4.7 – Результат моделювання роботи блоку *tab_mod_7*

Опис COI за системою основ $m = 3, m = 4, m = 5, m = 7$, що включає розроблені функціональні блоки представлено нижче.

```
INCLUDE "tab_mod_3";
INCLUDE "tab_mod_4";
INCLUDE "tab_mod_5";
INCLUDE "tab_mod_7";
SUBDESIGN prosessor_1
(
x3a[1..0], x3b[1..0], x4a[1..0], x4b[1..0],
x5a[2..0], x5b[2..0], x7a[2..0], x7b[2..0],
pr[1..0] : INPUT ;
y3[1..0], y4[1..0], y5[2..0], y7[2..0],
error3, error4, error5, error7 : OUTPUT;
)
VARIABLE
tabmod_3 : tab_mod_3;
tabmod_4 : tab_mod_4;
tabmod_5 : tab_mod_5;
tabmod_7 : tab_mod_7;
BEGIN
```

```
tabmod_3.a[1..0] = x3a[];
tabmod_3.b[1..0] = x3b[];
tabmod_4.a[1..0] = x4a[];
tabmod_4.b[1..0] = x4b[];
tabmod_5.a[2..0] = x5a[];
tabmod_5.b[2..0] = x5b[];
tabmod_7.a[2..0] = x7a[];
tabmod_7.b[2..0] = x7b[];
tabmod_3.pr[] = pr[];
tabmod_4.pr[] = pr[];
tabmod_5.pr[] = pr[];
tabmod_7.pr[] = pr[];
error3 = tabmod_3.error3;
error4 = tabmod_4.error4;
error5 = tabmod_5.error5;
error7 = tabmod_7.error7;
y3[1..0] = tabmod_3.y[1..0];
y4[1..0] = tabmod_4.y[1..0];
y5[2..0] = tabmod_5.y[2..0];
y7[2..0] = tabmod_7.y[2..0];
end;
```

Результат моделювання роботи СОІ з набором основ $m_1 = 3$, $m_2 = 4$, $m_3 = 5$, $m_4 = 7$ для операндів $A_{17} = (2,1,2,3)$ та $B_{13} = (1,1,3,6)$ при виконанні арифметичних операцій множення, складання та віднімання показаний на рис. 4.8.

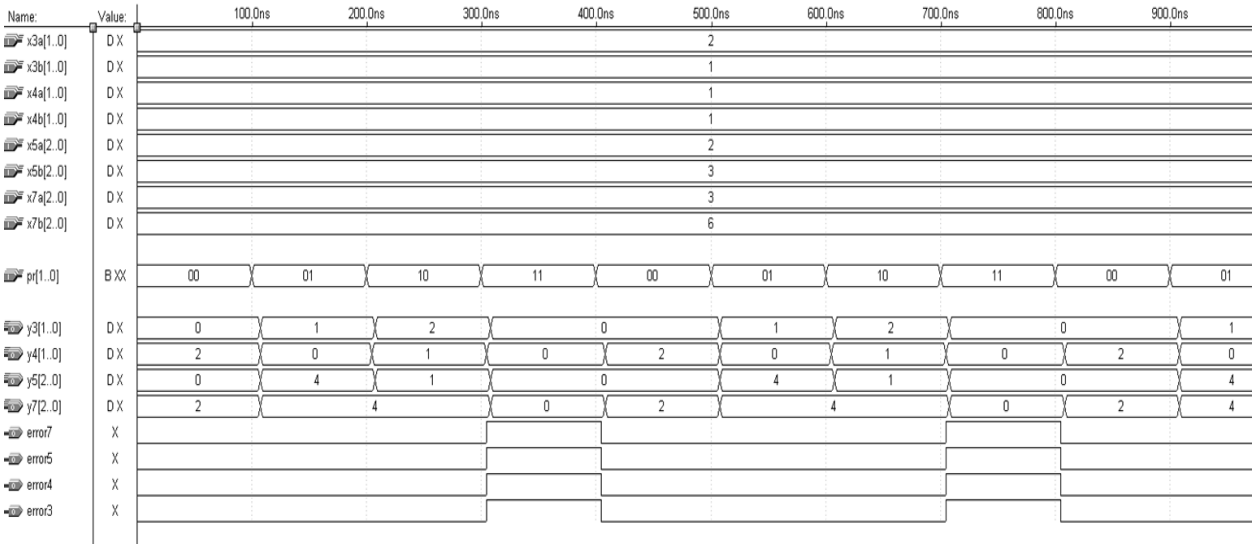

Рисунок 4.8 – Результат моделювання роботи СОІ

Для реалізації алгоритму RSA був розроблений блок додавання чи віднімання чисел з постійною основою, який можна побачити в додатку А.

4.3 Вибір мікросхеми та аналіз параметрів для реалізації спеціалізованого обчислювального інструменту

Вибір мікросхеми здійснювався за допомогою системи автоматизованого проектування фірмою Altera, що поставлялася, - MAX+plusII, на основі розробленої HDL-моделі СОІ ТА. Мікросхема EPM3064ATC100-4 вибрана з сімейства MAX3000A. Основні параметри мікросхеми EPM3064A приведені в таблиці 4.1.

Кількість входів мікросхеми – 22, виходів – 14. В цілому ресурси мікросхеми використовуються на 68%, що дає можливість їх додаткового використання для поліпшення окремих властивостей (зокрема відмовостійкій і надійності) СОІ ТА у СЗК. Мікросхема складається з 4-х логічних блоків A, B, C, D які представлені на рис. 4.9. Логічні блоки задіяні на 44%.

Таблиця 4.1 – Основні параметри мікросхеми EPM3064A

Найменування параметра	Значення параметрів
Логічна ємкість, еквівалентних вентилів	1250
Число макрокомірок	64
Число логічних блоків	4
Затримка розповсюдження сигналу вхід-вихід, t_{PD} [нс]	4,5
Час установки глобального тактового сигналу, t_{SU} [нс]	3
Затримка глобального тактового сигналу до виходу, t_{CO1} [нс]	2,8
Максимальна глобальна тактова частота, f_{CNT} [Мгц]	192,3

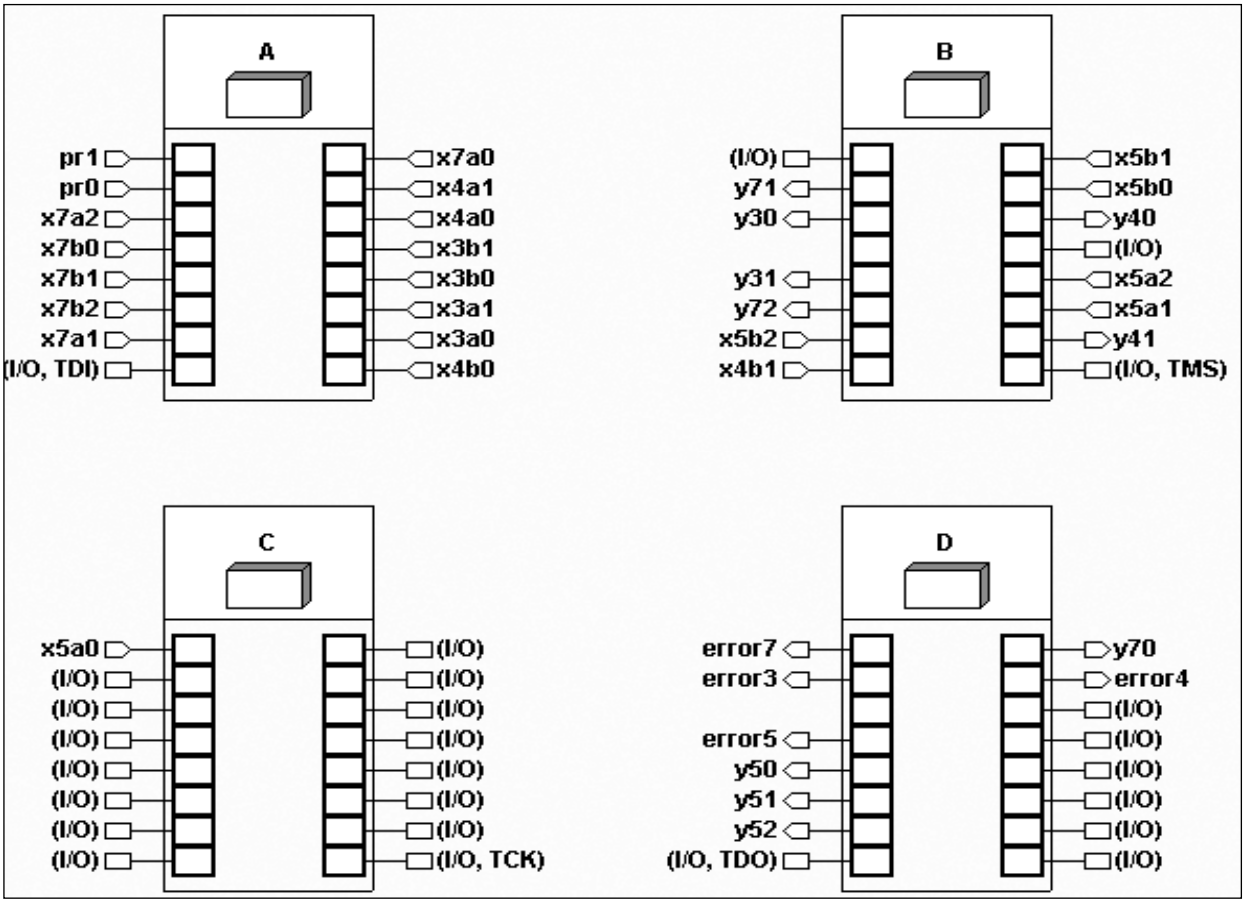

Рисунок 4.9 – Логічних блоків мікросхеми.

Отримані часи затримок реалізації кожній з арифметичних операцій, які приведені в таблиці 4.2, ці результати підтверджують теоретичні дослідження ефективності використання СЗК для побудови швидкодіючих процесорів обробки цифрової інформації.

Таблиця 4.2 – Час затримки реалізації арифметичних операцій

	error3	error4	error5	error7	y30	y31	y40	y41	y50	y51	y52	y70	y71	y72
pr0	4,7ns	4,7ns	4,7ns	4,7ns	7,8ns	6,5ns\7,8ns	4,7ns	6,5ns\7,8ns	6,5ns\7,8ns	6,5ns\7,8ns	6,5ns\7,8ns	7,8ns	7,8ns	7,8ns
pr1	4,7ns	4,7ns	4,7ns	4,7ns	6,5ns\7,8ns	6,5ns\7,8ns	4,7ns	6,5ns\7,8ns	6,5ns\7,8ns	6,5ns\7,8ns	6,5ns\7,8ns	7,8ns	7,8ns	7,8ns
x3a0					6,3ns\7,8ns	6,3ns\7,8ns								
x3a1					6,3ns\7,8ns	6,3ns\7,8ns								
x3b0					6,3ns\7,8ns	6,3ns\7,8ns								
x3b1					6,3ns\7,8ns	6,3ns\7,8ns								
x4a0							4,5ns	7,6ns						
x4a1								4,5ns\7,6ns						
x4b0							4,5ns	6,3ns\7,6ns						
x4b1								6,3ns\7,6ns						
x5a0									6,3ns\7,6ns	6,3ns\7,6ns	6,3ns\7,6ns			
x5a1									6,3ns\7,6ns	6,3ns\7,6ns	6,3ns\7,6ns			
x5a2									6,3ns\7,6ns	6,3ns\7,6ns	6,3ns\7,6ns			
x5b0									6,3ns\7,6ns	6,3ns\7,6ns	6,3ns\7,6ns			
x5b1									6,3ns\7,6ns	6,3ns\7,6ns	6,3ns\7,6ns			
x5b2									6,3ns\7,6ns	6,3ns\7,6ns	6,3ns\7,6ns			
x7a0												7,7ns	7,7ns	7,7ns
x7a1												7,7ns	7,7ns	7,7ns
x7a2												7,7ns	7,7ns	7,7ns
x7b0												7,7ns	7,7ns	7,7ns
x7b1												7,7ns	7,7ns	7,7ns
x7b2												7,7ns	7,7ns	7,7ns

Отримані результати синтезу СОІ ТА можуть бути використані для побудови СП в СЗК для розрядної сітки при $l \geq 2$ і їх можна рекомендувати для використання як арифметичні розширювачі базових обчислювальних систем.

ВИСНОВКИ

У дипломній роботі проведено комплексне дослідження можливостей підвищення ефективності реалізації алгоритмів криптографії на основі використання системи залишкових класів. Запропонований підхід дозволяє оптимізувати один з найпоширеніших алгоритмів - RSA, розширивши сферу його застосування в системах з обмеженими ресурсами.

Здійснено ґрунтовний аналіз теоретичних основ криптографії та алгоритму RSA. Показано, що незважаючи на криптографічну стійкість, даний алгоритм є обчислювально вимогливий, що ускладнює його реалізацію в системах з обмеженими ресурсами.

Досліджено можливості системи залишкових класів для оптимізації обчислень з великими цілими числами, необхідними в алгоритмі RSA. Проаналізовано переваги даної технології для побудови швидкодіючих пристроїв обробки даних.

Розроблено спеціалізований засіб цифрової обробки на основі системи залишкових класів, призначений для ефективної реалізації алгоритму RSA. Засіб включає систему модулів, спеціалізовані регістри та високопродуктивну арифметику залишкових класів.

Створено HDL-модель пристрою обробки інформації в системі залишкових класів. Модель дозволяє виконувати операції з числами за модулями 3, 4, 5 та 7. Проведено успішне моделювання розробленої HDL-моделі.

Для практичної реалізації запропонованого рішення обрана мікросхема EPM3064ATC100-4. За результатами аналізу її характеристик зроблено висновок про доцільність використання даної мікросхеми для побудови високопродуктивного криптографічного засобу з алгоритмом RSA на основі розробленого спеціалізованого обчислювача в системі залишкових класів.

Отримані результати свідчать про ефективність запропонованого підходу для оптимізації реалізації складних криптографічних алгоритмів. Розробка може бути використана як теоретична та практична основа для створення

спеціалізованих криптографічних систем, придатних для застосування в пристроях з обмеженими ресурсами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Frank M.G., Groom K.O., Stephan S.A. Jones Network and Data Security for Non-Engineers. Atlanta, 2016, pp. 81-86. URL: https://doi.org/10.1201/9781315381138 (дата звернення 14.03.2024).
2. Bushra S.I., Farheen S.H. Comparison Between RSA Algorithm and Modified RSA Algorithm Used in Cloud Computing . Boston, 2019, pp. 218-224. URL: https://doi.org/10.1007/978-3-030-33846-6_24 (дата звернення 14.03.2024).
3. Rivest, R.L., Shamir, A.A., Adleman, L.M. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM. Chicago, 2008, pp. 120-126. URL: https://doi.org/10.15407/infocom22.04.102 (дата звернення 15.03.2024).
4. Guczal, V.V., Kushnir, I.V., Yatsuk, M.V. Implementation of the RSA cryptoalgorithm based on a specialized data processing tool in the residue class system. Informatics and Computer Engineering. Dallas, 2021, pp. 102-110. URL: https://doi.org/10.1007/978-3-319-22829-3 (дата звернення 17.03.2024).
5. Conway R.K., Nelson A.J. Improved RNS FIR filter architectures. IEEE Transactions on Circuits and Systems. Detroit, 2004. P. 26–28.
6. Piestrak S.J., Berezowski K.A. Architecture of efficient RNS-based digital signal processor with very low-level pipelining. Proceedings of the International Conference, Galway, Republic of Ireland, 2008. P. 127–132.
7. Кошман С. О., Барсов В.І., Краснобаєв В.А. Диверсність табличних методів реалізації арифметичних операцій у системі залишкових класів. Вісник Харківського національного технічного університету сільського господарства імені Петра Василенка. Проблеми енергозабезпечення та енергозбереження в АПК України. Харків: ХНТУСГ, 2008. Вип. 73, т. 2. С. 70–72.
8. Кошман С. О., Краснобаєв В.А., Фурман І.О. Аналіз табличних алгоритмів реалізації модульних операцій у автоматизованих системах обробки цифрової інформації. Вісник Харківського державного технічного

університету сільського господарства. Проблеми енергозабезпечення та енергозбереження в АПК України. Харків: ХДТУСГ, 2004. Вип. 27. С. 174–178.

9. Кошман С. О. Алгоритм оптимізації обладнання систем обробки інформації АСКОЕ, що функціонують у модулярній. Вісник Харківського державного технічного університету сільського господарства. Проблеми енергозабезпечення та енергозбереження в АПК України. Харків: ХДТУСГ, 2005. Вип. 37. С. 240–244.
10. Buhrow B.A., Gilbert B.B., Haider C.T. Parallel modular multiplication using 512-bit advanced vector instructions: RSA fault-inxection countermeasure via interleaved parallel multiplication. Xournal of Cryptographic Engineering. 2021. P. 46-53.
11. Haches G.A., Quisquate X.X. Montgomery multiplication with no final subtraction. Cryptographic Hardware and Embedded SystemCHES’2000. LNCS-1965, Springer-Verlag. 2000. P. 293-301.
12. Elfard S.S. Xustification of Montgomery Modular Reductions. Advanced Computing. 2012. P. 41-45.
13. Laszlo H.W. Long Modular multiplication for Cryptographic Applications. Cryptographic Hardware and Embedded System- CHES’2004. LNCS-3156, Springer-Verlag. 2004. P. 45-61.
14. Giorgi P.I., Imbert L.S., Izard T.U. Parallel modular multiplication on multi-core processors. IEEE Symposium on Computer Arithmetic. TX, United States, Austin, 2013. P. 135-142.
15. Buhrow B.B., Gilbert B.I., Haider C.A. Parallel modular multiplication using 512-bit advanced vector instructions: RSA fault-inxection countermeasure via interleaved parallel multiplication. Xournal of Cryptographic Engineering. 2021. P. 46-53.
16. Elfard S.E. Xustification of Montgomery Modular Reductions. Advanced Computing. 2012. P. 41-45.
17. Kawamura S.K., Takabayashi S.O., Shimbo A.P. A fast modular exponentiation algorithm. IEEE Transaction on Information Theory. 2015. P. 2136-2142.

18. Анісімов А.В. Алгоритмічна теорія великих чисел. Академперіодика. 2001. С. 153.
19. Agrawal D.G., Archambeault B.O., Rao J.R. The EM Side-Channel(s). Cryptographic Hardware and Embedded Systems. San Francisco, 2002. P. 29 - 45.
20. Muir J.A. Techniques of Side Channel Cryptanalysis: thesis requirement for the degree of Master of Mathematics in Combinatorics and Optimization. Waterloo, Ontario, Canada, 2001. P. 92-93.
21. Kelsey J.J., Schneier B.R., Wagner D.E. Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security. 2000. V. 8. P. 141-158.
22. Tiri K.O. Side-Channel Attack Pitfalls. Design Automation Conference, June 4-8, 2007: Proceedings. San Diego, California, USA, 2007. P. 15-20.
23. Wollinger T.D., Paar C.H. How Secure Are FPGAs in Cryptographic Applications? Field Programmable Logic and Applications. Lisbon, Portugal, 2003. P.91-100.
24. Groza B.V., Petrica D.A. Cryptanalysis of an Autentication Protocol. Symbolic and Numeric Algorithms for Scientific Computing. 2006. P. 147-157.

ДОДАТОК А

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ОБОРОНИ
АЗЕРБАЙДЖАНСЬКОЇ РЕСПУБЛІКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
"ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ"

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

НАЦІОНАЛЬНИЙ АЕРОКОСМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ М. Є. ЖУКОВСЬКОГО
"ХАРКІВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ"

УНІВЕРСИТЕТ МІСТА ЖИЛІНА

СУЧАСНІ НАПРЯМИ РОЗВИТКУ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЗАСОБІВ УПРАВЛІННЯ

Тези доповідей тринадцятої міжнародної
науково-технічної конференції
26 – 27 квітня 2023 року
Том 1: секції 1, 3, 4

Баку – Харків – Жиліна – 2023

IMPLEMENTATION OF THE RSA CRYPTO-ALGORITHM BASED ON A SPECIALIZED DATA PROCESSING TOOL IN THE SYSTEM OF RESIDUAL CLASSES

Koshman S., Krasnobayev V., Kinchyk A.
V.N. Karazin Kharkiv National University, Kharkiv, Ukraine

The RSA (Rivest-Shamir-Adleman) algorithm is one of the most popular cryptographic algorithms used to ensure the security of data transmission in information systems. The conducted analysis showed that the implementation of this algorithm requires significant computing resources, therefore reducing the time of its implementation is an urgent and important task. The implementation of this task is carried out by optimizing the use of memory, using specialized processors or dividing the computing work between several devices. Various methods are used as a mathematical apparatus for the implementation of the RSA algorithm, including Montgomery's fast arithmetic and Tom and Rivest's recursive formulas. For the technical implementation of the RSA algorithm, there are specialized processors that are specifically designed to perform cryptographic operations. For example, Intel AES-NI processors that support a hardware implementation of the RSA algorithm. They speed up calculations by reducing the time required for encryption and decryption operations. But it should be noted that all these tools process data presented in the positional numbering system, which in turn does not allow parallelizing the above-mentioned algorithms at the level of microoperations [1, 2].

The purpose of the report is to reduce the implementation time of the RSA crypto-algorithm when using specialized tools for processing integer data in the system of residual classes.

The report presents specialized tools for processing integer data (STPID), which are based on the use of a non-positional system of residual classes (SRC). It is noted that the use of such STPID ensures a significant acceleration of cryptographic data processing algorithms, in particular RSA, which in turn positively affects the speed of data processing systems. It is shown that the efficiency of using SRC is conditioned by such properties as independence, low-bit and equality of residues. At the same time, the representation of input data in the form of low-bit residues ensures an increase in the performance of the RSA algorithm implementation based on the use of existing methods. The conducted research showed that this approach to the construction of STPID allows to increase the efficiency of the RSA algorithm implementation.

References

1. Frank M. Groom, Kevin Groom, Stephan S. Jones Network and Data Security for Non-Engineers. 2016, p. 81-86. DOI: 10.1201/9781315381138

2. Bushra S., Farheen S. Comparison Between RSA Algorithm and Modified RSA Algorithm Used in Cloud Computing 2019, p. 218-224. DOI: 10.1007/978-3-030-33846-6_24