НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кваліфікаційна наукова
праця на правах рукопису

Жолубак Іван Михайлович

УДК 004.315.5 : 004.382 : 512.624 : 621.3.049.771.14

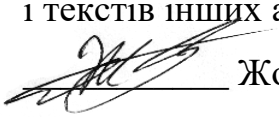
ДИСЕРТАЦІЯ

Методи та засоби створення реконфігурованих вузлів криптографічного захисту інформації для кібер-фізичних систем

05.13.05 – комп’ютерні системи та компоненти

Дисертація на здобуття наукового ступеня кандидата технічних наук

Подається на здобуття наукового ступеня кандидата технічних наук. Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.


Жолубак І. М.

Науковий керівник:
Глухов Валерій Сергійович
доктор технічних наук, професор

Львів-2024

АНОТАЦІЯ

Жолубак Іван Михайлович. Методи та засоби створення реконфігурованих вузлів криптографічного захисту інформації для кібер-фізичних систем. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.05 «Комп'ютерні системи та компоненти». – Національний університет «Львівська політехніка», Львів, 2024.

У дисертації розв'язується важливе науково-технічне завдання створення реконфігурованих вузлів криптографічного захисту інформації (КЗІ), які оперують у кіберфізичних системах (КФС) елементами розширених полів Галуа $GF(p^n)$, та у порівнянні з розширеними двійковими полями $GF(2^m)$ мають більшу криптографічну стійкість. При цьому $p^n \approx 2^m$, де $p > 2$ – просте число, конфігурована характеристика поля, n, m – порядок утворюючого поле полінома, $m \leq 1024$. Основним вузлом для аналізу обрано помножувач елементів таких розширених полів Галуа $GF(p^n)$, який побудовано на основі реконфігурованого вузла – модифікованої комірки Гілда (МКГ). Запропоновано 3 варіанти структури МКГ та показано їх особливості в порівнянні із класичною коміркою Гілда (КГ). Наведено порівняння апаратних витрат різних варіантів МКГ та помножувачів.

У **Вступі** виконано аналіз сучасних наукових досліджень у галузі розробки реконфігурованих компонентів для КЗІ на основі розширених полів Галуа, в тому числі засобів, що використовують еліптичні криві (ЕК). Обґрунтовано важливість використання генераторів ядер помножувачів для елементів розширених полів Галуа $GF(p^n)$, які знаходять застосування у системах КЗІ на основі ЕК. Сформульовано ціль та задачі дослідження, проаналізовано головні наукові досягнення та вказано їхню практичну цінність. Також висвітлено взаємозв'язок даної роботи з існуючими науковими програмами, планами і темами досліджень. Подається інформація про процес апробації результатів, публікації та використанні їх у практичній діяльності.

У роботі наведено структуру та архітектуру КФС, у складі якої є вузли КЗІ. ЕК широко використовуються у вузлах захисту інформації. В Україні стандарт

ДСТУ 4145-2002 регулює використання ЕЦП на основі ЕК. Він обмежує використання розширених полів Галуа двійковими полями $GF(2^m)$ з порядком утворюючого поле полінома $m \leq 509$ ($GF(2^{509})$), проте міжнародні стандарти рекомендують використовувати двійкові поля з набагато більшими порядками утворюючих поліномів ($m \leq 998$). Сучасні темпи розвитку комп'ютерної техніки та загроза створення квантових комп'ютерів ведуть до створення більш стійких засобів КЗІ, до збільшення порядку (p^n) розширених полів Галуа $GF(p^n)$, які використовуються. Один з найбільш очевидних методів злому засобів КЗІ є метод перебору усіх ключів. Програмне виконання операцій над елементами розширених полів Галуа $GF(p^n)$ має більшу трудоємність, у порівнянні з $GF(2^m)$ та забезпечує більшу стійкість до злому. Апаратна реалізація реконфігурованих вузлів КЗІ забезпечує ще більшу криптографічну стійкість засобів КЗІ. За елементну базу для створення вузлів КЗІ у складі КФС у роботі було обрано програмовані логічні інтегральні схеми (ПЛІС), оскільки вони забезпечують високу продуктивність та швидкодію при виконанні вузькоспеціалізованих задач у порівнянні з програмною реалізацією. Для генерації *VHDL*-описів вузлів КЗІ, що працюють з використанням розширених полів Галуа $GF(p^n)$, для їхньої наступної реалізації у ПЛІС було розроблено мовою *C++* програми-генератори ядер помножувачів елементів розширених полів Галуа.

У **Першому** розділі розглянуто сучасний стан та перспективи розвитку засобів та методів створення реконфігурованих вузлів КЗІ. Показано місце розширених полів Галуа $GF(p^n)$ у алгоритмах КЗІ КФС, розглянуто методи створення реконфігурованих вузлів на ПЛІС. Особливу увагу приділено правилам виконання арифметичних операцій у розширених полях Галуа $GF(p^n)$. Показано, що операції множення та ділення найбільш трудомісткі, при цьому ділення найчастіше виконується програмно. Тому саме операції множення приділено найбільше часу та уваги у даній роботі.

Також розглянуто питання складності алгоритмів та апаратно-програмна модель алгоритмів, питання злому систем КЗІ, особливості тестування операцій-

них елементів для розширених полів Галуа $GF(p^n)$, показано структуру комірки Гілда, підходи до створення генераторів ядер.

У **Другому розділі** розглянуто методи створення реконфігурованих вузлів КЗІ для КФС, загальну методику проведення дисертаційних досліджень, описано вимоги до створення реконфігурованих вузлів КЗІ, обґрунтовано доцільність створення паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ та запропоновано методи створення таких помножувачів. Вдосконалено методи оцінки часової та апаратної складностей та запропоновано метод тестування генераторів ядер таких помножувачів. Також наведено теоретичні підрахунки апаратної складності при реалізації помножувачів за трьома структурами МКГ:

1. МКГ є цілісним елементом (“чорна скринька”) і для кожного виходу цієї комірки формується своя булева функція та проводиться мінімізація цих булевих функцій методом Квайна–Мак-Класкі–Петрика (ЧС);

2. МКГ створена на основі функціональних вузлів: помножувача та суматора. Проводиться мінімізація методом Квайна–Мак-Класкі–Петрика булевих функцій, які описують роботу помножувача та суматора (ФВ);

3. МКГ створена на основі комбінаційних логічних елементів. МКГ реалізується, як комбінаційний паралельний, матричний помножувач за модулем характеристики поля та суматор за модулем характеристики поля (ЛВ).

Запропоновано та доопрацьовано наступні методи:

1. Отримав подальший розвиток метод оцінювання часової складності множення елементів розширених полів Галуа $GF(p^n)$ апаратним способом, за яким помножувач складається з МКГ, що дало можливість визначити поле $GF(3^n)$, у якому відношення часів множення програмним та апаратним способами перевищує таке відношення в інших полях щонайменше в 1,27 раза, чим забезпечує найбільшу криптографічну стійкість засобів КЗІ при інших однакових умовах.

2. Отримав подальший розвиток метод оцінювання апаратної складності помножувачів елементів розширених недвійкових полів Галуа $GF(p^n)$, де $p > 2$, який на відміну від відомих розглядає помножувачі для полів з приблизно одна-

ковим порядком p^n і які складаються з МКГ, що дало можливість визначити поле $GF(3^n)$, де помножувачі мають щонайменше на 6 % меншу апаратну складність, у порівнянні з помножувачами для інших недвійкових полів.

3. Вперше запропоновано метод створення для ПЛІС генераторів моделей (ядер) реконфігурованих паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ для вузлів КЗІ у КФС, за яким на відміну від відомих генерується 3 варіанти помножувачів з 3 структурами МКГ, що дало можливість створити описи моделей помножувачів та визначити реальну апаратну складність кожного із помножувачів та обрати найкращу реалізацію для кожного конкретного поля.

4. Вперше запропоновано метод тестування генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$, який на відміну від відомих використовує 2 різні еталони (2 різні математичні пакети) для перевірки згенерованих помножувачів на основі МКГ, що дало можливість підтвердити правильну роботу генераторів помножувачів та самих помножувачів для всіх варіантів їх реалізації.

Розглянуто теоретичні основи підрахунку апаратних витрат на реалізацію помножувачів елементів розширених полів Галуа $GF(p^n)$.

Проведено порівняння часових складностей елементів розширених полів Галуа $GF(p^n)$ у математичному пакеті *Maple*. За результатами порівняння найбільшу часову складність мають помножувачі для полів Галуа $GF(3^n)$.

Проведено аналіз апаратних витрат на множення елементів розширених полів Галуа $GF(p^n)$ апаратним способом. Порівняння проведено за трьома структурами МКГ.

Для структури МКГ ЧС помножувачі для поля $GF(3^n)$ мають кращі (менші) показники апаратної складності ніж для поля $GF(2^m)$, для структури МКГ ФВ помножувачі для полів $GF(3^n)$, $GF(5^n)$ та $GF(7^n)$ мають кращі (менші) показники апаратної складності ніж поля $GF(2^m)$. Для структури МКГ ЛВ коефіцієнт відносної апаратної складність помножувачів прямує до константного зна-

чення $4/7$, що свідчить про перевагу помножувачів для недвійкових полів $GF(p^n)$ над помножувачами для поля $GF(2^m)$.

У **Третьому розділі** описано процес розробки засобів створення (генераторів ядер) помножувачів елементів розширених полів Галуа $GF(p^n)$ для вузлів КЗІ КФС. Генератори були реалізовані мовою *C++*. Генератори створюють *VHDL*-описи (ядра) помножувачі за запропонованим у роботі методом створення для ПЛІС генераторів моделей (ядер) реконфігурованих паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ для вузлів КЗІ у КФС.

Мінімізація булевих функцій відбувається методом Квайна–Мак-Класкі–Петрика. Генерування помножувачів елементів розширених полів Галуа $GF(p^n)$ з великими характеристиками поля p та порядками p^n є дуже часозатратною задачею. Наприклад, генерація помножувача для поля $GF(53^{174})$ триває 1845 с. У роботі визначено час генерування помножувачів і для інших полів.

Також описано процес і наведено результати тестування згенерованих помножувачів запропонованим у роботі методом. Було протестовано роботу помножувачів для полів $GF(3^{30})$, $GF(7^{17})$, $GF(23^{10})$, $GF(53^{8})$ і показано правильність роботи генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ та самих помножувачів. Наведено шляхи можливого покращення характеристик помножувачів шляхом конвеєризації їх структури.

Четвертий розділ присвячено дослідженню створених в ході виконання роботи операційних вузлів (помножувачів) для полів Галуа, які застосовуються у криптографічних засобах захисту інформації на базі ЕК. Дослідження проводилися під час впровадження результатів дисертаційної роботи.

Наукові положення та висновки, сформульовані в дисертації, її результати використано під час реалізації проектних завдань у міжнародній компанії "JETSOFTPRO" (Україна, Польща, США) та в українській компанії ТзОВ “Кіберенергія” (Львів, Україна), що підтверджено відповідними актами впровадження. Також ці результати використовувалися у науково-дослідній роботі на кафедрі електронних обчислювальних машин Інституту комп'ютерних технологій, авто-

матики та метрології Національного університету "Львівська політехніка" ДБ/КІБЕР (номер державної реєстрації 0115U000446), що підтверджено відповідним актом впровадження. Також результати дисертації використовувались під час підготовки та викладання навчальних курсів з дисципліни "Дослідження і проектування комп'ютерних систем та мереж" на освітньо-кваліфікаційному рівні "Магістр" для спеціальності 123 "Комп'ютерна інженерія" для спеціалізацій "Комп'ютерні системи та мережі", "Кіберфізичні системи" та "Системне програмування", що підтверджено відповідним актом впровадження.

За відношенням k часових витрат на ПЛІС *Virtex UltraScale+ XCVU9P* при програмній та апаратній реалізаціях множення елементів полів $GF(p^n)$ у порівнянні з полями $GF(2^m)$ найбільше відношення k часових витрат для структури МКГ ЧС мають поля з характеристикою 3 (в 1,8 разів більше), з структурою ФВ – поля з характеристикою 3 (в 1,27 разів більше), з структурою ЛВ – поля з характеристикою 3 (в 1,36 разів більше). Тобто, найкращим з огляду на криптографічну стійкість є поле $GF(3^n)$, яке в найгіршому випадку забезпечує в 1,27 разів більший внесок в криптографічну стійкість засобів КЗІ ніж поле $GF(2^m)$.

Ключові слова: Модифікована комірка Гілда (МКГ), розширені поля Галуа $GF(p^n)$, булеві функції, генератор помножувачів елементів розширених полів Галуа, кіберфізичні системи (КФС), спеціалізовані процесори (СП).

ABSTRACT

Zholubak Ivan Mykhailovych. Methods and tools for creating reconfigurable nodes for cryptographic information protection in cyber-physical systems. – Qualification scientific work in manuscript form.

Dissertation for the degree of candidate of technical sciences in the specialty 05.13.05 "Computer systems and components". – Lviv Polytechnic National University, Lviv, 2024.

In the dissertation, an important scientific and technical task is addressed: the creation of reconfigurable nodes for cryptographic information protection (CIP) that operate in cyber-physical systems (CPS) using elements of extended Galois fields $GF(p^n)$. Compared to extended binary fields $GF(2^m)$, these have greater cryptographic strength. Here, $p^n \approx 2^m$, where $p > 2$ is a prime number, the configurable characteristic of the field, and *n*, *m* are the orders of the field-generating polynomial, with $m \leq 1024$. The main node for analysis is a multiplier for elements of such extended Galois fields $GF(p^n)$, built based on a reconfigurable node – a modified Guild cell (MGC). Three versions of the MGC structure are proposed, and their features compared to the classical Guild cell (GC) are shown. A comparison of the hardware costs of different MGC variants and multipliers is provided.

In the **Introduction**, an analysis of modern scientific research in the field of developing reconfigurable components for cryptographic information protection (CIP) based on extended Galois fields, including methods using elliptic curves (EC), is conducted. The importance of using multiplier core generators for elements of extended Galois fields $GF(p^n)$, which are used in CIP systems based on EC, is substantiated. The research goals and objectives are formulated, and the main scientific achievements are analyzed along with their practical value. The relationship of this work with existing scientific programs, plans, and research topics is also highlighted. Information about the process of validating the results, publications, and their practical use is provided.

The work outlines the structure and architecture of CPS, which includes CIP nodes. ECs are widely used in information protection nodes. In Ukraine, the DSTU

4145-2002 standard regulates the use of digital signatures based on EC. It limits the use of extended Galois fields to binary fields $GF(2^m)$ with the order of the field-generating polynomial $m \leq 509$ ($GF(2^{509})$), whereas international standards recommend using binary fields with much higher orders of generating polynomials ($m \leq 998$). The modern pace of development in computer technology and the threat of quantum computers lead to the creation of more robust CIP tools, increasing the order (p^n)) of extended Galois fields $GF(p^n)$ used. One of the most obvious methods of breaking CIP tools is the brute-force key search method. Software execution of operations on elements of extended Galois fields $GF(p^n)$ is more labor-intensive compared to $GF(2^m)$ and provides greater resistance to breaking. The hardware implementation of reconfigurable CIP nodes provides even greater cryptographic strength for CIP tools. Field Programmable Gate Arrays (FPGAs) were chosen as the element base for creating CIP nodes within CPS, as they provide high performance and speed in executing specialized tasks compared to software implementation. For generating VHDL descriptions of CIP nodes that work with extended Galois fields $GF(p^n)$ for their subsequent implementation in FPGAs, C++ programs were developed to generate multiplier cores for elements of extended Galois fields.

In the **First chapter**, the current state and prospects for the development of tools and methods for creating reconfigurable nodes for CIP are considered. The role of extended Galois fields $GF(p^n)$ in CIP algorithms for CPS is shown, and methods for creating reconfigurable nodes on FPGA are examined. Particular attention is paid to the rules for performing arithmetic operations in extended Galois fields $GF(p^n)$. It is shown that multiplication and division operations are the most labor-intensive, with division most often performed in software. Therefore, the operation of multiplication receives the most time and attention in this work.

The complexity of algorithms and the hardware-software model of algorithms, the issue of breaking CIP systems, the peculiarities of testing operational elements for extended Galois fields $GF(p^n)$, the structure of the Guild cell, and approaches to creating core generators are also considered.

In the **Second chapter**, methods for creating reconfigurable CIP nodes for CPS, the general methodology for conducting dissertation research, the requirements for creating reconfigurable CIP nodes, the feasibility of creating parallel multipliers for elements of extended Galois fields $GF(p^n)$ are described, and methods for creating such multipliers are proposed. Methods for evaluating temporal and hardware complexities have been improved, and a method for testing core generators of such multipliers has been proposed. The theoretical calculations of hardware complexity in the implementation of multipliers according to three MGC structures are also provided:

1. The MGC is an integral element ("black box") and each output of this cell forms its Boolean function, which is minimized using the Quine–McCluskey–Petrick method (BB).

2. The MGC is created based on functional units: a multiplier and an adder. Boolean functions that describe the operation of the multiplier and adder are minimized using the Quine–McCluskey–Petrick method (FU).

3. The MGC is created based on combinational logic elements. The MGC is implemented as a combinational parallel matrix multiplier modulo the characteristic of the field and an adder modulo the characteristic of the field (LE).

The following methods are proposed and refined:

1. The method for evaluating the temporal complexity of multiplication of elements of extended Galois fields $GF(p^n)$ by hardware means, where the multiplier consists of MGC, which made it possible to determine the field $GF(3^n)$, in which the ratio of multiplication times by software and hardware means exceeds that in other fields by at least 1.27 times, thus providing the highest cryptographic strength of CIP tools under other identical conditions.

2. The method for evaluating the hardware complexity of multipliers for elements of non-binary extended Galois fields $GF(p^n)$, where $p > 2$, which, unlike known methods, considers multipliers for fields with approximately the same order p^n and consisting of MGC, which made it possible to determine the field $GF(3^n)$, where multipliers have at least 6% lower hardware complexity compared to multipliers for other non-binary fields.

3. A method for creating FPGA generators of models (cores) of reconfigurable parallel multipliers for elements of extended Galois fields $GF(p^n)$ for CIP nodes in CPS is proposed for the first time, generating 3 variants of multipliers with 3 MGC structures, which made it possible to create descriptions of multiplier models, determine the real hardware complexity of each multiplier, and choose the best implementation for each specific field.

4. A method for testing core generators of multipliers for elements of extended Galois fields $GF(p^n)$ is proposed for the first time, using 2 different standards (2 different mathematical packages) to verify the generated multipliers based on MGC, which made it possible to confirm the correct operation of the multipliers and the generators themselves for all variants of their implementation.

The theoretical foundations for calculating hardware costs for the implementation of multipliers for elements of extended Galois fields $GF(p^n)$ are considered.

A comparison of the temporal complexities of elements of extended Galois fields $GF(p^n)$ in the mathematical package Maple is conducted. According to the comparison results, multipliers for the fields $GF(3^n)$ have the highest temporal complexity.

An analysis of hardware costs for multiplying elements of extended Galois fields $GF(p^n)$ by hardware means is conducted. The comparison is made according to three MGC structures.

For the MGC BB structure, multipliers for the field $GF(3^n)$ have better (lower) hardware complexity indicators than for the field $GF(2^m)$, for the MGC FU structure, multipliers for the fields $GF(3^n)$, $GF(5^n)$, and $GF(7^n)$ have better (lower) hardware complexity indicators than the field $GF(2^m)$. For the MGC LE structure, the relative hardware complexity coefficient of the multipliers tends towards a constant value of $4/7$, indicating the superiority of multipliers for non-binary fields $GF(p^n)$ over multipliers for the field $GF(2^m)$.

In the **Third chapter**, the process of developing tools for creating (core generators) multipliers for elements of extended Galois fields $GF(p^n)$ for CIP nodes in CPS

is described. The generators were implemented in C++. The generators create VHDL-descriptions (cores) of multipliers according to the method proposed in the work for creating FPGA generators of models (cores) of reconfigurable parallel multipliers for elements of extended Galois fields $GF(p^n)$ for CIP nodes in CPS.

Minimization of Boolean functions is carried out using the Quine–McCluskey–Petrick method. Generating multipliers for elements of extended Galois fields $GF(p^n)$ with large field characteristics p and orders p^n is a very time-consuming task. For example, generating a multiplier for the field $GF(53^{174})$ takes 1845 seconds. The work also determines the generation time for multipliers for other fields.

The process and results of testing the generated multipliers using the method proposed in the work are also described. The operation of multipliers for the fields $GF(3^{30})$, $GF(7^{17})$, $GF(23^{10})$, $GF(53^{8})$ was tested, and the correctness of the core generators for multipliers of elements of extended Galois fields $GF(p^n)$ and the multipliers themselves was shown. Ways to improve the characteristics of multipliers by pipelining their structure are suggested.

The **Fourth chapter** is devoted to the study of the operational nodes (multipliers) for Galois fields created during the execution of the work, which are used in cryptographic information protection tools based on EC. The studies were conducted during the implementation of the dissertation results.

The scientific principles and conclusions formulated in the dissertation, and its results, were used in the implementation of project tasks in the international company "JETSOFTPRO" (Ukraine, Poland, USA) and the Ukrainian company LLC "Cyberenerhiya" (Lviv, Ukraine), which is confirmed by relevant acts of implementation. These results were also used in the research work at the Department of Electronic Computing Machines of the Institute of Computer Technologies, Automation, and Metrology of the National University "Lviv Polytechnic" DB/CYBER (state registration number 0115U000446), confirmed by the corresponding act of implementation. Additionally, the dissertation results were used in the preparation and teaching of the course "Research and Design of Computer Systems and Networks" at the master's educational and qualification level for the specialty 123 "Computer Engineering" for specializa-

tions "Computer Systems and Networks", "Cyber-Physical Systems", and "System Programming", confirmed by the corresponding act of implementation.

Regarding the ratio k of time costs on the FPGA Virtex UltraScale+ XCVU9P for software and hardware implementations of element multiplications for fields $GF(p^n)$ compared to fields $GF(2^m)$, the highest ratio k of time costs for the MGC BB structure is for fields with characteristic 3 (1.8 times more), for the FU structure – fields with characteristic 3 (1.27 times more), for the LE structure – fields with characteristic 3 (1.36 times more). Thus, the best field in terms of cryptographic strength is $GF(3^n)$, which in the worst case provides 1.27 times greater contribution to the cryptographic strength of CIP tools than the field $GF(2^m)$.

Key words: Modified Guild Cell (MGC), extended Galois fields $GF(p^n)$, Boolean functions, generator of multipliers for elements of extended Galois fields, cyber-physical systems (CPS), specialized processors (SP).

СПИСОК ПУБЛІКАЦІЙ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Монографії

1. В.С. Глухов, І.М. Жолубак, Мохаммед Кадім Рахма Рахма. Принципи побудови та проектування операційних вузлів для полів Галуа, що використовуються в задачах криптографічного захисту інформації на основі еліптичних кривих. Кіберфізичні системи: багаторівнева організація та проектування [Текст]: монографія – А.О. Мельник та інші. За редакцією професора А. О. Мельника. Львів: «Магнолія 2006», 2019. 238 с. С. 58- 131.

Публікації в журналах, що входять до наукометричних баз даних Scopus

1. Elias, R., Hlukhov, V., Rahma, M., Zholubak, I. Hardware Components for Post-Quantum Elliptic Curves Cryptography // Advanced Computer Information Technologies (ACIT 2018), Ceske Budejovice, Czech Republic, June 1–3, 2018. P. 236–239. (Scopus).

2. Zholubak, I., Rahma, M. K., Hlukhov, V. Automation System for Configuration of Cryptographic Data Protection Unit Model // Proceedings of 4th International Workshop on Theory of Reliability and Markov Modeling for Information Technologies (WS TheRMIT 2018), in frameworks of the 14th International Conference on ICT in Education, Research, and Industrial Applications (ICTERI 2018), Kyiv, Ukraine, May 14–17, 2018. С. 700–707. (Scopus).

3. Zholubak, I.M., Hlukhov, V.S. Galua Field Multipliers Core Generator. International Journal of Computer Network and Information Security, 2023. – Vol. 3. – Pp. 1–14. DOI: 10.5815/ijcnis.2023.03.01, (Scopus).

Публікації в матеріалах конференцій, що входять до наукометричних баз даних Scopus

1. Rahma, M., Zholubak, I., Hlukhov, V. Devices for multiplicative inverse calculation in binary Galois fields // The 9th IEEE International Conference on Dependable Systems, Services and Technologies (DESSERT’2018), Kyiv, Ukraine, 24–27 May 2018. P. 275–278. (Scopus).

2. Zholubak, I.M., Hlukhov, V.S. Comparison of hardware complexity of

multipliers GF(p^m). In Proceedings of the 12th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2023. – Pp. 812–816. Dortmund, Germany. DOI: 10.1109/IDAACS-SWS50031.2020.9297059, (Scopus).

3. Zholubak, I.M., Hlukhov, V.S. Verification of Synthesized by the IP-core Generator Multipliers of Extended Galois Fields GF(p^n) Elements. In Proceedings of the 13th International Conference Dependable Systems, Services and Technologies (DESSERT), 2023. – Athens, Greece, (Scopus).

4. Zholubak, I.M., Hlukhov, V.S. Validation of Multipliers for Elements of Extended Galois Fields GF(p^n) and Multipliers IP-core Generator. In Proceedings of the 18th IEEE International Conference on Computer Science and Information Technologies (CSIT), 2023. – Lviv, Ukraine, (Scopus).

Статті у журналах, що включені до переліку наукових фахових видань України

1. В.С. Глухов, І.М. Жолубак, А.Т. Костик. Особливості опрацювання елементів трійкових полів Галуа на сучасній елементній базі. Вісник Національного університету “Львівська політехніка” “Комп’ютерні системи та мережі”, № 830. Львів, 2015. С. 33 – 39.

2. Жолубак І. М., Глухов В.С. Визначення розширеного поля Галуа GF(d^m) з найменшою апаратною складністю помножувача. Вісник Національного університету «Львівська політехніка» “Інформаційні системи та мережі”, № 854. Львів, 2016. С. 63 – 69.

3. Жолубак І. М., Глухов В. С. Апаратні витрати помножувачів полів Галуа GF(d^m) з великою основою. Вісник Національного університету «Львівська політехніка» “Комп’ютерні науки та інформаційні технології”, № 864. Львів, 2017. С. 77 – 82.

4. Жолубак І. М., Глухов В. С. Реалізація у ПЛІС помножувачів елементів полів Галуа високих порядків. Вісник Національного університету «Львівська політехніка» “Комп’ютерні системи та мережі”, № 881. Львів, 2017. С. 41 – 47.

5. Hlukhov, V., Kostyk, A., Zholubak, I., Rahma, M. Galois Fields Elements

Processing Units for Cryptographic Data Protection in Cyber-Physical Systems // Advances in Cyber-Physical Systems. 2017. V. 2. № 2. P. 47–53.

6. Родріг Еліас, Валерій Глухов, Мохаммед Рахма, Іван Жолубак. Ємнісна складність та вбудований контроль пристроїв для опрацювання елементів розширених полів Галуа. Електротехнічні та комп`ютерні системи. – Одеса : – 2018. Вид-во Наука і техніка. 29(105), с. 95 – 102.

7. Р.М. Еліас, В.С. Глухов, М. Рахма, І.М. Жолубак. Вбудований контроль пристроїв для опрацювання елементів розширених полів Галуа. Вісник Національного університету «Львівська політехніка» “Комп’ютерні системи та мережі”, № 905. Львів, 2018. С. 64 – 72.

8. Жолубак І. М., Курман П. В. Система безконтактних платежів на основі технології NFC. Науковий журнал «Комп’ютерні системи та мережі», № 1. Львів, 2022. С. 28 – 37, DOI: https://doi.org/10.23939/csn2022.01.028

9. Жолубак І. М., Матвієць В. Ю. Трекер для сонячних електростанцій. Науковий журнал «Комп’ютерні системи та мережі», № 1. Львів, 2022. С. 37 – 46, DOI: https://doi.org/10.23939/csn2022.01.037

10. Bohdan Marii, Ivan Zholubak. Features of Development and Analysis of REST Systems. Advances in Cyber-Physical Systems. Volume 7. Number 2. Lviv Polytechnic National University. 2022. pp. 121 – 129, DOI: https://doi.org/10.23939/acps2022.02.121

11. Жолубак І. М., Аналіз алгоритмів множення в полях Галуа для криптографічного захисту інформації. Вісник Національного університету «Львівська політехніка» “Інформаційні системи та мережі”, № 13. Львів, 2023. С. 338 – 349, DOI: https://doi.org/10.23939/sisn2023.13.338

Публікації у матеріалах конференцій, тезах доповідей та виданнях, що не включені до переліку наукових фахових видань України

1. Kostyk, A., Zholubak, I. Features of multiplication execution of operations in binary and ternary Galois fields // 5th International Youth Science Forum LITTERIS ET ARTIBUS 2015, Lviv, Ukraine, November 26–28, 2015.

2. В.С. Глухов, І.М. Жолубак. Порівняння апаратних витрат помножувачів

елементів розширених полів Галуа. 17-а міжнародно науково-практична конференція «Сучасні інформаційні та електронні технології» Одеса, Україна, 23—27 травня 2016 р. С. 133 – 134.

3. І.М. Жолубак, В.С. Глухов. Визначення розширеного поля Галуа $GF(d^m)$ з найменшою апаратною складністю помножувача. Інформаційні технології та комп'ютерне моделювання: матеріали статей Міжнародної науково-практичної конференції, 23 – 28 травня 2016 року. - Івано-Франківськ. 2016. С. 80 - 81.

4. Глухов В.С., Жолубак І.М. Дослідження апаратної складності помножувачів елементів розширених полів Галуа $GF(d^m)$. Другий науковий семінар Кібер-фізичні системи: досягнення та виклики, Львів, Національний університет «Львівська політехніка», 21-22 червня 2016 р. Матеріали Другого наукового семінару, с. 98 – 105.

5. Kostyk, A., Zholubak, I. The research of the binary codes program complication and application in cyber-physical systems // 6th International Youth Science Forum LITTERIS ET ARTIBUS 2016, Computer Science & Engineering (CSE-2016), Lviv, Ukraine, November 24–26, 2016.

6. Zholubak, I., Hlukhov, V. Research Hardware Complexity of Multipliers of Extended Galois Field GF(dm) // 6th International Youth Science Forum LITTERIS ET ARTIBUS 2016, Computer Science & Engineering (CSE-2016), Lviv, Ukraine, November 24–26, 2016.

7. Zholubak, I., Hlukhov, V. Hardware complexity of multipliers of extended Galois field in FPGA // 7th International Youth Science Forum LITTERIS ET ARTIBUS 2017, Computer Science & Engineering (CSE-2017), Lviv, Ukraine, November 23–25, 2017. P. 420–421.

8. Zholubak, I., Rahma, M., Hlukhov, V. Automation system program models configuration of cryptography cells in cyber-physical systems // 14th International Conference on ICT in Education, Research, and Industrial Applications (ICTERI 2018), Kyiv, Ukraine, May 14–17, 2018. P. 669–679.

9. Rodrigue Elias, Valerii Hlukhov, Mohammed Rahma, Ivan Zholubak. FPGA Cores for Fast Multiplicative Inverse Calculation in Galois Fields. Міжнародна нау-

ково-практична конференція «Електротехнічні та комп’ютерні системи: Теорія та практика» ЕЛТЕКС – 2018. м. Одеса, 29 травня - 1 червня 2018. Електротехнічні та комп`ютерні системи. – Одеса : – 2018. Вид-во Наука і техніка. 27(103), с. 227-233.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ЕЦП	електронний цифровий підпис
ЕК	еліптична крива
АЛП	арифметико-логічний пристрій
КЗ	комп’ютерний засіб
КС	комп’ютерна система
СКС	спеціалізовані комп’ютерні системи
НВІС	надвелика інтегральна схема
ПЛІС	програмована логічна інтегральна схема
СП	спеціалізований процесор, спецпроцесор
СК	спеціалізований комп’ютер
ЦП	центральний процесор
$GF(p^n)$	Просте поле Галуа *(Galois Field* – поле Галуа) p – просте число, характеристика і порядок простого поля Галуа
$GF(p^n)$	Розширене поле Галуа (*Galois Field* – поле Галуа) p – просте число, характеристика поля n – натуральне число, порядок полінома, який утворює розширене поле p^n – порядок розширеного поля Галуа
КГ	комірка Гілда
МКГ	модифікована комфрка Гілда
КФС	кіберфізичні системи
ЧС	“чорна скринька”
ФВ	функціональні вузли
ЛВ	логічні вузли
КЗІ	криптографічний захист інформації
ПК	персональний комп’ютер
ІКТ	інформаційно-комунікаційні технології

ВСТУП

Математичною основою багатьох алгоритмів криптографічного захисту інформації (КЗІ) є операції над полями, які мають скінченну кількість елементів – скінченні поля або поля Галуа. Арифметика полів Галуа використовується в таких галузях, як шифрування та дешифрування, виявлення та виправлення помилок, а також стиснення даних.

Актуальність роботи. У дисертації розв’язується важливе науково-технічне завдання створення реконфігурованих вузлів криптографічного захисту інформації (КЗІ), які оперують у кіберфізичних системах (КФС) елементами розширених полів Галуа $GF(p^n)$, та у порівнянні з розширеними двійковими полями $GF(2^m)$ мають більшу криптографічну стійкість. При цьому $p^n \approx 2^m$, де $p > 2$ – просте число, конфігурована характеристика поля, n, m – порядок утворюючого поле полінома, $m \leq 1024$. Основним вузлом для аналізу обрано помножувач елементів таких розширених полів Галуа $GF(p^n)$, який побудовано на основі реконфігурованого вузла - модифікованої комірки Гілда (МКГ). Запропоновано 3 варіанти структури МКГ та показано їх особливості в порівнянні із класичною коміркою Гілда (КГ). Наведено порівняння апаратних витрат різних варіантів МКГ та помножувачів.

Існує багато прикладів кіберфізичних систем, де захист інформації та безпека грають критичну роль. Наведемо деякі з них:

1. Промислові контрольні системи. Системи управління виробництвом та автоматизації в промисловості, такі як системи *SCADA (Supervisory Control and Data Acquisition)*, які контролюють та регулюють фізичні процеси у виробництві.

2. Електроенергетичні системи. Сучасні електроенергетичні системи, які використовують сучасні технології для моніторингу та управління розподіленими джерелами енергії, вимагають високого рівня кіберзахисту.

3. Медичні кіберфізичні системи. Сучасні медичні пристрої та системи збору та обробки медичних даних, які пов'язані з *IoT* та хмарними технологіями,

потребують високого рівня безпеки для захисту конфіденційності та цілісності медичної інформації.

4. Транспортні системи. Сучасні системи управління транспортом, такі як системи моніторингу руху, автоматичні транспортні системи, системи безпеки транспорту, які використовують камери та сенсори, щоб забезпечити безпеку.

5. Системи *smart grid*. Управління енергоефективністю, освітленням, водоспоживанням і відходами в сучасних інтелектуальних містах вимагає захисту від кібератак та несанкціонованого доступу.

6. Фінансові та банківські системи. Системи фінансового обліку та електронні банківські платформи, які забезпечують обробку фінансових транзакцій та зберігання конфіденційної інформації.

7. Автономні автомобілі. Розробка кіберфізичних систем для автономних автомобілів вимагає високого рівня безпеки, щоб убезпечитись від кіберзагроз та забезпечити безпеку пасажирів.

Однією з складових КЗІ є електронний цифровий підпис (ЕЦП), який використовується для забезпечення автентичності документів, повідомлень або транзакцій.

Алгоритми КЗІ, наприклад, алгоритми опрацювання електронних цифрових підписів (ЕЦП) з використанням еліптичних кривих (ЕК), мають багаторівневу структуру, в якій різні рівні виконують специфічні математичні операції над багаторозрядними кодами. Це включає в себе операції над елементами розширених полів Галуа $GF(p^n)$, де $p \geq 2$, операції над точками ЕК та операції над результатами обробки точок ЕК. У цьому контексті для різних засобів КЗІ необхідно забезпечити можливість конфігурації операційних пристроїв, які реалізують вказані алгоритми. Важливим завданням є забезпечення зміни параметрів еліптичної кривої, характеристики p та порядку утворюючого поле полінома n для поля Галуа та структури самого пристрою.

Пристрої, які опрацьовують елементи полів Галуа також є важливими будівельними блоками багатьох інших засобів КЗІ. Традиційно, розробники апаратних засобів КЗІ намагалися скористатися простотою реалізації пристроїв, для

двійкових розширених полів Галуа $GF(2^m)$, щоб зменшити апаратні витрати та підвищити продуктивність. В останні роки для збільшення криптографічної стійкості був відновлений інтерес до впровадження КЗІ на основі розширених полів Галуа $GF(p^n)$ з характеристикою p, відмінною від 2, які використовуються в таких застосунках як електронні підписи з використанням ізогеній ЕК та шифрування/розшифрування коротких повідомлень. Тому науково-технічне завдання створення реконфігурованих вузлів КЗІ, які оперують у КФС елементами розширених полів Галуа $GF(p^n)$ і забезпечують збільшення криптографічної стійкості є надзвичайно важливою та актуальною задачею.

Існує кілька причин чому цей напрямок став привабливим. По-перше, вони забезпечують більшу різноманітність на час реалізації і це спрямовано на підвищення безпеки. Наприклад, певні атаки, які були успішно проведені з еліптичними кривими, визначеними над бінарними полями $GF(2^n)$, не можуть переноситись до еліптичних кривих, визначених над $GF(p^n)$, де p – просте число, відмінне від 2 [1]. Таким чином, розглядаючи альтернативні варіанти реалізації, ми захищаємось від можливих майбутніх атак. По-друге, і, можливо, важливіше, при використанні розширених полів з характеристикою $p > 2$ з'являються додаткові переваги, такі як коротші розміри підписів, які просто не можуть бути досягнуті з полями з характеристикою $p = 2$. По-третє, це дає можливість використовувати конфігуровані або реконфігуровані криптопроцесори з реалізацією операцій у полях Галуа з різними характеристиками p поля.

На сучасному етапі, коли цифрові підписи вже активно використовуються у вбудованих комп'ютерних системах та КФС, виникає важлива потреба в їх швидкій обробці у реальному часі. Ця ситуація зумовлює необхідність використання швидкодіючих апаратних рішень, таких як спеціалізовані криптографічні процесори (криптопроцесори).

Проектування спецпроцесора (СП), який виконує операції над точками ЕК, вимагає використання таких спеціальних розділів математики як поля Галуа, еліптичні криві тощо. Елементи полів Галуа та точки на ЕК подаються у вигляді багаторозрядних двійкових кодів, які можуть складатися із сотень або тисяч бі-

тів, що вимагає розробки оригінальних засобів для виконання операцій над елементами полів Галуа та над точками на ЕК.

На практиці вже відомі та широко використовуються рішення для захисту від несанкціонованого використання та пошкодження інформації. Але сучасні методи обробки ЕЦП на основі полів Галуа з характиристикою $p > 2$ почали впроваджуватися недавно і на сьогоднішній день методи апаратного рішення цієї задачі опрацьовано недостатньо.

В Україні вже діє стандарт, який регламентує використання простих не двійкових полів Галуа – з 2020 року запроваджено новий Національний стандарт ДСТУ 9041:2020 [2]. Новий алгоритм використовує криптографічні перетворення у групі точок еліптичних кривих, використовуючи замість кривих у формі Вейєрштрасса – криві у формі Едвардса. Це дає переваги у швидкодії більш ніж у 3 рази.

Також діють інші стандарти для КЗІ – ДСТУ ISO/IEC 15946-5:2019 (ISO/IEC 15946-5:2017) [3]. Також стандартизовані процедури шифрування ДСТУ 7624:2014 [4] і гешування ДСТУ 7564:2014 [5].

Вищесказане визначає актуальність створення методів і засобів проектування КЗІ на основі ЕК та розширених полів Галуа $GF(p^n)$. І у роботі пропонуються рішення цієї задачі, які стосуються саме розширених полів Галуа $GF(p^n)$.

Зв'язок дисертаційної роботи з науковими програмами, планами і темами. Дисертаційна робота відповідає науковим напрямкам кафедри електронних обчислювальних машин Національного університету "Львівська політехніка" – “Питання теорії, проектування та реалізації комп'ютерних систем та мереж, а також комп'ютерних засобів, вузлів, приладів і пристроїв вимірювальних, інформаційних, керуючих, телекомунікаційних та кіберфізичних систем”. Робота була виконана в рамках держбюджетної науково-дослідної роботи ДБ/КІБЕР під назвою "Інтеграція методів і засобів вимірювання, автоматизації, опрацювання та захисту інформації в базисі кіберфізичних систем" (номер державної реєстрації 0115U000446).

Мета і завдання дослідження. Метою дисертаційної роботи є підвищення

криптографічної стійкості засобів КЗІ, які використовуються в складі КФС, шляхом розвитку методів та засобів створення реконфігурованих операційних пристроїв (а саме, апаратних паралельних помножувачів на основі МКГ) для роботи з елементами розширених полів Галуа $GF(p^n)$ з характеристиками p та з порядками n утворюючого поле полінома такими, що $p^n \approx 2^m$, де $p > 2$, $m \leq 1024$.

Для досягнення поставленої мети слід вирішити наступні задачі:

1. Вдосконалити методи оцінювання часової складності апаратного множення елементів розширених полів Галуа $GF(p^n)$.

2. Вдосконалити методи оцінювання апаратної складності апаратного множення елементів розширених полів Галуа $GF(p^n)$, де $p > 2$.

3. Розробити метод створення для ПЛІС генераторів моделей (ядер) реконфігурованих паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ для вузлів КЗІ у КФС за трьома варіантами структури МКГ: "чорна скринька" (ЧС), на основі функціональних вузлів (ФВ), на основі логічних вузлів (ЛВ), що використовуються в вузлах КЗІ.

4. Розробити метод тестування генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$.

5. Розробити генератори ядер (генератори моделей) помножувачів елементів розширених полів Галуа $GF(p^n)$ такими, що порядок поля $p^n < 2^{1024}$. Генератори повинні створювати моделі помножувачів на основі МКГ з трьома варіантами структури МКГ: ЧС, ФВ, ЛВ. За допомогою генераторів ядер згенерувати ряд помножувачів елементів розширених полів Галуа $GF(p^n)$ з трьома структурами МКГ та провести їх імплементацію на ПЛІС, порівняти результати реалізації. Показати збіжність теоретичних та практичних результатів оцінювання апаратної та часової складностей помножувачів. Показати досягнення поставленої у роботі мети.

6. На основі розроблених і вдосконалених методів провести аналіз апаратних складностей створених помножувачів елементів розширених полів Галуа $GF(p^n)$, $p > 2$, та визначити поля, які найкраще підходять для реалізації помно-

жувачів. Також провести аналіз часових складностей помножувачів елементів розширених полів Галуа $GF(p^n)$, $p > 2$, за відношенням часових витрат при програмній та апаратній реалізаціях.

Об'єкт дослідження – процеси та засоби створення реконфігурованих вузлів КЗІ у КФС.

Предмет дослідження – методи та засоби створення апаратних помножувачів для елементів розширених полів Галуа $GF(p^n)$.

Методи дослідження. При проектуванні апаратних помножувачів для елементів розширених полів Галуа $GF(p^n)$ враховувалися висновки теорії комп'ютерних систем, теорії обчислювальних систем, теорії обчислювальних машин, теорії проектування спеціалізованих комп'ютерних систем, теорії складності алгоритмів та програмно-апаратної складності комп'ютерних систем. Для реалізації елементів вузлів апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС використовувалась теорія проектування НВІС. Для розробки методів обробки елементів розширених полів Галуа $GF(p^n)$ враховувалися положення і висновки теорії інформації, теорії чисел, теорії залишків, теорії обчислень, теорії груп, для проектування спецпроцесорів, а також для вирішення задач проектування апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ застосовувалися результати теорії кодування, для створення моделей вузлів апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ та для аналізу їх роботи була використана теорія програмування, теорія моделей, обчислювальна математика, моделювання алгоритмів та апаратних засобів.

Отримані результати були перевірені шляхом моделювання згідно з теорією випробувань.

Дослідження, що були проведені, базуються на результатах теорії цифрових автоматів, на теоретичній моделі взаємодії відкритих систем та багаторівневій платформі КФС. Також були використані методи виконання математичних операцій у розширених полях Галуа $GF(p^n)$ у поліноміальному базисі. У проведених дослідженнях використовуються математичні напрацювання теорії чисел,

теорії алгоритмів та засобів моделювання цифрових схем.

Наукова новизна одержаних результатів. На основі проведених досліджень розв'язано важливу науково-прикладну задачу створення реконфігурованих (на ПЛІС) паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ в складі засобів КЗІ КФС, які забезпечують засобам КЗІ більшу криптографічну стійкість. При цьому отримано такі нові наукові результати:

1. Отримав подальший розвиток метод оцінювання часової складності множення елементів розширених полів Галуа $GF(p^n)$ апаратним способом, за яким помножувач складається з МКГ, що дало можливість визначити поле $GF(3^n)$, у якому відношення часів множення програмним та апаратним способами перевищує таке відношення в інших полях щонайменше в 1,27 раза, чим забезпечує найбільшу криптографічну стійкість засобів КЗІ при інших однакових умовах.

2. Отримав подальший розвиток метод оцінювання апаратної складності помножувачів елементів розширених недвійкових полів Галуа $GF(p^n)$, де $p > 2$, який на відміну від відомих розглядає помножувачі для полів з приблизно однаковим порядком p^n і які складаються з МКГ, що дало можливість визначити поле $GF(3^n)$, де помножувачі мають щонайменше на 6 % меншу апаратну складність, у порівнянні з помножувачами для інших недвійкових полів.

3. Вперше запропоновано метод створення для ПЛІС генераторів моделей (ядер) реконфігурованих паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ для вузлів КЗІ у КФС, за яким на відміну від відомих генерується 3 варіанти помножувачів з 3 структурами МКГ, що дало можливість створити описи моделей помножувачів та визначити реальну апаратну складність кожного із помножувачів та обрати найкращу реалізацію для кожного конкретного поля.

4. Вперше запропоновано метод тестування генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$, який на відміну від відомих використовує 2 різні еталони (2 різні математичні пакети) для перевірки згенерова-

них помножувачів на основі МКГ, що дало можливість підтвердити правильну роботу генераторів помножувачів та самих помножувачів для всіх варіантів їх реалізації.

Практичне значення одержаних результатів. Апаратні реалізації алгоритмів КЗІ, здебільшого на основі ПЛІС, характеризуються низьким електроспоживанням, малими габарити, високою продуктивністю, високою захищеністю, що є невід'ємними атрибутами КФС.

Отримані у дисертаційній роботі наукові та практичні результати створюють методологічну базу для розробки вузлів КЗІ, а саме, апаратних помножувачів для елементів розширених полів Галуа $GF(p^n)$, які дозволяють підвищити надійність, криптографічну стійкість та захищеність сучасних апаратних засобів КЗІ, у тому числі, засобів опрацювання ЕЦП на основі ЕК.

Практична цінність даної роботи полягає у тому, що за запропонованими методами та результатами теоретичних та практичних досліджень для реконфігурованих вузлів засобів КЗІ на основі ЕК визначено поле $GF(3^n)$, де помножувачі мають щонайменше на 6 % меншу апаратну складність, а також де відношення часів множення програмним та апаратним способами перевищує таке відношення в інших полях щонайменше в 1,27 рази, чим забезпечує найбільшу криптографічну стійкість засобів КЗІ, які працюють у полі $GF(3^n)$ при інших однакових умовах, а саме:

1. Створено і апробовано технологічні засоби (генератори ядер) для генерації ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ такими, що порядок поля $p^n < 2^{1024}$ для 3 структур МКГ, що підтверджено актами впровадження.

2. Створено і перевірено низку моделей помножувачів у вигляді VHDL-описів помножувачів елементів розширених полів Галуа $GF(p^n)$ такими, що порядок поля $p^n < 2^{1024}$, що підтверджено актом впровадження.

3. За розробленим методом проведено оцінку апаратної складності помножувачів елементів розширених полів Галуа $GF(p^n)$ такими, що порядок поля

$p^n < 2^{1024}$ та з приблизно однаковим порядком p^n, що підтверджено актом впровадження.

4. За розробленим методом проведено оцінку часової складності помножувачів елементів розширених полів Галуа $GF(p^n)$ такими, що порядок поля $p^n < 2^{1024}$ та з приблизно однаковим порядком p^n, що підтверджено актами впровадження.

5. Виконано перевірку розроблених помножувачів та генераторів помножувачів елементів розширених полів Галуа $GF(p^n)$, що підтверджено актами впровадження.

6. Розроблено методичні вказівки для використання в навчальному процесі кафедри ЕОМ (Методичні вказівки до лабораторних робіт “Проектування і моделювання елементів комп’ютерних систем та мереж” з дисципліни "Дослідження і проектування комп’ютерних систем та мереж" для студентів освітньо-кваліфікаційного рівня «Магістр», спеціальність 123 «Комп’ютерна інженерія», спеціалізації «Комп’ютерні системи та мережі», «Кіберфізичні системи» та «Системне програмування» / укл.: Глухов В.С., Жолубак І.М., Костик А.Т, Рахма М.К.Р. - Львів: видавництво Національного університету “Львівська політехніка”, 2019. - 25 с.), що підтверджено актом впровадження.

Особистий внесок здобувача. Усі наукові результати, викладені у дисертаційній роботі, отримано автором особисто і повністю розкрито у публікаціях. У публікаціях, що написано в співавторстві, автору дисертації належать методи оцінювання часової та апаратної складностей помножувачів елементів розширених полів Галуа $GF(p^n)$, методи створення таких помножувачів на основі МКГ та методи їх тестування.

У публікаціях, що написано в співавторстві, автору дисертації належать основні теоретичні результати (методи і підходи до рішення поставлених задач), а саме. У роботах [6], [7] та [8] проведено дослідження, які показують, що у сучасних ПЛІС, при реалізації побудованих на основі МКГ помножувачів елементів розширених полів Галуа $GF(p^n)$ з приблизно однаковими порядками полів із збільшенням характеристики p апаратні витрати збільшуються. У роботі [9] по-

казано обчислення незвідних поліномів для полів Галуа $GF(p^n)$. *У* роботах [10] та [11] показано, що у деяких місцях, де зростає характеристика поля *p*, апаратні витрати зменшуються. У роботі [12] розглянуто особливості побудови операційних пристроїв для опрацювання елементів трійкових полів Галуа $GF(3^n)$ на сучасній елементній базі. У роботі [13] наведено порівняння апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ з великими характеристиками, з метою визначення поля, у якому помножувачі, у разі їх реалізації на сучасних ПЛІС матимуть найменшу апаратну складність. У роботі [14] зроблено порівняння апаратних витрат помножувачів елементів полів Галуа з основами 2, 3, 5 7, 13 на ПЛІС У роботі [15] показано, що двійкові коди елементів розширених полів Галуа $GF(p^n)$ є надлишковими, частина з них ніколи не з'являється при нормальній роботі пристроїв опрацювання елементів таких полів. Невикористані (заборонені) кодові комбінації можна задіяти для робочого діагностування (вбудованого контролю) цих пристроїв. У роботі [16] запропоновано спосіб побудови послідовного помножувача елементів поля Галуа $GF(3^n)$. Запропонований метод перевірки операцій над елементами полів Галуа $GF(2^m)$ і $GF(3^n)$ використовує математичний пакет *Maple*. У дослідженні [17] продемонстровано, що для покращення процесу діагностики пристроїв, які задіяні у обробці елементів у розширених полях Галуа $GF(p^n)$, бажано використовувати поля з характеристикою *p*, де *p* – це перше просте число, що перевищує *2*, такі як $p = 3$ або $p = 5$. В дослідженні [18] описано основи створення та дизайну операційних блоків для полів Галуа, використовуваних у криптографічному захисті даних на основі еліптичних кривих. У роботах [19], [20] та [21] розглянуто різновиди кіберфізичних систем. У роботі [22] проведено порівняння складностей множення елементів розширених полів Галуа $GF(p^n)$ з різними характеристиками *p* поля на ПК. У роботах [23] та [24] узагальнено теоретичні основи створення помножувачів елементів розширених полів Галуа на ПЛІС. Наведено 3 варіанти побудови МКГ та порівняння цих варіантів. Розроблено інструмент для генерації *VHDL*-описів помножувачів елементів таких полів, для подальшого використання та впрова-

дження цих описів у пристроях захисту даних на ПЛІС. Показано перевірку коректності роботи генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ та згенерованих ним помножувачів на основі часових діаграм середовища *Active-HDL* та математичного пакету *Maple* [25] та часових діаграм середовища *Active-HDL* та бібліотеки *C++ GaloisCPP* [26]. У роботі [27] показано, що програмне множення елементів трійкових полів Галуа $GF(3^n)$ має найбільший час виконання. У роботі [28] показано, що всі розглянуті методи, побудовані на використанні обчислень квадратів і квадратних коренів із використанням швидких паралельних помножувачів у спеціальному процесорі полів Галуа, забезпечують обчислення оберненого елемента в двійкових полях Галуа $GF(2^m)$ в поліноміальному базисі за час, незалежний від значення операндів. У роботі [29] розроблено автоматизовану систему налаштування *VHDL*-описів криптографічних вузлів. З її допомогою створено сімейство помножувачів елементів полів Галуа. Було проведено аналіз апаратних складностей цих помножувачів для їх реалізації на ПЛІС. Також показано, що розширені поля Галуа $GF(p^n)$ з приблизно однаковою кількістю елементів і малими характеристиками поля мають низьку апаратну складність і високу програмну складність реалізації. У роботі [30] протестовано методи ділення елементів полів Галуа $GF(p^n)$, щоб вибрати методи з найкращою апаратною та часовою складностями для поліноміального базису. У роботі [31] запропоновано програмну систему автоматичного моделювання конфігурації осередків криптографії в кіберфізичних системах. У роботі [32] показано, що використання ізогеній суперсингулярних еліптичних кривих є орієнтованою на програмну реалізацію.

Апробація результатів дисертації. Основні положення дисертації було представлено та обговорено на:

1. 5th, 6th, 7th International youth science forum “litteris et artibus” – Львів: - 2015, 2016, 2017.

2. Міжнародній науково-практичній конференції “Інформаційні технології та комп’ютерне моделювання” – Івано-Франківськ – Яремче: - 2016.

3. 17-й міжнародній науково-практичній конференції “Современные ин-

формационные и электронные технологи" – Одесса: - 2016.

4. Кіберфізичні системи досягнення та виклики – Львів: - 2016.

5. Захист інформації і безпека інформаційних систем – Львів: - 2017.

6. 12-th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS) – Dortmund, Germany: 7-9 september 2023.

7. 13-th International Conference on Dependable Systems, Services and Technologies (DESSERT'2023) - Greece, Athens: 13-15 october 2023.

8. IEEE 18-th International Conference on Computer Science and Information Technologies (CSIT), held in frames of IEEE Lviv Polytechnic Week - Lviv, UKRAINE: 19-21 October 2023.

Публікації. За темою дисертаційної роботи опубліковано 28 наукових праць, з них 1 монографія, 14 статей у фахових наукових журналах та вісниках, 13 – у працях та тезах конференцій та семінарів, з них у Scopus – 1 стаття Q1 (квартиль 1), 2 статті Q4 (квартиль 4) та 4 тези конференцій.

Структура та обсяг роботи. Дисертаційна робота складається із вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Чотири розділи присвячено розгляду змістовної сутності створення реконфігурованих вузлів КЗІ, які оперують у КФС елементами розширених полів Галуа $GF(p^n)$. Вони у порівнянні з вузлами, які оперують елементами двійкових полів Галуа $GF(2^m)$, мають більшу криптографічну стійкість. Найбільше уваги приділено побудові помножувачів елементів розширених полів Галуа $GF(p^n)$ та створенню генераторів ядер (*VHDL*-описів) цих вузлів. Роботу викладено на 299 сторінках, з них основний текст – 155 сторінки, рисунків – 76, таблиць – 16. Список використаних джерел – 252 найменувань.

РОЗДІЛ 1

СУЧАСНИЙ СТАН ТА ПЕРСПЕКТИВИ РОЗВИТКУ ЗАСОБІВ ТА МЕТОДІВ СТВОРЕННЯ РЕКОНФІГУРОВАНИХ ВУЗЛІВ КРИПТОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ

1.1 Конфігуровані та реконфігуровані системи

Криптографічний захист інформації є невід’ємною частиною КФС. Вузли або спеціалізовані комп’ютери (СК) криптографічного захисту інформації можуть бути реалізовані програмно або апаратно. У кожного із цих варіантів є свої переваги та недоліки.

Такі системи можуть бути реконфігурованими та конфігурованими [33], [34].

Конфігуровані СК – це комп'ютерні системи, які перед використанням повинні бути налаштовані для виконання певних специфічних завдань або функцій. Такі комп'ютери можуть мати спеціальне обладнання або програмне забезпечення, що дозволяє їм ефективно виконувати конкретні завдання.

Реконфігуровані СК [35], [36] – це комп'ютерні системи, які можуть бути переналаштовані для виконання нових завдань або для адаптації до нових вимог. Це можуть бути зміни в апаратній частині, програмному забезпеченні або в обидвох.

Загалом, ідея конфігурованих та реконфігурованих СК полягає в тому, щоб максимально використовувати потенціал обладнання та програмного забезпечення, для вирішення конкретних завдань або вимог у відповідних галузях. СК можуть бути реалізовані програмно або апаратно. Програмна реалізація – це реалізація алгоритму мовами програмування високого рівня на ПК. Апаратна реалізація – це схемна реалізація алгоритму, в тому числі з використанням ПЛІС.

Програмна реалізація є більш гнучкою, легко конфігурується, не потрібно ніякого спеціалізованого обладнання, проте вона характеризується невисокою швидкістю в порівнянні з апаратною реалізацією.

У даній роботі розглядається процес створення апаратно реалізованих ре-

конфігурованих вузлів КЗІ.

При програмній реалізації засобів КЗІ, в порівнянні з апаратною, є можливість досить легко взнати внутрішню реалізацію алгоритму [37]. Апаратна реалізація є більш трудомісткою та потребує спеціалізованого обладнання, проте виділяється високою швидкодією та закритою архітектурою алгоритму шифрування [38].

Закрита архітектура алгоритму [37] – це підхід до проектування програмного коду або системи, коли всі деталі внутрішньої реалізації алгоритму є прихованими від користувача або інших частин програми. Іншими словами, зовнішній інтерфейс алгоритму зроблений настільки простим і абстрактним, що користувач або інші компоненти системи не мають доступу до деталей його внутрішньої роботи.

Архітектура алгоритму [39] – це високорівневий опис структури та організації алгоритмічного рішення. Це план або дизайн, який описує, як алгоритм буде виконувати певне завдання, які етапи виконання він має, які дані обробляються та як вони передаються між різними частинами алгоритму.

Для реконфігурування [33] використовуються кристали ПЛІС, які зберігають або змінюють конфігурацію. Після конфігурування програмований логічний пристрій стає спеціалізованим обчислювальним модулем, наприклад, апаратно-орієнтованим СП.

Створення високопродуктивних реконфігурованих систем є новим науково-технічним напрямком. Цей напрямок передбачає розробку нового покоління програмно-апаратних засобів, які отримали назву "реконфігурованих прискорювачів" [40]. Реконфігуровані прискорювачі все частіше використовуються для розв'язання складних завдань обробки даних з метою зменшення навантаження на процесори загального призначення і підвищення продуктивності комп'ютерних систем. Цей напрямок дозволяє здійснювати реконфігурацію, тобто заміну функціональності реконфігурованого прискорювача новими спеціалізованими процесорами. Це відкриває можливість надання комп'ютерній системі, що включає в себе реконфігурований прискорювач, зовсім нові властивості та досягати

високих технічних характеристик. Однією з переваг реконфігурованих прискорювачів є можливість синтезу як програмованих, так і апаратно-орієнтованих спеціалізованих процесорів, а також універсальних процесорів. Реалізація апаратно-орієнтованих спеціалізованих процесорів є особливо ефективною, оскільки вона дозволяє досягти надзвичайної продуктивності при мінімальних витратах на обладнання та енергоспоживання.

Існують такі методи реконфігурування ПЛІС [33]:

1. Статичне. Комп’ютерна система конфігурується перед випуском на ринок. Конфігурація ніколи не замінюється після того, як виріб проданий.

2. Оновлення. Конфігурація замінюється час від часу для налагодження системи або функціонального оновлення.

3. Динамічне. Комп’ютерна система може змінювати конфігурацію (реконфігуруватись) після випуску на ринок. Можливий набір конфігурацій, що дає змогу перепрограмувати ПЛІС під час її роботи.

На базовому рівні ПЛІС може бути 2 типи конфігураційних елементів: дрібнозернисті та грубозернисті. Зернистість тут визначається як найменший функціональний блок, розміщений у конфігурованому елементі.

1.2 Кіберфізичні системи

Кіберфізичні системи (КФС, *Cyber-Physical System, CPS*) – це системи, що складаються з різних природних об'єктів, керуючих контролерів та штучних підсистем, що утворюють єдину систему. У КФС відбувається тісна взаємодія та координація між обчислювальними та фізичними ресурсами. Комп'ютери забезпечують моніторинг і керування фізичними процесами, використовуючи петлі зворотного зв'язку [41].

У КФС відбувається гармонійне співіснування двох типів моделей [42]. З одного боку – це традиційні інженерні моделі (механічні, будівельні, електричні, біологічні, хімічні, економічні та інші), а з іншого – моделі комп'ютерні.

Попередниками КФС слід вважати системи вбудованого реального часу, розподілені обчислювальні системи, автоматизовані системи управління техніч-

ними процесами та об'єктами, а також бездротові сенсорні мережі.

З технічної точки зору КФС мають багато спільного зі структурами типу грід, реалізованими за допомогою інтернету речей (*Internet of Things, IoT*) [43], Індустрії 4.0, [44] промислового інтернету (*Industrial Internet*) [45], міжмашинної взаємодії (*Machine-to-Machine, M2M*) [46], туманного [47] і хмарного комп'ютингу [48] (*fog і cloud computing*). Але цими технічними засобами ні в якому разі не можна обмежувати уявлення про КФС. Для цих складних систем потрібні нові кібернетичні підходи до моделювання, оскільки саме моделі є центральним моментом в науці та інженерії.

На рис. 1.1 наведено використання КЗІ у системах інтернету речей, які є різновидом КФС. З графіка видно, що не так багато систем використовують засоби КЗІ. Це підтверджує актуальність дисертаційної робот.

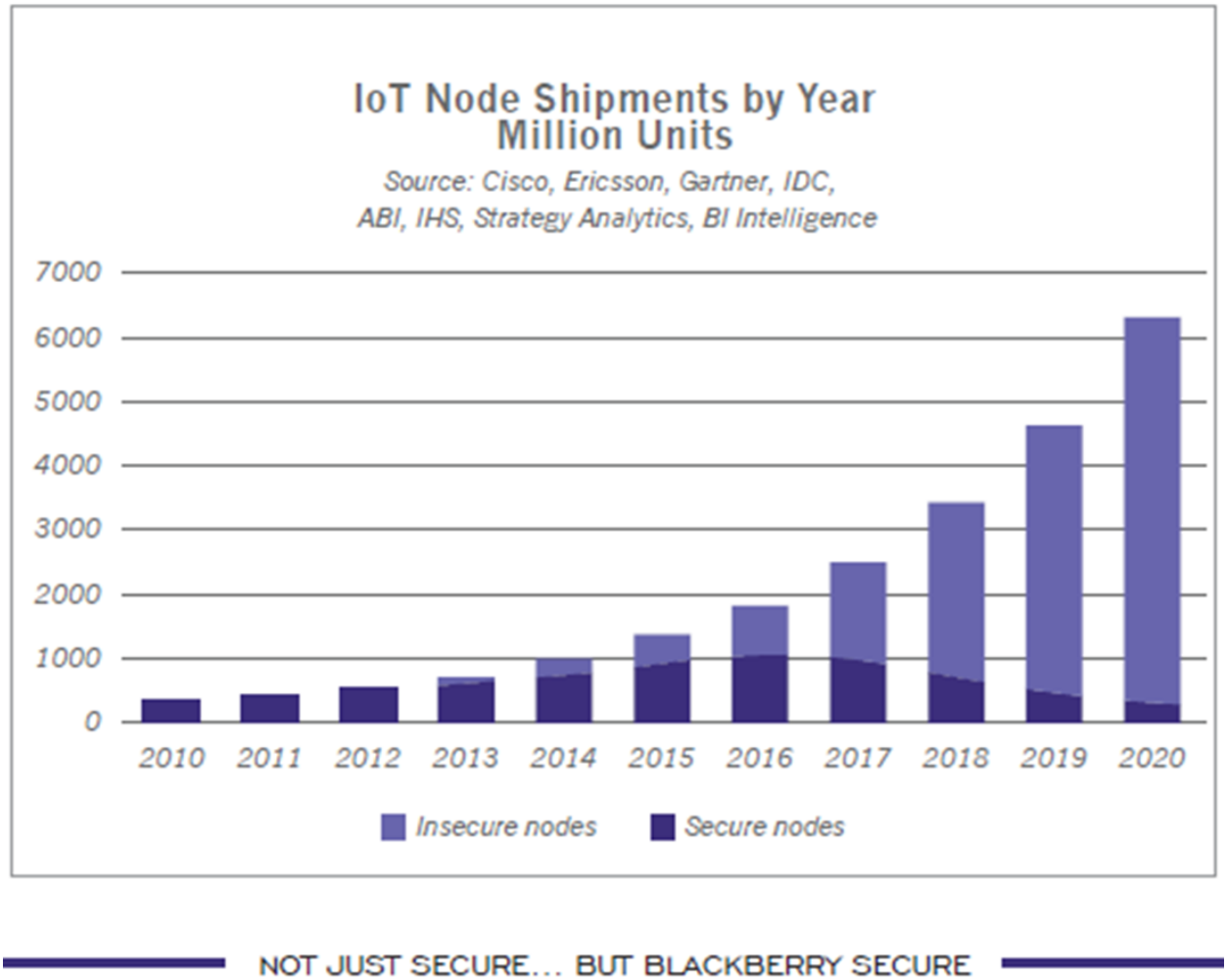


Рис. 1.1 Використання КЗІ у системах інтернету речей [49]

У [50] вказуються різні парадигми для характеристики КФС, включаючи диференціальні рівняння, стохастичні процеси та модель на основі агентів. Наводиться простий огляд різних підходів, не зосереджуючись на особливостях спеціалізованих дисциплін, таких як теорія керування чи багатоагентні системи. Метою статті [51] є надання фундаментальних концепцій кібербезпеки для системи розподілу кіберфізичної системи. Обговорюються вразливості кібервторгнень і стратегії підвищення стійкості системи. Представлено еволюцію інформаційно-комунікаційних технологій (ІКТ) для енергомережі та заходи кібербезпеки, а також ІКТ в середовищі енергосистеми. Включено огляд стандартів і протоколів зв'язку інтелектуальної мережі, а також розглянуто виявлення кібервторгнень у системи розподілу. У [52] представлено три рівні, які складають кожну КФС. Мета роботи полягає в тому, щоб сформулювати концепції, для побудови наукової теорії КФС, яка систематично характеризується фізичним рівнем, рівнем даних і рішенням, а також необхідними процесами, які визначають перехресний рівень. У роботі [53] описано 6-рівневу структуру кіберфізичної системи.

У [54] узагальнено та проаналізовано пріоритети технологічних досліджень та інновацій (*R&I*) у Європейському Союзі та Сполучених Штатах у секторах *IoT* та КФС. Робота [55] надає короткий історичний огляд тісно пов'язаних з КФС галузей, а саме теорії управління, теорії інформації та кібернетики.

Невід'ємною особливістю кіберфізичної системи є обмін даними, в тому числі і бездротовий. Виникає необхідність захисту даних. Проблема створення вузлів КЗІ є актуальною для КФС. Суттєвою особливістю КФЗ є низьке споживання, малі габарити, висока продуктивність, висока захищеність і як наслідок апаратна реалізація алгоритмів здебільшого на ПЛІС [14]. Використання ПЛІС веде до необхідності створення засобів для їх проектування, тобто генераторів ядер [56].

1.3 Методи забезпечення безпеки інформації

Методи забезпечення безпеки інформації поділяються на фізичні, програмно-технічні, управлінські, технологічні, рівень користувача, мережеві, проце-

дурні. На програмно-технічному рівні здійснюється перевірка та ідентифікація дійсності користувачів, протоколювання і аудит, управління доступом, криптографія, екранування, забезпечення високої доступності.

Криптографічні методи [57], [58] шифрування використовуються для перетворення звичайного тексту (під час передачі чи зберігання) в криптографічно захищений текст (шифротекст), який незрозумілий для незаконних користувачів без ключа. Це дозволяє забезпечити конфіденційність інформації.

Гібридне шифрування використовує як симетричне [59], [60], [61], [62], [63], так і асиметричне шифрування. Зазвичай великі обсяги даних шифруються симетричним методом, а ключ для симетричного шифрування обмінюється за допомогою асиметричного шифрування.

Велика група методів використовує еліптичні криві *(Elliptic Curve Cryptography, ECC)* [64] для створення ключів і шифрування. Дозволяє досягнути високого ступеня безпеки з меншими ресурсами, порівняно з іншими методами.

Ці методи можуть використовуватися в різних комбінаціях для забезпечення конфіденційності, цілісності та автентичності інформації під час передачі та зберігання. Вибір конкретного методу залежить від контексту використання, потреб безпеки та можливостей платформи.

Важливість захисту кіберфізичних систем (КФС). КФС повинні забезпечувати стабільну роботу при жорстких умовах експлуатації та повинні гарантувати захищеність інформації у каналах зв’язку, що стає дедалі актуальнішим через неперервне зростання кількості нападів на комп'ютерні системи, у тому числі і на КФС [65].

Типи криптографічних засобів захисту інформації. Для побудови пристроїв КЗІ використовують три види засобів [66]:

1) програмні (універсальний процесор з програмою, яка завантажується);

2) апаратні (цифровий автомат, СП);

3) програмно-апаратні (процесор з прошитою програмою, яка не змінюється).

Їх основні відмінності:

1) спосіб виконання задач;

2) надійність реалізованих методів;

3) ціна, яка є дуже важливою характеристикою в умовах сучасної жорсткої конкуренції на ринку.

Як правило дорожче – апаратне, дешеве рішення - програмне. Вища вартість апаратних пристроїв компенсується вищим ступенем захисту інформації.

Використання спеціалізованої апаратури, зокрема спецпроцесора (СП), забезпечує захист інформації при її апаратній обробці алгоритмами [67].

Переваги апаратної реалізації:

1) велика швидкодія;

2) висока захищеність від електромагнітних випромінювань;

3) висока захищеність від фізичного впливу на пристрій захисту інформації [29];

4) висока зручність використання – більш прозоре виконання операцій;

4) просте встановлення;

5) мала споживана потужність;

6) незмінність алгоритму шифрування;

7) апаратний давач випадкових чисел (ДВЧ), який застосовується при генерації ключів;

8) можливість безпосереднього завантаження ключів шифрування в СП;

9) ключі шифрування зберігаються в пам'яті СП;

10) ідентифікація і аутентифікація користувача відбувається до завантаження операційної системи;

11) можливість заборони зміни процесу завантаження комп'ютера;

12) функції перевірки цілісності операційної системи та застосунків;

13) можливість ведення доступного лише адміністратору безпеки журналу дій користувачів;

14) забезпечення більшої, у порівнянні з програмними продуктами, швидкості шифрування.

Основним недоліком апаратної реалізації є наявність програмного інтерфейсу.

Основи теорії створення апаратних вузлів на основі ядер ПЛІС заклав проф. Мельник А.О. у роботі [33].

У даній дисертаційній роботі вирішується важлива науково-прикладна задача проектування апаратних операційних пристроїв для вузлів КЗІ, для яких характерна багаторівнева структура, використання складного математичного апарату: таких розділів математики як еліптичні криві та поля Галуа.

Актуальність використання ЕЦП [68], [69], [70], [71] зростає останнім часом, оскільки їх використання є одним з методів КЗІ епохи постквантової криптографії [72].

Криптографія на основі еліптичних кривих. Криптографія на основі еліптичних кривих [73] *(Elliptic Curve Cryptography, ECC)* – це сучасний метод шифрування та підпису, який базується на властивостях математичних об'єктів, відомих як еліптичні криві. ЕЦП виявляється особливо ефективним в порівнянні з багатьма іншими методами, оскільки забезпечує високий рівень безпеки при використанні відносно коротких ключів, що робить його особливо підходящим для обмежених обчислювальних ресурсів, таких як мобільні пристрої.

Основні принципи ЕЦП:

1. Еліптичні криві [74] – це геометричні об'єкти, що визначаються рівняннями вищого порядку та мають особливі властивості. Вони використовуються для генерації ключів та виконання операцій шифрування та підпису.

2. Криптографічні операції [75]: ЕЦП використовує математичні операції на точках еліптичних кривих для забезпечення безпеки. Однією з таких операцій є додавання точок на кривій, що утворює закриту групу.

3. Усі операції над точками еліптичних кривих виконуються у полях Галуа.

Стандарти для систем захисту інформації на основі ЕК. На засоби КЗІ накладаються вимоги та обмеження такі самі, як і на КС – застосування принципів побудови відкритих систем [76], що складаються з апаратних і програмних продуктів і технологій, розроблених відповідно до загальноприйнятих та загаль-

нодоступних стандартів.

В Україні з 1 січня 2004 року діє стандарт, який регламентує використання ЕК для ЕЦП [77]. Організаційну підтримку застосуванню ЕЦП надає стандарт [78], який розкриває механізми роботи ЕЦП, у тому числі і на основі ідентифікаторів та сертифікатів. Також стандартизовані процедури шифрування ДСТУ 7624:2014 [79], алгоритм симетричного блокового перетворення ДСТУ 7624:2014 [4] і гешування ДСТУ 7564:2014 [5]. Для практичної реалізації [80], [30] корисними є рекомендації та вимоги міжнародних стандартів [81], які передбачають використання полів Галуа з характеристикою до 998. Український стандарт на ЕЦП [82] обмежується використанням полів Галуа з характеристикою не більше 509. У даній роботі розробляються засоби і методи обробки елементів розширених полів Галуа, які мають характеристику та порядок, які відповідають вимогам міжнародних стандартів, у чому і полягає актуальність даної роботи.

Теорія та практики захисту інформації на основі ЕК у КФС. В основі процедур отримання і перевірки ЕЦП відповідно до [82] лежать операції над елементами поля Галуа $GF(2^n)$ і точками ЕК. Дуже вагомий внесок у впровадження теорії полів Галуа в практику зробив Николайчук Я.М. [83], [84], [85], [86], Возна Н.Я. [84], Касянчук М.М. [86], значну увагу виконанню операцій над точками еліптичних кривих та елементами полів Галуа приділено у роботах Глухова В.С. [1], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103].

Значний внесок в теорію і практику обробки цифрових підписів зробили розробники Державних стандартів України А.І. Кочубінський, О.С. Шаталов, А. Анісімов, М. Карнаух, Г. Гулак , І. Горбенко, О. Потій, Д., Шевченко, Л. Ковальчук, Д. Балагура, Ю. Горбенко, А. Лєншин та І. Остапенко. Стандарт цифрового підпису [104] ґрунтується на еліптичних кривих та рекомендує для використання еліптичні криві в оптимальному нормальному базисі. У [82] зазначено національний стандарт України, який визначає процес створення та перевірки електронного цифрового підпису. Цей підпис ґрунтується на властивостях

груп точок еліптичних кривих над полями $GF(2^n)$. [105] описує використання кривих Едвардса для захищеної реалізації механізмів електронного цифрового підпису у відповідності до стандарту ДСТУ 4145-2002. У [106] наведено алгоритми сліпих мультипідписів, що використовують звичайні цифрові підписи, які відповідають стандартам ДСТУ 4145-2002. Міжнародний стандарт ISO/IEC 15946 [79], [107] визначає криптографічні методи з відкритим ключем, що ґрунтуються на еліптичних кривих. Цей стандарт обмежується лише розглядом криптографічних методів, що використовують властивості груп точок еліптичних кривих над скінченними полями, порядок яких є степенем простого числа (включаючи спеціальні випадки, коли порядок поля є простим числом та коли характеристика поля дорівнює два).

Теоретичним і практичним аспектам проектування криптографічних засобів присвячено роботи А.О.Мельника, Р.Б.Поповича, В.А.Мельника, І. Д. Горбенка. У роботі [108] проведено аналіз застосування технології процесорних ядер для побудови системи криптографічного захисту інформації. На основі проведеного аналізу вибрано перелік алгоритмів та структура системи криптографічного захисту інформації. Запропонований новий підхід до проектування надвеликої інтегральної схеми системи криптографічного захисту інформації на основі ядра програмованого процесора та декількох ядер криптографічних акселераторів, що характеризується гнучкістю та високою ефективністю. У роботі [109] розглянуто основні теоретичні положення сучасної криптографії та описано сучасні симетричні шифрувальні алгоритми.

У роботі [110] наводиться теорія із захисту інформації в інформаційно-телекомунікаційних системах. У роботі [111] подано обґрунтування та визначено вимоги до засобів криптографічного захисту інформації.

Математичні основи криптографії викладено у [112]. Основні питання коротко викладено у енциклопедичному виданні [113].

1.4 Складність алгоритмів та апаратно-програмна модель алгоритмів

Велике значення при проектуванні криптографічних засобів захисту має

їхня порівняльна оцінка, так само, як і оцінка складності алгоритмів, які реалізують ці засоби. Основи теорії складності з прив'язкою до КС (апаратно-програмна модель алгоритмів) заклав М.В.Черкаський [114], [115], [116], [117]. У роботах [114] та [115] запропоновано концепцію алгоритму з використанням узагальненої параметричної моделі. Дано визначення характеристик складності, в тому числі часовій, апаратній, ємнісній, програмній та структурній. У роботі [116] наводяться особливості створення моделі апаратного алгоритму та моделі апаратно-програмного універсального обчислювача на основі апаратно-програмної моделі (SH-моделі). Також описується розробка способів оптимізації характеристик складності операційних пристроїв та процесорів. У роботі [117] показано, що універсальна SH-модель відрізняється від універсальних математичних моделей алгоритмів за призначенням і способом синтезу. Розглянуті принципи побудови універсальної SH-моделі з погляду теорії складності апаратно-програмних алгоритмів. Сформульовані вимоги оптимізації до універсальної *SH*-моделі, що складається з двох *SH*-моделей – функціональної та керуючої.

За М. В. Черкаським [114] необхідно розрізняти:

1) апаратну складність;

2) програмну складність;

3) структурну складність;

4) часову складність.

Подальшим розвитком теорії є введення в обіг поняття ємнісної складності [17].

Апаратна складність – це кількість елементів деякого ієрархічного рівня представлення апаратних засобів, які утворюють алгоритмічний пристрій [114]:

$A = |E|$, де E – множина елементів схеми

Визначення відображає ієрархічну побудову комп'ютерних засобів. Якщо під алгоритмічним пристроєм розуміти операційний пристрій, то як елементи розглядаються однорозрядні комірки або вентилі. Для рівня регістрових передач елементами є операційні пристрої. На системному рівні алгоритмічним пристроєм є, наприклад, гіперкуб, елементами якого є процесорні елементи (вузлові

комп’ютери та інше).

Разом з тим, при оцінюванні апаратної складності можливий інший підхід. Апаратну складність мікропроцесорів доцільно визначати кількістю транзисторів, розташованих на кристалі.

Дану роботу сконцентровано на порівнянні вузлів КЗІ за апаратною складністю.

1.5 Криптографічний захист інформації у кіберфізичних системах

Криптографічний захист інформації – це процес використання криптографічних методів та технологій для забезпечення конфіденційності, цілісності та доступності інформації. Основна мета криптографічного захисту полягає в тому, щоб унеможливити несанкціонований доступ до даних, запобігти їхньому відтворенню або модифікації, а також забезпечити можливість перевірки автентичності користувачів та джерел інформації [118], [119].

Основні аспекти криптографічного захисту інформації включають:

Конфіденційність: Забезпечення того, що лише авторизовані особи мають доступ до конфіденційної інформації. Це досягається шифруванням даних, тобто перетворенням їх у нерозбірливий вигляд для тих, хто не має права доступу.

Цілісність: Забезпечення недоторканості даних. Це означає, що дані не можуть бути непомітно змінені або підроблені без відома та дозволу власника.

Доступність: Гарантування доступу до інформації для авторизованих користувачів та уникнення намагань блокування доступу незаконними шляхами.

Аутентифікація: Підтвердження ідентичності сторін взаємодії. Це може включати використання паролів, біометричних даних, сертифікатів тощо.

Невідмовність: Забезпечення можливості довести джерело або відправника інформації. Це важливо для уникнення заперечень з боку відправника про надсилання певного повідомлення.

Криптографічний захист використовує різноманітні алгоритми шифрування, цифрові підписи, хеш-функції та інші методи для досягнення вищевказаних цілей. Це особливо важливо у КФС, де збільшується кількість обміну інформаці-

єю через мережі та зберігання даних в електронному вигляді.

Криптографія – наука про математичні методи забезпечення конфіденційності. У роботі [53] наведено 6-рівневу модель КФС. Засоби КЗІ використовуються на рівнях з 1 по 5.

КЗІ включає в себе різні компоненти та вузли, які використовуються для захисту інформації, конфіденційності, цілісності та доступності інформації:

1. **Криптографічні алгоритми**: Вони використовуються для шифрування та розшифрування інформації. Прикладами є алгоритми шифрування, такі як *AES, DES, RSA, ECC*.

2. **Ключові управляючі системи**: Це системи для створення, керування та розподілу ключів для шифрування. Наприклад, системи управління ключами *(Key Management Systems - KMS)*.

3. **Шифратори і дешифратори**: Шифратори використовуються для перетворення звичайного тексту в криптований і навпаки. Дешифратори роблять зворотне перетворення.

4. **Модулі безпеки і таємниць**: Це фізичні або програмні модулі, які забезпечують безпеку на рівні апаратного забезпечення, такі як *TPM (Trusted Platform Module)* або *HSM (Hardware Security Module)*.

5. **Сертифікаційні органи та інфраструктура ключів**: Вони використовуються для видачі і управління цифровими сертифікатами, які підтверджують ідентичність та публічний ключ користувача.

6. **Системи аутентифікації (хешування)**: Вони дозволяють перевірити, чи має особа чи система право доступу до інформації. Це може включати паролі, біометричну аутентифікацію, смарт-карти тощо.

7. **Цифровий підпис**: Він використовується для забезпечення цілісності, автентичності та невідмовності документів та повідомлень в цифровому середовищі.

8. **Системи моніторингу та аналізу забороненої діяльності**: Вони використовуються для виявлення аномальних або заборонених дій, що можуть вказувати на порушення безпеки інформації.

9. **Засоби контролю доступу**: Вони використовуються для керування тими, хто має доступ до конкретної інформації або ресурсів.

10. **Системи реєстрації подій і журналювання**: Ці системи фіксують події, що стосуються безпеки, для подальшого аналізу і виявлення можливих інцидентів.

11. **Засоби захисту від витоку інформації**: Це включає в себе заходи для запобігання незаконному витоку чутливої інформації, такі як *Data Loss Prevention (DLP)* системи.

У роботі основну увагу приділено вузлам, які працюють з полями Галуа.

Криптографія поділяється на традиційну та постквантову [120], [32], [121], [122], [123]. Постквантова криптографія займається реалізацією криптографічних алгоритмів на квантових комп'ютерах [124], [125], [126], [127], [128]. Криптографія на основі еліптичних кривих – розділ криптографії, який вивчає асиметричні криптосистеми [129], засновані на еліптичних кривих над скінченими полями [130], [131], [132], [133], [134]. Електронний цифровий підпис (ЕЦП) – вид електронного підпису, отриманого за результатом криптографічного перетворення набору електронних даних. ЕЦП широко використовується для захисту інформації. В Україні стандарти ДСТУ 4145-2002 [82] та ДСТУ ISO/IEC 15946-1: 2015 [79] регулюють використання ЕЦП, в основу якого покладені операції над точками ЕК. Стандарт обмежується максимальним порядком полінома, який утворює двійкове поле Галуа $GF(2^m)$, $m = 509$, проте міжнародні стандарти рекомендують використовувати поля з порядком утворюючого поле полінома до $m = 998$ [81], [135], [136], [137]. Це підштовхує до побудови операційних пристроїв для обробки елементів полів Галуа з більшим порядком утворюючого поле полінома. Відповідно актуальною залишається задача мінімізації апаратної [6], [7], [12], часової [138], [139], [140], [141], структурної [142], [143], [144], [27], програмної складності.

1.6 Засоби КЗІ на основі еліптичних кривих та розширених полів Галуа $GF(p^n)$

Виконання операцій у розширених полях Галуа (також відомих як поле $GF(p^n)$, де p – просте число) над точками еліптичних кривих (ЕК) є важливою частиною криптографії і має застосування у сучасних криптографічних алгоритмах, таких як ЕЦП (електронний цифровий підпис).

Основні операції, які можна виконати над точками еліптичних кривих у розширених полях Галуа $GF(p^n)$, включають:

1. **Додавання точок (*Point Addition*)**: Додавання двох точок на еліптичній кривій. Основна операція в криптографії з еліптичними кривими. Додавання двох точок P і Q дає третю точку R, таку що $R = P + Q$. Нехай $P = (X_P, Y_P)$, $P \neq$ O і $Q = (X_Q, Y_Q)$, $Q \neq$ O, $P \neq$ Q – дві точки еліптичної кривої в афінних координатах.

Якщо $Q = -P$, то $R = O$. Якщо $Q \neq -$P, то координати (X_R, Y_R) точки R обчислюються за формулами [82]:

$$X_R = \left(\frac{Y_P + Y_Q}{Y_P + X_Q}\right)^2 + \frac{Y_P + Y_Q}{Y_P + X_Q} + X_P + X_Q + \text{A},$$

$$Y_R = \left(\frac{Y_P + Y_Q}{X_P + X_Q}\right)(X_P + X_R) + X_R + Y_P.$$

Як видно операції множення та ділення (яке також зводиться до множення) елементів полів Галуа $GF(p^n)$ займають суттєве місце при додаванні точок еліптичних кривих.

2. **Подвоєння точок (*Point Doubling*)**: Подвоєння точки P на еліптичній криві. Операція виконується для обчислення $2P$. Для точки P обчислюється точка $2P$.

Якщо $X_P = 0$, то $2P = O$. Якщо $X_P \neq 0$, то координати (X_R, Y_R) подвоєної точки $R = 2P$ обчислюються за формулами [82]:

$$X_R = {X_P}^2 + \frac{\text{B}}{{X_P}^2},$$

$$Y_R = {X_P}^2 + \left(X_P + \frac{Y_P}{X_P}\right) X_R + X_R.$$

Як видно операції множення та ділення (яке також зводиться до множення) елементів полів Галуа $GF(p^n)$ займають суттєве місце при подвоєнні точок еліп-

тичних кривих.

3. **Множення точок на ціле число (*Point Multiplication*)**: Операція виконується для обчислення добутку точки *P* на ціле число *k*. Ця операція може бути використана для створення публічного ключа або для виконання операцій підписування та перевірки підпису.

4. **Обчислення оберненої точки (*Point Inversion*)**: Операція, за якою обчислюється точка, обернена до заданої точки *P* на еліптичній кривій. Тобто, якщо $P = (x, y)$, то обернена точка – $Q = (x, -y)$.

5. **Перевірка належності точки кривій (*Point Membership Check*)**: Визначення, чи належить точка заданій еліптичній кривій. Це важливо для перевірки правильності введених даних.

6. **Знаходження порядку точки (*Point Order Finding*)**: Визначення порядку точки на еліптичній кривій, тобто знаходження числа *n* такого, що $nP = O$ (де *O* – точка відома як "нульова" або "ідентична").

Ці операції відіграють ключову роль у криптографії над точками еліптичних кривих, де вони використовуються для створення ключів, підпису повідомлень, аутентифікації та інших криптографічних операцій.

1.7 Еліптичні криві Едвардса

Еліптичні криві Едвардса зазвичай використовують розширені поля Галуа $GF(p^n)$, де $p \neq 2$ – просте число, а *n* – додатне ціле число.

Еліптичні криві Едвардса виглядають наступним чином в проективних координатах:

$$x^2 + y^2 = c^2(1 + dx^2y^2)$$

d – константи, обрані таким чином, що вони відповідають властивостям безпечних еліптичних кривих.

Для кривих Едвардса [2], беручи в якості нейтрального елемента точку $(0, 1)$, суму точок (x_1, y_1) та (x_2, y_2) можна представити формулою [210]:

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 + x_1x_2}{1 - dx_1x_2y_1y_2}\right).$$

При подвоєнні точок їх координати однакові, тому можна користуватись вище наведеними формулами додавання і для подвоєння.

Як видно операції множення та ділення (яке також зводиться до множення) елементів полів Галуа $GF(p^n)$ займають суттєве місце при додаванні та подвоєнні точок еліптичних кривих Едвардса.

Важливо відзначити, що для ефективної реалізації криптографічних примітивів, таких як ЕЦП, вибір параметрів еліптичних кривих впливає на безпеку та ефективність. Вибір характеристики поля p, а також констант d повинен враховувати вимоги безпеки та ефективності криптографічного застосування.

У 2020 році запроваджено новий Національний стандарт ДСТУ 9041:2020 [2]. Цей алгоритм призначений для шифрування коротких (до 616 біт) повідомлень для будь-яких алгоритмів шифрування, в тому числі визначених національними стандартами України.

Новий алгоритм використовує криптографічні перетворення у групі точок еліптичних кривих, використовуючи замість кривих у формі Вейєрштрасса найновітніші розробки у галузі еліптичної криптографії – криві у формі Едвардса. Це дає суттєві переваги у швидкодії більш ніж у 3 рази. Новий стандарт, розроблений з урахуванням усіх найсучасніших вимог до стійкості криптографічних алгоритмів.

Стандарт ДСТУ 9041:2020 [2] узгоджується з усіма наявними національними стандартами України. Нововведенням цього стандарту є його область застосування – інкапсуляція ключів, застосування передового математичного апарату, а також новий алгоритм для генерації псевдовипадкових послідовностей. Цей алгоритм відрізняється від подібного в ДСТУ 4145 [82], оскільки він базується, виключно, на національних криптографічних алгоритмах і не містить посилань на пострадянські стандарти, термін дії яких майже завершився.

Цей новий стандарт не входить до категорії так званих постквантових стандартів. Втім, його надійність опиниться під загрозою лише з появою квантових комп’ютерів, що мають 700 або більше кубітів (наразі вдалося досягти приблизно 50 "робочих" кубітів). Однак основною перевагою цього стандарту перед пос-

тквантовими алгоритмами є значно менша довжина ключа, яка може бути в десятки чи навіть сотні разів коротшою, ніж у постквантових алгоритмів.

Поява стандарту ДСТУ 9041:2020 [2] підкреслює актуальність даної роботи.

1.8 Поля Галуа

Арифметичні операції у полях Галуа $GF(p^n)$ широко використовуються в теорії кодування [145], цифровій обробці сигналів [146] та у криптосистемах з відкритим ключем [147], [148]. В основі виконання арифметичних операцій над елементами розширених полів Галуа $GF(p^n)$ лежить виконання операцій над його елементами за модулем характеристики поля – $mod\ p$.

Додавання в розширених полях Галуа $GF(p^n)$ виконується як порозрядна операція додавання за модулем p. Операції знаходження оберненого елемента і піднесення до степеня потребують більше часу, ніж базові операції: множення і додавання [149]. Виконання операції знаходження оберненого елемента [150], [151], [152] виконується за допомогою ітеративного множення. Отже, ефективне виконання операцій множення в розширених полях Галуа $GF(p^n)$ є ключовим аспектом для криптографічних пристроїв, що базуються на використанні цих полів.

Стаття [153] має на меті надати стислий погляд на проектування ефективних архітектур в арифметиці кінцевого поля. У цій статті запропоновано арифметику поля Галуа з використанням незвідного полінома для створення *S*-блоку для *AES* з використанням 128, 192 і 256-бітних ключів. У статті [154] розглядається реконфігурована цифрова схема, яка виконує різні арифметичні операції *GF* з тим самим набором апаратних засобів. Використовуючи це обладнання, можна виконувати декілька операцій паралельно. У роботі [155] запропоновано основу для перевірки коректності арифметичних операцій поля Галуа в цифрових схемах.

1.9 Операції над елементами розширених полів Галуа $GF(p^n)$

Розширені поля Галуа можна поділити на двійкові $GF(2^n)$ та з простою характеристикою поля $GF(p^n)$, $p>2$. Двійкові поля мають декілька переваг у по-

рівнянні з полями з іншою характеристикою. Найбільш вагомим є той факт, що сучасні комп'ютери побудовані на основі двійкової системи числення. За допомогою n біт всі можливі елементи поля $GF(2^n)$ можуть бути представлені без надлишковості. Це неможливо у простих полях де $p \neq 2$. Наприклад поле $GF(2^n)$ за допомогою 2 біт може представити абсолютно усі можливі комбінації поля. Поле $GF(3)$ також буде використовувати 2 біти для представлення 3 значень поля. У такому випадку 1 значення залишається незадіяним, тобто ресурси використовуються неоптимально. Інша перевага двійкових полів Галуа в тому, що реалізація обчислень у цих полях є простішою в порівнянні з розширеними полями Галуа [6], для виконання найбільш поширених операцій додавання, множення та піднесення до квадрату. Апаратна реалізація арифметичних операцій у розширених двійкових та простих полях Галуа з використанням конфігурованих елементів є недостатньо вивченою. У даній дисертаційній роботі це буде детально досліджуватись.

Існують 2 способи представлення полів Галуа - поліноміальний і нормальний базиси [82], які використовуються для оптимізації операцій над елементами цього поля.

Елементи $\{t^{n-1},\ldots,t^2,t,1\}$ основного розширеного поля Галуа $GF(p^n)$ утворюють поліноміальний базис, елементи $\{\theta, \theta^2, \theta^{2^2}, \ldots, \theta^{2^{n-1}}\}$ основного розширеного поля Галуа $GF(p^n)$ утворюють нормальний базис (θ і t– корені полінома p, що утворює поле). Решту елементів основного поля Галуа $GF(p^n)$ можна подати як у поліноміальному базисі (у вигляді $a_{n-1}t^{n-1}+\ldots+a_2t^2+a_1t+a_0$), так і у нормальному базисі (у вигляді $a_0\theta+a_1\theta^2+a_2\theta^{2^2}+\ldots+a_{n-1}\theta^{2^{n-1}}$), де a_i – розряди коду елемента поля ($i = 0, 1, \ldots, n-1$).

Обираючи між поліноміальним та нормальним базисами, важливо враховувати конкретні вимоги задачі та обчислювальні ресурси, які доступні для виконання операцій над розширеними полями Галуа $GF(p^n)$. Обидва ці базиси мають свої переваги та недоліки і можуть бути корисними в різних галузях застосування. Для реалізації помножувачів ми обрали поліноміальний базис.

У роботі [156] розглядається виконання арифметичних операцій у $GF(2^n)$ в поліноміальному базис. У [157] представлено нову архітектуру послідовних помножувачів для поліноміального базису розширеного двійкового поля $GF(2^n)$, створеним незвідними тричленами. Тричленами називаються поліноми третього ступеня, які не розкладаються на множники меншого ступеня у цьому полі. Такі тричлени також називаються "незвідними тричленами" або "неприводимими тричленами".

У [158] для засобів КЗІ на основі еліптичних кривих пропонується масштабований біт-паралельний словно-послідовний помножувач елементів скінченного поля Галуа $GF(2^n)$ з можливістю виявлення помилок.

Робота [159] описує процесор для побудови криптографічних систем на основі еліптичних кривих. Пропонований процесор складається із спеціальних компонент, найважливішою з яких є модульний блок множення, реалізований за допомогою алгоритму систолічного множення Монтгомері. Іншою новизною запропонованої архітектури є те, що вона реалізує виконання арифметичних операцій у полі $GF(3^n)$, що забезпечує приріст продуктивності.

Робота [160] має на меті зменшити розрив у швидкості між множенням за алгоритмом Монтгомері та модульним множенням у простому полі Галуа $GF(p)$, шляхом реалізації помножувача Монтгомері на ПЛІС з використанням надлишкової системи числення. У документі [161] запропоновано дві оборотні схеми приховування даних у зашифрованому зображенні за допомогою секретного обміну через поля Галуа $GF(p)$ і $GF(2^8)$. У роботі [162] пропонується два алгоритми множення елементів полів $GF(p^n)$ та їх апаратну реалізацію для нормального базису, де p ∈ {2,3}. У цьому випадку запропоновані помножувачі розроблено з використанням послідовної та цифрово-послідовної структур. Алгоритми множення множників представлених у нормальному базисі для $GF(2^n)$ і $GF(3^n)$ базуються на двох запропонованих алгоритмах для обчислення матриць множення, з метою прискорення часу виконання та зменшення ресурсів. У роботі [163] запропоновано схему представлення великих цілих чисел на основі атрибутів, яка

підходить для простих чисел *NIST* і простих чисел Пірпонта, що використовуються в суперсингулярній ізогенії Діффі-Хеллмана (*SIDH*) для постквантової криптографії. У роботі [164] представлено алгоритм малої складності стиснення з втратами з протографними кодами *LDPC* (коди *P-LDPC* з втратами) над $GF(2)$. Запропонована схема використовує для кодування даних в алгоритмі перетасованого посиленого поширення *(SRBP),* який асимптотично досягає теоретичної швидкості спотворення функції. У статті [165] розглянуто узагальнення двійкових нерівномірно пов'язаних кодів перевірки парності з низькою щільністю *(LDPC)* над полем Галуа $GF(2^n)$. У статті [166] автори реалізували швидку реалізацію скалярного множення для будь-якої загальної кривої Монтгомері в полі Галуа $GF(p)$ без використання будь-якого спеціального модуля. У статті [167] автори досліджують найбільш підходящі операції множення у полях $GF(p)$, які відповідають скалярному множенню з фіксованою комою, а також з рухомою комою і надають схеми реалізації для обох випадків. Крім того, розглядають вибір параметрів, задіяних в обох схемах, які впливають як на швидкість скалярного множення, так і на витрати пам'яті. У роботі [168] виконано реалізацію помножувача елементів скінченого поля Галуа $GF(2^n)$ низької обчислювальної складності, у якій використано примітивні поліноми, як незвідні поліноми. Цей модифікований помножувач реалізовано в *MATLAB*.

1.10 Операції додавання та віднімання у полях Галуа $GF(p^n)$

Припустимо, що F є множиною з двома бінарними операціями $+$ і $*$ (додавання і множення). F є полем, якщо:

1) F є абелевою групою за додаванням +;

2) $F\times = F\backslash\{0\}$ є абелевою групою за множенням×;

3) Виконується дистрибутивність для всіх a, b, c з множини F: $a\times(b + c) = a\times b + a\times c, (a + b)\times c = a\times c + b\times c$.

Абелева група – група, в якій бінарні операції є комутативними. Іншими словами група абелева, якщо $a\times b = b\times a$, для будь-яких двох елементів.

Якщо число елементів F скінченне то F називається скінченним полем або

полем Галуа. Під час множення двох елементів поля Галуа у поліноміальному базисі порозрядні операції додавання та множення виконуються за модулем p.

Елементи множини поліноміального базису можна подати у вигляді розрядів: $(a_{\text{n-1}}, \ldots, a_2, a_1, a_0)$.

Запишемо формули для операцій додавання елементів розширених полів Галуа $GF(p^n)$. Припустимо що в нас є елементи A та B поля $GF(p^n)$:

$A = (a_{\text{n-1}}, .., a_2, a_1, a_0)$,

$B = (b_{\text{n-1}}, \ldots, b_2, b_1, b_0)$, тоді

$A + B = (S_{n-1}, \ldots, S_2, S_1, S_0)$,

$S_i = a_i \oplus_p b_i$, де $\oplus_p$ позначає операцію додавання за модулем p.

Розглянемо операцію додавання над елементами A та B в полі $GF(3^n)$:

$A = (a_{\text{n-1}} \ldots a_2 a_1 a_0)$

$B = (b_{\text{n-1}} \ldots b_2 b_1 b_0)$

Де $a_n = \{\alpha_1 \alpha_0\}_n$, а α_0 та α_1- двійкові розряди поля $GF(3^n)$:

$$\begin{array}{cccccc} & a_{n-1} & \ldots & a_2 & a_1 & a_0 \\ \oplus_3 & & & & & \\ & b_{n-1} & \ldots & b_2 & b_1 & b_0 \\ \hline & S_{n-1} & \ldots & S_2 & S_1 & S_0 \end{array}$$

$S_k = \sum_{i+j=k} a_i \oplus b_j$,якщо $0 \leq k \leq 2\text{n-}1$ - формула для знаходження суми кожного розряду елементів поля.

Операція додавання детально описана у [226].

1.11 Операція множення у розширених полях Галуа $GF(p^n)$

Множення у розширених полях Галуа $GF(p^n)$ може бути послідовним та паралельним. Послідовне виконання множення описане у роботах [169], [170], [171]. Паралельне множення описано у наступних роботах: [172], [149], [173], [174], [175], [176], [177], [178]. Операція ділення є операцією множення на обернений елемент [179], [180], [181], операція знаходження кореня квадратного описана у [182].

Один з методів паралельного множення елементів розширених полів Галуа $GF(p^n)$, складається з двох етапів: знаходження проміжного добутку та знахо-

дження остачі від його ділення на простий поліном, що утворює дане розширене поле Галуа $GF(p^n)$. Для знаходження проміжного результату кожен розряд першого елемента множиться на кожен розряд другого елемента за модулем n $GF(p^n)$ у стовпчик.

$S_{пром._k} = \sum_{i+j=k} a_i b_j$,якщо $0 \leq k \leq 2n-1$ – формула для знаходження k-того розряду проміжного результату множення, $\sum_{i+j=k}$ позначає операцію додавання за модулем n $GF(p^n)$.

Далі, проміжний добуток, який ми отримали, ділиться на простий многочлен степеня n, що утворює дане розширене поле Галуа $GF(p^n)$. Результатом множення буде остача від ділення. Ділення зводиться до додавання до проміжного результату добутку многочлена та старшої частини проміжного результату, як показано у формулах нижче.

$A * B = ((a_{n-1}, \ldots, a_2, a_1, a_0) * (b_{n-1}, \ldots, b_2, b_1, b_0)) = (S_{n-1}, \ldots, S_2, S_1, S_0)$,

$S_{пром.} = (S_{2n-1}, \ldots, S_2, S_1, S_0)$,

$S_{пром.} = S_{ст.} \&\& S_{мол.}$, де && позначає об'єднання результату (конкантенацію),

$S_{ст.} = (S_{2n-1}, \ldots, S_n)$,

$S_{мол.} = (S_{n-1}, \ldots, S_0)$,

$S = S + p * S_{ст.} = p * (S_{2n-1}, \ldots, S_n)$.

Для знаходження проміжного результату, множення відбувається за модулем p у стовпчик. Після цього, результат ділимо на простий многочлен степеня n $GF(p^n)$. Остача від ділення буде результатом множення. Ділення можна замінити покроковим додаванням цього полінома до проміжного добутку починаючи із старшого розряду у випадку, якщо він рівний 1, поки результат не буде числом в межах поля.

$A * B = ((a_{n-1}, \ldots, a_2, a_1, a_0) * (b_{n-1}, \ldots, b_2, b_1, b_0)) = (S_{n-1}, \ldots, S_2, S_1, S_0)$

$S_k = \sum_{i+j=k} a_i b_j$,якщо $0 \leq k \leq 2n-1$ - формула для знаходження проміжного результату множення.

На рис. 1.2 схематично показане множення двох елементів поля $GF(2^n)$ з

використанням комірок Гілда.

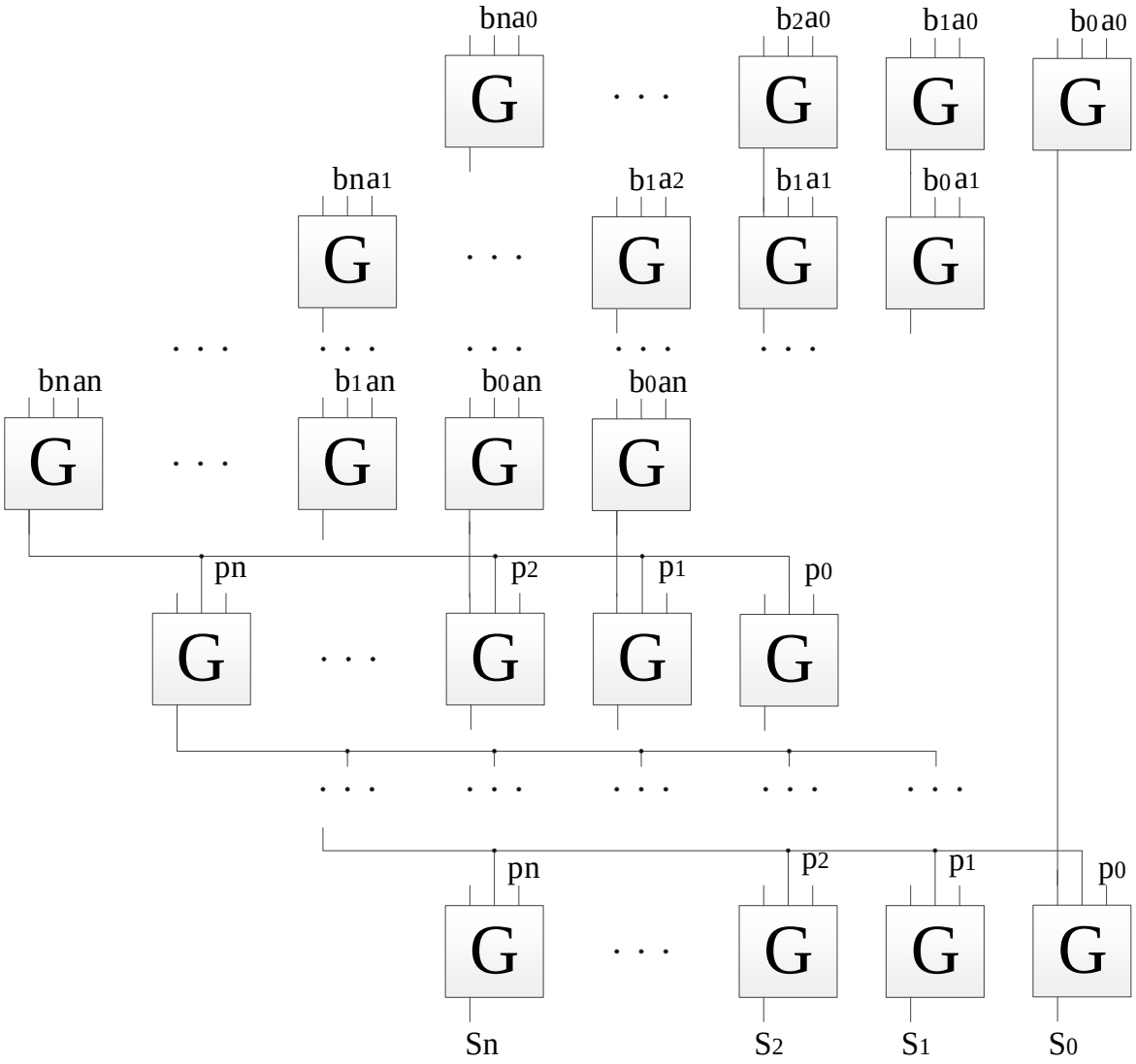


Рис. 1.2 Схема множення двох елементів поля $GF(2^m)$ з використанням КГ

Спочатку обчислюємо проміжний результат множення, потім отриманий результат ділимо за модулем простого полінома – остача від ділення буде результатом. Для простішої реалізації на схемі виконується покрокове додавання многочлена, якщо розряди які вийшли за межі поля рівні 1. Поліном для поля розміру n буде мати вигляд: $c_n x^n + \ldots + c_2 x^2 + c_1 x^1 + c_0 x^0$.

1.12 Методи виконання множення у розширених полях Галуа $GF(p^n)$

Множення у розширених полях Галуа $GF(p^n)$ є важливою складовою багатьох криптографічних алгоритмів в різних галузях математичних та інформаційних наук. Існує кілька методів виконання множення в таких полях, і вони мо-

жуть бути класифіковані наступним чином:

Множення у розширених полях Галуа $GF(p^n)$ можна класифікувати за структурою, архітектурою та обсягом ресурсів [183]:

1. За структурою апаратного забезпечення:

   1) Послідовний помножувач: В цьому випадку біти вхідних чисел обчислюються послідовно, один за одним, за допомогою логічних елементів.

   2) Паралельний помножувач: В цьому випадку біти вхідних чисел обчислюються паралельно.

2. За архітектурними особливостями:

   1) Помножувач на основі додавання: Використовує послідовні додавання для отримання результату множення. Він зазвичай використовує позиційне представлення чисел.

   2) Помножувач на основі логічних відображень: Використовує таблиці або логічні відображення для виконання операцій множення у полях Галуа. Цей метод може бути ефективним для малих полів Галуа.

   3) Множення на основі операцій над бінарними числами: Використовує операції *XOR*, *AND*, та інші для виконання операцій множення над бінарними числами.

3. За обсягом ресурсів:

   1) Множення на основі оптимізації апаратного забезпечення: Використовує оптимізовані апаратні рішення для максимальної швидкодії і мінімального споживання ресурсів.

   2) Множення на основі обчислювальних алгоритмів: Використовує алгоритми множення з оптимізованими обчислювальними методами.

Множення в розширених полях Галуа $GF(p^n)$ може бути реалізовано різними методами, кожен має свої перевагами та недоліки для конкретних застосувань. Ці методи можна поділити на категорії, виходячи з їхнього підходу та основних принципів [183]:

1. Методи на основі таблиць.

- Множення за допомогою таблиць множення: Цей простий метод включає в себе попереднє обчислення таблиць множення для всіх можливих пар елементів у полі Галуа. Під час множення відповідні записи з таблиць переглядаються та комбінуються для отримання результату. Цей метод є простим, але вимагає значної пам'яті для зберігання таблиць.

2. Бітові методи:

- Множення методом зсуву та додавання [184]: Також відоме, як бінарне множення, цей метод використовує двійкове представлення елементів у полі Галуа. Він включає в себе зсув одного операнда біт за бітом та умовне додавання його до поточної суми на основі значення поточного біта. Цей метод є ефективним за використанням пам'яті і підходить для програмних реалізацій.

- Метод Монтгомері (*Montgomery Multiplication*) [185]: Цей метод особливо ефективний для модульного множення у полях Галуа. Він використовує методи операцій з бітами та модульних редукцій для прискорення обчислень.

3. Поліноміальні методи:

- Множення на основі поліноміального базису [186]: Елементи у полях Галуа можуть бути представлені у вигляді поліномів, і множення виконується шляхом множення цих поліномів за модулем незвідного полінома, який визначає поле. Цей метод використовується в багатьох апаратних реалізаціях.

- Множення на основі нормального базису [187]: У цьому підході елементи представлені в нормальному базисі, що спрощує операції множення до послідовності зсувів та додавань. Це ефективно для полів певного розміру та використовується в апаратних рішеннях.

4. Алгоритми швидкого множення поліномів:

- Алгоритм Карацуби (*Karatsuba Algorithm*) [188]: Цей алгоритм застосовує множення поліномів рекурсивно, зменшуючи кількість необхідних множень. Його можна адаптувати для множення поліномів у полях Галуа.

- Алгоритм Штрассена (*Schönhage–Strassen Algorithm*) [189]: Цей алгоритм використовує швидкі перетворення Фур'є (*FFT*) для прискорення множення поліномів. Він також може бути застосований до множення поліномів у полях Га-

луа.

5. Методи на основі матриць.

- Матричне множення [190]: Цей метод включає в себе представлення елементів поля Галуа як векторів та виконання матричного множення для обчислення добутків. Методи на основі матриць можуть бути особливо ефективними для певних розмірів полів та використовуються в деяких апаратних реалізаціях.

6. Спеціалізовані апаратні методи.

- Спеціалізовані апаратні реалізації [191]: Спеціалізоване апаратне забезпечення, таке як програмовані логічні схеми (*FPGAs*) або інтегральні мікросхеми зі спеціальним призначенням (*ASICs*), можуть бути розроблені для ефективного виконання операцій множення в полях Галуа. Ці реалізації високо оптимізовані для конкретних розмірів полів та застосувань.

7. Методи Монте-Карло.

- Випадкові алгоритми [192]: У деяких застосуваннях можуть використовуватися методи Монте-Карло для ефективного наближеного множення в полях Галуа. Ці методи є ймовірнісними та надають результати з певним рівнем точності.

8. Множення на основі комірок Гілда.

- Множення виконується у матричних помножувачах, кожен вузол якого є коміркою Гілда [6].

Вибір методу множення залежить від факторів, таких як розмір поля Галуа, доступні обчислювальні ресурси, конкретні вимоги застосунку (наприклад, швидкість, використання пам'яті) та те, чи виконання реалізується у програмному або апаратному середовищі. Кожен метод має свої власні компроміси, і вибір належного може бути важливим для оптимізації продуктивності в різних обчислювальних завданнях та застосунках.

У цій роботі ми реалізовуємо помножувачі з паралельною структурою, на основі комірок Гілда, за допомогою апаратної реалізації на ПЛІС.

1.13 Комірка Гілда та модифікована комірка Гілда для розширених двійкових полів Галуа *GF(2ⁿ)*

Помножувач [193], [194] для двійкових розширених полів Галуа $GF(2^m)$ може бути реалізований на основі модифікованих комірок Гілда (МКГ) [102]. МКГ на відміну від комірки Гілда (КГ) не має виходу переносу. МКГ для полів Галуа $GF(2^m)$ повинна мати 3 входи та 1 вихід (рис. 1.3).

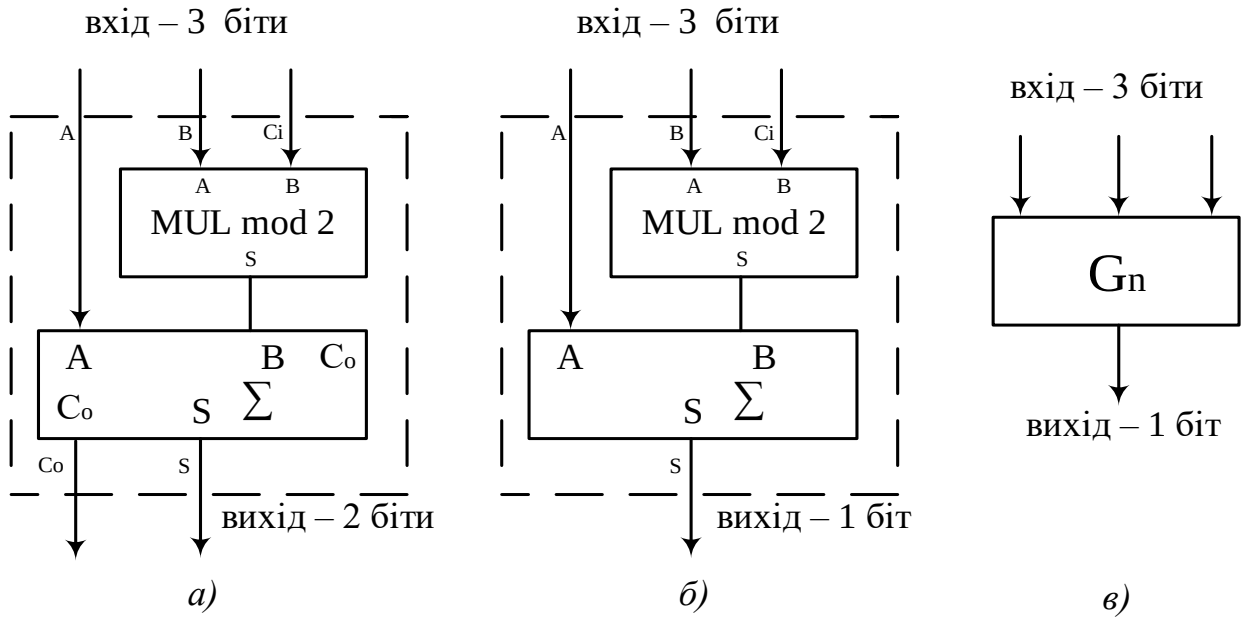


Рис. 1.3 а) комірка Гілда, б) модифікована комірка Гілда для обробки елементів розширених двійкових полів Галуа $GF(2^m)$, в) символ модифікованої комірки Гілда $GF(2^m)$

1.14 Криптопроцесор

Вузол КЗІ – це криптопроцесор (рис. 1.4) [195], який містить в своєму складі пристрій керування (*Arithmetic Control Unit*), пам’ять (*RAM*), пристрій додавання (*Adder*), пристрій множення (*Multiplier*), пристрій піднесення до квадрату (*Squarer*).

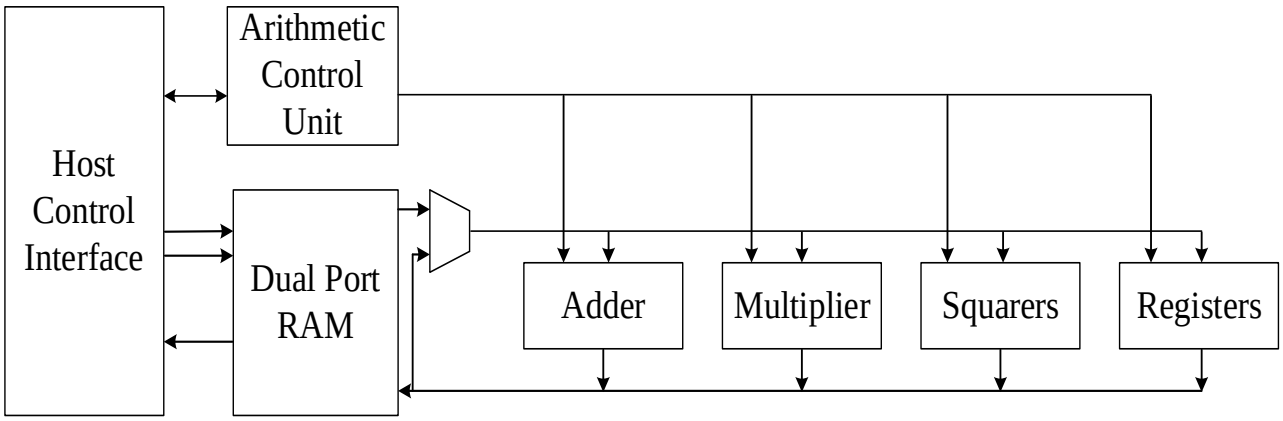


Рис. 1.4 Криптопроцесор

1.15 Генератори ядер та методи генерації описів функціональних вузлів

Генератори ядер (*IP cores*) – програмні або апаратні засоби для генерування конфігурованих або реконфігурованих *IP*-ядер [33]. *IP*-ядра (англ. *IP cores*), *IP*-блоки (*IP* – англ. *intellectual property*) – *VHDL*-описи моделей функціональних вузлів [23].

Розрізняють три основні класи *IP*-блоків [33]:

1) програмні *IP*-блоки – блоки, специфіковані мовою опису апаратури;

2) схемотехнічні блоки – блоки, специфіковані на схемотехнічному рівні, без залежності від конкретної топологічної реалізації;

3) фізичні блоки (*Hard IP-Core*) - блоки, специфіковані фізично для реалізації у НВІС (наприклад, *GDSII* для *ASIC*).

Існуючі засоби проектування на системному рівні можна поділити на 2 методи [33]:

1) Метод переведення опису проекту на мові високого рівня в логічні вентилі ПЛІС;

2) Метод створення спеціалізованого процесора шляхом конфігурування базової моделі.

У ПЛІС під *Hard IP-Core* розуміють спеціалізовані області кристала, виділені для виконання певних функцій [124]. У цих місцях на кристалі реалізовані блоки незмінної структури, спроектовані за методологією *ASIC (Application-Specific Integrated Circuit)*, оптимізовані для виконання заданих функцій, які не

мають засобів її програмування. У разі використання цього виду ядер розмір площі, що використовується на кристалі, зменшується, покращуються характеристики швидкодії, але відбувається втрата універсальності.

Створенням генераторів ядер на ПЛІС займались такі вчені як Мельник А.О. [33], Мельник В.А. [33], Глухов В.С. [138], Еліас Р. [139], Рахма М. [140]. У роботі [33] детально розглянуто всі аспекти створення генераторів ядер та генерування з їх допомогою конфігурованих та реконфігурованих вузлів на ПЛІС. Як приклад наведено систему "Хамелеон" [196]. У роботі [197] проаналізовано особливості засобів генерування програмних моделей обчислювальних пристроїв, порівняно характеристики цих засобів та запропоновано рекомендації щодо застосування цих засобів під час побудови самоконфігуровної комп'ютерної системи.

Методи генерації описів функціональних вузлів.

Способи опису функціональних вузлів. Описати проект можна:

1) схематично;

2) мовою опису апаратних засобів (*VHDL*);

3) у вигляді графа автомата.

Найбільш універсальним є *VHDL*-опис. Сучасні засоби проектування забезпечують трансляцію схем і графів у *VHDL*-описи [198], [199]. Також існують генератори *VHDL*-описів стандартних вузлів цифрової техніки – генератори ядер [33].

Мови опису апаратних засобів. Мови *VHDL* і *Verilog (Verilog HDL)* відносяться, до мов опису апаратури [200]. Вони призначені не для написання програм для *FPGA* та НВІС, а для проектування логіки самих цих пристроїв. Ці мови призначені для моделювання електронних схем на вентильному рівні, регістрових передач, корпусів мікросхем. Тому ці мови можна назвати мовами наскрізного функціонально-логічного проектування.

Останнім часом популярності набули системи високорівневого проектування з мов високого рівня *C* або *C++ (HLS – High Level Synthesis)*. Синтез високого рівня - це методика використання програмованої логіки без використання

традиційних мов опису пристрою (*Verilog/VHDL*) і не потребує попереднього знання практики проектування *FPGA/VLSI*. Інструменти *HLS* складають функцію *C/C++* в логічні елементи, спрямовані на ефективне використання програмованого пристрою для швидкої роботи та використання економічних ресурсів. Це дає можливість звичайним програмістам написати власну логіку, не будучи фахівцями з *FPGA*. У документі [201] представлено метод перевірки еквівалентності для перевірки *GCSE* у розкладі синтезу високого рівня шляхом посилення критеріїв еквівалентності шляху. У статті [202] автори пропонують лінійний алгоритм планування часу для циклічного конвеєра без зворотного відстеження. Результати тестів показують, що цей алгоритм може бути більш ніж у 1000 разів швидшим за ітераційний алгоритм на основі *SDC* у *LegUp*, досягаючи того самого. Якщо порівнювати з промисловим інструментом *Vivado HLS*, цей алгоритм все ще може бути в середньому більш ніж у 500 разів швидшим із порівнюваною якістю результатів. У роботі [203] запропоновано техніку спільного використання модулів функціонального рівня в *HLS*. У роботах [196] та [33] розглядається система високорівневого синтезу спеціалізованих процесорів – “Хамелеон”.

На рис. 1.5 наведений процес перетворення програми на мові *C* у *VHDL*-опис.

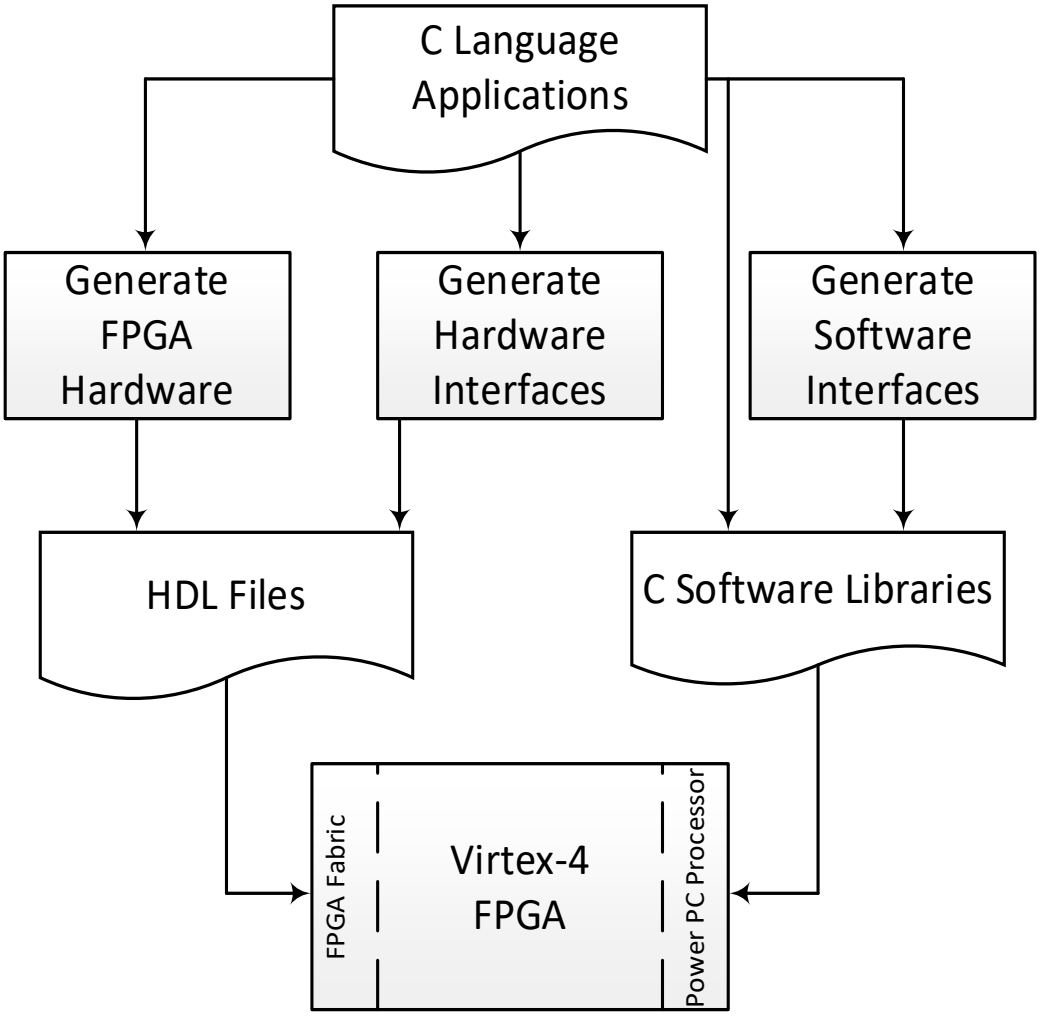


Рис. 1.5 Набір засобів CoDeveloper фірми Impulse Accelerated Technologies

Для математичних обчислень використовується стандартна бібліотека *math.h*. Для переведення змодельованих на мові C/C++ описів розробляються транслятори. З'являється тенденція для переходу від описів у вигляді *RTL* до описів у вигляді *HDL* з наступною автоматичною генерацією *RTL*-описів [224] і, навіть, автоматичною генерацією переліку зв'язків для проектування топології ПЛІС.

1.16 Основи проектування засобів КЗІ як спеціалізованих комп'ютерних засобів

Спеціалізовані комп'ютерні засоби [232] використовуються для реалізації систем вузької спеціалізації. До таких засобів відносяться вузли КЗІ. СКЗ працюють за певним наперед визначеним алгоритмом.

Алгоритм. Алгоритм - набір інструкцій, що описують порядок дій виконавця для досягнення певного результату [229]. Незалежні інструкції можуть виконуватись паралельно, якщо вони виконують різні дії.

Побудова алгоритму. В основу процесу проектування СКС, з розбиттям її на достатню кількість дрібних складових можна покласти наступну послідовність дій [213]:

1) вихідним станом процесу проектування є формулювання мети алгоритму або конкретніше – результату, який повинен бути отриманий при його виконанні;

2) проводиться збір фактів, які стосуються будь-яких характеристик алгоритму;

3) створюється образна модель процесу, у якій, використовуються графічні та інші способи представлення, образні „картинки”, які дозволяють краще зрозуміти особливості виконання алгоритму;

4) в образній моделі виділяється найбільш суттєва частина, для якої підбирається найбільш точне формулювання;

5) проводиться визначення змінних, які необхідні для формального представлення даного кроку алгоритму;

6) вибирається одна з конструкцій – проста послідовність дій, умовна конструкція або цикл;

7) для інших незрозумілих частин алгоритму, які залишились – перерахована послідовність дій повторюється.

Складність алгоритму (*SH*-модель алгоритму). *SH*-модель об’єднує теорії абстрактних алгоритмів з розв’язанням задач на реальних комп’ютерах. Під час синтезу, аналізу та для вдосконалення SH-моделей рекомендується застосовувати п'ять критеріїв складності: апаратну, часову, ємнісну, програмну і структурну [116], [117], які взаємопов’язані і взаємозалежні. У роботі увага в першу чергу зосереджена на мінімізації апаратної складності, при цьому приймається, що інші характеристики мають допустиму для рішення поставлених задач величину.

1.17 Елементна база для створення апаратно-програмних та апаратних вузлів КЗІ

ПЛІС є перепрограмованою мікросхемою. Розробка високотехнологічних інструментів дизайну сприяла зміні правил програмування ПЛІС за новими технологіями, які перетворюють графічні блок-схеми або навіть *C*-код у цифрову апаратну схему [216].

Використання ПЛІС у всіх галузях економіки зумовлене тим фактом, що ПЛІС об'єднує найкращі частини *ASIC (Application-Specific Integrated Circuit)* [217] та процесорних систем. ПЛІС забезпечують швидкість і надійність. ЦП також має таку ж гнучкість програмного забезпечення як і ПЛІС. На відміну від процесорів, ПЛІС мають справді паралельний характер виконання команд. Кожне незалежне завдання обробки призначається окремій частині мікросхеми і може функціонувати автономно без будь-якого впливу з боку інших логічних блоків. В результаті продуктивність однієї частини мікросхеми не впливає на іншу, при додаванні додаткового функціоналу.

Елементи жорсткої логіки реалізуються у вигляді ПЛІС. Останнім часом стали доступними до використання гібридні структури - системи на кристалі: мікроконтролери, які містять в своєму складі додаткові елементи, в тому числі, і ПЛІС [216], [217], [159], [153], та ПЛІС, які містять в своєму складі мікроконтролери та інші додаткові елементи [225].

Мікроконтролери. Мікроконтролери – це компактні інтегральні мікросхеми, які містять у собі процесор, пам'ять і периферійні пристрої, і призначені для виконання специфічних завдань в електронних системах.

Захищеність ПЛІС. Логіка роботи (ПЛІС, ПЛМ) давно використовується для проектування СКС та вузлів КЗІ [243], [244], [245].

Ядра на ПЛІС (*IP cores, IP*-блоки, *IP*-ядра, *VC*) – готові модулі для створення систем на кристалі (*VC* - віртуальні компоненти).

Ядра поділяють на 3 основних класи:

1) програмні *IP*-блоки (*soft blocks*), специфіковані мовою опису апаратури, наприклад, мовою *VHDL* у вигляді *VHDL*-описів;

2) схемотехнічні блоки (*firm blocks*) – це блоки, які визначені на рівні схемотехніки, без залежності від конкретної топологічної реалізації;

3) фізичні (топологічні) блоки (*hard blocks*) – це блоки, які визначені на фізичному рівні реалізації ПЛІС.

Ядра для засобів КЗІ. Відомі ядра для реалізації вузлів систем КЗІ на ПЛІС. Це, в основному, ядра для шифрування, відповідно до алгоритмів *DES*, *3DES*, *AES*, а також для гешування відповідно до алгоритмів *SHA-1* [253], SHA-256 [254] та *MD*-5 [255], [251].

З наступних публікацій відомі ядра на ПЛІС для роботи з полями Галуа $GF(2^m)$ [230], [124], [125], [152] та ядра для виконання операцій над точками ЕК – [235], [202].

Недоліками ПЛІС є необхідність повторної конфігурації після кожного ввімкнення живлення та обмежений рівень захисту інтелектуальної власності.

Перехід від *FPGA* до *ASIC.* Це є найбільш ефективним рішенням для вирішення проблеми повторної конфігурації після кожного ввімкнення живлення. Сучасні технології дозволяють здійснити цей перехід після проектування та налагодження *FPGA*. Процес проектування *ASIC* при такому підході ілюструє [246], [247].

Захист інтелектуальної власності. Відомі ПЛІС [242], що включають в себе вбудований стійкий ідентифікаційний номер. Крім того, можливе внесення користувацького ідентифікаційного номеру до ПЛІС. Ці номери можуть бути використані для розробки схеми захисту від несанкціонованого копіювання.

Для захисту інтелектуальної власності (ядер) від несанкціонованого клонування та використання починають використовувати електронний цифровий підпис (ЕЦП) [237], [241]. Також ЕЦП використовується для виявлення зловмисного трафіку в комп’ютерних мережах [238].

1.18 Багаторівнева структура засобів КЗІ

Для проведення аналізу обрано модель взаємодії відкритих систем. Згідно з цією моделлю, спеціалізовані комп'ютерні інструменти та спецпроцесори пред-

ставлені у вигляді функціональних каналів із багаторівневою та багатозадачною структурою (див. рис. 1.6) [212].

Проектування СП, який виконує операції над точками ЕК, вимагає використання таких спеціальних розділів математики як поля Галуа та ЕК. Елементи скінченних полів Галуа та точки на ЕК кодуються за допомогою бінарних кодів великої довжини (від сотень до тисяч бітів). Для операцій із цими елементами та точками у СП потрібні унікальні технічні рішення.

За результатами аналізу формується уява про дворівневу структуру (рис. 1.7, 1.8) вузлів КЗІ на основі ЕК. Верхній рівень забезпечує обмін інформацією із зовнішнім середовищем. Нижній (власне спеціалізований, СП) – забезпечує виконання специфічних для даної задачі операцій. Розвитком цієї ідеї є погляд на СП як на багаторівневу відкриту систему.

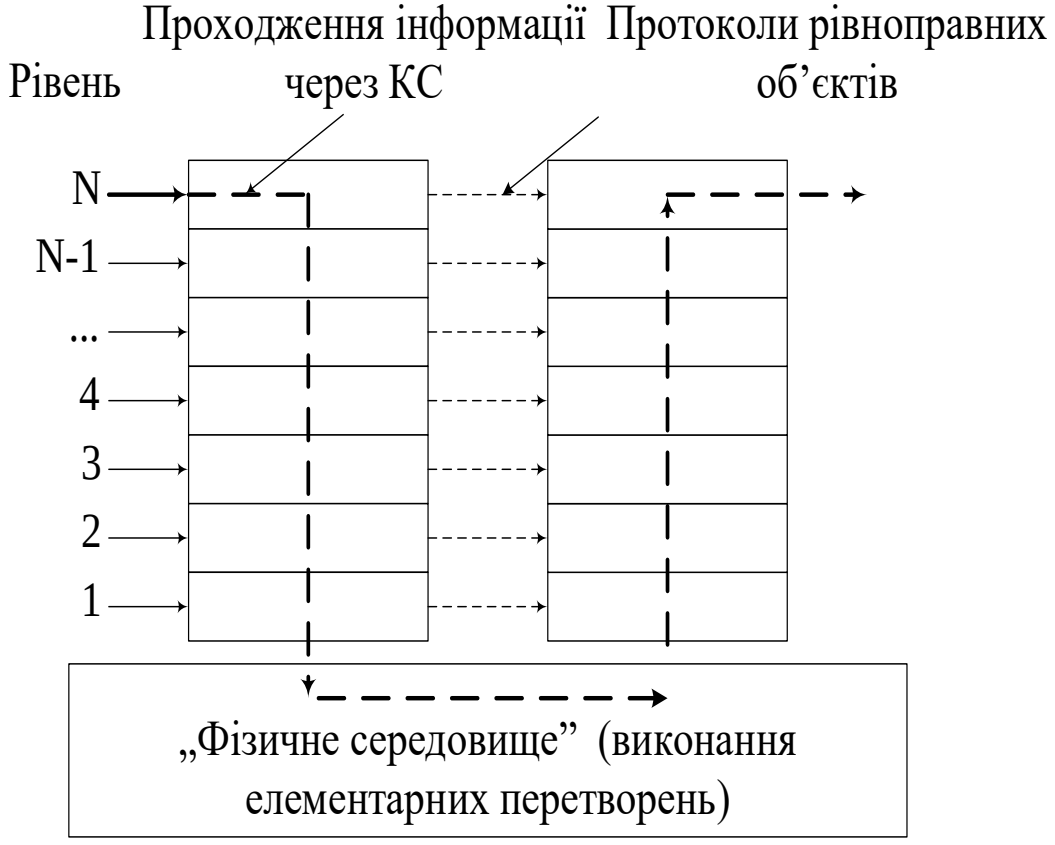


Рис. 1.6. Функціональний канал

Пристрій для формування і перевірки цифрового підпису (обчислювач цифрового підпису входить до складу шифропроцесора, який забезпечує конфіден-

ційність гарантоздатних систем (рис. 1.7). Шифропроцесор має дворівневу структуру. На верхньому рівні знаходиться протокольний універсальний процесор, який забезпечує зв'язок з зовнішнім світом і керує нижнім рівнем. На нижньому рівні знаходиться декілька спецпроцесорів, одним із них є обчислювачем цифрового підпису. Протокольний процесор і спецпроцесори реалізовані на програмованій логічній інтегральній схемі (ПЛІС). У свою чергу спецпроцесор також має аналогічну дворівневу структуру (рис. 1.8). Протокольний *RISC*-процесор, який входить до складу спецпроцесора, може бути не обов'язково універсальним. Помножувач елементів поля Галуа $GF(2^m)$ входить до складу ядра спецпроцесора (рис. 1.8). Спецпроцесор призначений для формування і перевірки цифрових підписів, але викладені нижче принципи можуть бути задіяні для проектування і інших спеціалізованих пристроїв.

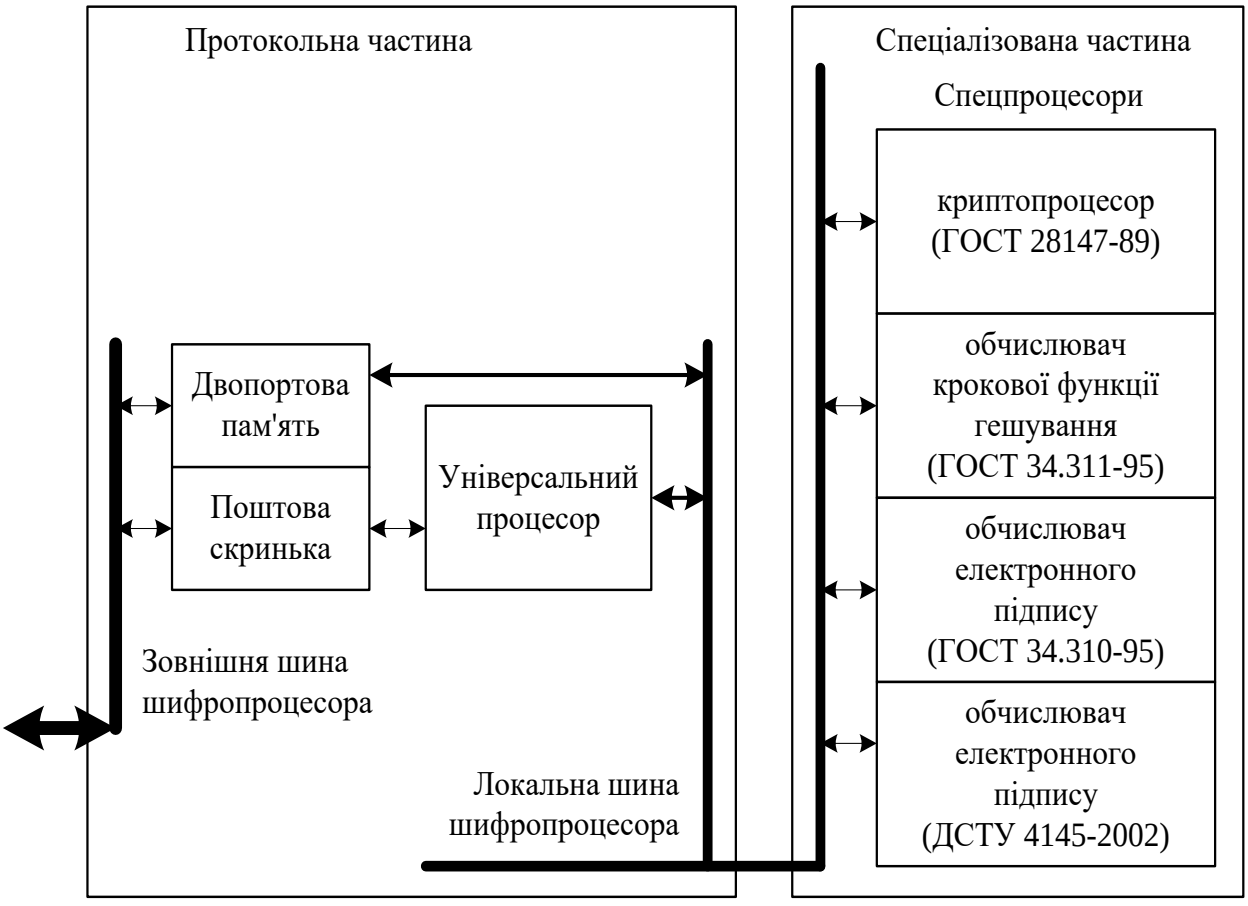


Рис. 1.7. Шифропроцесор (засіб КЗІ)

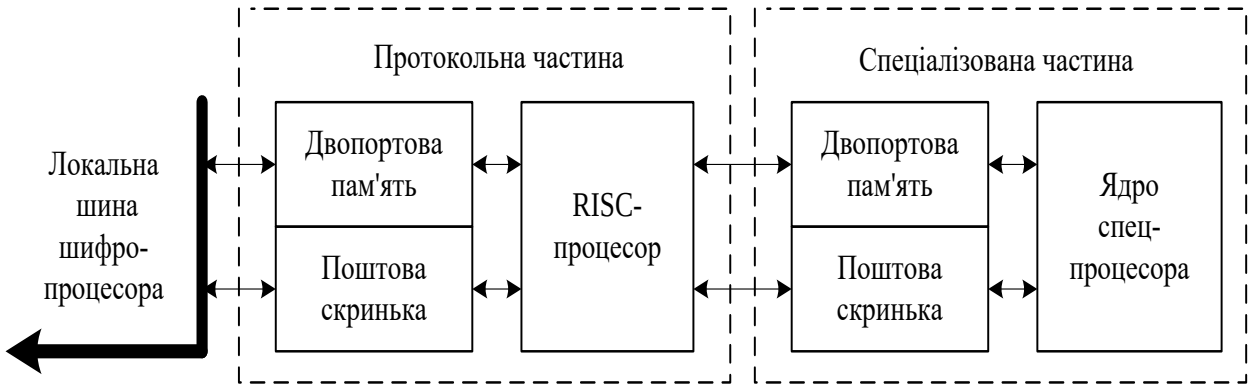


Рис. 1.8. Спецпроцесор

1.19 Методи злому систем КЗІ

Останнім часом, у зв'язку з розвитком Інтернету, стало можливим ефективно використовувати для злому систем КЗІ метод «грубої сили» (наприклад, метод перебору ключів розшифрування) шляхом розпаралелення операцій, який описаний у роботі [1]. Такий підхід, як правило, реалізується в такий спосіб. Встановлюється сервер з підключенням до Інтернету, з якого будь-хто бажаючий може завантажити програму, що розшифровує текстове повідомлення, шляхом простого перебору ключів. Програма зазвичай поставляється як у вигляді вихідних текстів, так і компільованою під найбільш поширені операційні системи. Після запуску програми, встановлюється зв'язок з сервером, отримуємо від нього набір ключів для перебору і після закінчення роботи повертаємо результат.

Метод «розподіленого злому» часто використовують некомерційні організації для з'ясування криптостійкості алгоритмів [222]. Описаний спосіб, що спирається виключно на ентузіазм учасників, вже дав результати [223].

Задача злому системи КЗІ, яка базується на використанні еліптичних кривих, полягає у логарифмуванні, тобто операції, яка використовує множення, для знаходження результату. У розширених полях Галуа $GF(p^n)$ ця операція виконується методом перебору. Тобто для злому криптографічного шифру на основі еліптичних кривих потрібно виконати дуже багато операцій над елементами розширених полів Галуа $GF(p^n)$. А це можливо зробити, тільки за участі великої кількості обчислювальних пристроїв. Тобто, такі операції потрібно виконувати у

мережі [233] ПК. У цій роботі оцінюються апаратні витрати на виконання найбільш складної із згаданих операцій – множення елементів розширених полів Галуа $GF(p^n)$ з метою визначення поля, у якому співвідношення витрат на множення програмним способом, під час злому системи КЗІ, до витрат на множення апаратними засобами, під час захисту інформації найбільше. Вважається, що використання саме такого поля у засобах КЗІ найбільше підсилює криптографічну стійкість алгоритмів, які реалізують ці засоби КЗІ.

1.20 Криптографічна стійкість

Криптографічна стійкість – це міра здатності криптографічної системи чи алгоритму протистояти спробам несанкціонованого декодування чи злому. Криптографічна стійкість залежить від ряду факторів, які включають складність алгоритму, довжину ключа, секретність реалізації та вразливості до конкретних типів атак [118].

Основні аспекти криптографічної стійкості:

1. Довжина ключа:

Ключовим параметром стійкості криптографічного алгоритму є довжина ключа. Загалом, чим більша довжина ключа, тим вища стійкість, оскільки збільшення довжини ключа експоненційно збільшує кількість можливих ключів, що ускладнює їх вгадування методом грубої сили (*brute force*). Також важливими є характеристики ключа, тому актуальною задачею є генерація криптоключів [239].

2. Складність алгоритму:

Алгоритми, які використовують складні математичні операції, можуть бути більш стійкими до певних типів атак, наприклад, криптоаналізу. Однак, важливо, щоб алгоритми були не тільки складними, а й достатньо добре дослідженими та перевіреними.

3. Атаки з вибраним відкритим текстом (*chosen-plaintext attacks*) і атаки з вибраним шифротекстом (*chosen-ciphertext attacks*):

Стійкість до цих атак є критичною для багатьох сучасних криптосистем. Криптосистема [240] вважається стійкою, якщо вона може витримати спроби декодування навіть коли атакуючий може генерувати шифротексти на основі відомих або вибраних відкритих текстів.

4. Квантова стійкість:

З розвитком квантових комп'ютерів з'явилася потреба в алгоритмах, які можуть протистояти квантовому декодуванню. Алгоритми, стійкі до квантових атак, називаються квантово-стійкими і вони стають все більш важливими у сфері криптографічної безпеки.

5. Виконання стандартів та сертифікації:

Дотримання визнаних міжнародних стандартів, таких як серія стандартів ISO/IEC 27000, NIST та інших, також важливо для забезпечення криптографічної стійкості.

1.21 Тестування операційних елементів для розширених полів Галуа $GF(p^n)$

Під час тестування операційних елементів, розроблених на ПЛІС, для елементів розширених полів Галуа $GF(p^n)$, потрібно виконувати тестування як апаратної, так і програмної частини.

Тестування апаратної частини ПЛІС може бути наступним [219]:

1. Функціональне тестування блоків. Тестування окремих блоків та компонентів на коректність реалізації їх функцій.

2. Логічне тестування. Перевірка логічної правильності проекту та відсутність помилок в схемі.

3. Тестування векторами входу. Застосування тестових векторів для стимулювання входів і перевірки відповідей.

4. Тестування сумісності. Випробування взаємодії між різними компонентами і блоками на ПЛІС.

5. Аналіз апаратних ресурсів. Вимірювання та оцінка використання апаратних ресурсів, таких як *LUT* (*Look-Up Tables*), регістри, *BRAM* (*Block RAM*), *DSP* (*Digital Signal Processing*) ресурси тощо.

6. Тестування в реальному часі. Випробування апаратної частини в реальному часі для визначення продуктивності і затримок.

7. Тестування відмовостійкості. Проведення тестів для виявлення відмов чи помилок і реакції системи на них.

Тестування програмної частини ПЛІС може бути наступним [220]:

1. Верифікація програмної логіки. Тестування програмного коду, який виконується на ПЛІС, зокрема *VHDL*-код.

2. Тестування взаємодії з ПЛІС. Випробування програмного забезпечення, яке взаємодіє з апаратною частиною ПЛІС через зовнішні інтерфейси (наприклад, з використанням *UART, SPI, I2C*).

3. Тестування прошивки. Верифікація правильності прошивки та її взаємодії з апаратною частиною.

4. Тестування алгоритмів. Тестування алгоритмів, які виконуються на ПЛІС, з використанням вхідних та вихідних даних.

5. Тестування сумісності інтерфейсів. Випробування сумісності між програмною та апаратною частинами через різні інтерфейси.

Для тестування арифметичних операцій у розширених полях Галуа $GF(p^n)$ часто використовують математичні пакети або бібліотеки, які дозволяють виконувати операції над елементами поля та перевіряти їхню правильність. Деякі з популярних математичних пакетів та бібліотек, які можна використовувати для цієї мети наведені у табл. 1.1.

Найбільше можливостей для роботи з великими розширеними полями Галуа $GF(p^n)$ надає пакет *Maple*, проте він забезпечує проведення обчислень тільки у поліноміальному базисі.

На рис. 1.9 наведено структурну схему процесу тестування [218]. Тестування починається із генерування тестових послідовностей, які подаються на об’єкт тестування та еталон. Далі результат подається на вузол порівняння. При невідповідності результатів робимо висновок, що тестований об’єкт працює неправильно.

Таблиця 1.1

Математичні пакети та бібліотеки для виконання арифметичних операцій у розширених полях Галуа $GF(p^n)$

Назва пакета (розширення пакету)	Фірма	Можливість роботи у $GF(p^n)$
Mathcad 15.0	*Parametric Technology Corporation*	$p = 2$, $n \leq 600$, поліноміальний базис
Mathlab R2022b (Communications Toolbox)	*The MathWorks, Inc.*	p – просте число, $n \leq 16$, поліноміальний базис
Mathematica 13	*Wolfram Research, Inc.*	p – просте число, n – обмежено можливостями комп'ютера і часом (перевірено до n=500), поліноміальний базис
Maple 18	*Waterloo Maple Inc.*	p – просте число, n – обмежено тільки часом (перевірено до n=4000), поліноміальний базис
GaloisCPP	*Open source*	p – просте число, n – обмежено тільки можливостями комп'ютера (перевірено до n=998), поліноміальний базис
Galois++	*Open source*	p – просте число, n – обмежено тільки можливостями комп'ютера (перевірено до n=998), поліноміальний базис
liboa	*Open source*	p – просте число, n – обмежено тільки можливостями комп'ютера (перевірено до n=998), поліноміальний базис

На рис. 1.10 наведено класифікацію еталонів [218]. Вони поділяються на модель та реальний взірець. Модель поділяється на програмну та "на папері". Програмна на функціональну та табличну. Реальний взірець може бути:

1) повним еквівалентом об'єкту, що тестується;

2) спрощеним об'єктом;

3) ускладненим об'єктом;

4) несправним об'єктом.

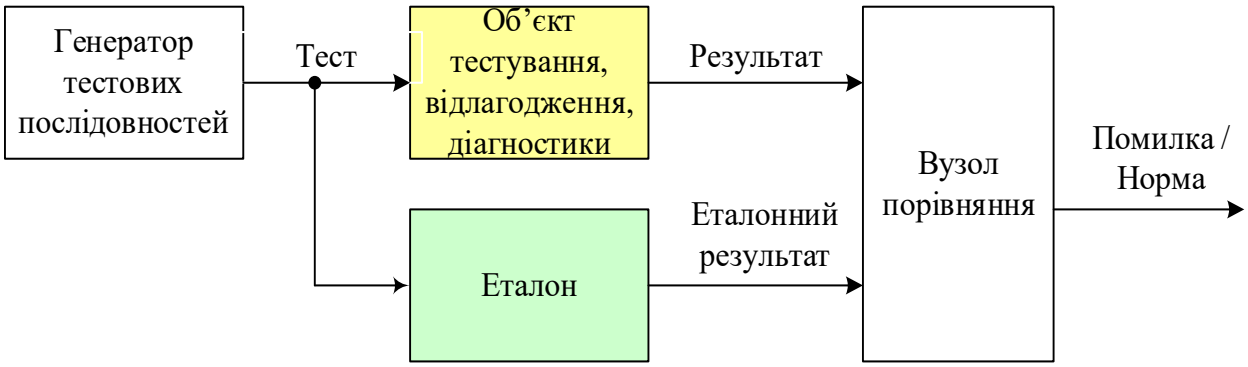


Рис. 1.9 Структурна схема процесу тестування

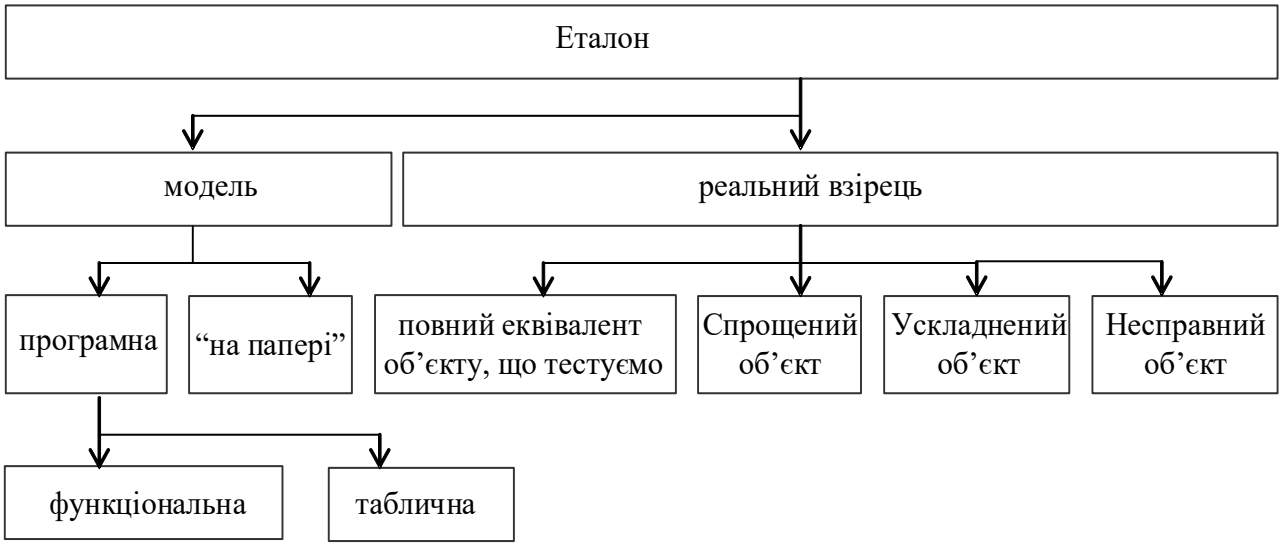


Рис. 1.10 Класифікація еталонів

1.22 Висновки до розділу 1

У першому розділі був проведений аналіз сучасного стану теорії, методів та засобів, що використовуються у проектуванні пристроїв для захисту інформації. Також був розглянутий набір найбільш важливих відкритих стандартів і алгоритмів.

У розділі наводиться визначення поняття “кіберфізичні системи” та місце КЗІ у кіберфізичних системах. Наводяться методи забезпечення інформаційної безпеки. Дано визначення поняттю ЕК та її місце у засобах КЗІ. Наведено основні операції, які виконуються над елементами розширених полів Галуа $GF(p^n)$, описано алгоритми їх виконання та визначено найбільш трудомісткі операції – операції множення та ділення. Оскільки операція ділення – це множення на обернений елемент, то вона зводиться до множення. Наведено методи виконання

операцій множення у розширених полях Галуа $GF(p^n)$. Описано комірку Гілда та модифіковану комірку Гілда, наведено їх відмінності. Дано визначення поняття “генератори ядер” та описано методи та засоби їх побудови. Наведено методи злому систем КЗІ. Описано основи проектування спеціалізованих комп’ютерних засобів. Наведена елементна база для створення апаратно-програмних та апаратних вузлів КЗІ. Наведено основні методи генерації описів функціональних вузлів. Наведено детальний опис виконання операцій додавання, віднімання та множення у розширених полях Галуа $GF(p^n)$.

Проведений аналіз дозволив визначити необхідність створення найбільш складних з вузлів КЗІ – помножувачів елементів розширених полів Галуа $GF(p^n)$ і основні задачі, які потрібно вирішити для розв’язання цієї актуальної науково-прикладної задачі:

1) визначити основні принципи помножувачів елементів розширених полів Галуа $GF(p^n)$, їхню структуровану модель та структурні алгоритми їх роботи;

2) розробити засоби для проектування помножувачів елементів розширених полів Галуа $GF(p^n)$;

3) провести експериментальне дослідження та впровадження розроблених вузлів криптографічного захисту інформації.

РОЗДІЛ 2

МЕТОДИ СТВОРЕННЯ РЕКОНФІГУРОВАНИХ ВУЗЛІВ КРИПТОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ ДЛЯ КІБЕРФІЗИЧНИХ СИСТЕМ

2.1 Загальна методика проведення дисертаційних досліджень

Розділ описує підходи та вимоги до проектування вузлів КЗІ на основі розширених полів Галуа $GF(p^n)$. Обґрунтовано доцільність та наведено особливості створення паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ з малими та великими характеристиками p полів. Запропоновано структурну схему МКГ для полів з характеристиками $p > 2$. Розроблено метод створення паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ на основі 3 структур МКГ та проведено порівняння апаратних витрат помножувачів з цими структурами:

1) МКГ як “чорна скринька” (ЧС). МКГ є повністю цілісним елементом, в якому несуттєвою є внутрішня структура комірки, а до уваги береться тільки кількість її входів та виходів;

2) МКГ на основі функціональних вузлів: помножувача та суматор (ФВ), з уточненням внутрішньої структури;

3) МКГ на основі логічних вузлів (ЛВ), які утворюють помножувач та суматор.

Також розроблено метод оцінювання часової складності та метод оцінювання апаратної складності помножувачів елементів розширених полів Галуа $GF(p^n)$, реалізованих на основі МКГ. Також розроблено метод тестування цих помножувачів.

2.2 Кіберфізичні системи у складі вузлів криптографічного захисту інформації

Захист інформації у КФС є критично важливим, оскільки ці системи поєднують фізичні процеси з обчислювальними елементами та мережею зв'язку. За-

безпечення безпеки таких систем вимагає комплексного підходу, що включає як технічні, так і організаційні заходи.

На рис. 2.1 наведено багаторівневу структуру КФС. Ця схема є уточненням відомої схеми [191]. Оскільки, однією з основних рис КФС є бездротовий зв'язок, то при передачі інформації у КФС актуальною стає задача захисту цієї інформації, особливо з постійною появою нових загроз (квантових комп’ютерів). Велика кількість вузлів КФС не мають вузлів захисту інфомації, що показано на рис. 1.1. На рис. 2.1 показані місця, де повинні бути розміщені засоби КЗІ. Здебільшого це місція міжшарового обміну інформацією.

Рис. 2.1 Багаторівнева платформа кіберфізичної системи

2.3 Підходи до проектування вузлів КЗІ на основі розширених полів Галуа $GF(p^n)$

Аналіз, представлений у розділі 1, демонструє, що ефективне вирішення значущої та сучасної науково-практичної задачі підвищення криптографічної стійкості засобів КЗІ, які використовуються в складі КФС, шляхом розвитку методів та засобів створення реконфігурованих операційних пристроїв для роботи з елементами розширених полів Галуа $GF(p^n)$ з характеристиками p та з порядками n утворюючого поле полінома такими, що $p^n \approx 2^m$, де $p > 2$, $m < 1024$ вимагає інтеграції різноманітних наукових дисциплін.

При проектуванні апаратних помножувачів для елементів розширених полів Галуа $GF(p^n)$ враховувалися висновки теорії комп'ютерних систем, теорії обчислювальних систем, теорії обчислювальних машин, теорії проектування спеціалізованих комп'ютерних систем, теорії складності алгоритмів та програмно-апаратної складності комп'ютерних систем. Для реалізації елементів вузлів апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС використовувалась теорія проектування НВІС. Для розробки методів обробки елементів розширених полів Галуа $GF(p^n)$ враховувалися положення і висновки теорії інформації, теорії чисел, теорії залишків, теорії обчислень, теорії груп, для проектування спецпроцесорів, а також для вирішення задач проектування апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ застосовувалися результати теорії кодування, для створення моделей вузлів апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ та для аналізу їх роботи була використана теорія програмування, теорія моделей, обчислювальна математика, моделювання алгоритмів та апаратних засобів.

Отримані результати були перевірені шляхом моделювання згідно з теорією випробувань.

Дослідження, що були проведені, базуються на результатах теорії цифрових автоматів, на теоретичній моделі взаємодії відкритих систем та багаторівневій платформі КФС. Також були використані методи виконання математичних

операцій у розширених полях Галуа $GF(p^n)$ у поліноміальному базисі. У проведених дослідженнях використовуються математичні напрацювання теорії чисел, теорії алгоритмів та засобів моделювання цифрових схем. При проектуванні апаратних помножувачів для елементів розширених полів Галуа $GF(p^n)$ враховувалися висновки теорії комп'ютерних систем, теорії обчислювальних систем, теорії обчислювальних машин, теорії проектування спеціалізованих комп'ютерних систем, теорії складності алгоритмів та програмно-апаратної складності комп'ютерних систем. Для реалізації елементів вузлів апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС використовувалась теорія проектування НВІС. Для розробки методів обробки елементів розширених полів Галуа $GF(p^n)$ враховувалися положення і висновки теорії інформації, теорії чисел, теорії залишків, теорії обчислень, теорії груп, для проектування спецпроцесорів, а також для вирішення задач проектування апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ застосовувалися результати теорії кодування, для створення моделей вузлів апаратних помножувачів елементів розширених полів Галуа $GF(p^n)$ та для аналізу їх роботи була використана теорія програмування, теорія моделей, обчислювальна математика, моделювання алгоритмів та апаратних засобів.

Отримані результати були перевірені шляхом моделювання згідно з теорією випробувань.

Дослідження, що були проведені, базуються на результатах теорії цифрових автоматів, на теоретичній моделі взаємодії відкритих систем та багаторівневій платформі КФС. Також були використані методи виконання математичних операцій у розширених полях Галуа $GF(p^n)$ у поліноміальному базисі. У проведених дослідженнях використовуються математичні напрацювання теорії чисел, теорії алгоритмів та засобів моделювання цифрових схем.

Для досягнення поставленої мети слід вирішити наступні задачі:

1. Провести комплексний аналіз сучасного стану теорії, методів та інструментів проектування апаратних засобів КЗІ.

2. Дослідити методи виконання арифметичних операцій у розширених полях Галуа $GF(p^n)$, особливо методи виконання множення у цих полях.

3. Дати визначення поняття МКГ та порівняти її із КГ.

4. Обґрунтувати доцільність та особливості створення паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$.

5. Вдосконалити метод оцінювання часової складності апаратного множення елементів розширених полів Галуа $GF(p^n)$.

6. Вдосконалити метод оцінювання апаратної складності множення елементів розширених полів Галуа $GF(p^n)$.

7. Провести аналіз апаратної складності помножувачів елементів розширених полів Галуа $GF(p^n)$, $p > 2$. Визначити поля, в яких апаратна складність помножувачів найменша.

8. Провести аналіз часової складності помножувачів елементів розширених полів Галуа $GF(p^n)$, $p > 2$. Визначити поля, в яких відношенням часових витрат при програмній та апаратній реалізаціях помножувачів найбільші.

9. Розробити метод створення паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ для вузлів КЗІ на основі МКГ за трьома варіантами структури МКГ: “чорна скринька” (ЧС), на основі функціональних вузлів (ФВ), на основі логічних вузлів (ЛВ).

10. Розробити метод тестування генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$.

11. Розробити генератори ядер (генератори моделей) помножувачів елементів розширених полів Галуа $GF(p^n)$ з довільними характеристиками p і порядками n утворюючого поле полінома, такими, що порядок поля $p^n \approx 2^{1024}$, $p^n < 2^{1024}$. Генератори повинні створювати моделі помножувачів на остнові МКГ з трьома варіантами структури МКГ: ЧС, ФВ, ЛВ.

12. За допомогою генераторів ядер згенерувати ряд помножувачів елементів розширених полів Галуа $GF(p^n)$ з трьома структурами МКГ.

13. Провести імплементацію помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС *Spartan* 6, *Cyclone* V та *Virtex* UltraScale та порівняти результати реалізації.

14. Показати збіжність теоретичних та практичних результатів оцінювання апаратної та часової складностей помножувачів.

15. Провести впровадження розроблених вузлів КЗІ.

2.4 Реалізація вузлів КЗІ на основі розширених полів Галуа $GF(p^n)$

Пропонується використовувати стандартну модель взаємодії відкритих систем, як основу для створення елементів криптографічного захисту інформації, заснованих на ЕК, а також розглядаються інструменти для їх застосування.

Адаптація цієї уніфікованої моделі для криптографічних вузлів, що використовують еліптичні криві, включає їх інтеграцію на початкових рівнях стандартної моделі взаємодії відкритих систем. Далі йде розробка та аналіз функціональних та структурних моделей для цих вузлів, що буде детально розглянуто у наступних частинах даної роботи.

Математичною основою для сучасних криптографічних систем, нарівні з іншими математичними дисциплінами, слугують поля Галуа та теорія ЕК.

Серед операцій у розширених полях Галуа $GF(p^n)$, які вимагають значного обчислювального часу – знаходження обернених елементів та множення. Ці операції застосовуються для виконання дій над точками ЕК (додавання, подвоєння, множення на число). У даній дисертаційній роботі розроблено структурні алгоритми для операцій у розширених полях Галуа $GF(p^n)$ з великими значеннями $p^n < 2^{1024}$ [6], [7], [12], [13] і наведено рекомендації щодо вибору оптимальних розширених полів Галуа $GF(p^n)$ у поліноміальному базисі. Також виконано імплементацію цих алгоритмів у вигляді операційних пристроїв для використання в засобах КЗІ.

2.5 Обґрунтування доцільності створення паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$

Схеми паралельних помножувачів дуже ілюстративні. Вони добре ілюст-

рують всі особливості виконання множення у розширених полях Галуа $GF(p^n)$ Тому саме ці помножувачі було обрано для аналізу часової та апаратної складностей. Було розроблено методи оцінювання складностей і ці методи тепер можна адаптувати під інші типи помножувачів. Відповідно, результати аналізів можуть відрізнятись від отриманих у цій роботи.

2.6 Паралельні помножувачі елементів розширених полів Галуа $GF(p^n)$ на основі МКГ

На рис. 2.2 наведений відомий паралельний (матричний) помножувач для множення цілих чисел без знаку, де *SMn* – комірка Гілда.

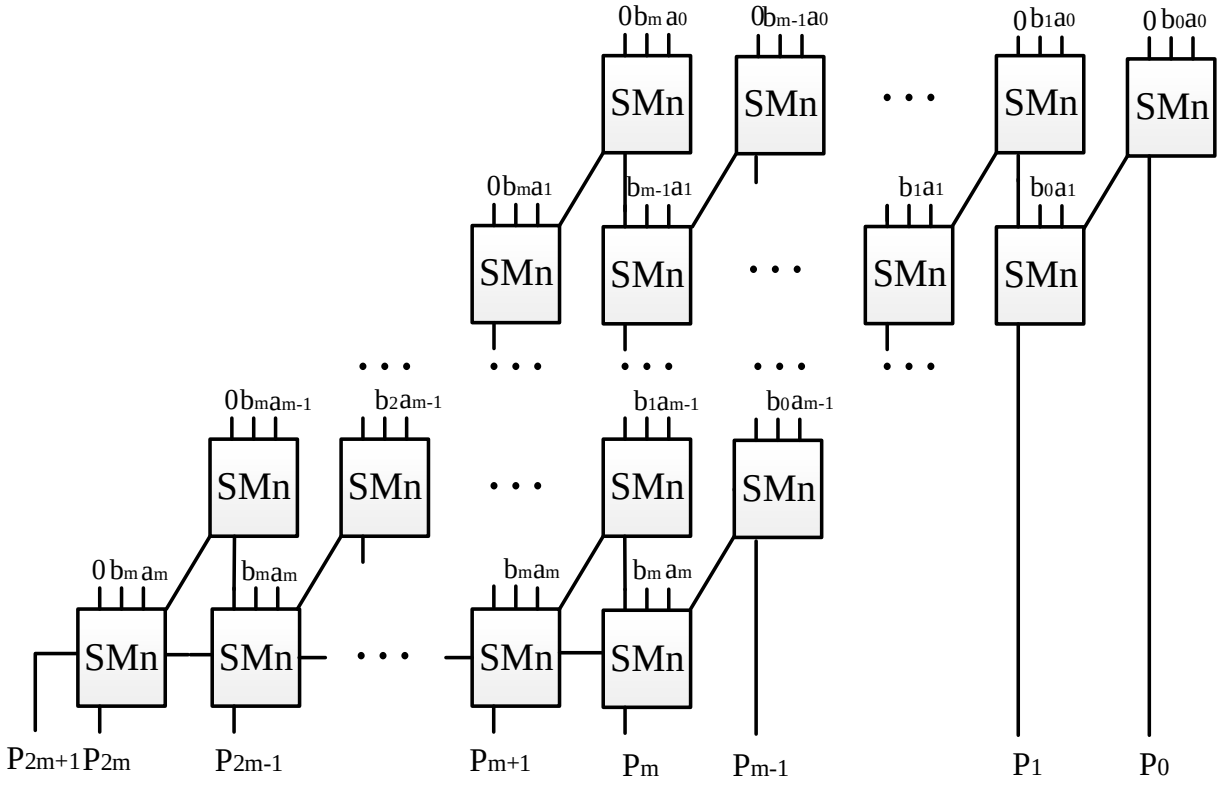


Рис. 2.2 Схема матричного помножувача на основі комірок Гілда

Матричний помножувач для елементів розширених полів Галуа $GF(p^n)$ в поліноміальному базисі будується на основі схеми рис. 2.2. Схему такого помножувача для полів $GF(3^n)$ наведено на рис. 2.3 та загальний випадок для полів $GF(p^n)$ наведено на рис. 2.4. Вони складаються з МКГ (G_n) та вузлів *f*.

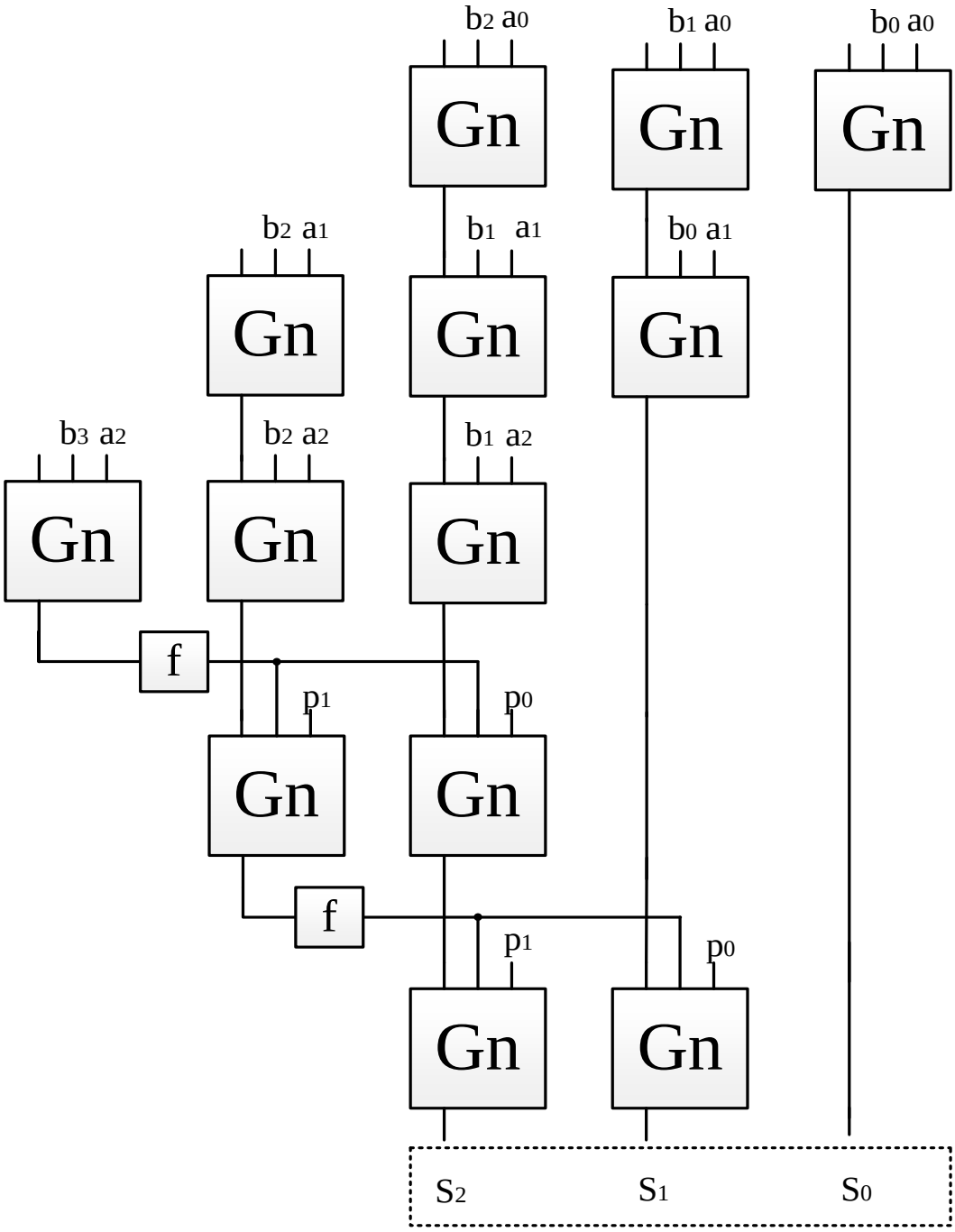


Рис. 2.3 Матричний помножувач для поля $GF(3^n)$ з використанням МКГ

Вузол *f*, що використовується для знаходження коефіцієнта, на який перемножується утворюючий поле поліном, при зведенні проміжного результату за модулем такого полінома. Вузол *f* обчислює інверсію за модулем характеристики поля: $f = (p - G_n) \bmod p = (-G_n) \bmod p$, де p – характеристика поля, G_n – результат на виході модифікованої комірки Гілда. Коефіцієнт при старшому розряді незвідного полінома завжди рівний 1 [96].

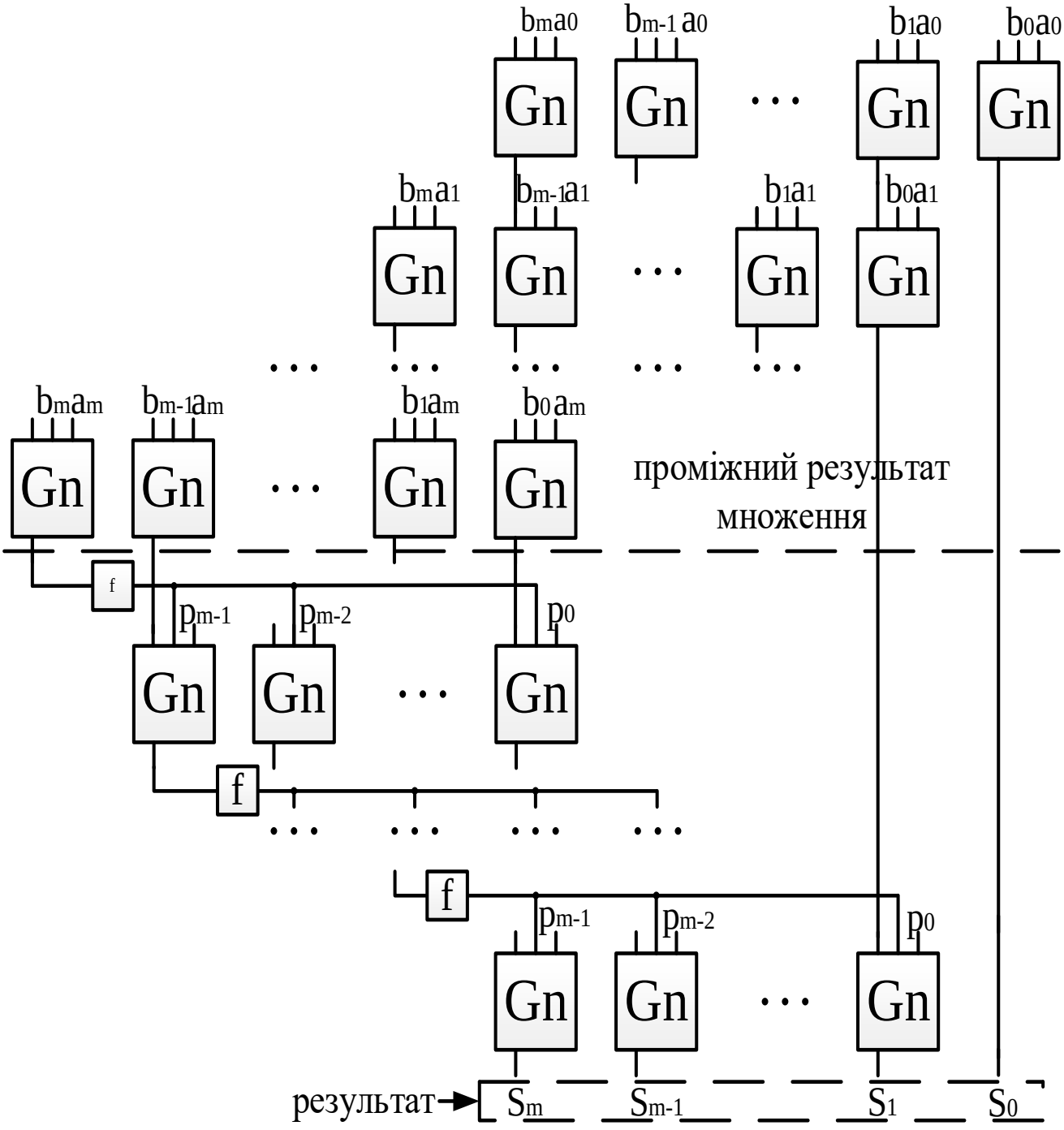


Рис. 2.4 Матричний помножувач для поля $GF(p^n)$ з використанням модифікованих комірок Гілда

2.7 Особливості створення паралельних помножувачів елементів розширених полів Галуа *GF(p^n)* на основі МКГ ЧС та ФВ

Помножувач для розширених полів Галуа $GF(p^n)$ може бути реалізований

на основі МКГ. МКГ для полів Галуа $GF(p^n)$ повинні мати *3x* двійкових входів та *x* виходів, $x = \lceil log_2 p \rceil$ (рис. 1.2). При використанні сучасних ПЛІС, логічні комірки яких побудовано на основі програмованих 6-входових комбінаційних схем (*LUT*), реалізація на ПЛІС МКГ, коли не уточняється структура МКГ, а береться до уваги тільки кількість її входів та виходів, потребує $q_{LUT} = (2^{3x-5}\boldsymbol{-1)} \cdot \boldsymbol{x}$ *LUT (LUT* з 6 входами).

Апаратні витрати будемо оцінювати у порівнянні із витратами помножувача для розширеного поля Галуа $GF(2^m)$, при умові $p^n \approx 2^m$ При цьому враховується різниця у величинах порядків досліджуваних полів.

Для варіанту ЧС коефіцієнт апаратних витрат $k_{mul} = k_g k_k$, де $k_g = \frac{k_{gp}}{k_{g2}}$, $k_k = \frac{k_{kp}}{k_{k2}}$ – коефіцієнти складності та кількості МКГ, k_{gp} та k_{g2}, k_{kp} та k_{k2} – кількість *LUT* у МКГ та кількість МКГ для полів Галуа $GF(p^n)$ та $GF(2^m)$, відповідно.

Для розширених полів Галуа $GF(2^m)$ $k_{g2} = 1$, для інших $k_{gp} = (2^{3x-5} - 1)x$, де $x = \lceil \log_2 p \rceil$. Звідси випливає що $k_{gp} = (2^{3\lceil \log_2 p \rceil - 5} - 1)\lceil \log_2 p \rceil$. Отже:

$$k_{gp} = (2^{3\lceil \log_2 p \rceil - 5} - 1)\lceil \log_2 p \rceil \quad (2.1)$$

В розширених полях Галуа $GF(2^m)$ для реалізації помножувача потрібно $k_{k2}=2m^2 - 2m + 1$ МКГ, а в полях Галуа $GF(p^n)$ – $k_{kx}=2n^2 - 2n + 1$ та додатково $(n-1) * (2^{3*\lceil \log_2 p \rceil - 5} - 1) * \lceil \log_2 p \rceil$ *LUT* для знаходження коефіцієнта, на який перемножуємо незвідний поліном. Апаратними витратами для знаходження коефіцієнта (на реалізацію елемента *f*) (рис. 2.4), можна, в даному випадку, знехтувати, оскільки вони малі в порівнянні з витратами на реалізацію самих модифікованих комірок Гілда. Отже:

$$k_k \approx \frac{2n^2-2n+1}{2m^2-2m+1} \quad (2.2)$$

$$k_{mul} \approx \frac{(2^{3\lceil \log_2 p \rceil - 5}-1)\lceil \log_2 p \rceil (2n^2-2n+1)}{2m^2-2m+1} \quad (2.3)$$

При цьому $p^n \approx 2^m$. Тоді $n \approx log_p 2^m = \frac{m}{log_2 p}$, $k_k \approx \frac{\frac{2m^2}{(log_2 p)^2} - \frac{2m}{log_2 p} + 1}{2m^2-2m+1} \approx$

$log_2^{-1} p$,

$$k_{mul} \approx \frac{(2^{3\lceil \log_2 p \rceil - 5} - 1)(\log_2 p)}{log_2 p} \approx 2^{3\lceil \log_2 p \rceil - 5}. \quad (2.4)$$

Для отримання більш точних значень k_{mul} потрібно проводити обчислення за формулою 2.3. Початкову ділянку графіка функції k_{mul} наведено на рис. 2.6, для проміжку p від 1 до 10, де суцільною лінією позначено відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$, а пунктирною – їх приблизна оцінка. З рис. 2.5 можемо бачити, що при збільшенні характеристики поля p апаратні витрати стрімко зростають. Рис 2.5 наводить показники апаратної складності для проміжку p від 1 до 342.

Найменші апаратні витрати, при реалізації МКГ ЧС для розширених полів Галуа $GF(p^n)$ будуть мати помножувачі для полів Галуа $GF(3^n)$, що видно із рис. 2.7.

Для реалізації МКГ ЧС потрібно $q_{ЧС} = (2^{3x-5} - 1)x\ LUT$. Для варіанту ФВ, коли помножувач та суматор, які мають $2x$ входів та x виходів кожний, для реалізації однієї МКГ буде потрібно $q_{ФВ} = 2(2^{2x-5} - 1)x\ LUT$.

Тоді: $\frac{q_{\text{ЧС}}}{q_{\text{ФВ}}} = \frac{(2^{3x-5}-1)\cdot x}{2(2^{2x-5}-1)\cdot x} \approx \frac{2^{3x-5}}{2(2^{2x-5})} = 2^{x-1} = 2^{\lceil \log_2 p \rceil - 1}$ – відношення витрат для реалізації однієї МКГ ЧС та МКГ ФВ. З формули видно, що внутрішня структура МКГ суттєво впливає на апаратні витрати. У порівнянні з МКГ ЧС, коли до уваги береться тільки кількість входів і виходів, додаткове врахування внутрішньої структури МКГ зменшує значення апаратної складності МКГ.

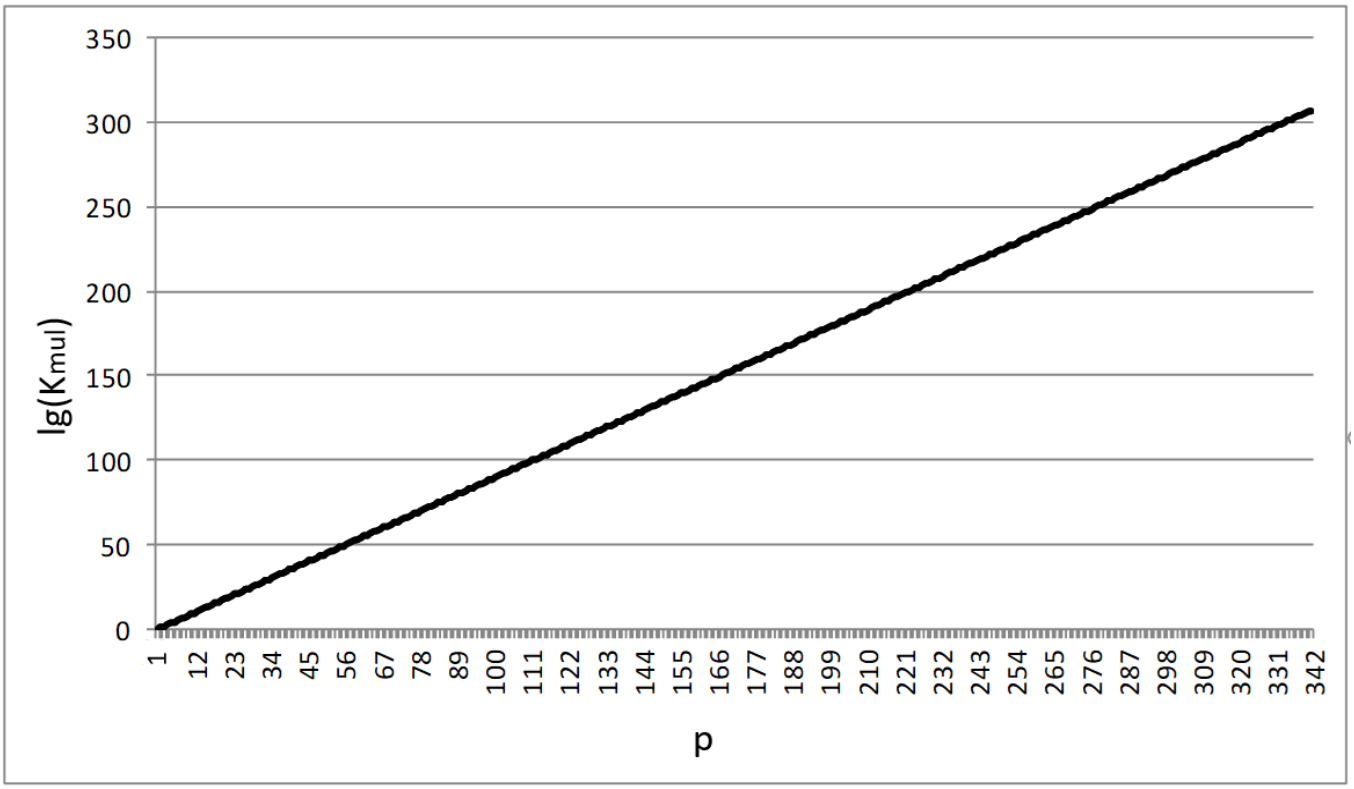


Рис. 2.5 Відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$ для структури МКГ, ЧС для проміжку p від 1 до 342

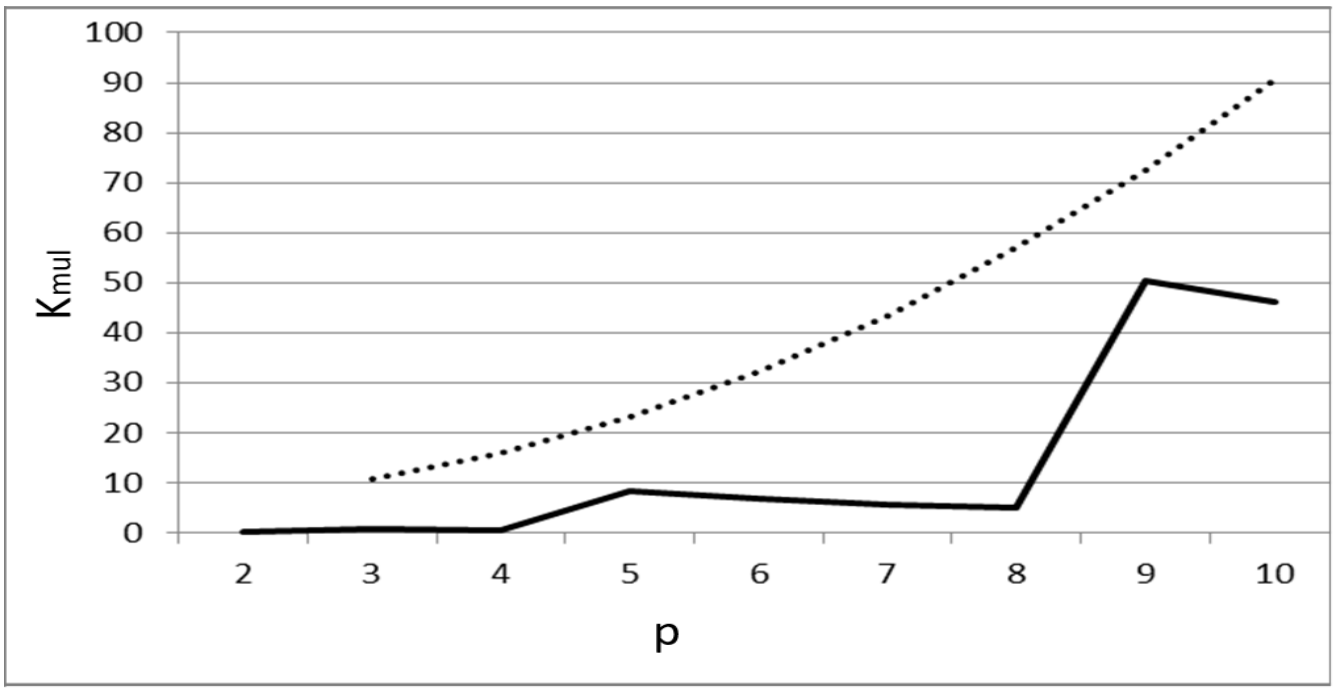


Рис. 2.6 Відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$ для структури МКГ, ЧС для проміжку p від 1 до 10

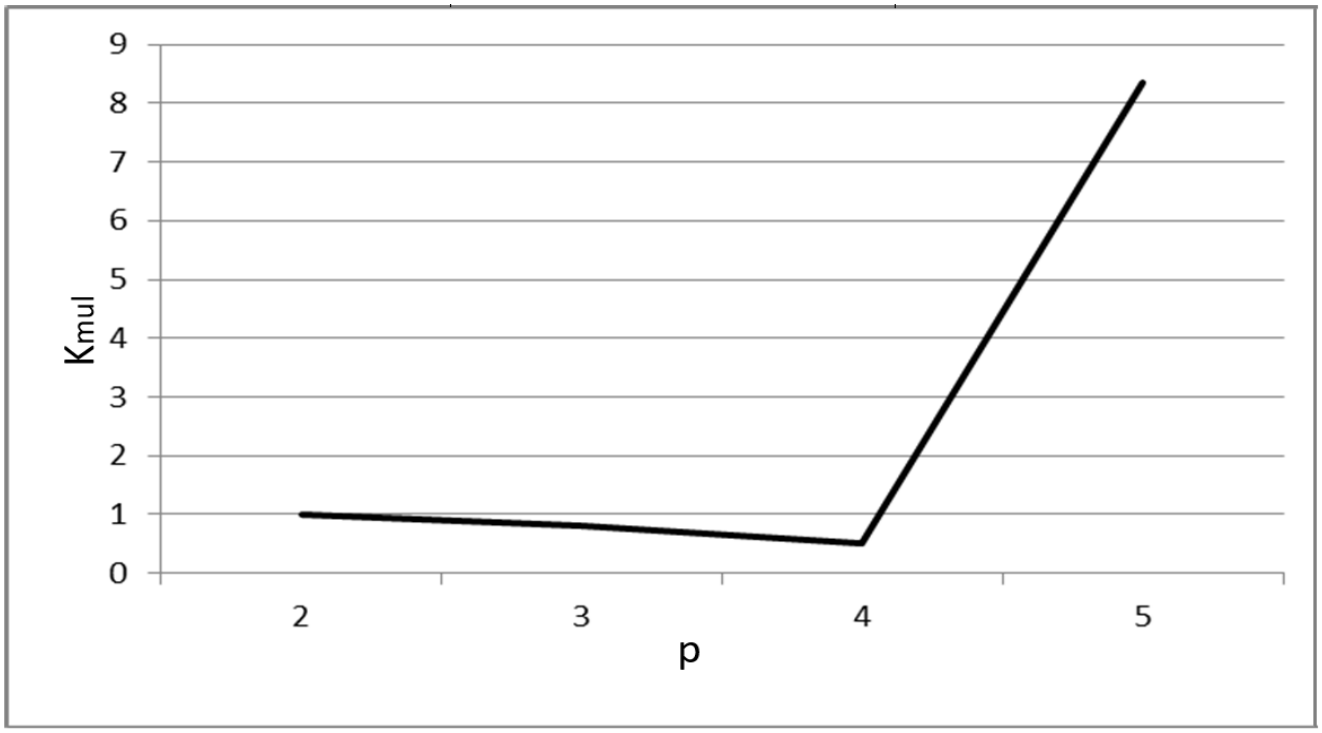


Рис. 2.7 Відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$ для структури МКГ ЧС, для проміжку p від 1 до 5

Проведемо оцінку апаратних витрат помножувача для МКГ ФВ. Для розширених полів Галуа $GF(2^m)$ $k_{g2} = 1$, для інших:

$$k_{gp} = (2^{2\lceil \log_2 p \rceil - 5} - 1)\lceil \log_2 p \rceil 2 \qquad (2.5)$$

У розширених полях Галуа $GF(2^m)$, для реалізації помножувача потрібно $2m^2 - 2m + 1$ МКГ, а в полях Галуа $GF(p^n)$ – $2n^2 - 2n + 1$ МКГ.

Отже: $k_k \approx \frac{2n^2-2n+1}{2m^2-2m+1}$.

При цьому $p^n \approx 2^m$. Тоді $n \approx log_p 2^m = \frac{m}{log_2 p}$, $k_k \approx \frac{(\frac{2m^2}{log_2^2 p} - \frac{2m}{log_2 p} + 1)}{2m^2-2m+1} \approx log_2^{-1} p$, $k_{mul} \approx \frac{(2^{2\lceil \log_2 p \rceil - 5} - 1)\lceil \log_2 p \rceil 2}{\log_2 p} \approx 2^{2\lceil \log_2 p \rceil - 4}$ (2.6)

Порівнюючи формули для знаходження k_{mul} для МКГ ЧС (2.4) та МКГ ФВ (2.6) видно, що для МКГ ФВ при великих значеннях p загалом зменшується значення апаратної складності помножувача в $2^{\lceil \log_2 p \rceil - 1}$ рази.

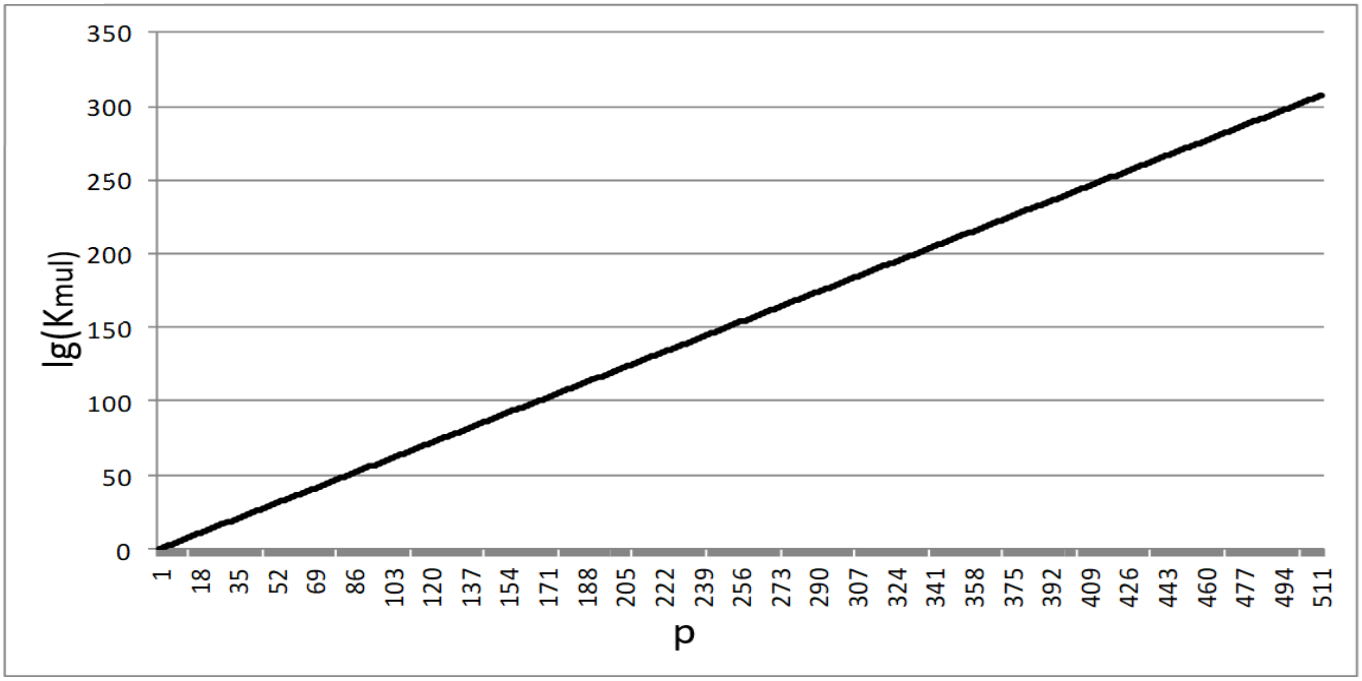


Рис. 2.8 Відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$ для структури МКГ ФВ, для проміжку p від 1 до 511

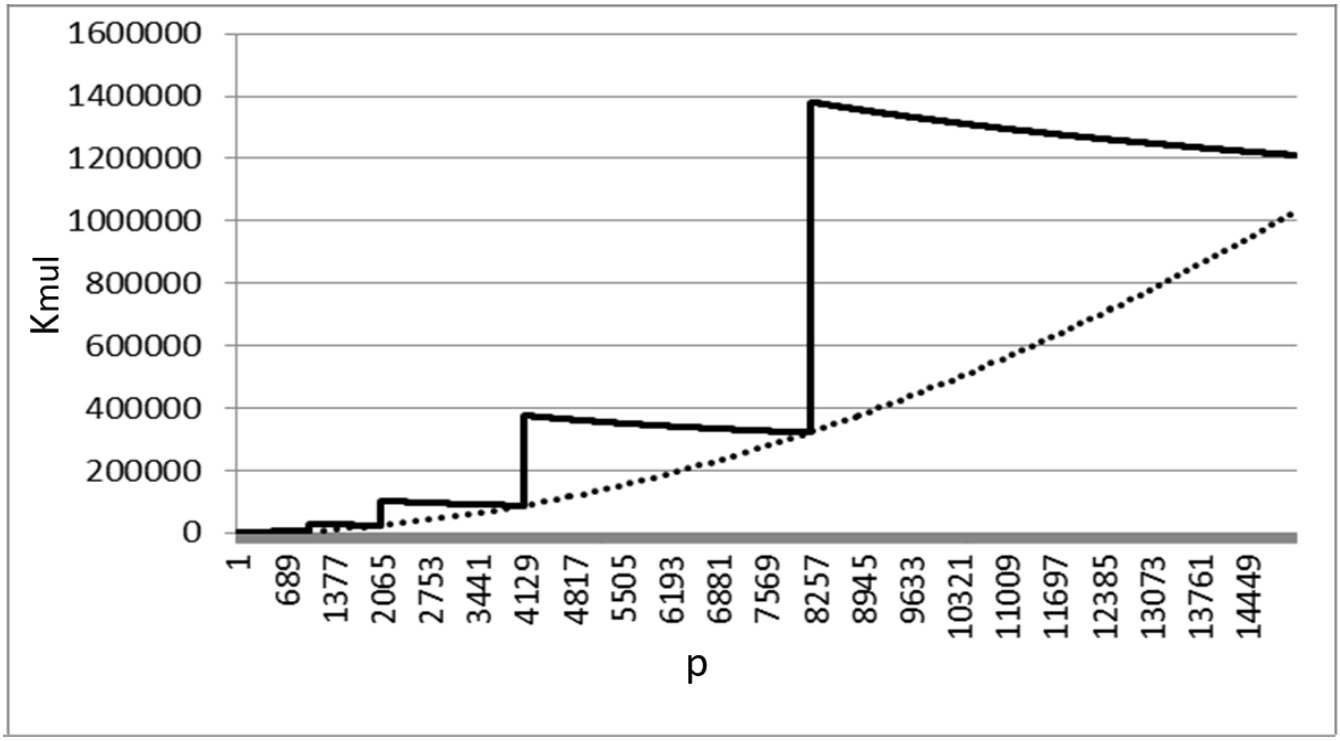


Рис. 2.9 Відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$ для структури МКГ ФВ, для проміжку p від 1 до 14449

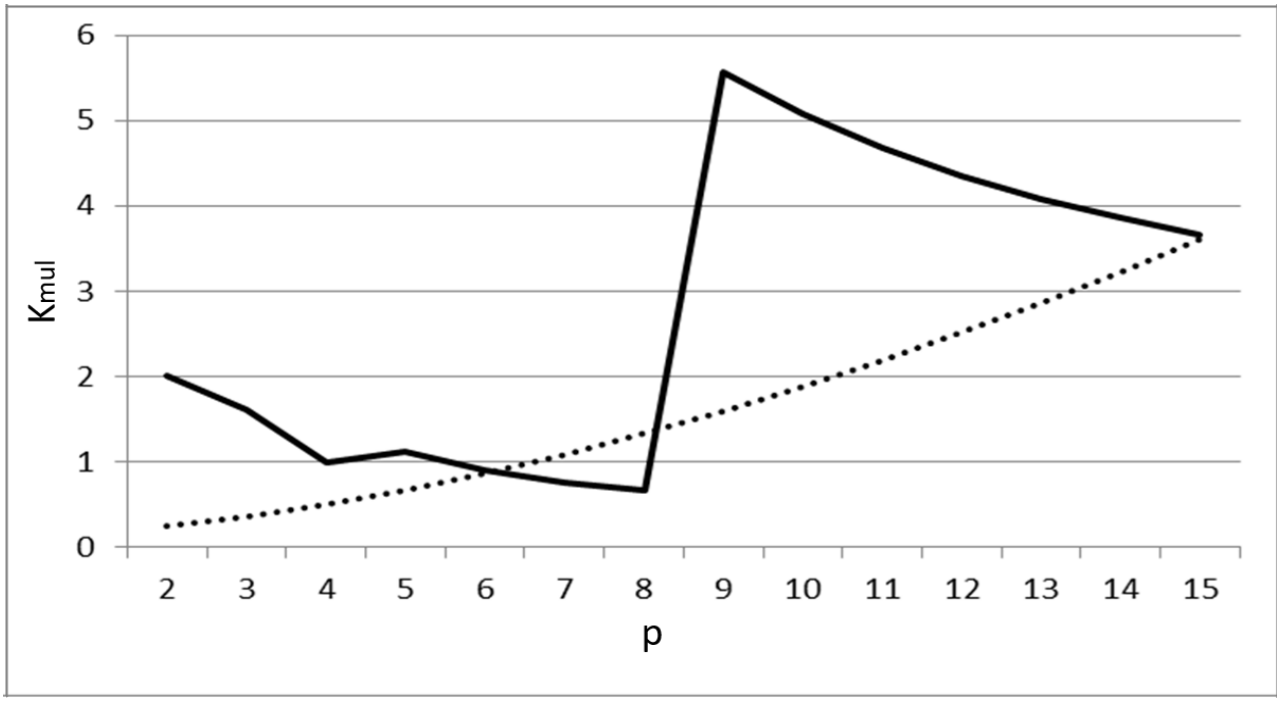


Рис. 2.10 Відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$ для структури МКГ ФВ, для проміжку p від 1 до 15

Для малих n (рис. 2.10) k_{mul} розраховувалося за точними формулами (3.5, 3.6).

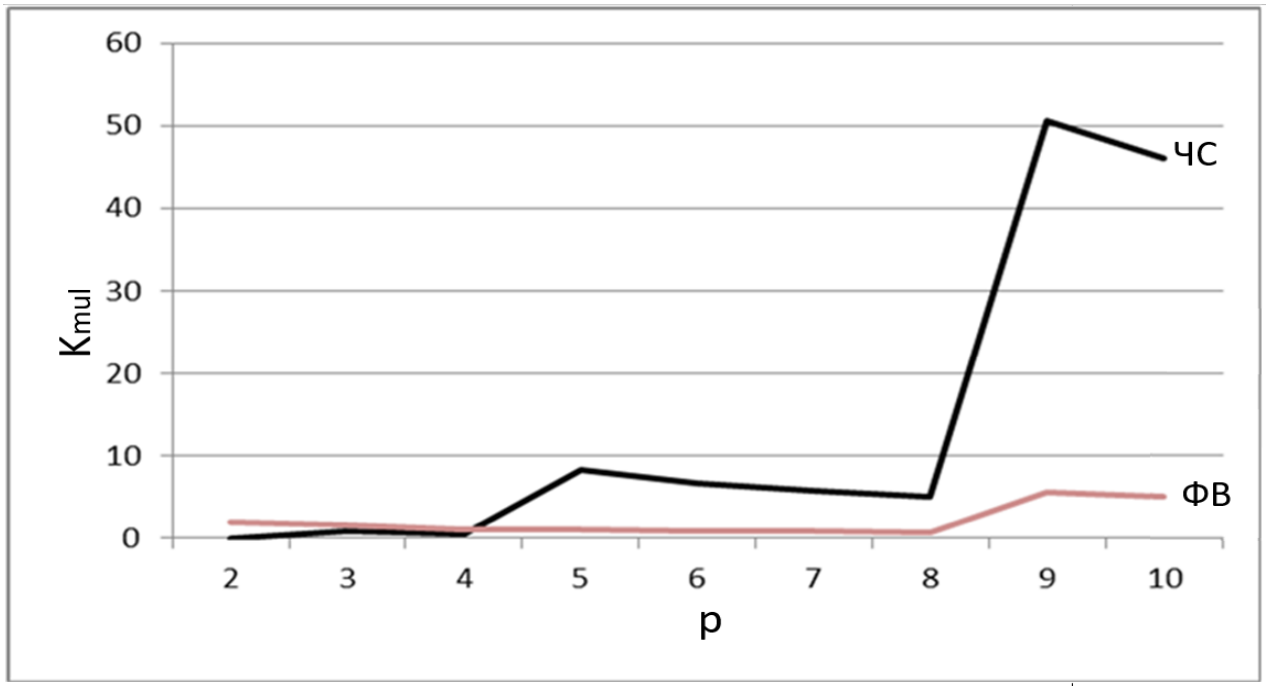


Рис. 2.11 Відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$ з структурами ЧС та ФВ

На рис. 2.8, 2.9, 2.10 показано апаратні витрати на створення помножувачів елементів розширених полів Галуа $GF(p^n)$ з різними характеристиками поля, але приблизно однаковими порядками, для структури МКГ ФВ. Початкову ділянку графіку функції k_{mul} наведено на рис. 2.10, де суцільною лінією позначено відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ та $GF(2^m)$, а пунктирною лінією – їх приблизна оцінка. З рис. 2.10 можемо бачити, що на початку графіка апаратні витрати при збільшенні характеристики поля стрімко зростають. Як видно з рис. 2.10 найменші апаратні витрати за 2-им варіантом оцінювання будуть мати розширені поля Галуа $GF(7^m)$.

На рис. 2.11 наведено відношення апаратних витрат помножувачів елементів розширених полів Галуа $GF(p^n)$ до $GF(2^m)$ із структурами ЧС та ФВ.

2.8 Особливості створення паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ на основі МКГ ЛВ

При побудові помножувачів елементів розширених полів Галуа $GF(p^n)$ з великою характеристикою поля постає потреба у створенні МКГ з великою кількістю входів. В такому випадку булеві функції стають складними. У даному випадку помножувач та суматор, що працюють за модулем p у МКГ, можна представити такими, що складаються з базових функціональних логічних вузлів (ЛВ): мультиплексорів, однорозрядних двійкових суматорів та інших.

За варіантом ЛВ помножувач можна представити як матричний помножувач з прямим та зворотнім ходом (рис. 2.13). При прямому ході обчислень виконується операція множення, а при зворотному – знаходження остачі від ділення методом без відновлення залишків. При прямому ході одну комірку *SMn* можна реалізувати на *KLUT1 = 2* елементах *LUT* (4 входи та 2 виходи), при зворотному – один елемент *SMch* на *KLUT2 = 2* елементах *LUT* (5 входів, 2 виходи), а елементи *Sn* – на *KLUT3 = 1* (3 входи, 1 вихід) та *Rn* – на *KLUT4 = 1* (2 входи, 1 вихід) елементів *LUT*. Суматор *SUM_G* модифікованої комірки Гілда (рис. 2.14) будується за допомогою ланцюжка повних однорозрядних двійкових суматорів. Два таких суматора (5 входів, 3 виходи) можна представити за допомогою 3 *LUT*, отже

KLUT5 = 3. Дані представлення наведено на рис. 2.12.

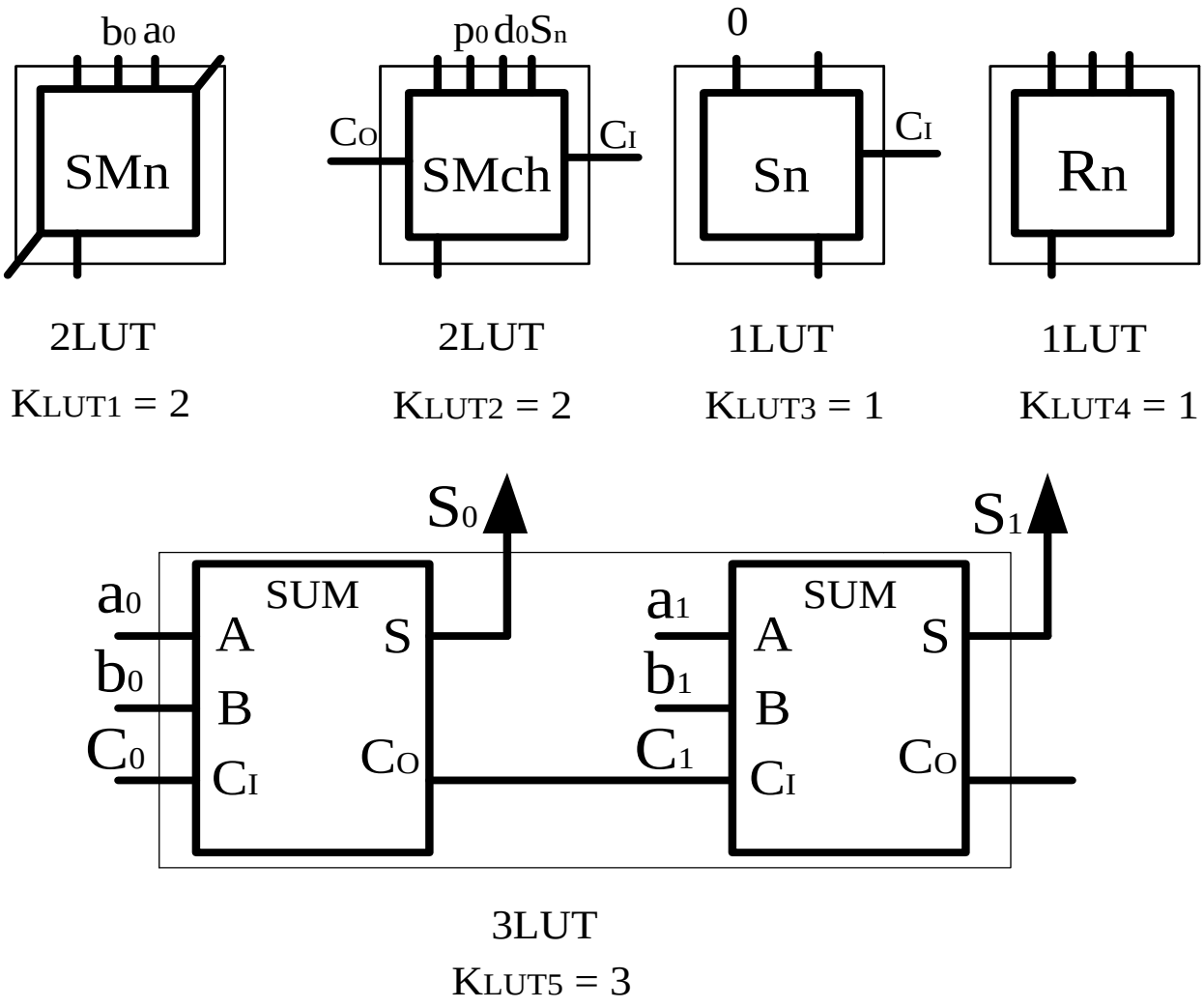


Рис. 2.12 Реалізація основних блоків в МКГ із структурою ЛВ на 6-ти входових *LUT*

SMn – елемент помножувача, який виконує операцію модульного множення та додавання і має виходи результату та переносу, тобто, це комірка Гілда. *SMch* – елемент, який виконує операцію додавання або віднімання числа в доповняльному коді при діленні без відновлення залишків. *Sn* – це вузол, який визначає тип операції, віднімання чи додавання, при діленні без відновлення залишків. *Rn* – елемент, який визначає чи потрібно проводити ще одну операцію додавання для знаходження результату.

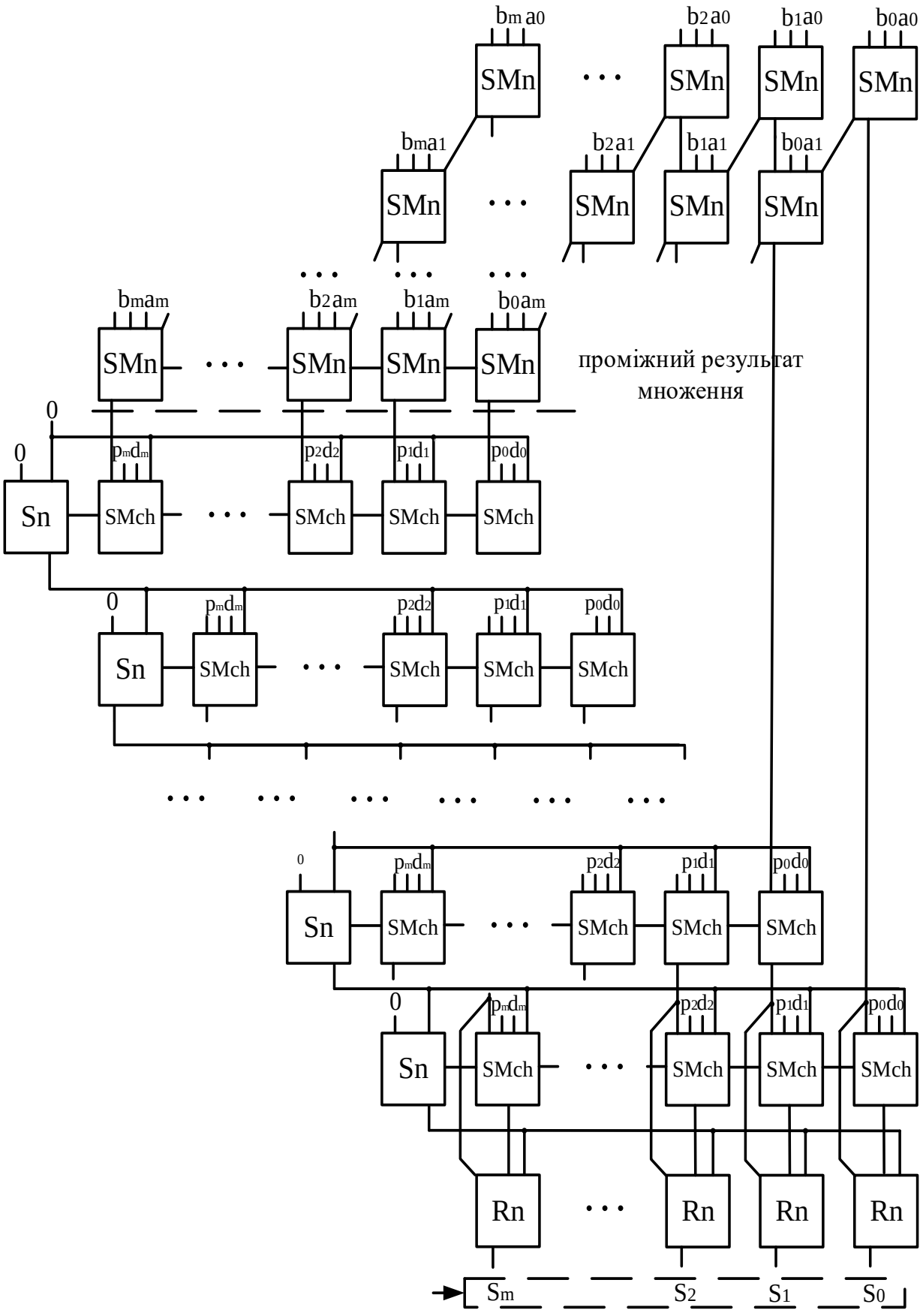


Рис. 2.13 Структура помножувача *MUL_G* МКГ $GF(p^n)$

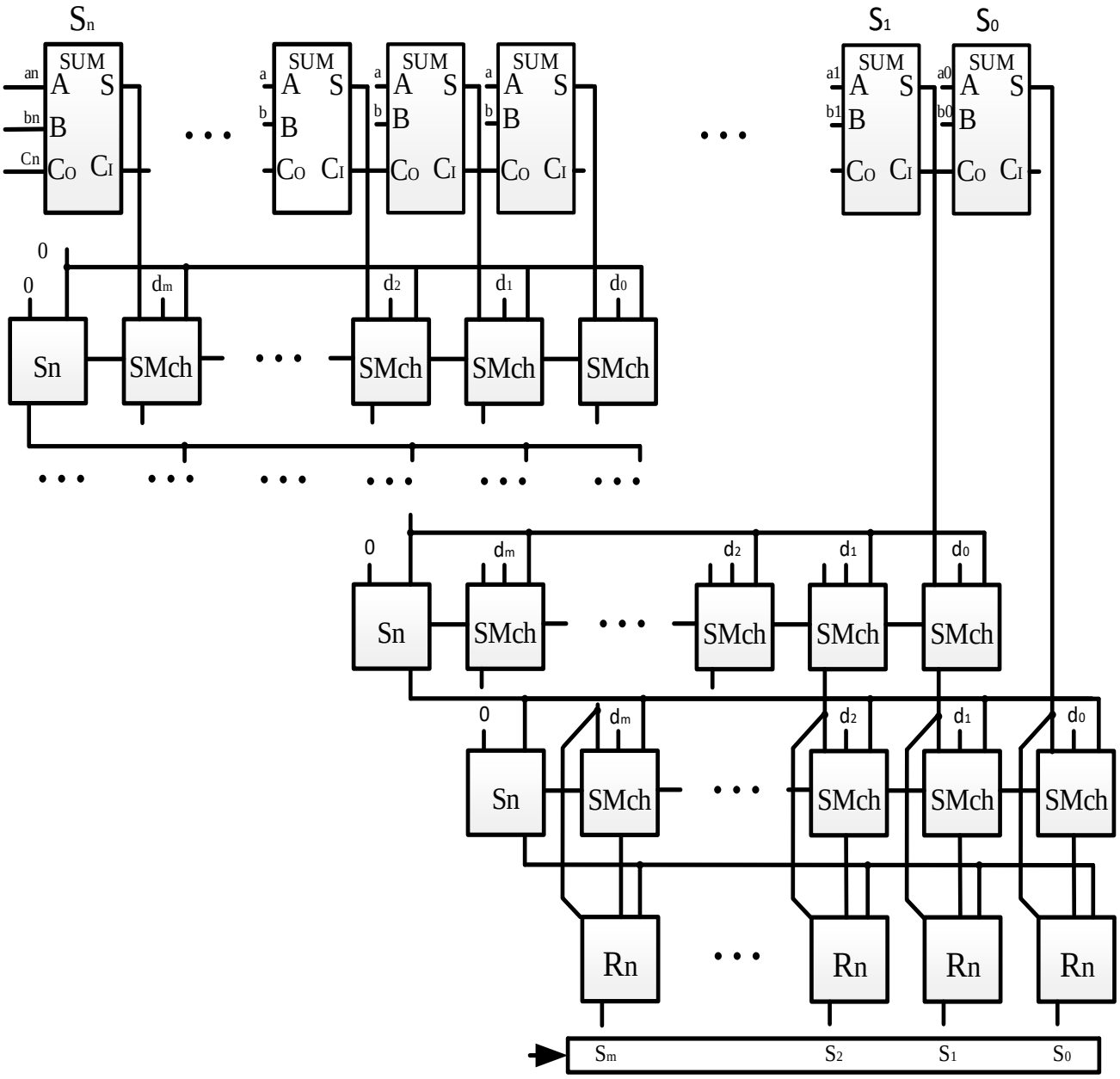


Рис. 2.14 Структура суматора *SUM_G* МКГ $GF(p^n)$

Коефіцієнт апаратних витрат помножувача для елементів поля $GF(p^n)$ відносно аналогічних витрат помножувача для елементів поля $GF(2^m)$ $k_{mul} = k_g k_k$, де $k_g = \frac{k_{gp}}{k_{g2}}$, $k_k = \frac{k_{kp}}{k_{k2}}$ – коефіцієнти складності та кількості МКГ, k_{gp} та k_{g2}, k_{kp} та k_{k2} – кількість *LUT* у МКГ та кількість МКГ для полів Галуа $GF(p^n)$ та $GF(2^m)$, відповідно.

Для розширених полів Галуа $GF(p^n)$ $k_{g2} = 7$, для інших: k_{gp}= KSMn ×

KLUT1 + KSMch × KLUT2 + KSn × KLUT3 + KRn × KLUT4 + KSUMn × KLUT5, KSMn, KSMch, KSn, KRn – кількість елементів, відповідно, SMn, SMch, Sn, Rn в помножувачі елементів розширених полів Галуа (рис. 2.15).

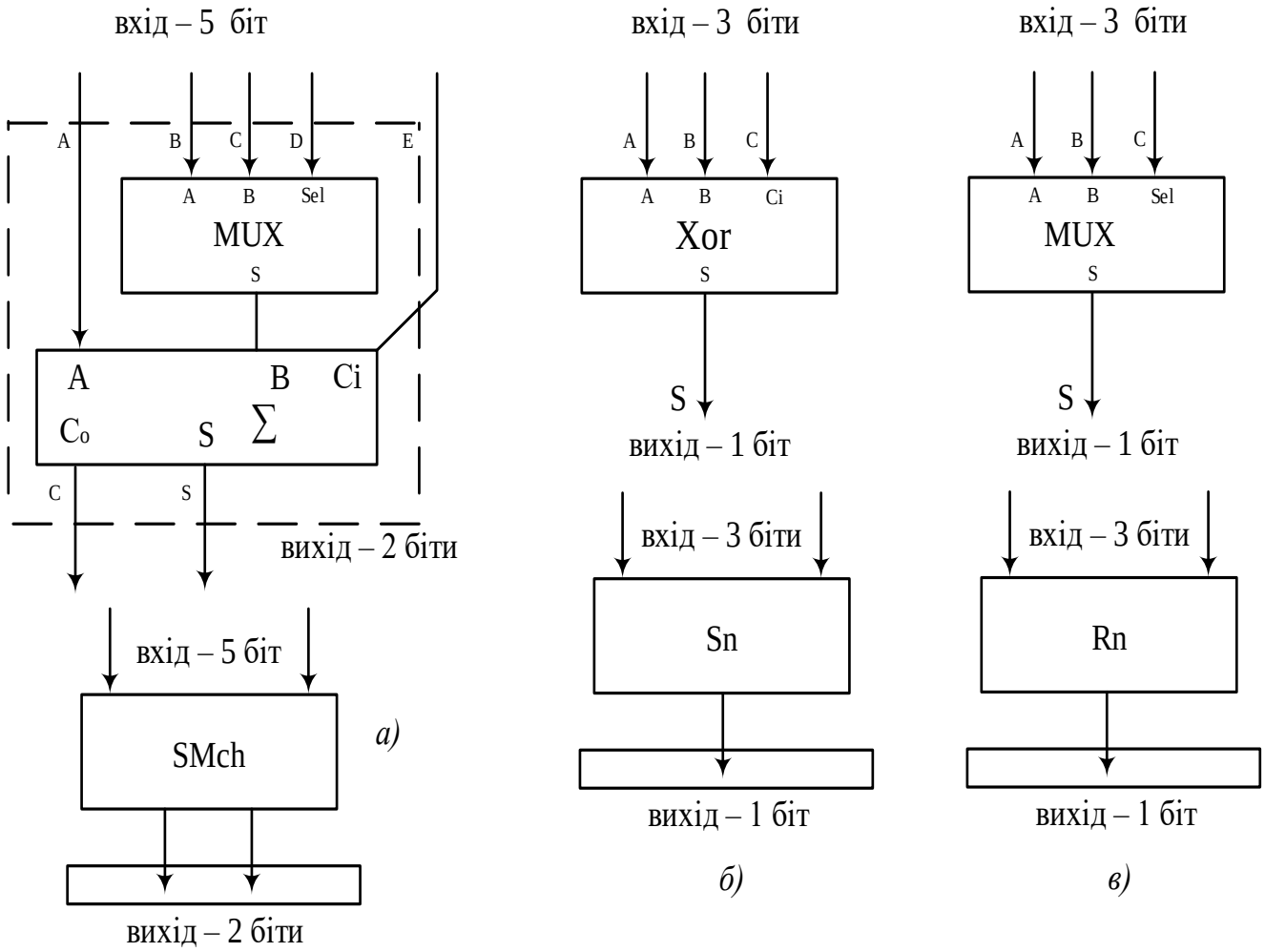


Рис. 2.15 а) елемент *SMch*, б) елемент *Sn*, в) елемент *Rn*

В розширених полях Галуа $GF(2^m)$ для реалізації помножувача потрібно k_{k2}=$2m^2-2m+1$ МКГ, а в полях Галуа $GF(p^n)$ з характеристикою поля p – k_{kp}=$2n^2-2n+1$. Також додатково $(n-1)(2^{3\lceil\log_2 p\rceil-5}-1)\lceil\log_2 p\rceil$ *LUT* для знаходження коефіцієнта, на який перемножуємо незвідний поліном (цими апаратними витратами можна, в даному випадку, знехтувати, оскільки вони малі в порівнянні з витратами на реалізацію самих МКГ)

k_{gp}= $(\lceil\log_2 p\rceil)^2\times 2$ + $(\lceil\log_2 p\rceil)^2\times 2$ + $\lceil\log_2 p\rceil\times 1$ + $\lceil\log_2 p\rceil\times 1$ + $\lceil\log_2 p\rceil\times\frac{3}{2}$

$k_{gp} = 2(\lceil \log_2 p \rceil)^2 + 2(\lceil \log_2 p \rceil)^2 + 2\lceil \log_2 p \rceil + \frac{3}{2}\lceil \log_2 p \rceil = 4(\lceil \log_2 p \rceil)^2 + \frac{7}{2}\lceil \log_2 p \rceil$. Отже: k_{gp}=$(2(\lceil \log_2 p \rceil)^2 + 2(\lceil \log_2 p \rceil)^2 + \frac{1}{2}\lceil \log_2 p \rceil + 2\lceil \log_2 p \rceil = 4\left(\lceil \log_2 p \rceil)^2 + \frac{7}{2}\lceil \log_2 p \rceil\right)/7$.

Отже:

$$k_k \approx \frac{2n^2-2n+1}{2m^2-2m+1} \approx \frac{n^2}{m^2} \approx \left(\frac{n}{m}\right)^2 \text{ для великих } n, \quad (2.6)$$

$$k_{mul} \approx \frac{(4(\lceil \log_2 p \rceil)^2+\frac{7}{2}\lceil \log_2 p \rceil)n^2}{7m^2} \quad (2.7)$$

При цьому $p^m \approx 2^n$. Тоді $m \approx log_p 2^n = \frac{n}{log_2 p}$, $k_k \approx log_2^{-2} p$, $k_{mul} \approx \frac{4(\lceil \log_2 p \rceil)^2+\frac{7}{2}\lceil \log_2 p \rceil}{7(log_2 p)^2}$. $lim_{p\to\infty} k_{mul} = 4/7$. Графік функції k_{mul} наведено на рис. 2.16.

Для $n < 10$ k_{mul} треба розраховувати за більш точними формулами (2.6, 2.7).

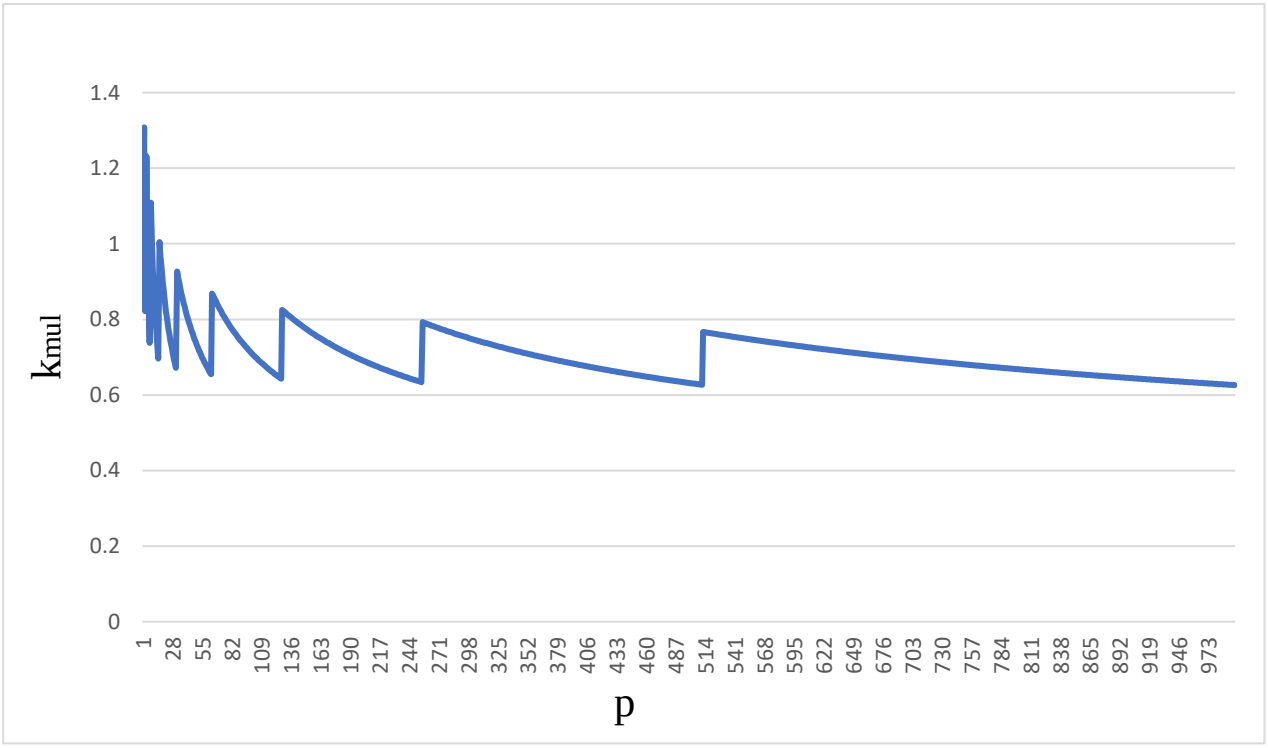


Рис. 2.16 Графік функції k_{mul}

Як видно з рис. 2.16 відношення апаратних витрат на реалізацію помножувачів у полі $GF(p^n)$ до $GF(2^m)$, при збільшенні характеристики поля будуть прямувати до асимптотичного значення $4/7$, що говорить про перевагу, з точки

зору апаратних витрат, використання розширених полів Галуа з великою характеристикою p поля порівняно з двійковими розширеними полями Галуа при реалізації МКГ у варіанті ЛВ.

2.9 Модифікована комірка Гілда для розширених полів Галуа $GF(p^n)$ з характеристикою $p > 2$ та варіанти її структури

МКГ для розширених полів Галуа $GF(p^n)$ з характеристиками $p > 2$ розглянуті у роботі [12]. КГ для розширених полів Галуа з характеристикою поля $p = 2$ має входи *A, B, Ci* та виходи – *S, Co*. МКГ не має виходу переносу – *Co*. МКГ для розширених полів Галуа $GF(p^n)$ повинна мати *3x* двійкових входи та *x* двійкових виходів, розрядністю $x = \lceil \log_2 p \rceil$ біт кожний (рис. 2.17).

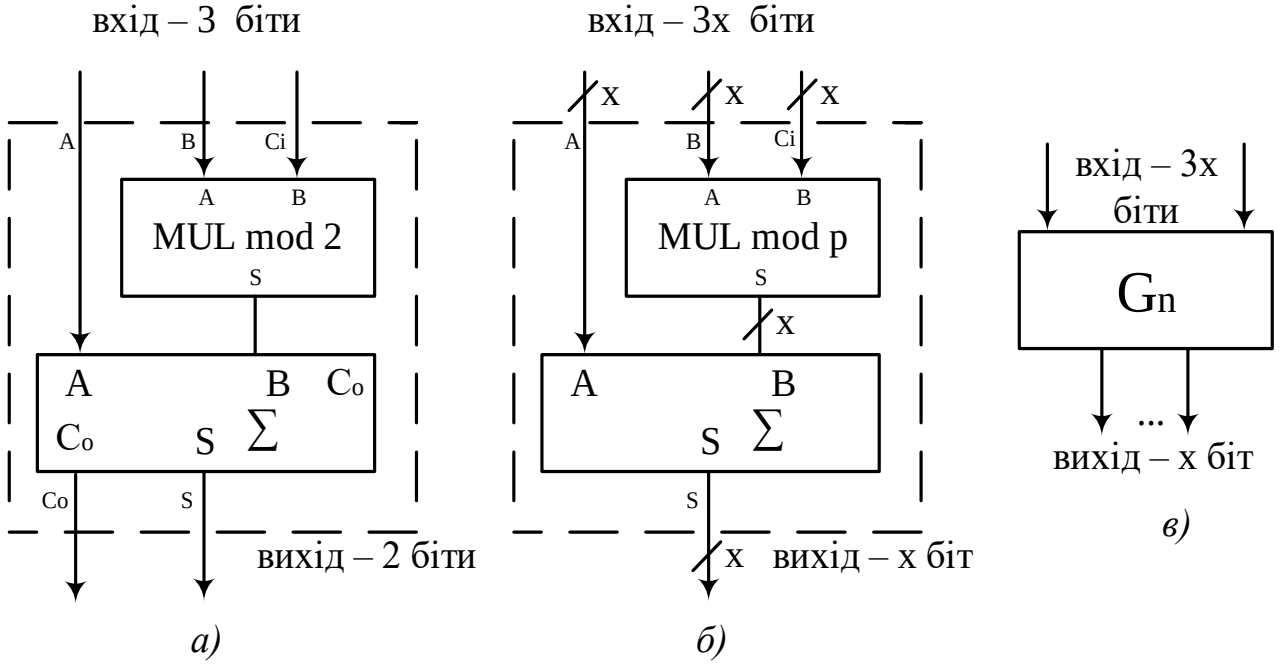


Рис. 2.17 а) комірка Гілда, б) модифікована комірка Гілда для розширених полів Галуа $GF(p^n)$, в) символ модифікованої комірки Гілда для розширених полів Галуа $GF(p^n)$

МКГ для розширених полів Галуа $GF(p^n)$ можна реалізувати на ПЛІС за 3 структурами:

1) МКГ є цілісним елементом (“чорна скринька”) і для кожного виходу цієї комірки формується своя булева функція та проводиться мінімізація цих булевих

функцій методом Квайна–Мак-Класкі–Петрика (ЧС);

2) МКГ створена на основі функціональних вузлів: помножувача та суматора. Проводиться мінімізація методом Квайна–Мак-Класкі–Петрика булевих функцій, які описують роботу помножувача та суматора (ФВ);

3) МКГ створена на основі комбінаційних логічних елементів. МКГ реалізується як комбінаційний паралельний, матричний помножувач за модулем характеристики поля p та суматор за модулем характеристики поля p (ЛВ).

2.10 Метод створення паралельних помножувачів для вузлів КЗІ на основі МКГ

Створення вузлів КЗІ складається із створення криптопроцесора, одним із вузлів якого є паралельний помножувач розширених полів Галуа $GF(p^n)$. Створення паралельних помножувачів для вузлів КЗІ поділяється на створення МКГ, створення інвертора за модулем p – вузла f, створення помножувача на основі МКГ та вузла f. Алгоритми створення вузла f, МКГ, помножувача та криптопроцесора наведено нижче.

Метод створення паралельних помножувачів для вузлів КЗІ на основі МКГ забезпечує синтез помножувачів на основі МКГ з 3 варіантами їх структури: ЧС, ФВ, ЛВ. Кожен варіант створення помножувача буде відрізнятись тільки створенням МКГ.

Створення паралельних помножувачів для вузлів КЗІ на основі МКГ із структурою ЧС

Алгоритм створення МКГ з структурою ЧС:

1.Задання характеристики p поля та порядку n утворюючого поле поліному.

2. Генерування булевих функцій, які описують МКГ як “чорну скриньку” $res = (((a \times b) \bmod p) + c) \bmod p$, де p – характеристика поля, a, b, c – входи МКГ, res – вихід МКГ.

3. Мінімізація булевих функцій, які описують МКГ, методом Квайна–Мак-Класкі–Петрика.

4. Генерування *HDL*-опису МКГ за створеними булевими функціями.

Алгоритм створення інвертора за модулем p – вузла f:

1. Завдання характеристики p поля та порядку n утворюючого поле полінома.

2. Генерування булевих функцій для вузла *f*:

$f = (p - G_n)\, mod\, p = (-G_n)\, mod\, p$, де p – характеристика поля, G_n – результат на виході МКГ при прямому ході обчислень, *f* – результат.

3. Мінімізація булевих функцій, які описують вузол *f*, методом Квайна–Мак-Класкі–Петрика.

4. Генерування *HDL*-опису вузла *f* за створеними булевими функціями.

Алгоритм створення помножувача:

1. Створення матриці для розміщення МКГ та вузлів *f*.

2. Наповнення матриці вузлами МКГ та *f*.

3. Опис зв’язків між МКГ та вузлами *f*.

4. Створення *HDL*-опису помножувача.

Далі можливе створення криптопроцесора, який використовує синтезований помножувач.

Алгоритм створення криптопроцесора:

1. Генерування *HDL*-опису арифметико-логічного пристрою, який виконує операції над елементами розширених полів Галуа $GF(p^n)$.

2. Генерування *HDL*-опису криптопроцесора, який виконує операції над точками ЕК з використанням створеного АЛП.

Схему АЛП криптопроцесора наведено на рис. 2.18. Рівні алгоритмів КЗІ на основі еліптичних кривих, які реалізуються у криптопроцесорі зображено на рис. 2.19. Рівень 6 та вищі рівні виходять за межі даної роботи та детально не розглядаються. Структурну схему криптопроцесора наведено на рис. 1.3.

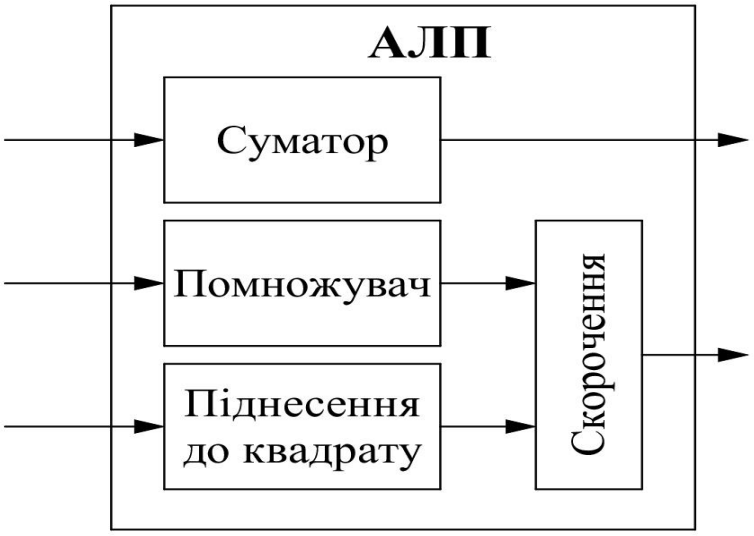


Рис. 2.18 АЛП криптопроцесора

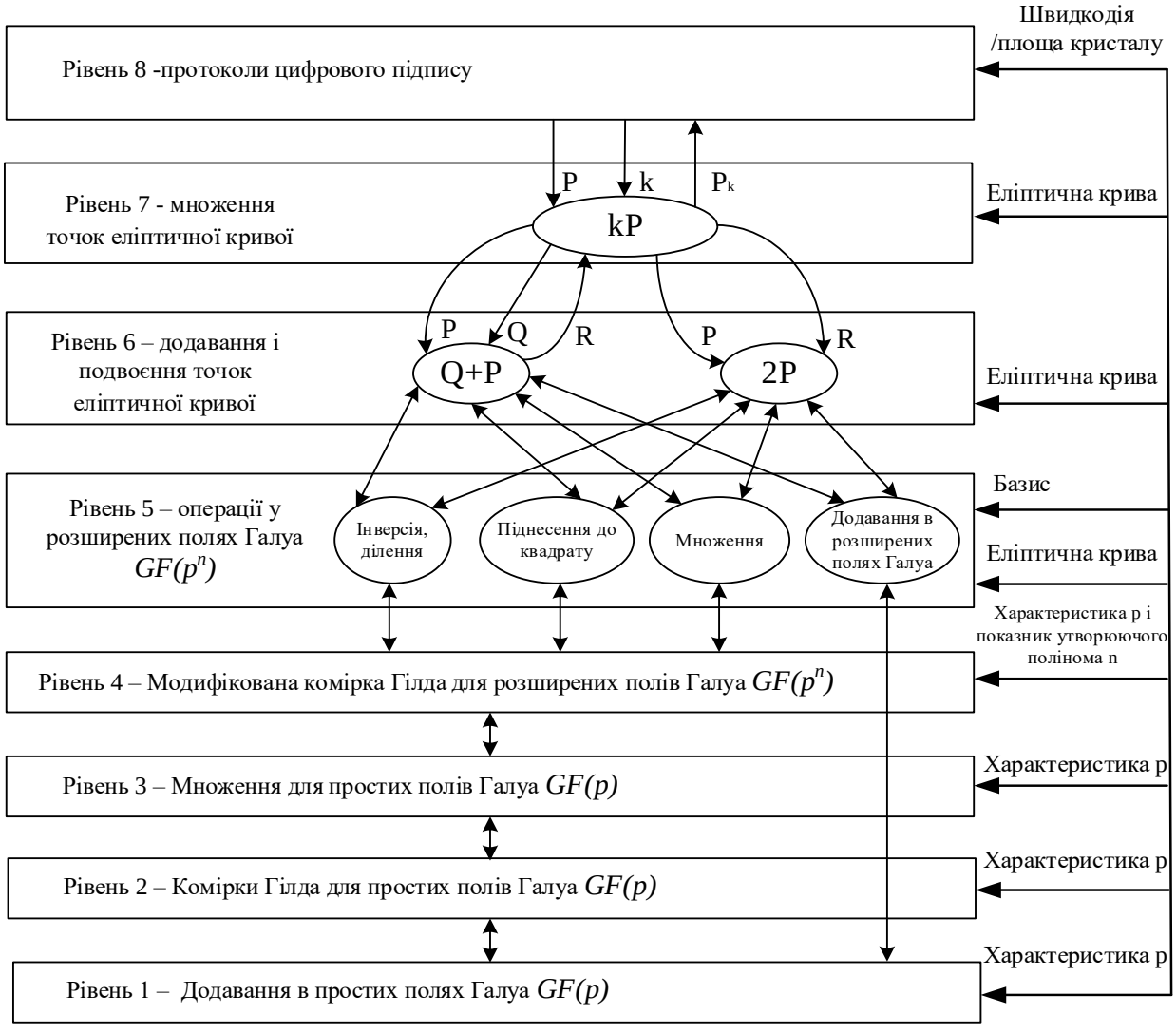


Рис. 2.19 Рівні алгоритмів КЗІ на основі еліптичних кривих

Створення паралельних помножувачів для вузлів КЗІ на основі МКГ з структурою ФВ

Алгоритми створення інвертора за модулем p – вузла f, помножувача та криптопроцесора, наведено вище. Вони є спільними для всіх структур МКГ.

Алгоритм створення МКГ з структурою ФВ складається з наступних кроків:

1. Задання характеристики p поля та порядку n утворюючого поле полінома.

2. Генерування булевих функцій, які описують помножувач МКГ:

$res_{mul} = (a \times b) \bmod p$, де p – характеристика поля, a, b – входи помножувача (*MUL*) МКГ, res_{mul} – вихід.

3. Мінімізація булевих функцій, які описують помножувач, методом Квайна–Мак-Класкі–Петрика.

4. Генерування *HDL*-опису помножувача (*MUL*).

5. Генерування булевих функцій, які описують суматор МКГ:

$res_{sum} = (res_{mul} + c) \bmod p$, де p – характеристика поля, res_{mul}, c – входи суматора (*SUM*) МКГ, *res* – вихід.

6. Мінімізація булевих функцій, які описують суматор, методом Квайна–Мак-Класкі–Петрика.

7. Генерування *HDL*-опису суматора (*SUM*).

8. Генерування *HDL*-опису МКГ із описів створеного помножувача (*MUL*) та суматора (*SUM*).

Створення паралельних помножувачів для вузлів КЗІ на основі МКГ з структурою ЛВ

Алгоритми створення інвертора за модулем p – вузла f, помножувача та криптопроцесора, наведено вище. Вони є спільними для всіх структур МКГ.

Алгоритм створення МКГ з структурою ЛВ складається з наступних кроків:

1. Завдання характеристики p поля та порядок n утворюючого поле полінома.

2. Створення матриці, у кожному елементі якої може бути або елемент *SMn* (КГ), який формує сигнали $C,S = ((a \times b) \bmod p) + c$, де p – характеристика поля, a, b, c – входи КГ, S – вихід результату, C – вихід переносу, або елемент *SMch*, який формує сигнали *Cout,S = A+E+(B&D or Cin&notD)*, де *A, B, Sel* – входи мультиплексора, *A, B, Ci* – входи суматора, *Cout, S* – виходи суматора.

3. Створення списку елементів *Sn*: *S = A xor B xor C, Rn: S = A & B or C & notB*, та *SUM*.

4. Визначення, де саме у матриці потрібно поставити вузли *SMn* та *SMch*.

5. Встановлення зв'язків з входами *A, B, C* та виходами *S* та *Co* вузла *SMn*, входами *A, B, C, D, E* та виходами *S* та *Co* вузла *SMch*.

6. Встановлення зв'язків з елементами *Sn* та *Rn*.

7. Встановлення зв'язків входів МКГ *A, B, C* та виходу МКГ *S*.

8. Генерування *HDL*-опису МКГ.

2.11 Метод оцінювання часової складності виконання множення елементів розширених полів Галуа $GF(p^n)$ апаратним способом

Метод складається з наступних кроків:

1. Генерування помножувача.

2. Створення проекту у середовищі *Xilinx Vivado*.

3. Тестування згенерованого помножувача.

4. Проведення імплементації згенерованого помножувача.

5. Визначення максимальної часової затримки Tапар. помножувача. Інформація про максимальну часову затримку береться з середовища *Xilinx Vivado* або із звітів, які формуються після імплементації.

6. Визначення часу Tпрогр. множення у математичному пакеті *Maple*.

7. Визначення відношення часових витрат при програмній та апаратній реалізаціях множення елементів розширених полів Галуа $GF(p^n)$ $k = \frac{T\text{прогр.}}{T\text{апар.}}$.

Припустимо, злом засобів КЗІ реалізується методом "грубої сили". Для оцінки стійкості засобу КЗІ до злому необхідно проаналізувати складність виконання арифметичних операцій у розширених полях Галуа $GF(p^n)$. Найскладні-

шою арифметичною операцією є операція множення. Тому проаналізуємо часову складність множення у розширених полях Галуа $GF(p^n)$.

Проведемо дослідження часової складності виконання операцій множення елементів розширених полів Галуа $GF(p^n)$ у математичному пакеті *Maple*. У табл. 2.1 наведено час виконання операції множення (у мілісекундах) елементів розширених полів Галуа $GF(p^n)$ та відносний час виконання множення (k) – це відношення часу виконання множення елементів розширеного поля Галуа $GF(p^n)$, до часу виконання множення елементів розширеного поля Галуа з характеристикою поля 2 $GF(2^m)$: $k = \frac{Tmul(GF(p^n))}{Tmul(GF(2^n))}$. Залежність коефіцієнта k від характеристики поля наведено на рис. 2.20.

Таблиця 2.1

Час виконання множення елементів розширених полів Галуа $GF(p^n)$ у математичному пакеті Maple

Поле	Час виконання множення мс.	Відносний час виконання множення, $k = \frac{Tmul(GF(p^n))}{Tmul(GF(2^n))}$
$GF(2^{998})$	0.259	1.00
$GF(3^{630})$	0.378	1.46
$GF(5^{430})$	0.306	1.18
$GF(7^{350})$	0.217	0.84
$GF(11^{280})$	0.153	0.59
$GF(13^{270})$	0.137	0.53
$GF(17^{240})$	0.116	0.45
$GF(19^{230})$	0.108	0.42
$GF(23^{220})$	0.098	0.38
$GF(29^{206})$	0.085	0.33

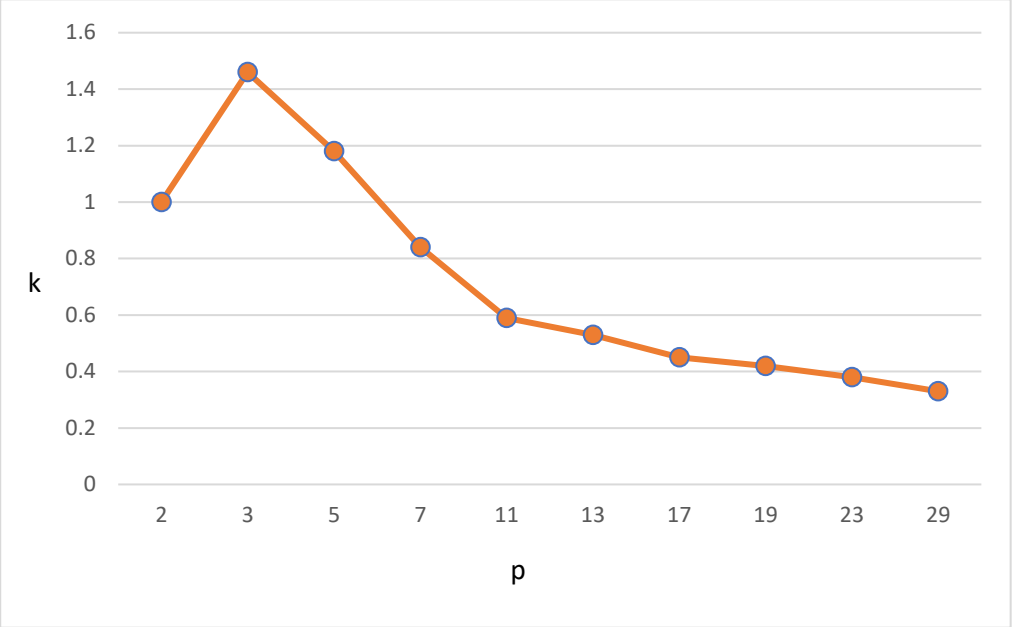


Рис. 2.20 Відносний час виконання множення елементів розширених полів Галуа $GF(p^n)$ у математичному пакеті Maple

З рис. 2.20 видно, що найбільшу часову складність множення елементів розширених полів Галуа $GF(p^n)$ має помножувач для поля $GF(3^n)$ з коефіцієнтом 1.46, потім йде $GF(5^n)$ – коефіцієнт 1.18, $GF(2^m)$ – коефіцієнт 1.

2.12 Метод оцінювання апаратної складності помножувачів елементів розширених недвійкових полів Галуа *GF(p^n)*

Метод складається з наступних кроків:

1. Задання параметрів полів p, n, m, таких, що $p^n \approx 2^m$.

2. Визначення апаратних витрат у полі $GF(2^m)$.

3. Визначення апаратних витрат у полі $GF(p^n)$.

4. Визначення відносної апаратної складності як відношення апаратних витрат для полів $GF(p^n)$ та $GF(2^m)$ за формулами (2.1) – (2.8) з врахуванням коефіцієнта Co (відносний порядок поля $GF(p^n)$, наведено у таблиці 4.5). Кращим вважається поле з меншим показником розрахованої відносної апаратної складності.

5. Перевірка розрахованого теоретичного показника відносної апаратної складності з результатами імплементації помножувача на ПЛІС:

1) створення помножувачів описаним вище методом;

2) створення проекту у середовищі *Xilinx Vivado*;

3) тестування створених помножувачів;

4) імплементація створених помножувачів (1 помножувач у проекті) (рис. 2.21).

6. Одним з результатів імплементації є звіт, де вказано кількість *LUT*, які було задіяно для імплементації помножувача (рис. 2.22). Кількість *LUT* є результатом оцінювання.

7. Визначення відносної апаратної складності результатів імплементації як відношення кількості *LUT* у помножувачах для полів $GF(p^n)$ та $GF(2^m)$.

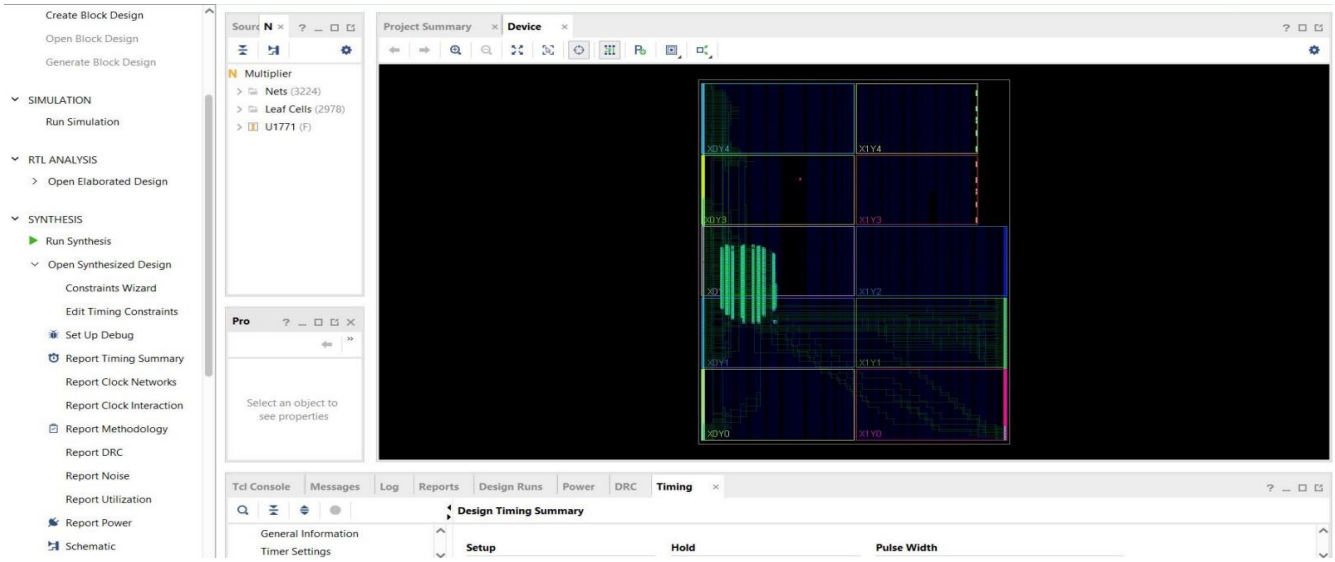


Рис. 2.21 Імплементація згенерованого помножувача

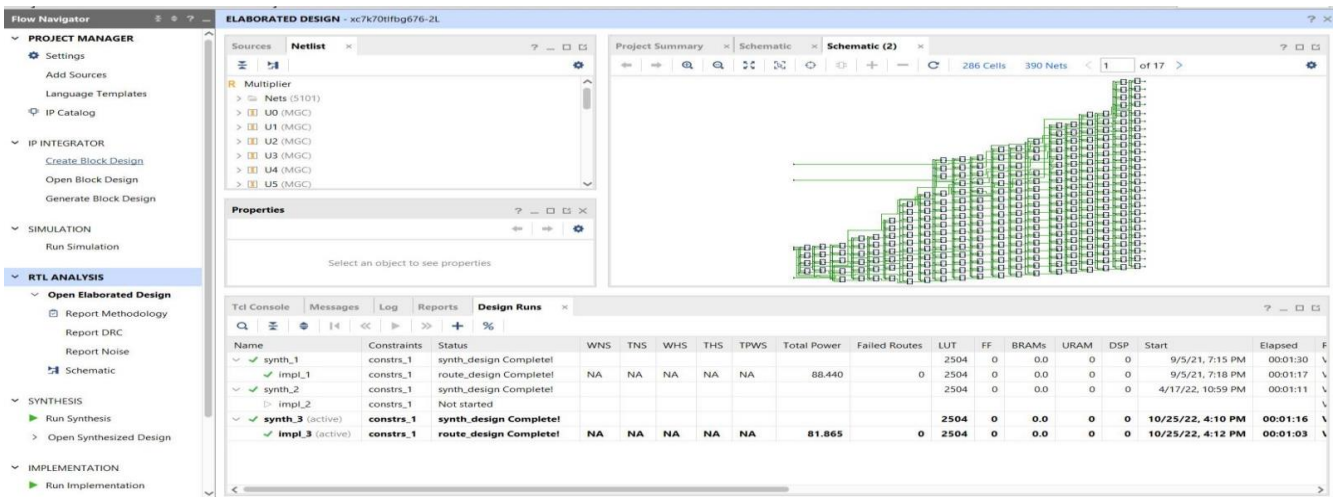


Рис. 2.22 Схема згенерованого помножувача

На рис. 2.21 наведено імплементований помножувач у середовищі *Xilinx Vivado*. На рис. 2.22 згенерований помножувач та репорити, у яких міститься основна інформація та характеристики згенерованого помножувача: апаратні витрати, часові затримки, енергоспоживання, кількість входів, кількість виходів, інформація про час генерування та успішність операції.

2.13 Метод тестування генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$

Метод складається з наступних кроків:

1. Визначення поліному, який утворює розширене поле Галуа $GF(p^n)$, за допомогою математичного пакету *Maple*.

2. Генерування помножувача одним із описаних вище методів.

3. Моделювання у середовищі *Active HDL* роботи помножувача на тестових прикладах, а саме, e_{max} – максимально великий код елемента поля. Проводимо моделювання множення елементів: $e_{max} \times (e_{max} - 1)$; $e_{max} \times 0$; $e_{max} \times 1$.

4. Виконання множення цих елементів у математичному пакеті *Maple*.

5. Виконання множення цих елементів за допомогою бібліотеки *C++ GaloisCPP*.

6. Порівняння результатів. Усі три результати повинні збігатися, що є ознакою правильної роботи помножувача, а також запропонованих методів створення помножувачів та технологічного засобу (генератора ядер), який реалізує ці методи.

Ознакою правильної роботи помножувача є те, що результати моделювання у *Active HDL* збігаються з результатами множення у математичному пакеті *Maple* та з результатами множення у бібліотеці *GaloisCPP*.

На рис. 2.23 наведено структурну схему процесу тестування генератора помножувачів елементів розширених полів Галуа $GF(p^n)$ з використанням двох еталонів. Тестування починається із генерування тестових послідовностей, які подаються на помножувач, згенерований генератором та еталони. Далі результати подаються на вузол порівняння. При невідповідності результатів робимо ви-

сновок, що тестований об’єкт працює неправильно. Як еталони використовуються результати множення елементів розширених полів Галуа $GF(p^n)$ у математичному пакеті *Maple* та з використанням бібліотеки *GaloisCPP*.

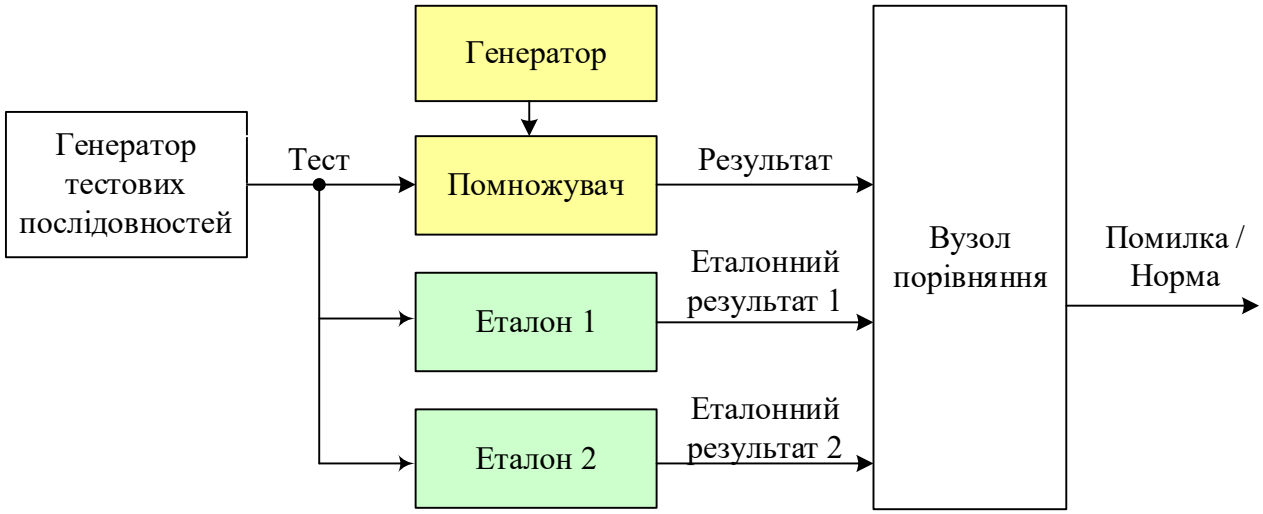


Рис. 2.23 Структурна схема процесу тестування генератора помножувачів елементів розширених полів Галуа $GF(p^n)$ з використанням двох еталонів.

2.14 Актуальність задачі створення генераторів ядер помножувачів елементів розширених полів Галуа *GF(p^n)*

У 2 розділі проведено теоретичні дослідження з метою виявлення розширених полів Галуа $GF(p^n)$ з найменшими апаратними витратами на реалізацію помножувачів у порівнянні із помножувачами для полів $GF(2^m)$. Дуже добре себе показали помножувачі для полів $GF(2^m)$, $GF(3^n)$, $GF(5^n)$, $GF(7^n)$. Для підтвердження цих результатів проведено практичні дослідження: створено помножувачі для цих полів Галуа, проведено їх імплементацію на ПЛІС та порівняно апаратні витрати на їх створення. Створювати помножувачі вручну для елементів розширених полів Галуа $GF(p^n)$, порядок яких приблизно 2^{998}, практично неможливо, оскільки такий помножувач буде мати 1992008 МКГ. Для створення таких великих помножувачів було розроблено інструменти для генерації їх *HDL*-описів для подальшого використання цих описів при реалізації засобів КЗІ на ПЛІС.

Генератор створює помножувачі за трьома варіантами структури МКГ:

- МКГ як “чорна скринька” (без виділення структурних елементів) – ЧС;

- МКГ на основі функціональних вузлів (помножувач та суматор) – ФВ;

- МКГ на основі логічних вузлів – ЛВ.

На рис. 2.24 наведено структурну схему генераторів ядер помножувачів елементів розширених полів Галуа *GF(p^n)*, яку розроблено для підтвердження або спростування правильності теоретичних обчислень.

Схеме складається з 3 блоків:

1. Введення характеристики *p* та порядку утворюючого поле полінома *n* та вибір структури МКГ помножувача.

2. Створення булевих функцій вузлів помножувача.

3. Створення помножувальної матриці та заповнення її згенерованими вузлами.

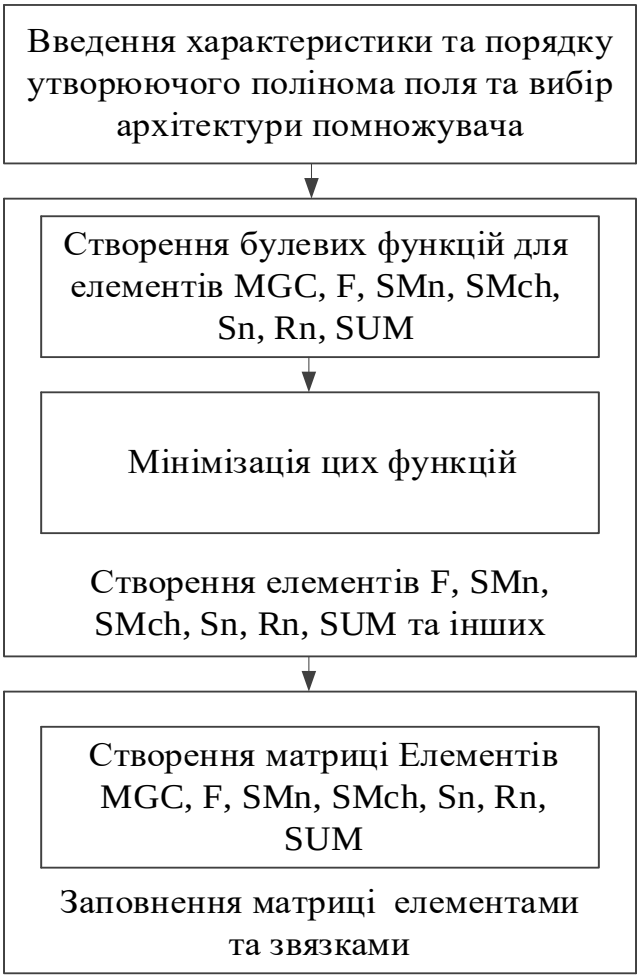


Рис. 2.24 Структурна схема генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$.

2.15 Висновки до розділу 2

У другому розділі:

Показано основні архітектурні принципи побудови вузлів КЗІ, їхню структуровану модель та структурні алгоритми їх роботи.

Розглянуто теоретичні основи підрахунку апаратних витрат на реалізацію помножувачів елементів розширених полів Галуа $GF(p^n)$.

Проведено порівняння часових складностей елементів розширених полів Галуа $GF(p^n)$ у математичному пакеті *Maple*. За результатами порівняння найбільшу часову складність мають помножувачі для полів Галуа $GF(3^n)$.

Проведено аналіз апаратних витрат на множення елементів розширених полів Галуа $GF(p^n)$ апаратним способом. Порівняння проведено за трьома структурами реалізації МКГ:

1) МКГ як “чорна скринька” (ЧС);

2) МКГ на основі функціональних вузлів: помножувача та суматора (ФВ);

3) МКГ на основі логічних вузлів (ЛВ).

Для структури МКГ ЧС помножувачі для поля $GF(3^n)$ мають кращі (менші) показники апаратної складності ніж для поля $GF(2^m)$, для структури МКГ ФВ помножувачі для полів $GF(3^n)$, $GF(5^n)$ та $GF(7^n)$ мають кращі (менші) показники апаратної складності ніж поля $GF(2^m)$. Для структури МКГ ЛВ коефіцієнт відносної апаратної складність помножувачів прямує до константного значення $4/7$, що свідчить про перевагу помножувачів для недвійкових полів $GF(p^n)$ над помножувачами для поля $GF(2^m)$. Ці порівняння є теоретичними і далі проведено їх практичне підтвердження.

РОЗДІЛ 3

РОЗРОБКА ЗАСОБІВ СТВОРЕННЯ ПОМНОЖУВАЧІВ ЕЛЕМЕНТІВ РОЗШИРЕНИХ ПОЛІВ ГАЛУА *GF(p^n)* ДЛЯ ВУЗЛІВ КЗІ КФС

3.1 Вимоги до технології створення генераторів ядер (*VHDL*-описів) помножувачів елементів розширених полів Галуа *GF(p^n)*

Для реконфігурації засобів КЗІ необхідно використовувати ПЛІС. Реконфігурація забезпечується імплементацією *VHDL*-описів, які описують схему всередині ПЛІС. Необхідно мати можливість проводити редагування цих описів, для чого потрібний спеціальний засіб – генератор *VHDL*-описів, або, по-іншому, генератор ядер.

У Розділі 3 описано процес створення генераторів *VHDL*-описів помножувачів елементів розширених полів Галуа $GF(p^n)$ з характеристиками p та порядками n утворюючого поле полінома, які є реконфігурованими. Генератор створює помножувачі за трьома структурами МКГ, які запропоновано у п. 2.12.

При цьому реалізовано алгоритм мінімізації булевих функцій методом Квайна–Мак-Класкі–Петрика. Також у цьому розділі було проведено тестування згенерованих помножувачів для полів з різними характеристиками.

Зростання сфер застосування операцій над розширеними полями Галуа $GF(p^n)$, у різних алгоритмах КЗІ зумовлює розробку та впровадження нових пристроїв для реалізації цих операцій. На практиці часто використовуються розширені поля Галуа $GF(2^m)$. Проте використання інших характеристик поля надає більше можливостей для КЗІ і, відповідно, здатне підвищити криптографічну стійкість засобів КЗІ. Хоч і існує багато засобів для створення помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС з використанням мов опису апаратних рішень (*VHDL, Verilog, HLS*), проте такий підхід є надзвичайно трудомістким. Тому виникла потреба у створенні генераторів таких помножувачів. Генератори дозволяють користувачу задати необхідні параметри помножувача (характеристику p поля, порядок n полінома, що утворює поле), після чого генерується опис помножувача мовою, наприклад *VHDL*, що дозволяє досліджувати

його характеристики, переваги та недоліки на ПЛІС.

Генератори ядер (*VHDL*-описів) помножувачів елементів розширених полів Галуа $GF(p^n)$ створюються за методом створення спеціалізованого пристрою, шляхом конфігурування базової моделі [33].

Генератор ядер повинен відповідати наступним вимогам:

1. Зручність використання розробки. Тобто, дружність інтерфейсу та широкі функціональні можливості.

2. Забезпечення можливості тестування розробленого пристрою.

3. Можливість оцінки апаратних витрат, споживаної потужності та продуктивності пристрою за допомогою синтезування, після імплементування ядра у ПЛІС.

4. Можливість вибору оптимального, за критеріями користувача, варіанту. Тобто кращого по співвідношення апаратних витрат, швидкодії та споживаної потужності.

Сучасні системи розробки мають наступні проблеми:

- складність процесу проектування. Від розробника вимагається глибоке знання апаратного забезпечення, на якому буде реалізований пристрій, та алгоритмів його роботи;

- час проектування;

- продуктивність розробленого пристрою.

При створенні помножувачів елементів розширених полів Галуа $GF(p^n)$ виникає серйозна проблема – вони дуже великі, проте містять дуже багато подібних вузлів. Апаратні витрати деяких з таких помножувачів містить табл. 3.1.

З табл. 3.1 видно, що кількість елементів є дуже великою. Вручну створювати такі помножувачі надзвичайно складно або практично неможливо. Тому було вирішено розробити генератори ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ з реконфігурованими характеристикою поля та порядком утворюючого поле полінома (порядок поля не більше 2^{998}).

Таблиця 3.1

Кількість МКГ у помножувачах елементів розширених полів Галуа $GF(p^n)$ з приблизно однаковими порядками полів та різними характеристиками полів

Поле	Кількість МКГ (приблизно)	Кількість входів та виходів МКГ
$GF(2^{998})$	2000000	12000000
$GF(3^{630})$	800000	9500000
$GF(5^{430})$	370000	6600000
$GF(7^{350})$	250000	4500000
$GF(11^{280})$	145800	3500000
$GF(13^{270})$	150000	3600000
$GF(17^{240})$	115200	3450000
$GF(19^{230})$	105800	3180000
$GF(23^{220})$	96800	2910000
$GF(31^{201})$	80800	2430000
$GF(37^{191})$	73000	2630000
$GF(41^{186})$	69200	2484000
$GF(43^{184})$	67700	2448000
$GF(47^{180})$	64800	2340000
$GF(53^{174})$	60500	2160000

3.2 Структурна схема генераторів ядер (*VHDL*-описів) помножувачів елементів розширених полів Галуа $GF(p^n)$

Генератори *VHDL*-описів помножувачів елементів розширених полів Галуа $GF(p^n)$ були реалізовані мовою програмування *C++*. Вони реалізують розроблений метод створення паралельних помножувачів для вузлів КЗІ на основі МКГ (п. 2.9). На рис. 3.1 наведено структурну схему генераторів. Генератори створюють помножувачі за трьома структурами МКГ (п. 2.1). Дані алгоритми детально описані у літературних джерелах [6], [7], [12], [13], [14]. Процес генерування по-

множувачів поділяється на такі етапи:

1) генерування булевих функцій для вузлів;

2) мінімізація цих функцій;

3) створення *VHDL*-описів вузлів помножувачів: *F*, *SUM*, *MUL*, *SMn*, *SMch*, *Sn*, *Rn* (тільки для ФВ та ЛВ);

4) створення *VHDL*-описів МКГ (*MGC*);

5) встановлення зв’язків між згенерованими вузлами (створення *VHDL*-описів помножувачів).

Програма генерує булеві функції за їх таблицями істинності. Далі відбувається мінімізація методом Квайна–Мак-Класкі–Петрика. На основі мінімізованих булевих функцій, створюються *VHDL*-описи вузлів *F*, *SUM*, *MUL*, *SMn*, *SMch*, *Sn*, *Rn* та генерується *VHDL*-опис МКГ. На останньому етапі генерується *VHDL*-опис самого помножувача, в якому описано всі МКГ, елементи *F* помножувача та зв’язки між ними.

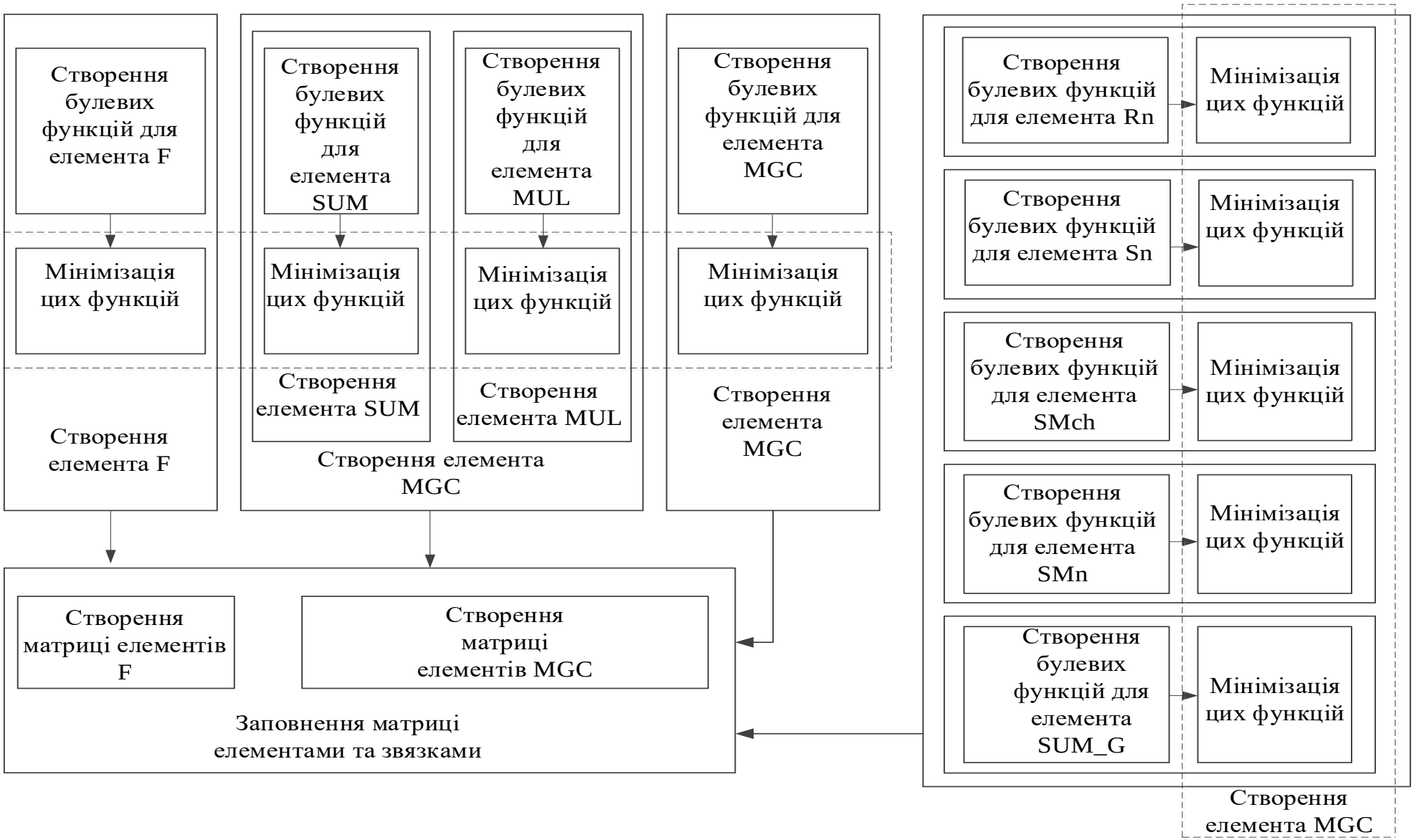


Рис. 3.1 Структурна схема генераторів ядер (*VHDL*-описів) помножувачів елементів розширених полів Галуа $GF(p^n)$

Діаграми класів генераторів ядер (*VHDL*-описів) помножувачів елементів розширених полів Галуа $GF(p^n)$ описують програмну реалізацію генераторів. Діаграми класів реалізовано для кожного генератора окремо. Кожен генератор відповідає певній структурі МКГ (рис. 3.2, 3.4, 3.7). З діаграм класів бачимо, що весь обмін даними відбувається через файли.

Для структури МКГ ЧС діаграма класів генератора зображена на рис. 3.2, для структури МКГ ФВ на рис. 3.4, для структури МКГ ЛВ на рис. 3.7.

3.3 Генерація та мінімізація булевих функцій МКГ

Генерація та мінімізація булевих функцій МКГ є дуже трудомісткою та часозатратною задачею. Для мінімізації був обраний метод Квайна–Мак-Класкі–Петрика. До переваг методу Квайна–Мак-Класкі–Петрика можна віднести такі особливості:

1) його можна застосовувати на великій кількості САПР, з використанням ЕОМ, для мінімізації повністю або частково певних функцій;

2) не важливо, задана функція в СДНФ або СКНФ;

3) зручно мінімізувати системи булевих функцій у зв'язку з простим виділенням загальних частин системи ФАЛ, що реалізується;

4) даний метод – алгоритмічно систематичний, він легко формалізується та алгоритмізується, не залежить від навичок розробника;

5) дозволяє послідовно здійснити всі етапи мінімізації (склеювання та виявлення зайвих імплікант, отримання мінімальних покриттів).

Незважаючи на наведені переваги, у розглянутого методу є два порівняно серйозні недоліки:

1) дуже важко вручну мінімізувати функції з шістьма і більше змінними;

2) метод Квайна–Мак-Класкі–Петрика алгоритмічно неінваріантний: час роботи зростає експоненційно зі збільшенням кількості вхідних даних. Якщо функція залежить від занадто великої кількості змінних, використовують евристичні алгоритми.

В даній роботі проводилась мінімізація булевих функцій для полів з характеристиками $p \leq 53$ (функцій, які залежать максимум від 18 змінних).

Алгоритм мінімізації методом Квайна–Мак-Класкі–Петрика наведено в [258].

Класом, який відповідає за мінімізацію є *Simplifier*. Він містить всі методи та дані, які необхідні для мінімізації функцій. Конструктор класу створює таблицю, в яку будуть записуватись терми для мінімізації за допомогою методу *add_table()* та таблицю, у якій буде міститись результат мінімізації методом *make_min_table()*. Метод *set_function()* завантажує функцію, яку нам потрібно мінімізувати у таблицю. *Get_current_level()* – повертає поточний найменший рівень таблиці мінімізації. *Compress_table()* – мінімізує таблицю з імплікантами булевої функції (Додаток Д.16-Д.17).

Функції *print_term_expr(), print_func_expr()* та *print_truth_table()* є допоміжними та використовуються для виводу на екран результатів мінімізації кожного терму, функції загалом та таблиці істинності даної логічної функції (Додаток Д.18).

TransFuncToMinimizeMGC(),transFuncToMinimizeF(), transFuncToMinimize MUL() та *transFuncToMinimizeSUM()* – використовуються для перетворення функцій зі зручного для людини вигляду у простіший вигляд, який зручніший для обробки комп’ютером. Наведемо приклад перетворення функції. Функція вигляду *f(A,B,C,D,E,F)=(not A(1) and not A(0) and not B(1) and B(0) and not C(1) and C(0)) or (not A(1) and not A(0) and B(1) and not B(0) and C(1) and not C(0))* перетворюється до такого вигляду *f(A,B,C,D,E,F)=~A ~B ~C D ~E F +~A ~B C ~D E ~F +~A B ~C ~D ~E ~F* (Додаток Д.18).

ConvertminimizedFunctionMGC(), convertminimizedFunctionF(), convertmini mizedFunctionMUL() та convertminimizedFunctionSUM() – функції, які перетворюють мінімізовану функцію вигляду *f(A,B,C,D,E,F)=~A~B~CD~EF+~A~BC ~DE~F+A~B~CD E~F* у вигляд *f(A,B,C,D,E,F)=(not A(1) and not A(0) and not B(1) and B(0) and not C(1) and C(0)) or (not A(1) and not A(0) and B(1) and not B(0) and C(1) and not C(0)) or (A(1) and not A(0) and not B(1) and B(0) and C(1)*

and not C(0)) (Додаток Д.18).

Функції *QM_F()* та *QM_MGC()* є функціями, які перетворюють мінімізовану функцію у текстовий формат, для використання у *VHDL* шаблонах розроблюваних пристроїв (Додаток Д.18).

3.4 Створення генератора ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ЧС

При створенні помножувача елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ЧС генеруються булеві функції логічних вузлів *f* та *MGC* у класах *generateFunctionsForF* та *generateFunctionsForMGC* відповідно. Згенеровані функції записуються у файли. Далі клас *transformation_and_minimization* зчитує ці функції з файлу та проводить їх мінімізацію методом Квайна–Мак-Класкі–Петрика та записує результат знову у ці файли. Класи *create_f* та *create_MGC,* на основі мінімізованих булевих функцій генерують *VHDL*-описи вузлів *F* та *MGC* та записують їх у файли. Класи *felement* та *MGC* описують відповідні вузли *f* та *MGC*, з яких клас *create_matrix* створює помножувальну матрицю. Клас *create_matrix* також встановлює для кожного з об’єктів класів *felement* та *MGC* зв’язки з іншими вузлами (рис. 3.2).

3.4.1 Створення інвертора за модулем p (вузла f)

Створення інвертора за модулем *p* (вузла *f*) полягає у генеруванні булевих функцій, які описують логіку його роботи, їх мінімізації та формуванні *VHDL*-опису цього вузла. В залежності від вхідного параметру, тобто характеристики поля *p,* формується булева функція, яка є інверсією вхідного значення за модулем *p*, тобто *res = p - input*.

GenerateFunctionsForF – це клас, який за допомогою методу *generateFunction()* створює булеві функції, для кожного із розрядів результату. Параметр *basis,* тобто характеристика поля, передається конструктору (Додаток Д.9 –Д.10).

Create_f – це клас, який за допомогою методу *creaF()* створює *VHDL*-опис вузла *F* за наступним шаблоном (Додаток Д.1-Д.2):

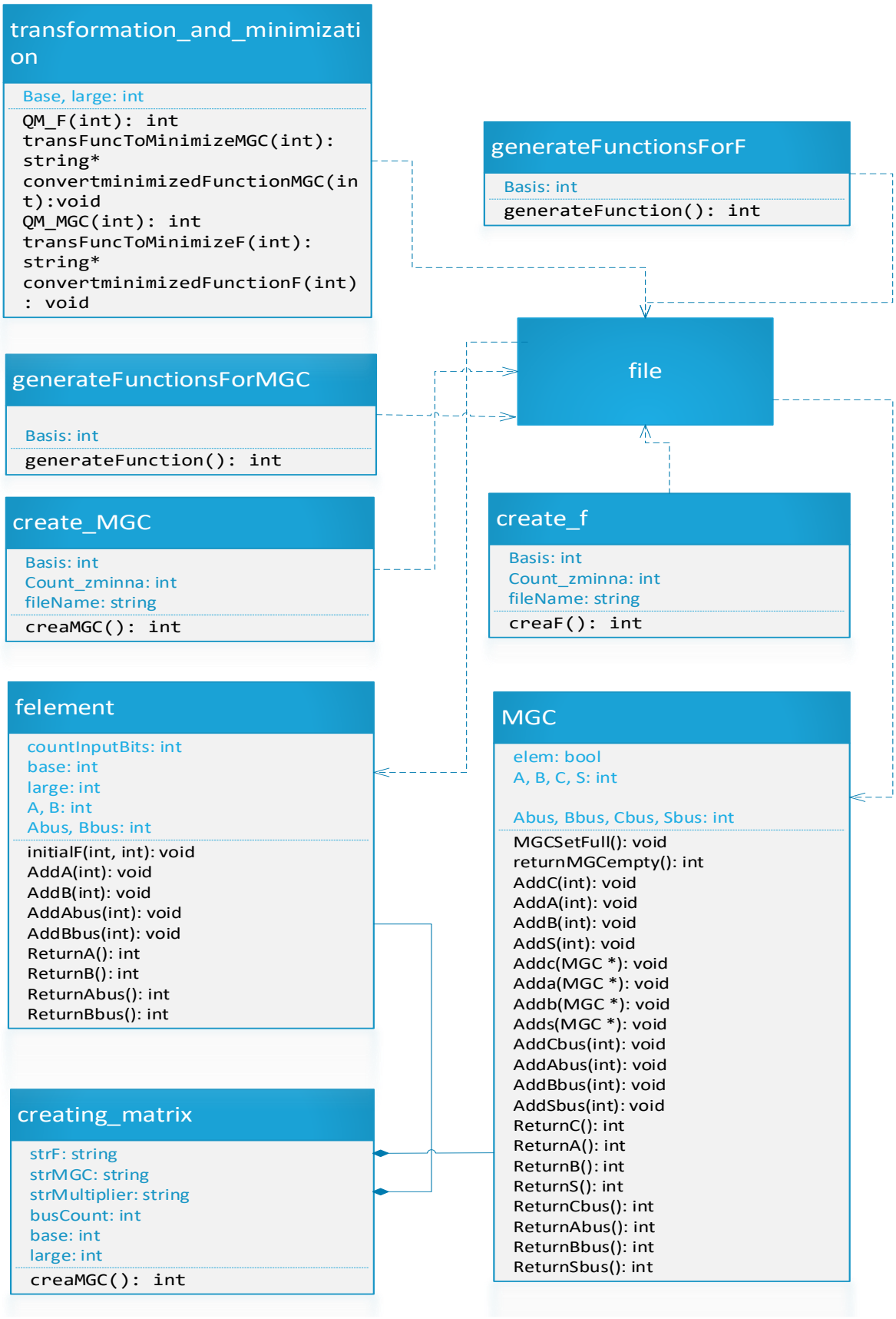


Рис. 3.2 Діаграма класів генератора *VHDL*-описів помножувачів елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ЧС

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity F is
    port(
      A : in STD_LOGIC_VECTOR(Count_zminna downto 0);
      B : out STD_LOGIC_VECTOR(Count_zminna downto 0)
      );
end F;
architecture F of F is
begin
     process(A)
    begin
      B(0) <= Функція ;
      B(1) <= Функція ;
    end process;
end F;
```

3.4.2 Створення МКГ (вузла MGC)

Створення вузла *MGC* дуже подібне до створення вузла *F*. Для нього генерується булева функція, яка для *MGC* є еквівалентом функції $res = (((a * b) \bmod p) + c) \bmod p$, де p – характеристика поля. Розроблено клас для створення МКГ – *generateFunctionsForMGC*, який має відповідний метод *generateFunction()*, який виконує генерування булевих функцій та їх запис у файл (Додаток Д.11-Д.12).

Create_f– це клас, який за допомогою методу *creaMGC()* створює *VHDL*-опис вузла *MGC* за наступним шаблоном:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity MGC is
    port(
      A : in STD_LOGIC_VECTOR(Count_zminna downto 0);
      B : in STD_LOGIC_VECTOR(Count_zminna downto 0);
      C : in STD_LOGIC_VECTOR(Count_zminna downto 0);
      S : out STD_LOGIC_VECTOR(Count_zminna downto 0)
      );
end MGC;
architecture MGC of MGC is
begin
     process(A, B, C)
```

```
    begin
      S(0) <= Function;
      S(1) <= Function;
    end process;
end MGC;
```

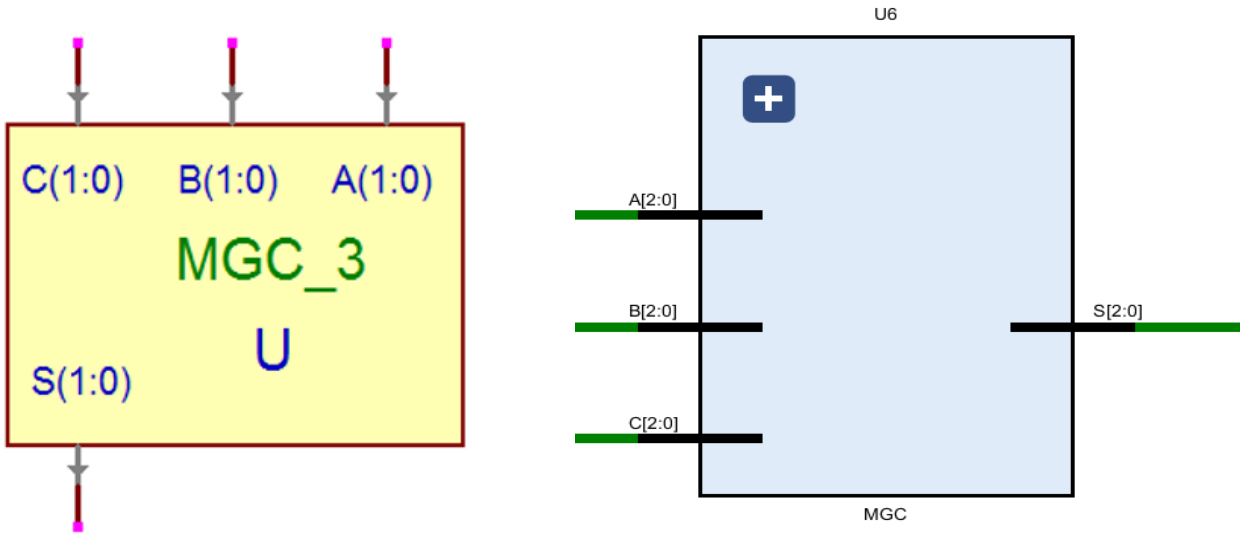
Створення матриці та заповнення логічними вузлами *F* та *MGC*

Класи *felement* та *MGC* описують усі зв'язки, які мають логічні вузли *F* та *MGC* на схемі. Для вузла *F* – це *n*-розрядний вхід *A* та вихід *B*, для елемента *MGC* – це *n*-розрядні входи *A, B, C* та вихід *S* (Додаток Д.7-Д.8).

Клас *Create_matrix* створює матрицю з $2\times[large - 1]^2$ логічних вузлів *MGC* та *[large - 1]* логічних вузлів *F*, де *large* – порядок утворюючого поле полінома. Клас містить 2 методи. Метод *fillMatrix()* проходиться по усіх елементах матриці та встановлює, де потрібно об'єкту значення *true*, тобто він існує або значення *false* - елемент там не потрібний. Встановлюються всі зв'язки між елементами та формується матриця об'єктів точно така ж як і на схемі (Додаток Д.5-Д.6).

Метод *printFile()* друкує згенерований *VHDL*-опис помножувача, який містить вузли *MGC* та *F* та зв'язує їх між собою сигналами.

На рис. 3.3а) зображено МКГ, яка створена у середовищі *Active HDL*, а на рис 3.3б) МКГ створена у середовищі *Xilinx Vivado*.



а) б)

Рис. 3.3 Створена МКГ з структурою ЧС: а) *Aldec Active HDL*, б) *Xilinx Vivado*

3.5 Створення генератора ядер помножувачів елементів розширених полів Галуа *GF(p^n) з* структурою МКГ ФВ

Для структури МКГ ФВ інвертор за модулем *p* – вузол *f* створюється аналогічно, як для архітектури МКГ ЧС. Вузли *MGC* створюються на основі вузлів *MUL* та *SUM*, для яких генеруються булеві функції у класах *generateFunctionForMUL* та *generateFunctionForSUM* відповідно. Ці функції записуються у файл. Після цього функції мінімізуються класом *transformation_and_minimization*. Класи *Create_MUL* та *Create_SUM* генерують *VHDL*-описи вузлів *MUL* та *SUM*. Далі клас *create_matrix* створює матрицю з вузлів *F* та *MGC*. Клас *create_matrix* також встановлює для кожного з об'єктів класів *F* та *MGC* зв'язки з іншими вузлами (рис. 3.4).

Створення вузлів *MUL* та *SUM* дуже подібне до створення вузла *F*. Для кожного з них формується булева функція, яка має вигляд для вузла *MUL – res = (a × b) mod p*, для *SUM – res = (a + b) mod p*, де *p* – характеристика поля. Розроблено 3 класи для їх реалізації *generateFunctionsForMGC*, *generateFunctionsForMUL*, *generateFunctions ForSUM*, які виконують генерування булевих функцій та їх запис у файл (Додаток Е.7- Е.8). МКГ з структурою ФВ у середовищі *Aldec Active HDL* зображена на рис. 3.5, у середовищі *Xilinx Vivado* - на рис 3.6.

Класи *createMUL*, *createSUM* створюють *VHDL*-описи відповідних елементів відповідно до шаблону (Додаток Е.3- Е.4):

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity MUL is
    port(
      A : in STD_LOGIC_VECTOR(2 downto 0);
      B : in STD_LOGIC_VECTOR(2 downto 0);
      S : out STD_LOGIC_VECTOR(2 downto 0)
      );
end MUL;
architecture MUL of MUL is
```

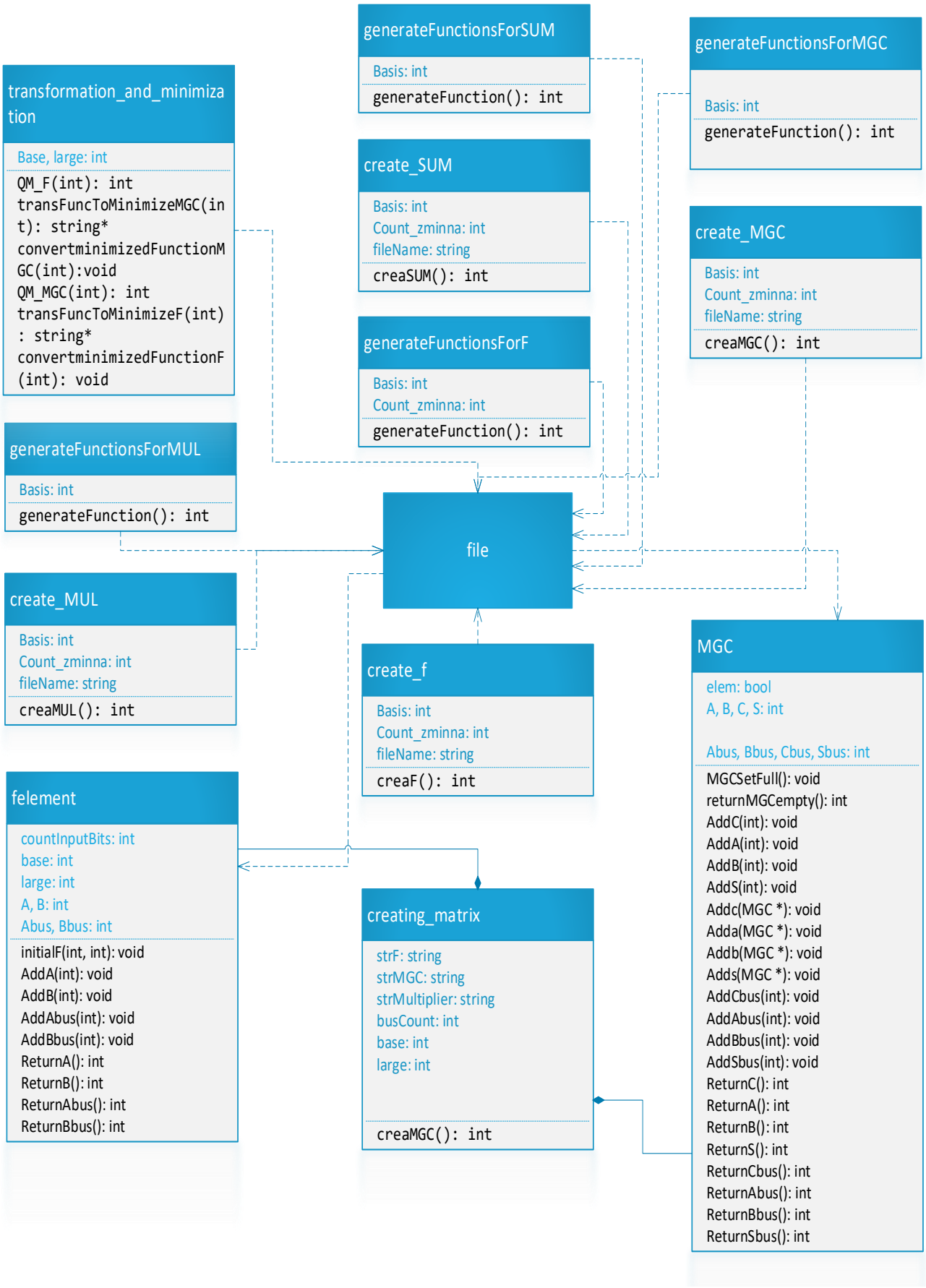


Рис. 3.4 Діаграма класів генератора ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ФВ

```
begin
        process(A, B)
      begin
        S(0) <= функція;
        S(1) <= функція;
        S(2) <= функція;
      end process;
end MUL;

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity SUM is
      port(
        A : in STD_LOGIC_VECTOR(2 downto 0);
        B : in STD_LOGIC_VECTOR(2 downto 0);
        S : out STD_LOGIC_VECTOR(2 downto 0)
        );
end SUM;
architecture SUM of SUM is
begin
        process(A, B)
      begin
        S(0) <= функція;
        S(1) <= функція;
        S(2) <= функція;
      end process;
end SUM;
```

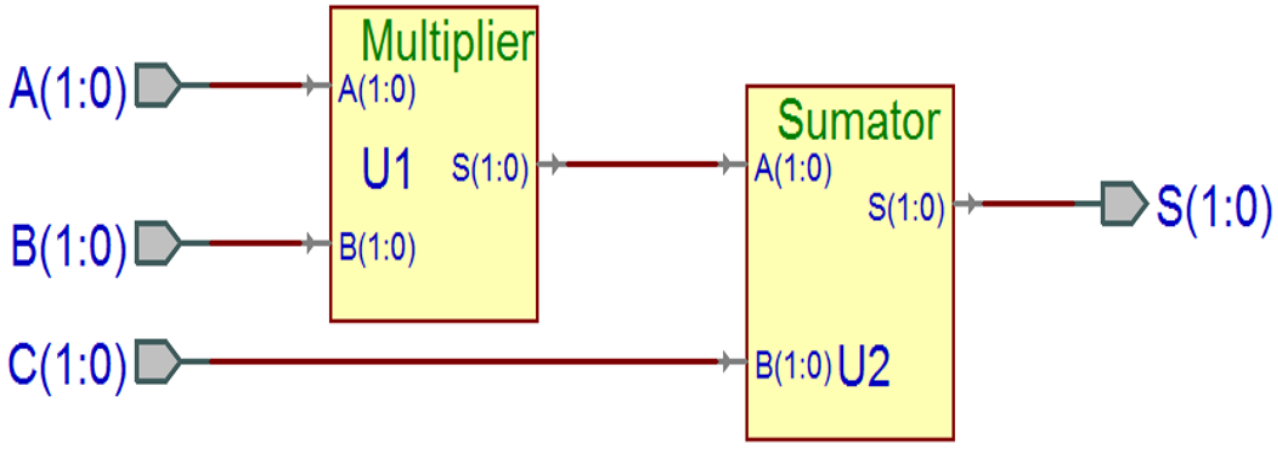


Рис. 3.5 Реалізація МКГ з структурою на основі функціональних вузлів в середовищі *Aldec Active HDL*

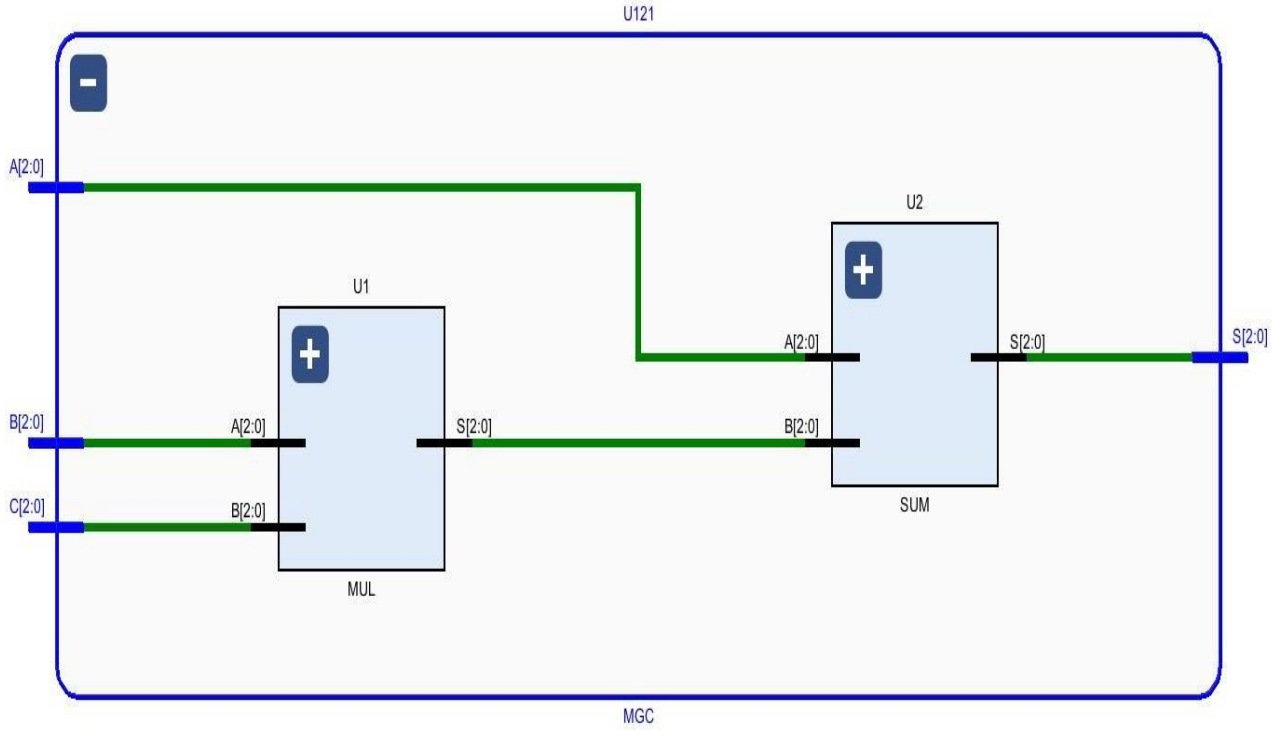


Рис. 3.6 Реалізація МКГ з структурою на основі функціональних вузлів в середовищі *Xilinx Vivado*

3.6 Створення генератора ядер помножувачів елементів розширених полів Галуа *GF(p^n) з* структурою МКГ ЛВ

Структура МКГ ЛВ дуже сильно відрізняється від структур МКГ ЧС та ФВ, проте підхід до створення самого помножувача такий самий. У цьому алгоритмі використовуються класи *generateFunctionForSMch*, *generateFunctionForSMn*, *generateFunctionForSn*, *generateFunctionForRn*, *generateFunctionForSUM*. Клас *transformation_and_minimization* мінімізує булеві функції. Клас *Create_MGC* створює на основі булевих функцій, згенерованих елементів вузол *MGC*. Далі на основі створеної МКГ формується помножувач. Діаграма класів генератора ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ЛВ зображена на рис. 3.7. Створена у середовищі *Xilinx Vivado* МКГ зображена на рис. 3.15.

3.7 Генерування помножувачів

Для інтеграції, згенерованих помножувачів, у розширених полях Галуа $GF(p^n)$ в ПЛІС було вибрано середовище розробки *Xilinx Vivado*.

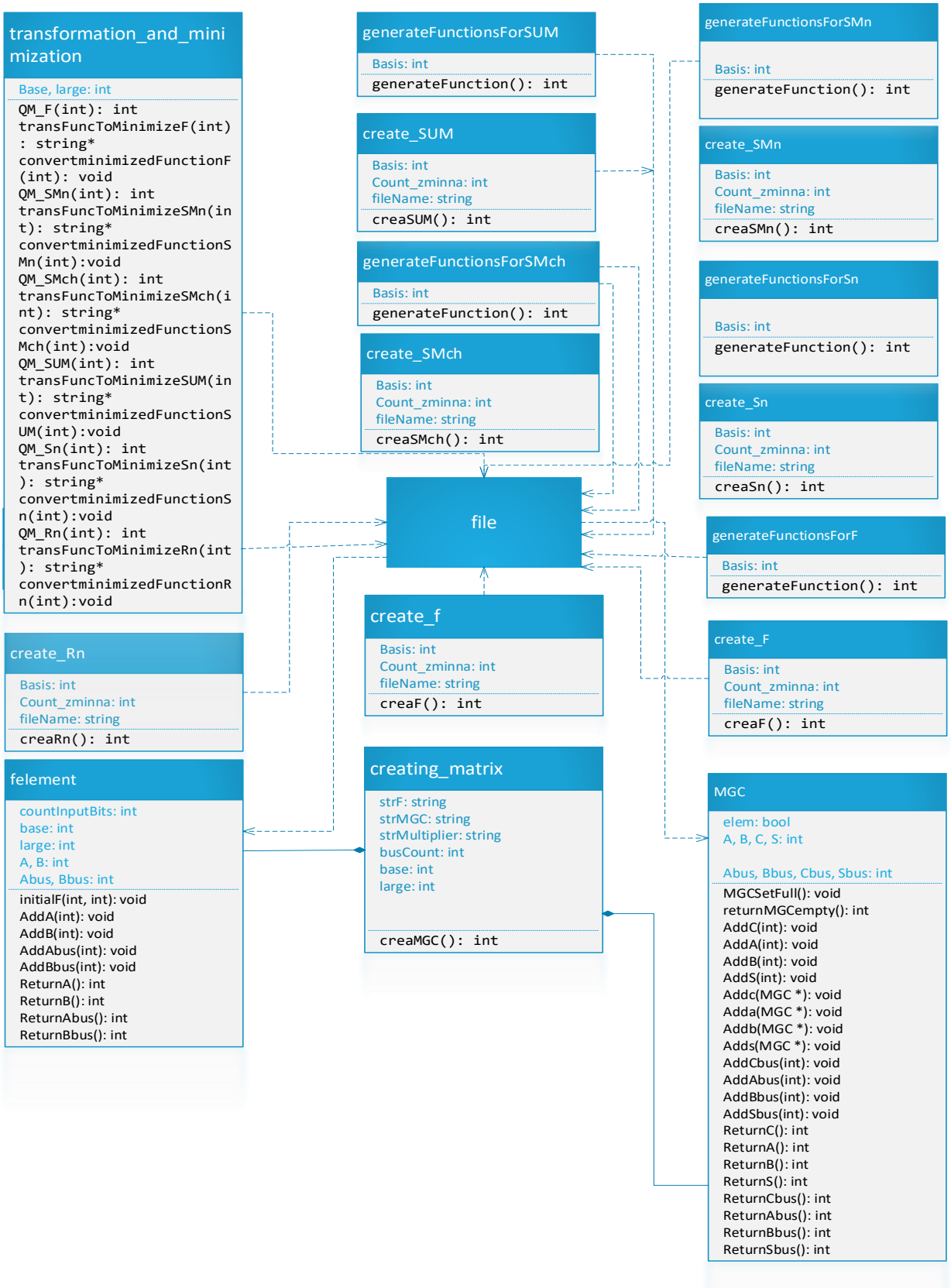


Рис. 3.7 Діаграма класів генератора ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ЛВ

На рис. 3.8 зображено інтерфейс генераторів помножувачів елементів розширених полів Галуа $GF(p^n)$. Синтезовані схеми використовуються в середовищі *Xilinx Vivado* для проектування топології кристалу ПЛІС та визначення уточнених апаратних та часових параметрів помножувачів. З рис. 3.8 видно, що користувач може обирати 1 з 3 генераторі.

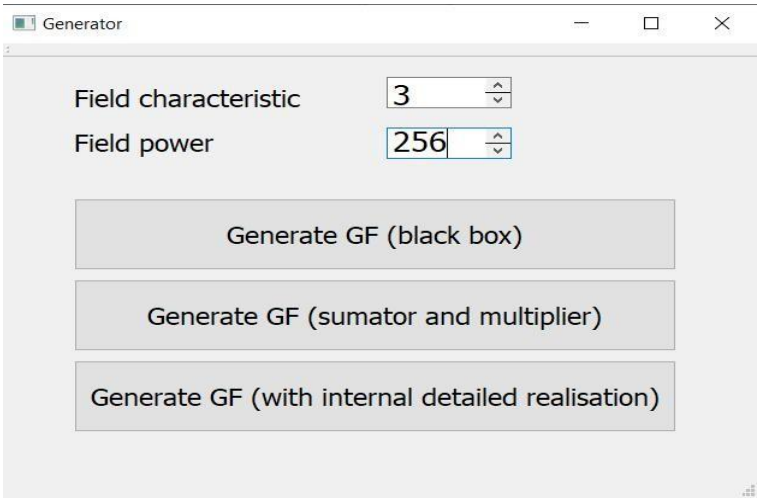


Рис. 3.8 Інтерфейс генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$

Генератором було згенеровано ряд помножувачів і проаналізовано синтезовані схеми в середовищі *Xilinx Vivado*. Проаналізуємо, наприклад, згенерований помножувач $GF(3^3)$, при реалізації МКГ ЧС (рис. 3.9) та помножувач $GF(7^5)$ при реалізації МКГ ФВ (рис. 3.11). На рис. 3.9 наведена схему помножувача, що складається з 15 МКГ і 2 вузлів F. З рис. 3.10 видно, що МКГ складається з логічних елементів. Апаратна складність МКГ зростає при збільшенні характеристики поля.

На рис. 3.11 наведено схему помножувача $GF(7^5)$, синтезовану у середовищі *Xilinx Vivado*. На рис. 3.15 наведена схема МКГ ЛВ, на рис. 3.12 – схема помножувача цієї МКГ, на рис. 3.13 – схема суматора цієї МКГ. На рис. 3.14 зображено МКГ ЛВ для поля $GF(3^{16})$.

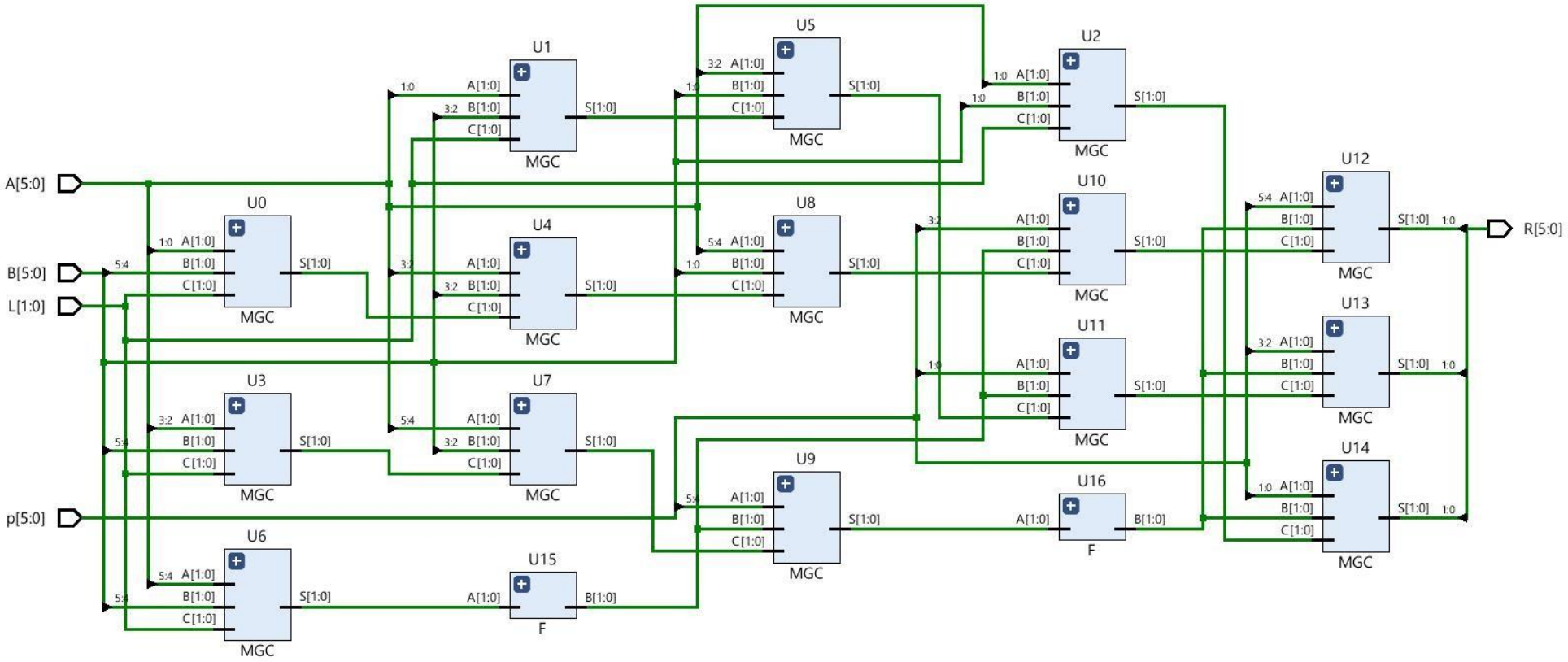


Рис. 3.9 Схема помножувача $GF(3^3)$. з МКГ ЧС

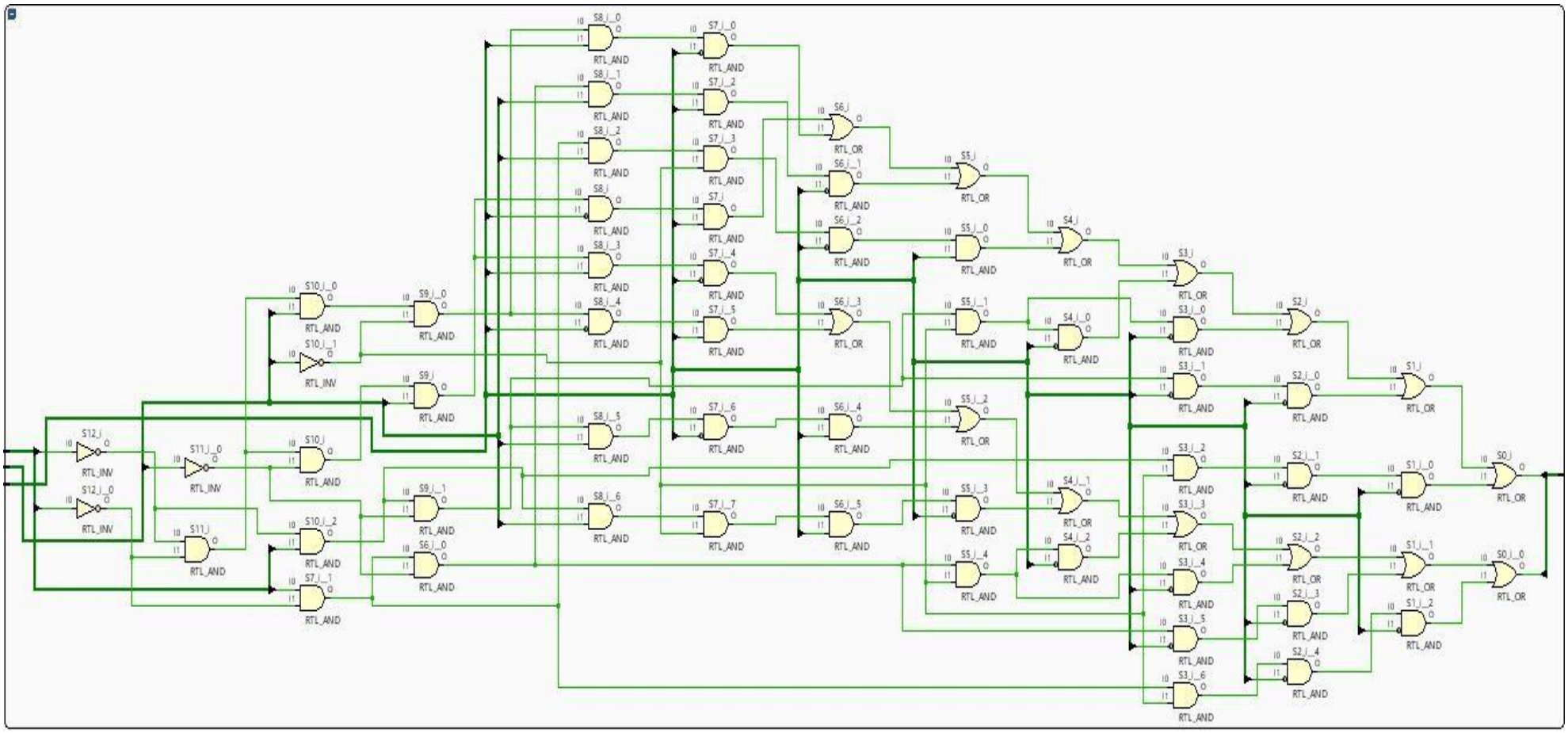


Рис. 3.10 Схема МКГ ЧС $GF(3^3)$

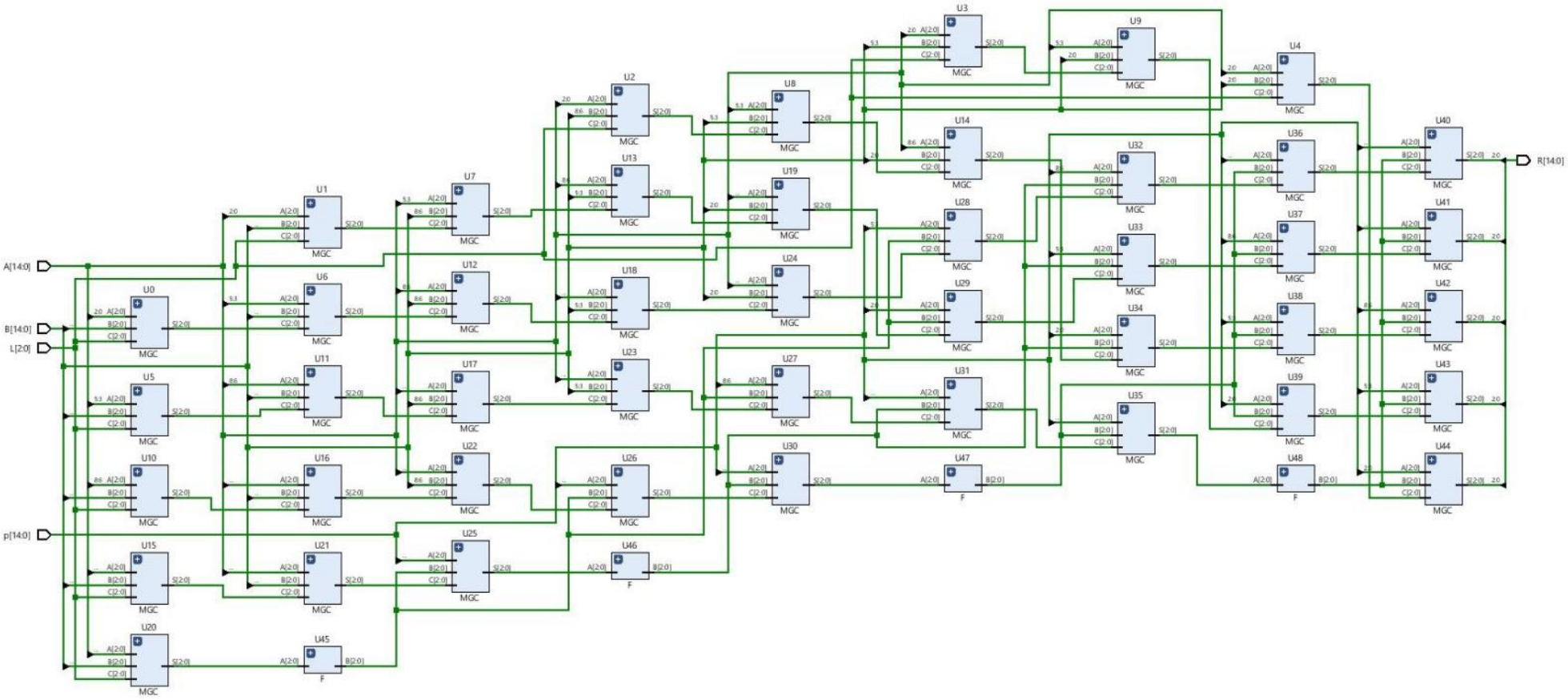


Рис. 3.11 Схема помножувача $GF(7^5)$ з МКГ ФВ

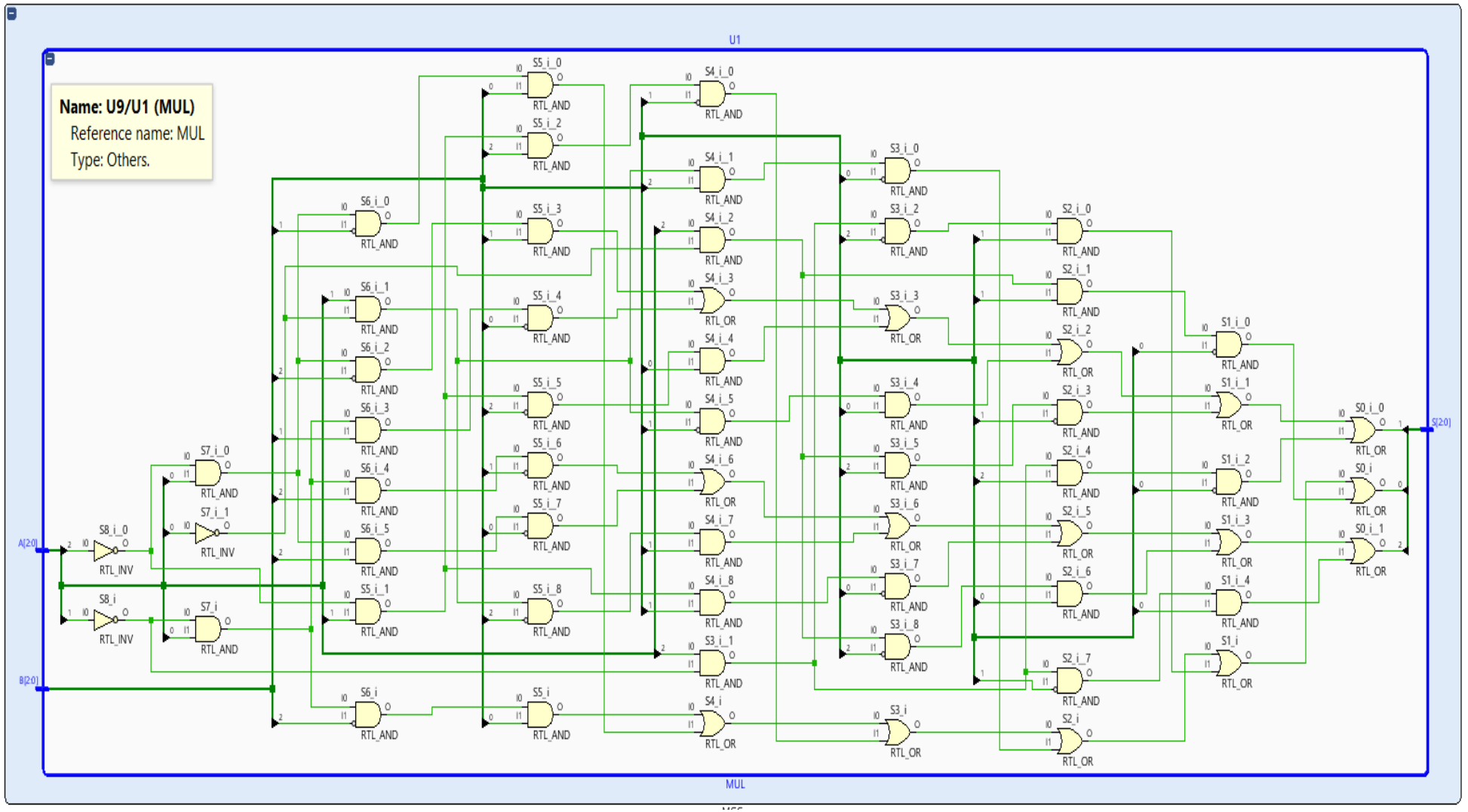


Рис. 3.12 Схема помножувача МКГ ФВ поля $GF(7^5)$

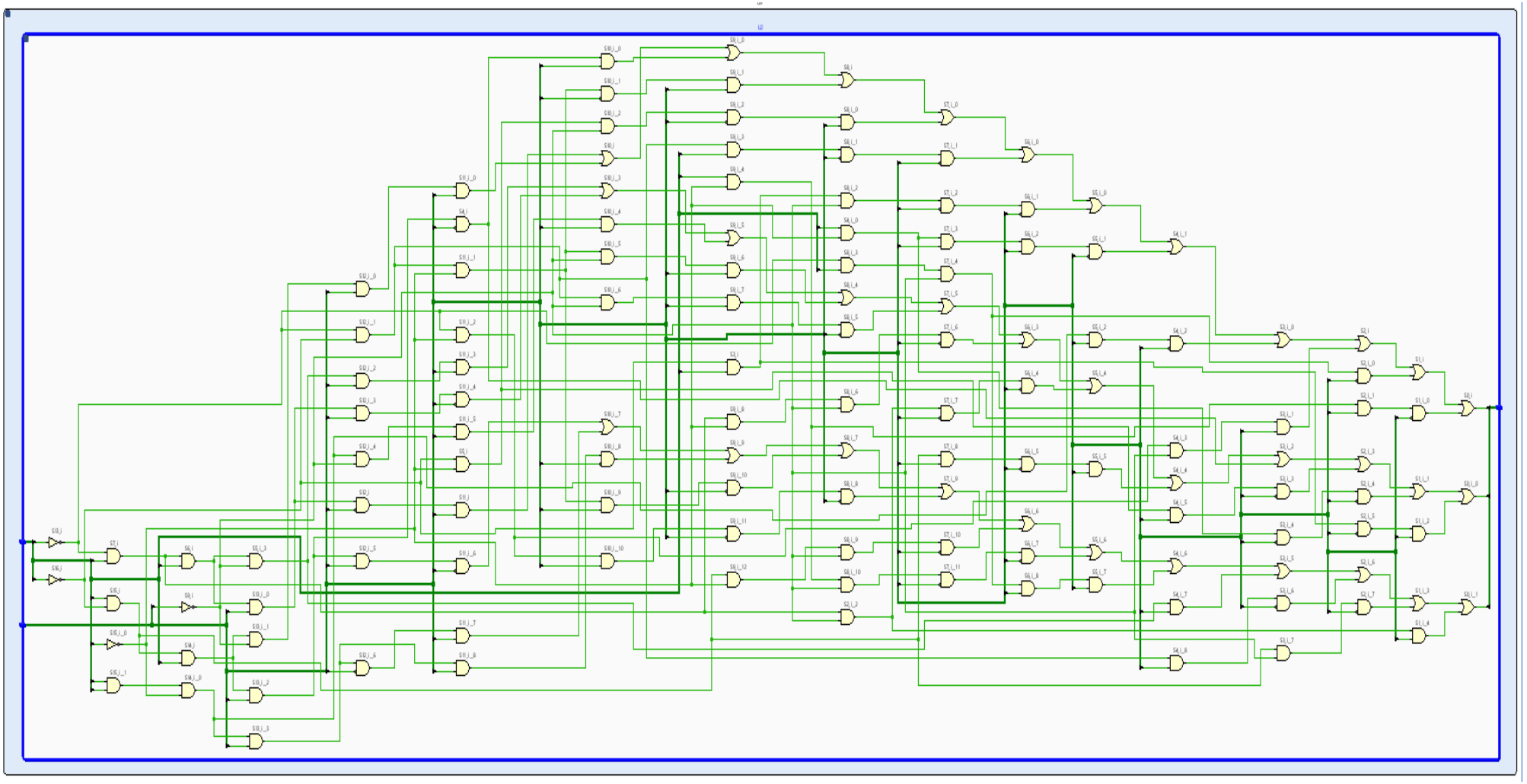


Рис. 3.13 Схема суматора МКГ ФВ поля $GF(7^5)$

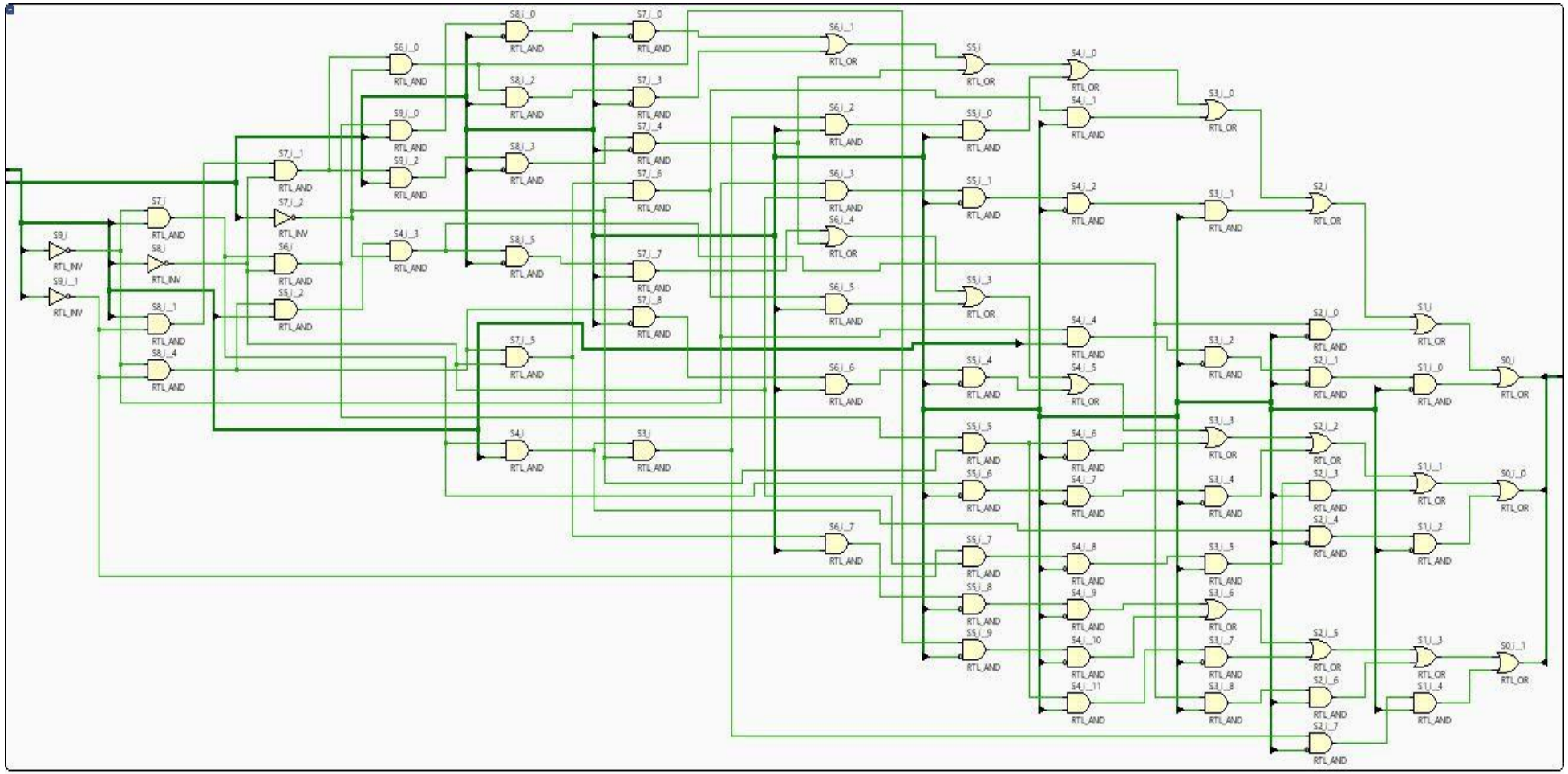


Рис. 3.14 Схема МКГ ЛВ поля $GF(3^{16})$

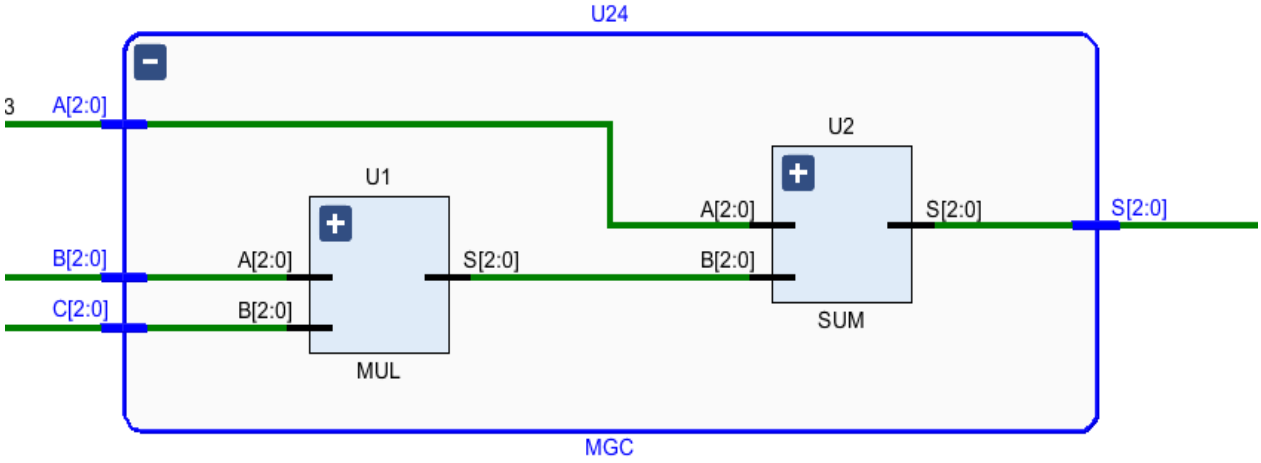


Рис. 3.15 Схема МКГ ФВ $GF(7^5)$

3.8 Часові затрати при генеруванні ядер помножувачів елементів розширених полів Галуа *GF(p^n)*

Генерування помножувачів елементів розширених полів Галуа $GF(p^n)$ з великими характеристиками поля та порядками є дуже часозатратною задачею. Найбільше часу при побудові МКГ займає процес мінімізації булевих функцій. У табл. 3.2 наведено часові затрати на генерування помножувачів за трьома структурами МКГ. Наприклад, генерація помножувача для поля $GF(53^{174})$ триває 1845 с.

Таблиця 3.2.

Часові затрати при генеруванні ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ з простою характеристикою поля

Поле	МКГ ЧС, секунд	МКГ ФВ, секунд	МКГ ЛВ, секунд
$GF(2^{998})$	154	90	45
$GF(3^{630})$	186	103	47
$GF(5^{430})$	216	120	46
$GF(7^{350})$	245	130	53
$GF(11^{280})$	283	153	56

$GF(13^{270})$	320	170	57
$GF(17^{240})$	381	202	58
$GF(19^{230})$	432	234	57
$GF(23^{220})$	473	256	60
$GF(31^{201})$	567	268	60
$GF(37^{191})$	756	357	63
$GF(41^{186})$	1245	645	62
$GF(43^{184})$	1305	710	64
$GF(47^{180})$	1567	785	59
$GF(53^{174})$	1845	956	63

3.9 Тестування згенерованих помножувачів та генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$

Тестування помножувачів відбувалося за розробленим методом (п. 2.12).

Для тестування згенерованих помножувачів було обрано бібліотеку *GaloisCPP* та математичний пакет *Maple*.

GaloisCPP – це бібліотека *C++* для виконання арифметичних операцій у полях Галуа. Ця бібліотека підтримує арифметику над кінцевим полем $GF(p^n)$, де p є простим числом, а $n \geq 1$, і робить це відносно ефективно, покладаючись на таблиці додавання та множення.

Maple - це комп'ютерна система алгебраїчних обчислень та система комп'ютерної алгебри. Вона використовується для вирішення різних математичних завдань, проведення алгебраїчних обчислень, символьного та числового обчислення, а також для виконання аналітичних досліджень і моделювання математичних та інженерних систем.

Детально покажемо процес перевірки правильності роботи помножувача для поля $GF(53^8)$. Знаходження незвідного полінома поля проводимо у математичному пакеті *Maple*. Множення елементів проводимо у математичному пакеті *Maple* та за допомогою бібліотеки *GaloisCPP*. Потім ці результати звіряємо

з результатами множення у середовищі *Active HDL*.

Знайдемо незвідний поліном для поля $GF(53^8)$. Поліном буде мати такий вигляд:

$$p = \alpha^8 + 2 \tag{3.1}$$

Тепер представимо цей поліном у вигляді елемента поля:

$$[2, 0, 0, 0, 0, 0, 0, 0, 1] \tag{3.2}$$

Виконаємо множення кількох елементів поля $GF(53^8)$. Виберемо перше число і позначимо літерою *d:*

$$d10 := 62259690411357$$

$$d16 := 389FF6BB155D$$

$$d2 := 1110001001111111110110101110110001010101011101$$

$$dChr := [49, 52, 52, 52, 52, 52, 52, 52]$$

$$c := (52\alpha^7 + 52\alpha^6 + 52\alpha^5 + 52\alpha^4 + 52\alpha^3 + 52\alpha^2 + 52\alpha + 49) mod 53$$

Виберемо друге число та позначимо літерою *f*:

$$f10 := 62259690411353$$

$$f16 := 389FF6BB1559$$

$$f2 := 1110001001111111110110101110110001010101011001$$

$$fChr := [45, 52, 52, 52, 52, 52, 52, 52]$$

$$g := (52\alpha^7 + 52\alpha^6 + 52\alpha^5 + 52\alpha^4 + 52\alpha^3 + 52\alpha^2 + 52\alpha + 45) mod 53$$

Результатом множення елементів d та f поля $GF(53^8)$ буде значення r:

$$r10 := 21482356195953$$

$$r16 := 1389C035C271$$

$$r2 := 1001110001001110000000011010111000010011100001$$

$$rChr := [18, 0, 3, 6, 9, 12, 15, 18]$$

$$g := (18\alpha^7 + 15\alpha^6 + 12\alpha^5 + 9n\alpha^4 + 6\alpha^3 + 3\alpha^2 + 18) mod 53$$

Результат моделювання роботи згенерованого помножувача у середовищі *Active HDL* показаний на рис. 3.16.

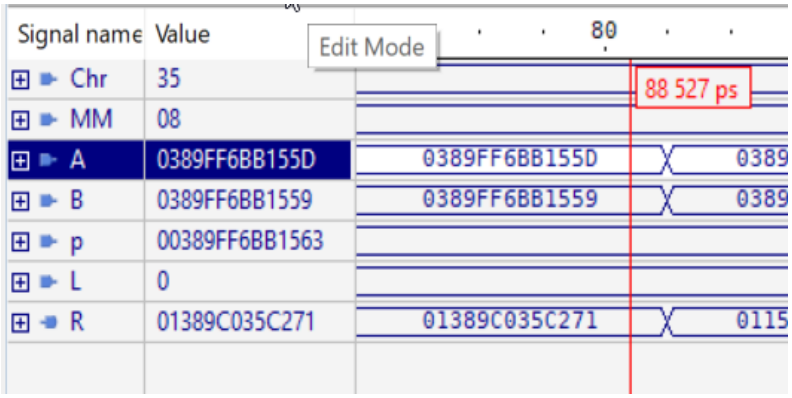


Рис. 3.16 Моделювання роботи згенерованого помножувача елементів розширеного поля Галуа $GF(53^8)$

Проведемо множення 2 наступних елементів d та f поля $GF(53^8)$. Отже:

$$d10 := 62259690411358$$

$$d16 := 389FF6BB155E$$

$$d2 := 11100010011111111101101011101100010101011110$$

$$dChr := [50, 52, 52, 52, 52, 52, 52, 52]$$

$$c := (52\alpha^7 + 52\alpha^6 + 52\alpha^5 + 52\alpha^4 + 52\alpha^3 + 52\alpha^2 + 52\alpha + 50) mod 53$$

Елемент f:

$$f10 := 62259690411354$$

$$f16 := 389FF6BB155A$$

$$f2 := 11100010011111111101101011101100010101011010$$

$$fChr := [46, 52, 52, 52, 52, 52, 52, 52]$$

$$g := (52\alpha^7 + 52\alpha^6 + 52\alpha^5 + 52\alpha^4 + 52\alpha^3 + 52\alpha^2 + 52\alpha + 46) mod 53$$

Результат множення $d \times f$:

$$r10 := 19087752721393$$

$$r16 := 115C36B873F1$$

$$r2 := 10001010111000011011010111000011100111111110001$$

$$rChr := [7, 51, 1, 4, 7, 10, 13, 16]$$

$$r := (16\alpha^7 + 13\alpha^6 + 10\alpha^5 + 7\alpha^4 + 4\alpha^3 + \alpha^2 + 51\alpha + 7) mod 53$$

Результат моделювання роботи згенерованого помножувача у середовищі *Active HDL* показаний на рис. 3.17.

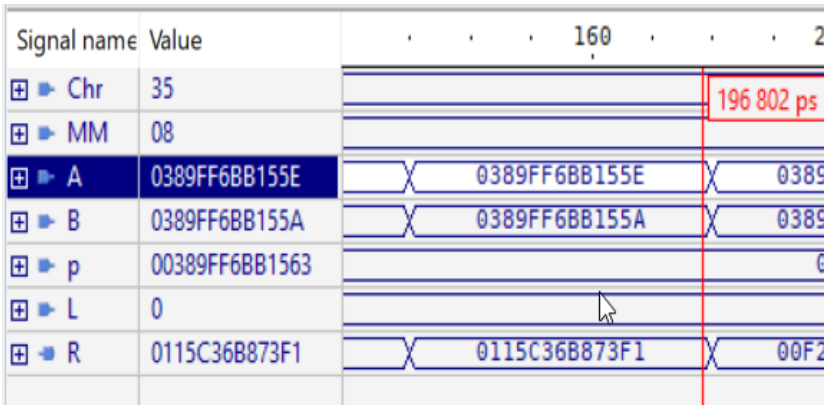


Рис. 3.17 Моделювання роботи згенерованого помножувача елементів розширеного поля Галуа $GF(53^8)$

Наступні:

$$\text{d10} := 62259690411359$$

$$\text{d16} := \text{389FF6BB155F}$$

$$\text{d2} := 1110001001111111110110101110110001010101011111$$

$$\text{dChr} := [51, 52, 52, 52, 52, 52, 52, 52]$$

$$c := (52\alpha^7 + 52\alpha^6 + 52\alpha^5 + 52\alpha^4 + 52\alpha^3 + 52\alpha^2 + 52\alpha + 51) mod 53$$

$$\text{f10} := 62259690411355$$

$$\text{f16} := \text{389FF6BB155B}$$

$$\text{f2} := 1110001001111111110110101110110001010101011011$$

$$g = (52\alpha^7 + 52\alpha^6 + 52\alpha^5 + 52\alpha^4 + 52\alpha^3 + 52\alpha^2 + 52\alpha + 47) mod 53$$

$$\text{r10} := 16693149392956$$

$$\text{r16} := \text{F2EAD3D603C}$$

$$\text{r2} := 111100101110101011010011110101100000000111100$$

$$\text{rChr} := [51, 49, 52, 2, 5, 8, 11, 14]$$

$$r := (14\alpha^7 + 11\alpha^6 + 8\alpha^5 + 5\alpha^4 + 2\alpha^3 + 52\alpha^2 + 49\alpha + 51) mod 53$$

Результат моделювання роботи згенерованого помножувача у середовищі *Active HDL* показаний на рис. 3.18.

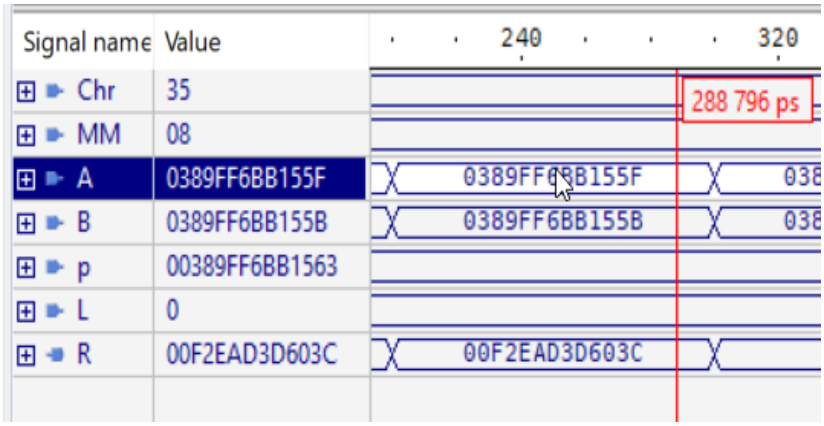


Рис. 3.18 Результат моделювання роботи згенерованого помножувача елементів розширеного поля Галуа $GF(53^8)$

Наступні:

$$d10 := 62259690411360$$

$$d16 := 389FF6BB1560$$

$$d2 := 111000100111111111011010111011000101010110000 0$$

$$dChr := [52, 52, 52, 52, 52, 52, 52, 52]$$

$$c := (52\alpha^7 + 52\alpha^6 + 52\alpha^5 + 52\alpha^4 + 52\alpha^3 + 52\alpha^2 + 52\alpha + 52) mod 53$$

$$f10 := 62259690411356$$

$$f16 := 389FF6BB155C$$

$$f2 := 111000100111111111011010111011000101010101110 0$$

$f_{Chr} := [48, 52, 52, 52, 52, 52, 52, 52]$

$$g := (52\alpha^7 + 52\alpha^6 + 52\alpha^5 + 52\alpha^4 + 52\alpha^3 + 52\alpha^2 + 52\alpha + 48) mod 53$$

$$r10 := 14298545915591$$

$$r16 := D0123C006C7$$

$$r2 := 110100000001001000111100000000000110110001 11$$

$$rChr := [44, 47, 50, 0, 3, 6, 9, 12]$$

$$r := (12\alpha^7 + 9\alpha^6 + 6\alpha^5 + 3\alpha^4 + 50\alpha^2 + 47\alpha + 44) mod 53$$

Результат моделювання роботи згенерованого помножувача у середовищі *Active HDL* показаний на рис. 3.19.

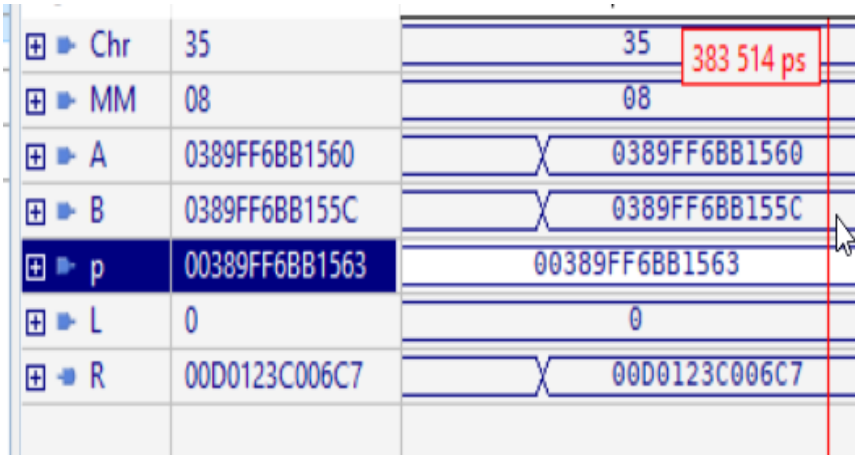


Рис. 3.19 Результат моделювання роботи згенерованого помножувача елементів розширеного поля Галуа $GF(53^8)$

На рис. 3.20 показано моделювання роботи згенерованого помножувача з розгорнутим побітовим представленням змінної R, тобто результату. На рис. 3.21 показаний результат множення елементів розширеного поля Галуа $GF(53^8)$ у математичному пакеті *Maple*.

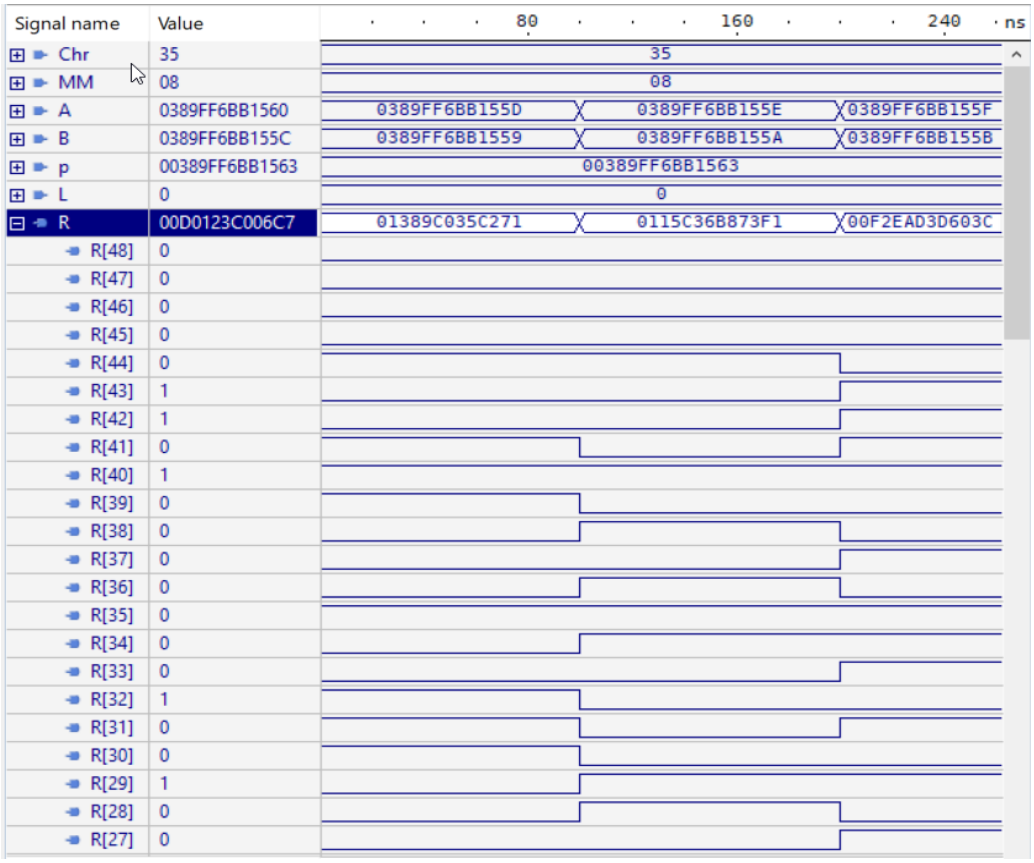


Рис. 3.20 Результат множення елементів розширеного поля Галуа $GF(53^8)$ *з* розгорнутим побітовим представленням змінної R

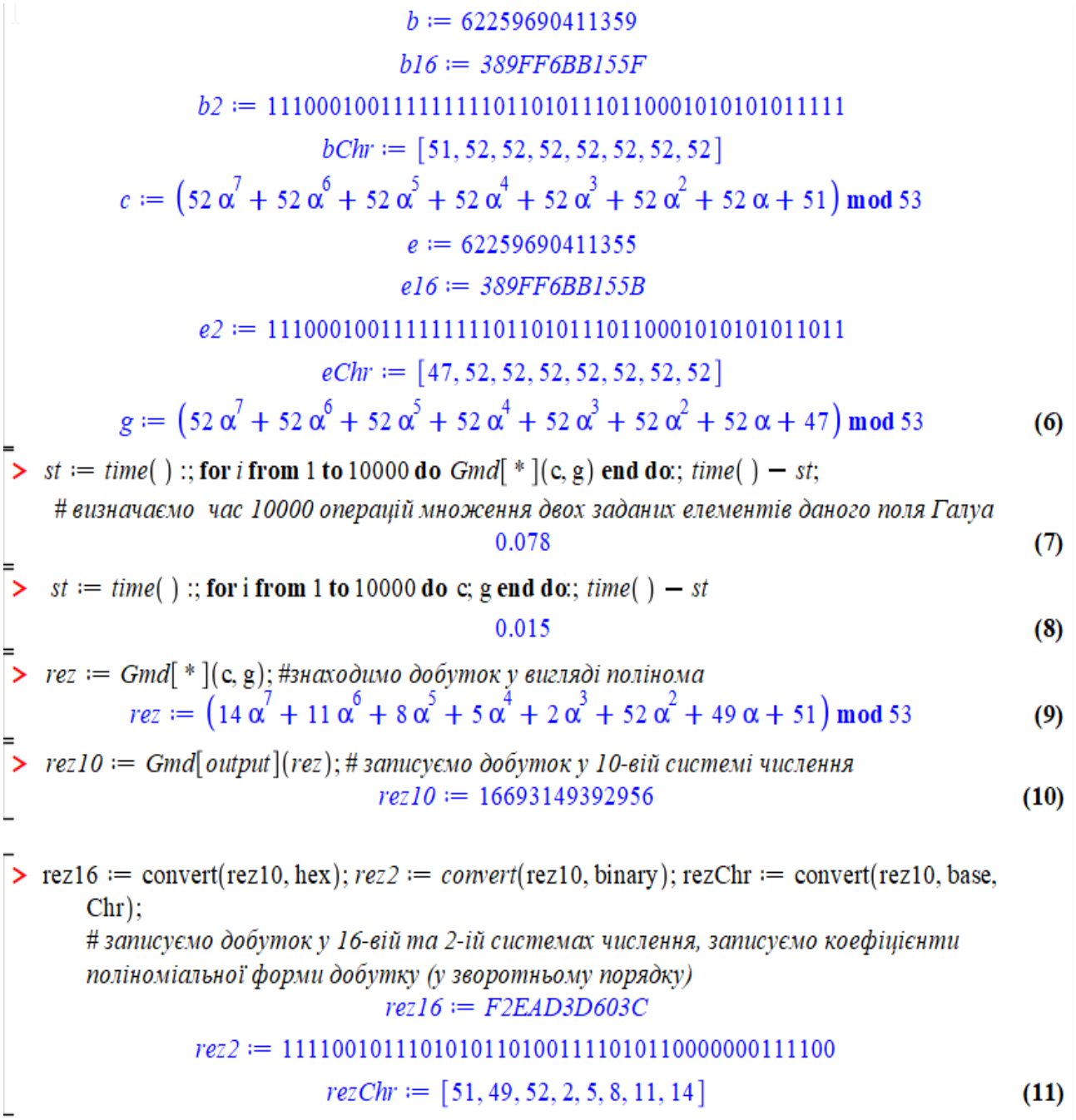


Рис. 3.21 Результат множення елементів розширеного поля Галуа $GF(53^8)$ у математичному пакеті *Maple*

Результати множення елементів поля $GF(3^{30})$ показано у таблиці 3.3. Незвідний поліном для цього поля представлений формулою 3.

$$p := \alpha^{30} + \alpha + 2 \tag{3.3}$$

Незвідний поліном поля $GF(3^{30})$ представлений у форматі поля має наступний вигляд:

$$\mathrm{p} := [2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$$

$$\text{p10} := 205891132094654$$

$$\text{p16} := \text{BB41C3CA78BE}$$

Таблиця 3.3.

Результати множення елементів розширеного поля Галуа $GF(3^{30})$

По-ле			Результат
$GF(3^{30})$		d	$d10 := 205891132094645$ $d16 := BB41C3CA78B5$ $dChr := [2,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2]$
		f	$f10 := 205891132094641$ $f16 := BB41C3CA78B1$ $fChr := [1,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2]$
			$r10 := 55432227871648$ $r16 := 326A52405BA0$ $rChr :$ $= [1,0,2,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0,1,2,0]$
		d	$d10 := 205891132094249$ $d16 := BB41C3CA7729$ $dChr := [2,1,0,0,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2]$
		f	$f10 := 205891132094569$ $f16 := BB41C3CA7869$ $fChr := [1,0,0,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2]$
		r	$r10 := 87107786650541$ $r16 := 4F395CAE33AD$

			$rChr := [2,0,1,1,0,2,1,1,0,2,0,1,2,0,1,2,0,1,2,0,$ $1,2,0,1,2,0,1,2,0,1]$
		d	$d10 := 205891132094609$ $d16 := BB41C3CA7891$ $dChr := [2,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,$ $2,2,2,2,2,2,2,2,2,2]$
		f	$f10 := 205891132094579$ $f16 := BB41C3CA7873$ $fChr :$ $= [2,0,1,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,$ $2,2,2,2,2,2,2,2,2,2]$
		r	$r10 := 87107786654864$ $r16 := 4F395CAE4490$ $rChr :$ $= [2,1,1,2,2,1,1,0,1,2,0,1,2,0,1,2,0,1,2,0,1,$ $2,0,1,2,0,1,2,0,1]$
		d	$d10 := 205891131205761$ $d16 := BB41C3BCE881$ $dChr :$ $= [0,2,0,0,0,2,1,1,2,2,2,0,1,2,2,2,2,2,2,2,2,$ $2,2,2,2,2,2,2,2,2]$
		f	$f10 := 205891131384650$ $f16 := BB41C3BFA34A$ $fChr :$ $= [2,0,2,1,0,0,1,2,2,2,2,1,1,2,2,2,2,2,2,2,2,2,$ $2,2,2,2,2,2,2,2]$
		r	$r10 := 86606811206159$ $r16 := 4EC4B837920F$

			$rChr:$ $= [2, 1, 0, 0, 0, 1, 2, 0, 2, 1, 0, 2, 0, 2, 1, 2, 2, 0, 2, 1, 1,$ $2, 2, 1, 0, 0, 1, 2, 0, 1]$

Результати множення елементів поля $GF(7^{17})$ показано у таблиці 3.4. Незвідний поліном для цього поля представлений формулою 4.

$$p := \alpha^{17} + \alpha + 3 \tag{3.4}$$

Незвідний поліном поля *GF(7[17])* представлений у форматі поля має наступний вигляд:

$$p := [3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$$

$$p10 := 232630513987217$$

$$p16 := D39383266E91$$

Таблиця 3.4.

Результати множення елементів розширеного поля Галуа $GF(7^{17})$

Поле			Результат
$GF(7^{17})$		d	$d10 := 232630513987119$ $d16 := D39383266E2F$ $dChr := [3, 1, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]$
		f	$f10 := 232630513987137$ $f16 := D39383266E41$ $fChr := [0, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]$
		r	$r10 := 206669594706647$ $r16 := BBF703C742D7$ $rChr := [2, 6, 2, 4, 3, 0, 5, 3, 1, 6, 4, 2, 0, 5, 3, 1, 6]$
		d	$d10 := 232630513898319$ $d16 := D3938325134F$ $dChr := [5, 6, 5, 6, 4, 1, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]$
		f	$f10 := 232630513916430$ $f16 := D39383255A0E$

			$fChr := [0,4,4,3,5,2,6,6,6,6,6,6,6,6,6,6]$
		r	$r10 := 50903772838622$ $r16 := 2E4BF54C3ADE$ $rChr := [4,4,1,4,6,6,1,4,6,0,5,4,2,0,5,3,1]$
		d	$d10 := 232629625098319$ $d16 := D3934E2B104F$ $dChr := [1,3,1,6,6,3,4,5,6,5,3,6,6,6,6,6,6]$
		f	$f10 := 232559736209430$ $f16 := D38308774016$ $fChr := [0,4,4,6,4,5,6,2,0,3,1,6,1,6,6,6,6]$
		r	$r10 := 45256645768854$ $r16 := 2929224B7696$ $rChr := [2,6,0,6,1,6,4,0,6,3,5,4,0,5,3,2,1]$
		d	$d10 := 232630513987206$ $d16 := D39383266E86$ $dChr := [6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6]$
		f	$f10 := 232630513987200$ $f16 := D39383266E80$ $fChr := [0,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6]$
		r	$r10 := 50904079007615$ $r16 := 2E4C078BFF7F$ $rChr := [1,3,1,6,4,2,0,5,3,1,6,4,2,0,5,3,1]$

Результати множення елементів поля $GF(23^{10})$ показано у таблиці 3.5. Незвідний поліном для цього поля представлений формулою 5.

$$p := \alpha^{10} + \alpha + 7 \tag{3.5}$$

Незвідний поліном поля *GF(23[10])* представлений у форматі поля має наступний вигляд:

$$\mathrm{p} := [7,1,0,0,0,0,0,0,0,0,1]$$

$$p10:=41426511213679$$

$$p16:=25AD\ 5C7C\ 346F$$

Таблиця 3.5.

Результати множення елементів розширеного поля Галуа $\boldsymbol{GF(23^{10})}$

Поле			Результат
$GF(23^{10})$		d	d_{10}:=41426511213648 d_{16}:=25AD5C7C3450 d_{Chr}:=[22, 22, 22, 22, 22, 22, 22, 22, 22, 22]
		f	f_{10}:=41426511213642 f_{16}:=25AD5C7C344A f_{Chr}:=[16, 22, 22, 22, 22, 22, 22, 22, 22, 22]
		r	r_{10}:=27556898184059 r_{16}:=191017209B7B r_{Chr}:=[13, 12, 21, 7, 16, 2, 11, 20, 6, 15]
		d	d_{10}:=41426511212650 d_{16}:=25AD5C7C306A d_{Chr}:=[13, 2, 21, 22, 22, 22, 22, 22, 22, 22]
		f	f_{10}:=41426511212872 f_{16}:=25AD5C7C3148 f_{Chr}:=[5, 12, 21, 22, 22, 22, 22, 22, 22, 22]
		r	r_{10}:=38776720545957 r_{16}:=234468A758A5 r_{Chr}:=[8, 9, 21, 20, 0, 8, 17, 3, 12, 21]
		d	d_{10}:=41426501213650 d_{16}:=25AD5BE39DD2 d_{Chr}:=[10, 9, 2, 6, 10, 21, 22, 22, 22, 22]
		f	f_{10}:=41426503435872

			f_{16}:=25AD5C058660 f_{Chr}:=[18, 4, 17, 4, 18, 21, 22, 22, 22, 22]
		r	r_{10}:=21557968287435 r_{16}:=139B5B0AB6CB r_{Chr}:=[21, 18, 20, 13, 13, 14, 13, 6, 22, 11]
		d	d_{10}:=41416511213650 d_{16}:=25AB08705052 d_{Chr}:=[8, 7, 16, 9, 7, 10, 1, 20, 22, 22]
		f	f_{10}:=41418733435872 f_{16}:=25AB8CE4BBE0 f_{Chr}:=[19, 10, 2, 10, 13, 10, 16, 20, 22, 22]
		r	r_{10}:=4844037855400 r_{16}:=467D727D8A8 r_{Chr}:=[3, 6, 1, 18, 3, 1, 16, 19, 15, 2]

3.10 Конвеєризація комбінаційних помножувачів

Згенеровані генераторами схеми є комбінаційними. Частота *F1* зміни операндів на входах помножувача визначається часом *T1* найбільшої затримки проходження сигналів від входу до виходу – *F1 = 1/T1*. Якщо не брати до уваги вузли *F*, то затримку будуть визначати тільки МКГ та їх кількість у найдовшому ланцюжку. Наприклад, у комбінаційній схемі помножувача для поля $GF(p^3)$ (рис. 3.22 показує її конвеєрну версію) найдовший ланцюжок складається з 5 МКГ – це стовпчик, на верхню МКГ якого подано розряди b_2 та a_0.

Таку схему можна трансформувати у конвеєрну, додавши на кожному рівні помножувача регістри *Rg*. Частота *F2* зміни операндів на входах помножувача визначається часом *T2* проходження сигналів від одного конвеєрного регістра до іншого – *F2 = 1/T2*. Приклад конвеєрної схеми для $GF(p^3)$ наведено на рис. 3.22. Як видно, затримка *T2* – це затримка однієї МКГ. Тому *T1/T2 = 5* і *F2/F1 = 5*, тобто, конвеєрний помножувач зможе працювати на в 5 разів біль-

шій частоті.

У даній роботі дослідження обмежено тільки комбінаційними помножувачами, оскільки їх можна трансформувати в інші типи помножувачів, провівши з за запропонованими у роботі методами аналогічні дослідження.

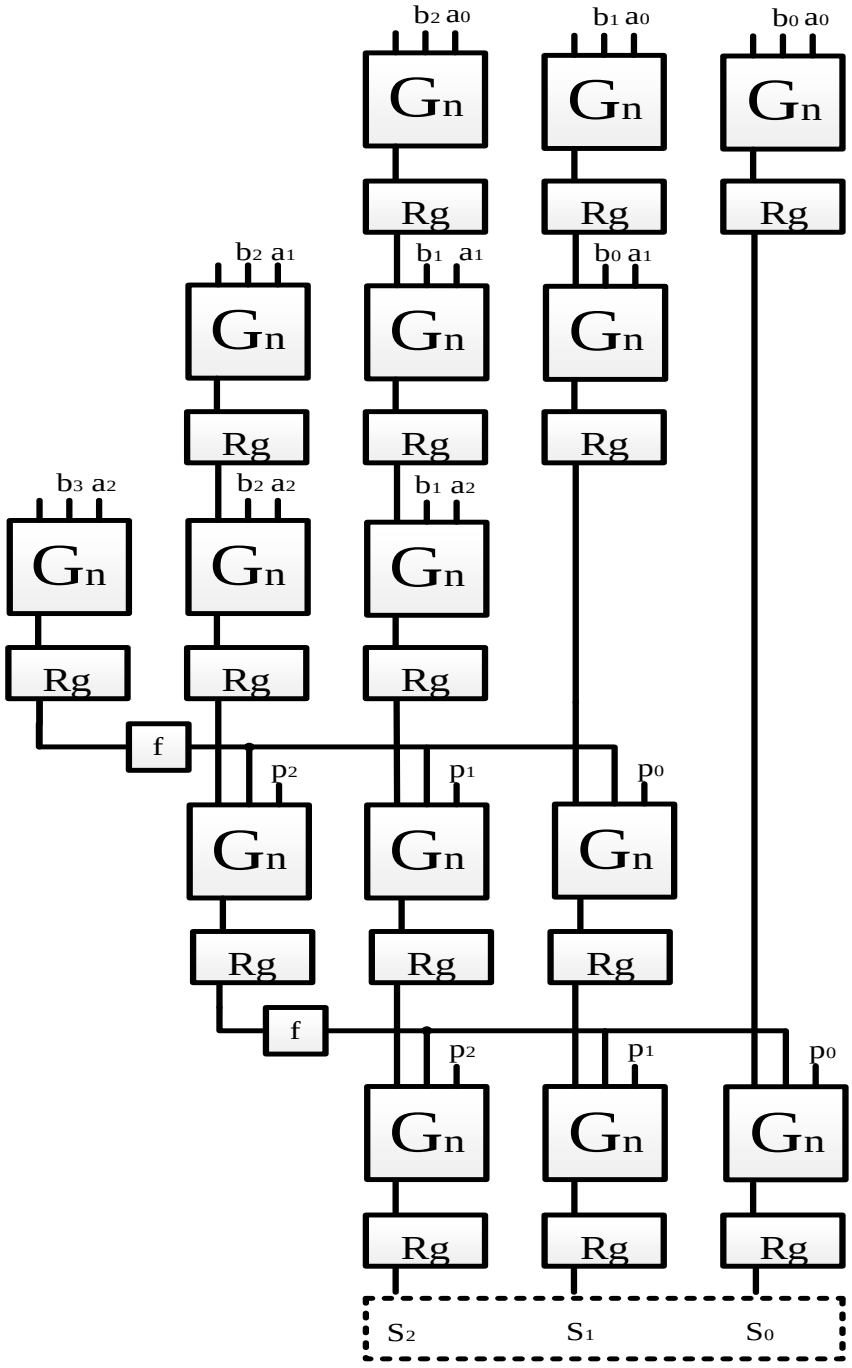


Рис. 3.22 Конвеєрна схемам помножувача полів Галуа $GF(p^3)$

3.11 Висновки до розділу 3

У третьому розділі описано процес розробки генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ з довільною характеристикою p та порядком утворюючим поле полінома n за 3 структурами МКГ. Наведено структурну схему генераторів ядер помножувачів та діаграми класів для кожного із трьох варіантів структури помножувача. Наведено основні часові характеристики генерування помножувачів. Проведено тестування розроблених генераторів та згенерованих ними ядер помножувачів для розширених полів Галуа $GF(p^n)$. Проведено тестування показало правильність роботи генераторів та згенерованих ними ядер.

Було описано процес і наведено результати тестування згенерованих помножувачів запропонованим у роботі методом. Було протестовано роботу помножувачів для полів $GF(3^{30})$, $GF(7^{17})$, $GF(23^{10})$, $GF(53^8)$, і показано правильність роботи генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ та самих помножувачів.

РОЗДІЛ 4

ДОСЛІДЖЕННЯ ТА ВПРОВАДЖЕННЯ РЕКОНФІГУРОВАНИХ ВУЗЛІВ КРИПТОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ НА ОСНОВІ РОЗШИРЕНИХ ПОЛІВ ГАЛУА, ЯКІ ВИКОРИСТОВУЮТЬСЯ ПРИ КРИПТОГРАФІЧНОМУ ЗАХИСТІ ІНФОРМАЦІЇ НА ОСНОВІ ЕЛІПТИЧНИХ КРИВИХ

4.1 Реалізація помножувачів на ПЛІС *Spartan 6* та *Cyclone V* та порівняння результатів реалізації

На рис. 4.1 представлено внутрішню структуру а) МКГ ЧС та б) МКГ ФВ для поля $GF(3^4)$.

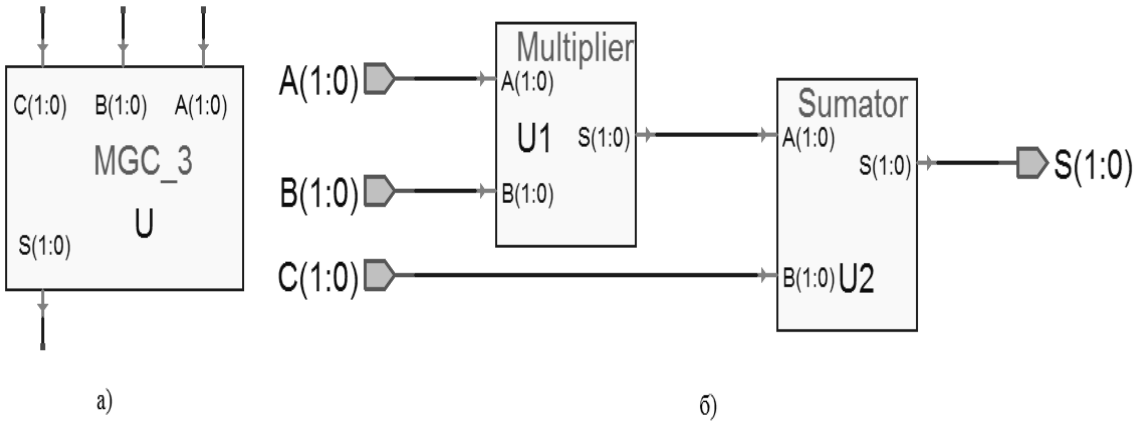


Рис. 4.1 Реалізація МКГ для розширених полів Галу *GF(3⁴)*: а) МКГ ЧС, б) МКГ ФВ

У першому варіанті при синтезі МКГ формується 1 функція *S(1:0)*, яка залежить від 6 змінних *(A(1:0), B(1:0), C(1:0))*, і є результатом множення за модулем 3 та додавання за модулем 3, у другому – 2 функції *S(1:0)* – одна на виході помножувача, друга – на виході суматора, кожна з яких залежить від 4 змінних, перша є результатом множення за модулем 3, а друга – результатом додавання за модулем 3.

На рис. 4.2 наведено схему помножувача елементів розширеного поля Галуа $GF(3^4)$ промодельовану в середовищі *Active HDL 9.1*.

Значення апаратних витрат та часових затримок на реалізацію помножу-

вачів $GF(2^{15})$, $GF(3^9)$, $GF(5^6)$, $GF(7^5)$, $GF(13^4)$, які мають схеми, аналогічні рис. 4.2, наведено на графіках – рис. 4.3, рис. 4.4 та таблицях 4.1 та 4.2. Топологія кристалів була розроблена у середовищах *Xilinx ISE* для *Spartan 6*, *Quartus* для *Cyclone 5*.

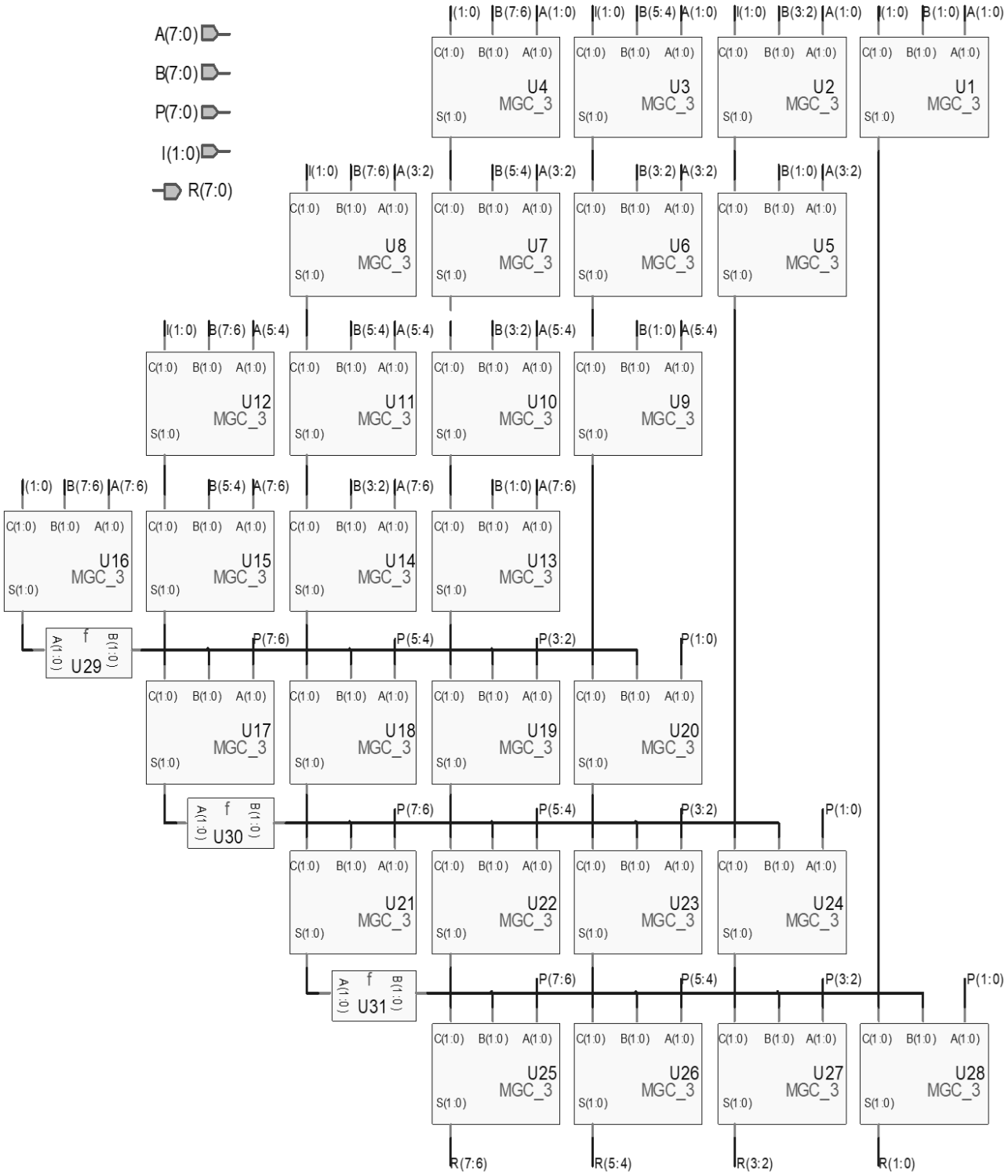


Рис. 4.2 Схема помножувача елементів розширеного поля Галуа $GF(3^4)$

З графіків рис. 4.3 та рис. 4.4 видно, що найменші апаратні витрати має помножувач для елементів поля $GF(2^{15})$. Найменші часові затримки має також помножувач поля $GF(2^{15})$. Також слід відзначити, що $GF(3^9)$ та $GF(7^5)$ мають хороші показники щодо апаратних витрат та швидкодії, які є трохи більшими ніж у двійкових полях.

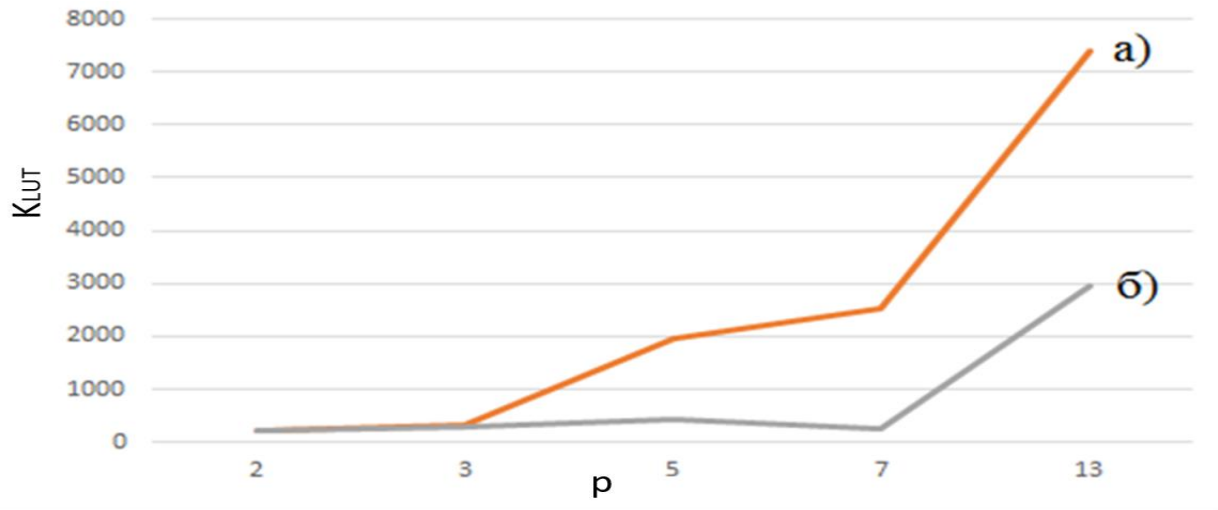


Рис. 4.3 Апаратні витрати помножувачів елементів розширених полів Галуа $GF(2^{15}),GF(3^9),GF(5^6),GF(7^5),GF(13^4)$ на ПЛІС *Spartan 6*: а) МКГ ЧС, б) МКГ ФВ

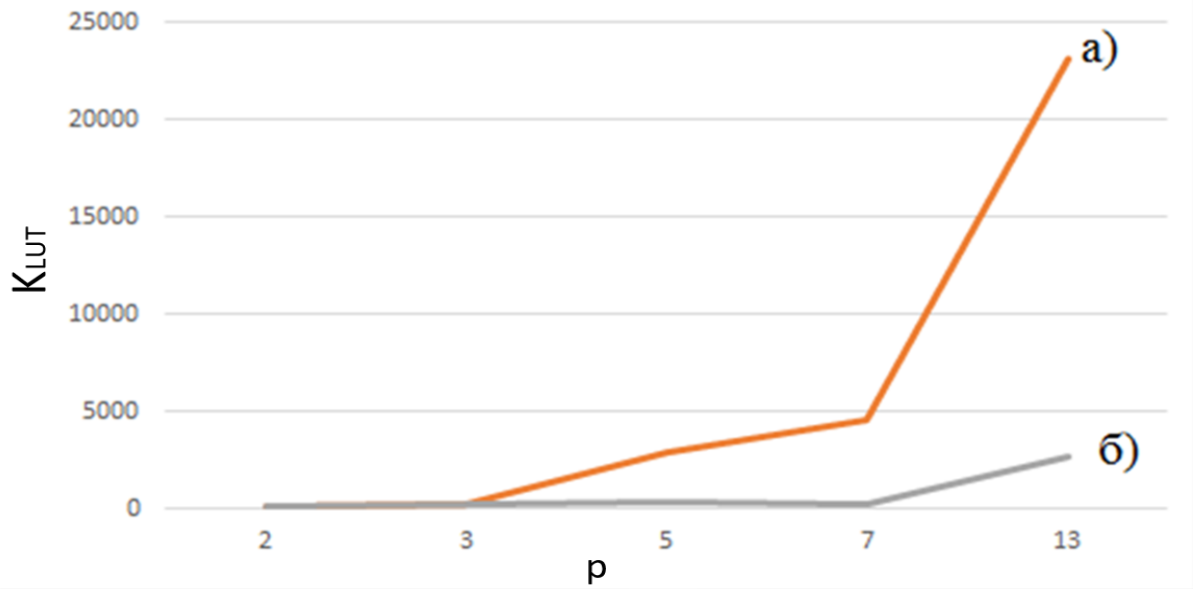


Рис. 4.4 Апаратні витрати помножувачів елементів розширених полів Галуа $GF(2^{15}),GF(3^9),GF(5^6),GF(7^5),GF(13^4)$ на ПЛІС Cyclone 5: а) МКГ ЧС, б) МКГ ФВ

З табл. 4.1 бачимо що 2, 3, 5, та 7-кові поля при реалізації помножувачів на ПЛІС *Spartan-6* мають найменші показники апаратних витрат *LUT* та часових затримок.

Таблиця 4.1

Апаратні витрати (LUT та SLICE) при реалізації помножувачів для розширених полів Галуа $GF(p^n)$ на ПЛІС Spartan 6

Поле	Структура МКГ	Кількість МКГ	Кількість *LUT* в помножувачі	Кількість *SLICE* в помножувачі	Кількість входів/виходів	Найбільша затримка, *ns.*
$GF(2^{15})$	ЧС	435	218	82	61	22.161
$GF(2^{15})$	ФВ	435	205	86	61	25.835
$GF(3^9)$	ЧС	153	312	138	74	31.186
$GF(3^9)$	ФВ	153	298	106	74	47.267
$GF(5^6)$	ЧС	66	1946	600	75	49.414
$GF(5^6)$	ФВ	66	439	171	75	42.067
$GF(7^5)$	ЧС	45	2534	963	63	37.389
$GF(7^5)$	ФВ	45	258	103	63	30.513
$GF(13^4)$	ЧС	28	7395	3031	68	41.970
$GF(13^4)$	ФВ	28	2949	1018	68	85.703

З таблиці 4.2 бачимо, що найменші апаратні витрати будуть в поля $GF(2^{15})$. Проте поля $GF(3^9)$, $GF(5^6)$, $GF(7^5)$ при реалізації помножувачів на ПЛІС *Cyclone V* мають також хороші показники апаратних витрат *LUT* та часових затримок.

Таблиця 4.2

Апаратні витрати (ALM та LAB) при реалізації помножувачів для розширених полів Галуа $GF(p^n)$ на ПЛІС Cyclone 5

Поле	Структура МКГ	Кількість МКГ	Кількість затрачених *ALM* в помножувачі	Кількість затрачених *LAB* в помножувачі	Кількість входів/виходів	З'єднання між блоками
GF(2^{15})	ЧС	435	122	15	61	406
GF(2^{15})	ФВ	435	142	17	61	418
GF(3^9)	ЧС	153	292	31	74	543
GF(3^9)	ФВ	153	193	24	74	613
GF(5^6)	ЧС	66	2879	325	75	6596
GF(5^6)	ФВ	66	370	43	75	5345
GF(7^5)	ЧС	45	4519	548	63	10450
GF(7^5)	ФВ	45	263	32	63	405
GF(13^4)	ЧС	28	23132	3318	68	83624
GF(13^4)	ФВ	28	2660	296	68	50456

4.2 Реалізація помножувачів на ПЛІС *Virtex UltraScale* та порівняння результатів реалізації

У табл. 4.3 наведено апаратні витрати *LUT*, при реалізації помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС *Virtex Ultrascale+*. З таблиці бачимо, що при реалізації помножувачів з МКГ ЧС апаратні витрати стрімко зростають при збільшенні характеристики поля *p*. Апаратні витрати є найменшими для полів Галуа $GF(2^m)$, $GF(3^n)$. При реалізації МКГ ФВ апаратні витрати також стрімко зростають, при збільшенні характеристики поля. Найменші апаратні витрати за цією архітектурою є у 2-кових, 3-кових, 5-кових, 7-кових полів. При реалізації помножувачів з структурою МКГ ЛВ апаратні витрати зі

збільшенням характеристики поля зменшуються, у порівнянні з апаратними витратами полів Галуа $GF(2^m)$. Отже за трьома наведеними структурами МКГ можна побудувати будь-який помножувач елементів розширених полів Галуа $GF(p^n)$, апаратні витрати якого можна спрогнозувати.

Таблиця 4.3

Апаратні витрати LUT при реалізації помножувачів елементів розширених полів Галуа на ПЛІС Virtex Ultrascale+

Поле	Кількість *LUT* в помножувачі, МКГ ЧС	Кількість *LUT* в помножувачі, МКГ ФВ	Кількість *LUT* в помножувачі, МКГ ЛВ
$GF(2^{499})$	250405	219345	1878456
$GF(3^{315})$	354215	354327	1837834
$GF(5^{215})$	1647332	462678	1807234
$GF(7^{175})$	1321117	293187	1786675
$GF(11^{140})$	Недостатньо *LUT*	Недостатньо *LUT*	1714345
$GF(13^{135})$	Недостатньо *LUT*	Недостатньо *LUT*	1632814
$GF(17^{120})$	Недостатньо *LUT*	Недостатньо *LUT*	1545345
$GF(19^{115})$	Недостатньо *LUT*	Недостатньо *LUT*	1436345
$GF(23^{110})$	Недостатньо *LUT*	Недостатньо *LUT*	1334239
$GF(31^{100})$	Недостатньо *LUT*	Недостатньо *LUT*	1245768
$GF(37^{95})$	Недостатньо *LUT*	Недостатньо *LUT*	1223523
$GF(41^{93})$	Недостатньо *LUT*	Недостатньо *LUT*	1123564
$GF(43^{92})$	Недостатньо *LUT*	Недостатньо *LUT*	1134345
$GF(47^{90})$	Ндостатньо *LUT*	Недостатньо *LUT*	1097356
$GF(53^{87})$	Не достатньо *LUT*	Недостатньо *LUT*	1058028

Генерування таких великих помножувачів є часозатратною задачею. Час генерування помножувачів за структурами МКГ ЧС та ФВ при збільшенні характеристики поля зростає. При генеруванні помножувачів за структурою МКГ ЛВ, час генерування помножувачів є приблизно однаковим для різних характе-

ристик поля. Час імплементації великих помножувачів на ПЛІС займає від кількох десятків хвилин до кількох годин. На рис. 4.5 наведено імплементований на ПЛІС *Virtex UltraScale+* помножувач $GF(5^{215})$.

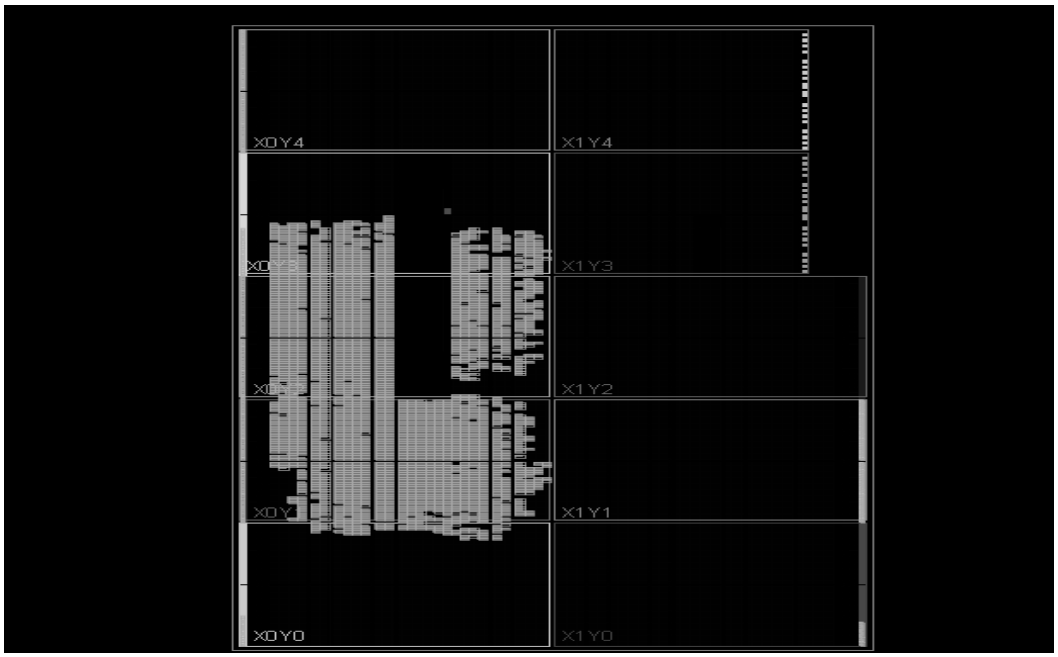


Рис. 4.5 Імплементація помножувача $GF(5^{215})$ на ПЛІС *Virtex UltraScale+*, з структурою МКГ ФВ

На рис. 4.6 наведено характеристики споживання потужності помножувачем на ПЛІС *Virtex UltraScale+*.

На рис. 4.7 наведена схема МКГ ЧС на ПЛІС *Virtex UltraScale+*. МКГ реалізована для розширеного поля Галуа $GF(17^{120})$.

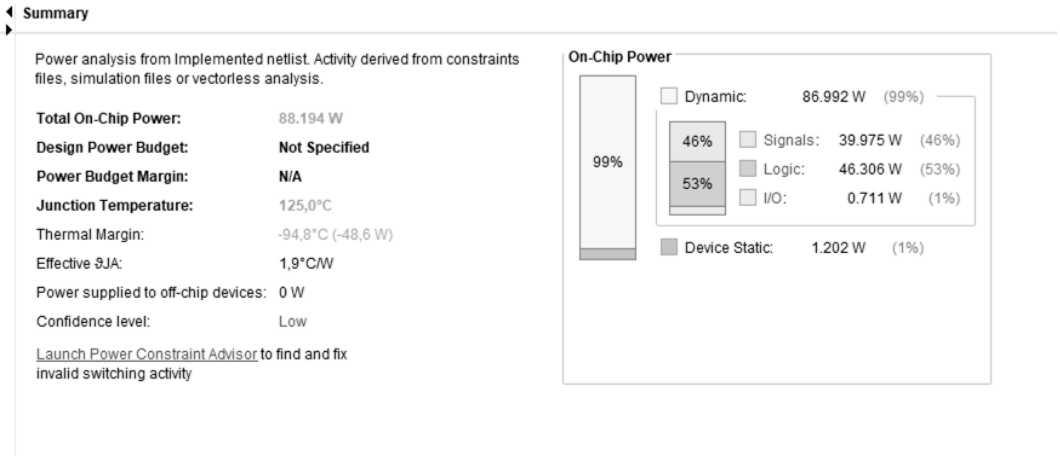


Рис. 4.6 Результат імплементації помножувача на ПЛІС *Virtex UltraScale+*. Характеристики споживаної потужності

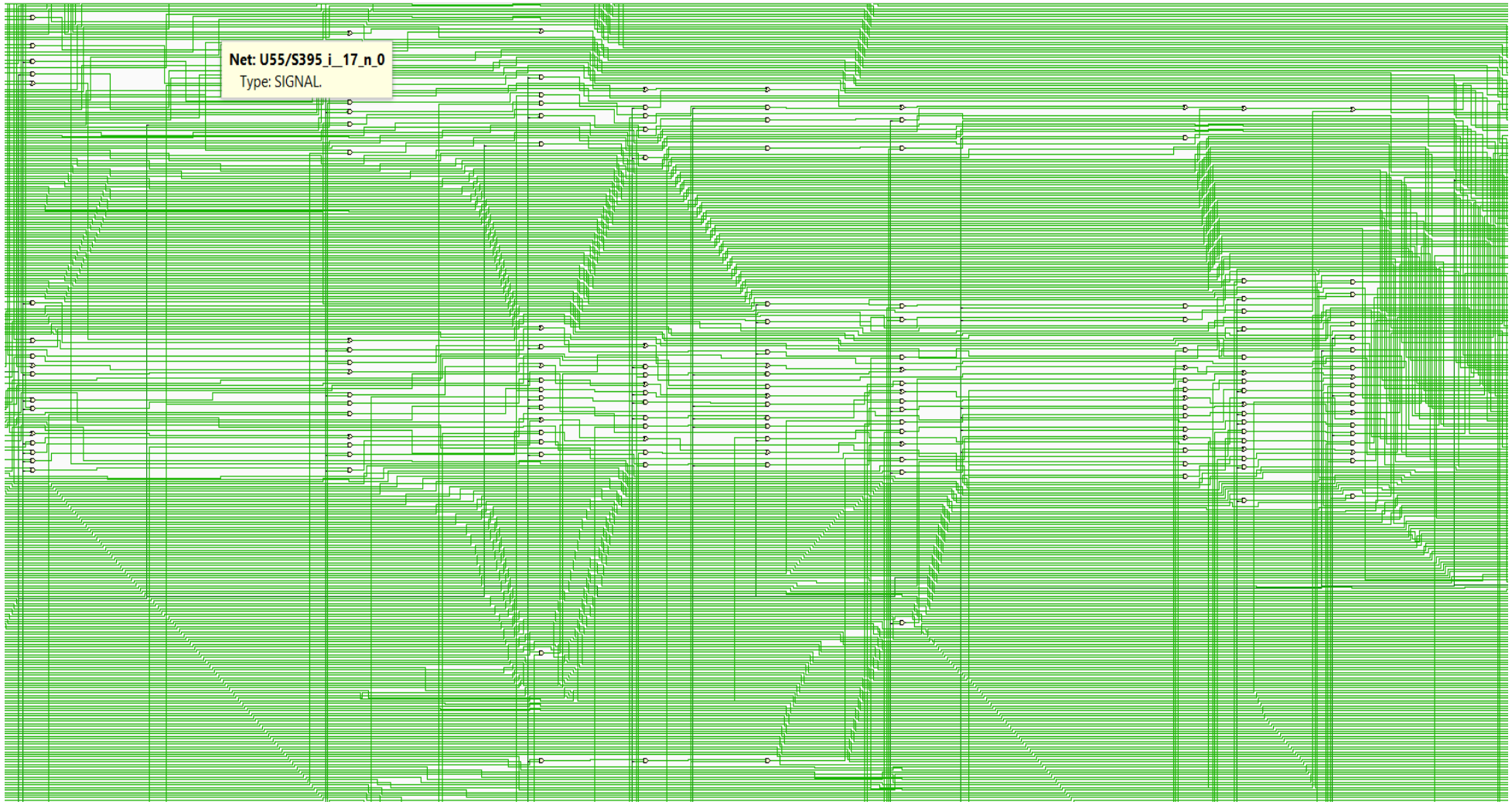


Рис. 4.7. Схема МКГ ЧС $GF(17^{120})$ реалізована на ПЛІС *Virtex UltraScale+*

На рис. 4.8 наведено фрагмент схеми МКГ ЛВ помножувача елементів розширеного поля Галуа $GF(11^{140})$ на ПЛІС *Virtex UltraScale+*..

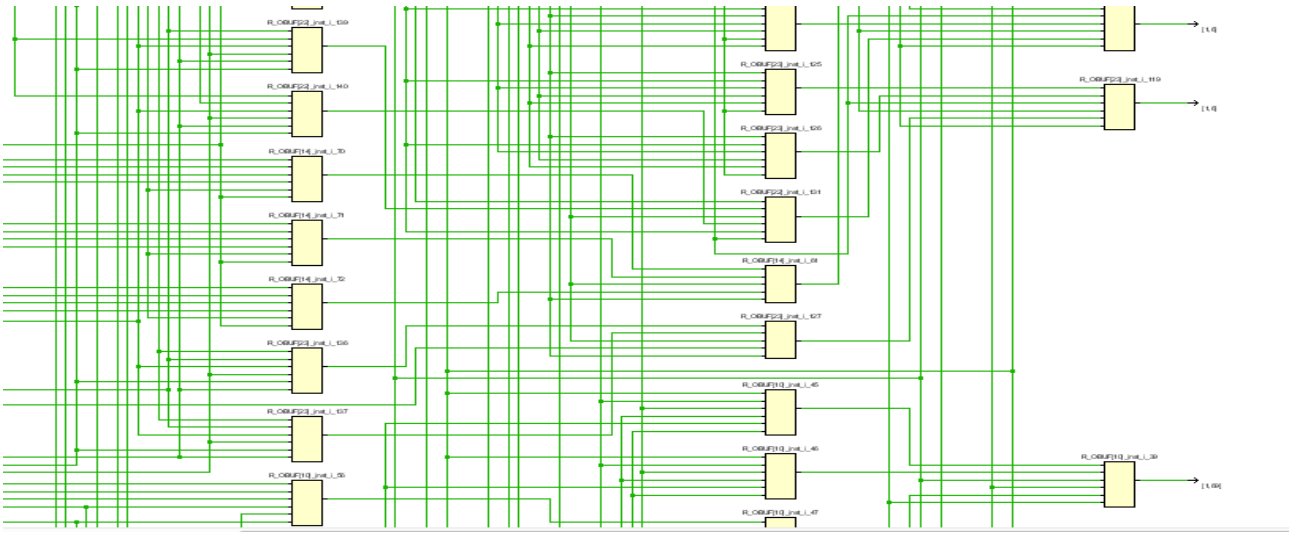


Рис. 4.8 Фрагмент схеми МКГ ЛВ помножувача елементів розширеного поля Галуа $GF(11^{140})$ на ПЛІС *Virtex UltraScale+*

На рис. 4.9 наведено фрагмент звіту імплементації згенерованого помножувача $GF(5^{20})$ на ПЛІС *Virtex UltraScale+*. З рисунка видно, що помножувач використовує 16454 *LUTs* з 101400 можливих, 4492 *Slices* з 25350 можливих.

Name	Slice LUTs (101400)	Slice (25350)	LUT as Logic (101400)	Bonded IPADs (26)
N Multiplier	16454	4492	16454	243

Рис. 4.9 Фрагмент звіту імплементації помножувача елементів розширеного поля Галуа $GF(5^{20})$ на ПЛІС *Virtex UltraScale+*

На рис. 4.10 зображено фрагмент схеми помножувача елементів розширених полів Галуа $GF(7^n)$ з розгорнутою схемою МКГ ЧС на ПЛІС *Virtex UltraScale+*. На рис. 4.12 – схема вузла *F* помножувача $GF(2^m)$ на ПЛІС *Virtex UltraScale+*.

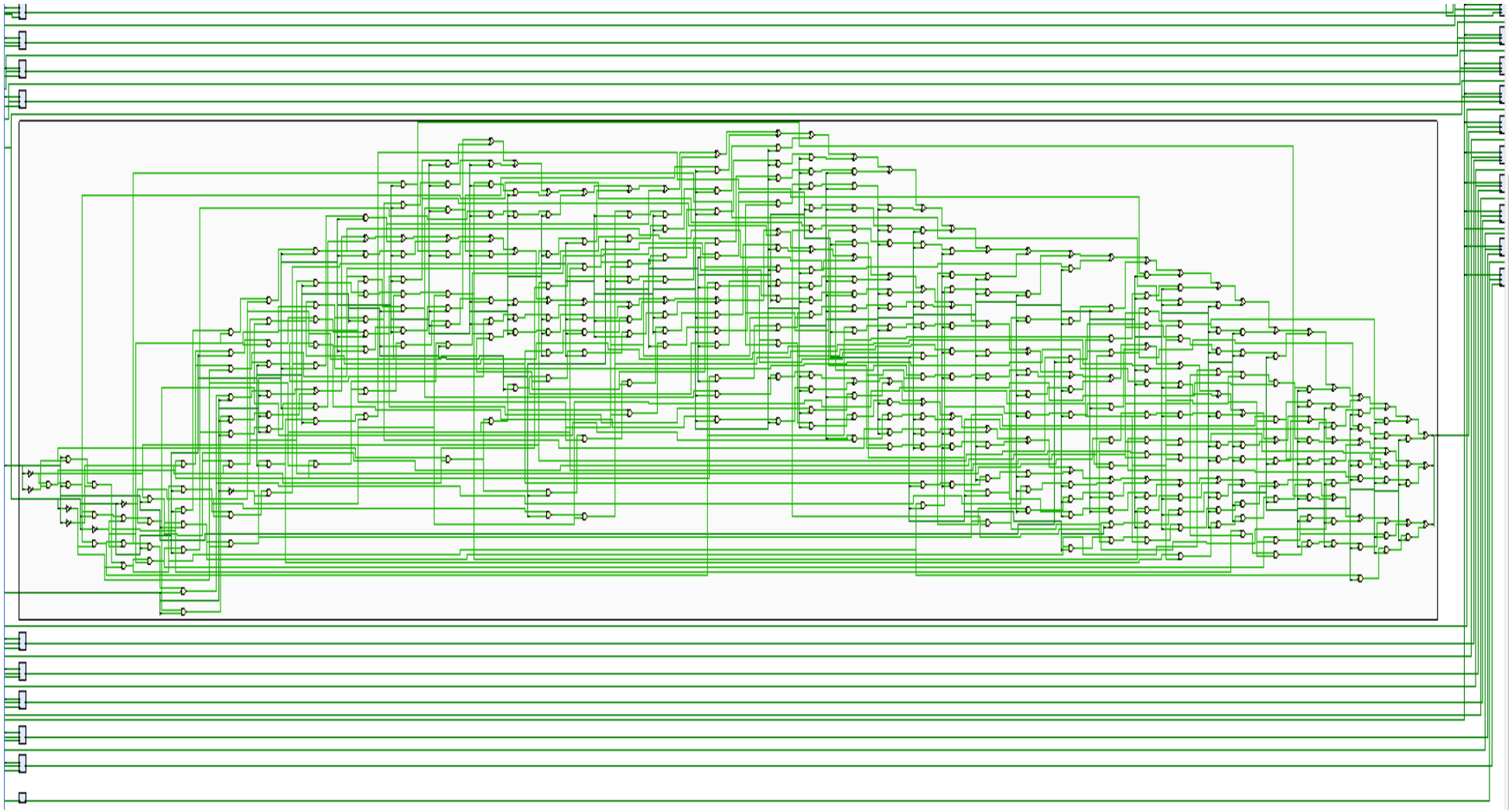


Рис. 4.10 Фрагмент схеми помножувачів елементів розширених полів Галуа $GF(7^n)$, з розгорнутою схемою МКГ ЧС на ПЛІС *Virtex UltraScale+*

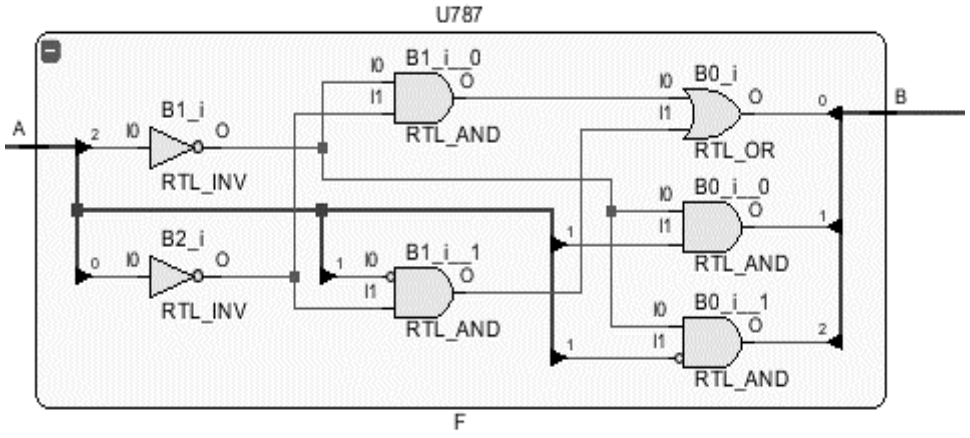


Рис. 4.11 Схема вузла *f* помножувача $GF(2^m)$ на ПЛІС *Virtex UltraScale+*

У таблицях 4.4 та 4.6 наведено реальні та теоретичні результати генерації ядер (*VHDL*-описів) помножувачів для ПЛІС *Virtex UltraScale+ XCVU9P*. При реалізації помножувачів з структурою МКГ ЧС, апаратна складність швидко зростає зі збільшенням характеристики поля. Ці витрати найменші для двійкових і трійкових полів Галуа. При реалізації помножувачів з структурою МКГ ФВ, апаратна складність також швидко зростає зі збільшенням характеристики поля. Найменшу апаратну складність за цією структурою МКГ мають помножувачі для полів з характеристиками $p = 2, 3, 5, 7$. При реалізації помножувачів на основі МКГ ЛВ, апаратна складність збільшується зі збільшенням характеристики поля. Отже, за трьома згаданими структурами МКГ можна для обраного поля з використанням створеного генератора ядер згенерувати *VHDL*-опис помножувача, який правильно сприймається засобами проектування ПЛІС, що дозволяє визначити реальні апаратні та часові характеристики помножувача. У таблицях також показано час генерації помножувача. Час збільшується зі збільшенням характеристики поля.

У таблиці 4.5 та 4.7 показано порівняння теоретичних KT_{mul} і реальних KR_{mul} апаратних витрат на реалізацію помножувачів елементів розширених полів Галуа $GF(p^n)$ відносно апаратних витрат на реалізацію помножувачів двійкових полів Галуа $GF(2^m)$ KT_2 і KR_2, МКГ реалізується за 3 структурами. KT_{mul}

$= NT_d/NT_2$, $KR_{mul} = NR_d/NR_2$.

NT_p – теоретична кількість LUT у помножувачі для поля $GF(p^n)$;

NR_p – реальна кількість LUT у помножувачі для поля $GF(p^n)$;

T – час генерування помножувача.

З таблиць 4.5, 4.7 та графіків рис. 4.12, 4.13, 4.14 видно, що для МКГ ЧС за апаратними витратами трійкові поля Галуа $GF(3^n)$ на 3 % кращі, ніж двійкові $GF(2^m)$. При реалізації помножувачів на МКГ ФВ у порівнянні з полями $GF(2^m)$ – поле з $GF(3^n)$ має на 11 % більшу апаратну складність, поле $GF(5^n)$ має на 20 % більшу апаратну складність, поле $GF(7^n)$ має на 18 % більшу апаратну складність. При реалізації помножувачів на основі МКГ ЛВ апаратна складність помножувачів є більшою ніж у помножувачів з структурами МКГ ЧС та МКГ ФВ.

Таблиця 4.4

Апаратні витрати LUT реальні NR_p та теоретичні NT_p та час генерування помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС Virtex UltraScale+ XCVU9P, які мають 2069000 LUTs

Поле	*p*	Розмір поля Галуа (O_p)	МКГ ЧС			МКГ ФВ			МКГ ЛВ		
			NR_p	T, *sec.*	NT_p	NR_p	T, *sec.*	NT_p	NR_p	T, *sec.*	NT_p
$GF(2^{50})$	2	1,12E+15	2504	1,0	4901	2190	0,5	2450	18784	0,5	29208
$GF(3^{32})$	3	1,85E+15	4034	1,4	3970	4032	0,7	1984	17950	0,5	36510
$GF(5^{22})$	5	2,38E+15	19936	1,6	41625	5615	0,8	2768	17867	0,5	35049
$GF(7^{18})$	7	1,62E+15	16851	3,0	27585	3522	1,2	1837	16689	0,5	23366
$GF(13^{14})$	3	3,93E+15	81133	8,0	185420	10211	2,0	10216	43368	0,5	58656

Таблиця 4.5

Порівняння реальних $\frac{KR_{mul}}{C_o}$ та теоретичних $\frac{KT_{mul}}{C_o}$ коефіцієнтів апаратних витрат LUT, при реалізації помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС Virtex UltraScale+ XCVU9P, які мають 2069000 LUTs

Поле	p	Порядок поля Галуа (O_p)	$C_o = \frac{O_p}{O_2}$	МКГ ЧС				МКГ ФВ				МКГ ЛВ			
				KT_{mul}	$\frac{KT_{\text{mul}}}{C_o}$	KR_{mul}	$\frac{KR_{\text{mul}}}{C_o}$	KT_{mul}	$\frac{KT_{\text{mul}}}{C_o}$	KR_{mul}	$\frac{KR_{\text{mul}}}{C_o}$	KT_{mul}	$\frac{KT_{\text{mul}}}{C_o}$	KR_{mul}	$\frac{KR_{\text{mul}}}{C_o}$
$GF(2^{50})$	2	1,12E+15	1	1	1	1	1	1	1	1	1	1	1	1	1
$GF(3^{32})$	3	1,85E+15	1,65	0,81	0,43	1,61	0,97	0,81	0,49	1,84	1,11	1,25	0,75	0,95	0,57
$GF(5^{22})$	5	2,38E+15	2,12	8,49	4	7,96	3,75	1,13	0,53	2,56	1,2	1,2	0,56	0.95	0,44
$GF(7^{18})$	7	1,62E+15	1,35	5,63	4,17	6,72	4,97	0,75	0,55	1,6	1,18	0,8	0,59	0,88	0,65
$GF(13^{14})$	13	3,93E+15	3,5	37,8	10,8	32,40	9,25	4,17	1,19	4,65	1,32	2,0	0,57	2.3	0.65

Таблиця 4.6

Апаратні витрати LUT реальні NR_p та теоретичні NT_p та час генерування помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС Virtex UltraScale+ XCVU9P, які мають 2069000 LUTs

Поле	p	Порядок поля Галуа (O_p)	МКГ ЧС			МКГ ФВ			МКГ ЛВ		
			NR_p	T, *sec.*	NT_p	NR_p	T, *sec.*	NT_p	NR_p	T, *sec.*	NT_p
$GF(2^{998})$	2	2,67E+300	1013715	98	1990013	888165	52	995006	*not fit*	57	15920104
$GF(13^{270})$	13	5,81E+300	*not fit*	889	75222491	*not fit*	423	4149177	*not fit*	61	31840208

Таблиця 4.7

Порівняння реальних $\frac{KR_{mul}}{C_o}$ та теоретичних $\frac{KT_{mul}}{C_o}$ коефіцієнтів апаратних витрат LUT, при реалізації помножувачів елементів розширених полів Галуа $GF(p^n)$ на ПЛІС Virtex UltraScale+ XCVU9P, які мають 2069000 LUTs

Поле	p	Розмір поля Галуа (O_p)	$C_o = \frac{O_p}{O_2}$	МКГ ЧС				МКГ ФВ				МКГ ЛВ			
				KT_{mul}	$\frac{KT_{mul}}{C_o}$	KR_{mul}	$\frac{KR_{mul}}{C_o}$	KT_{mul}	$\frac{KT_{mul}}{C_o}$	KR_{mul}	$\frac{KR_{mul}}{C_o}$	KT_{mul}	$\frac{KT_{mul}}{C_o}$	KR_{mul}	$\frac{KR_{mul}}{C_o}$
$GF(2^{998})$	2	2,67E+300	1	1	1	1	1	1	1	1	1	1	1	1	1
$GF(13^{270})$	13	5,81E+300	3,5	37,8	10,8	-	-	4,17	1,19	-	-	2.0	0.57	-	-

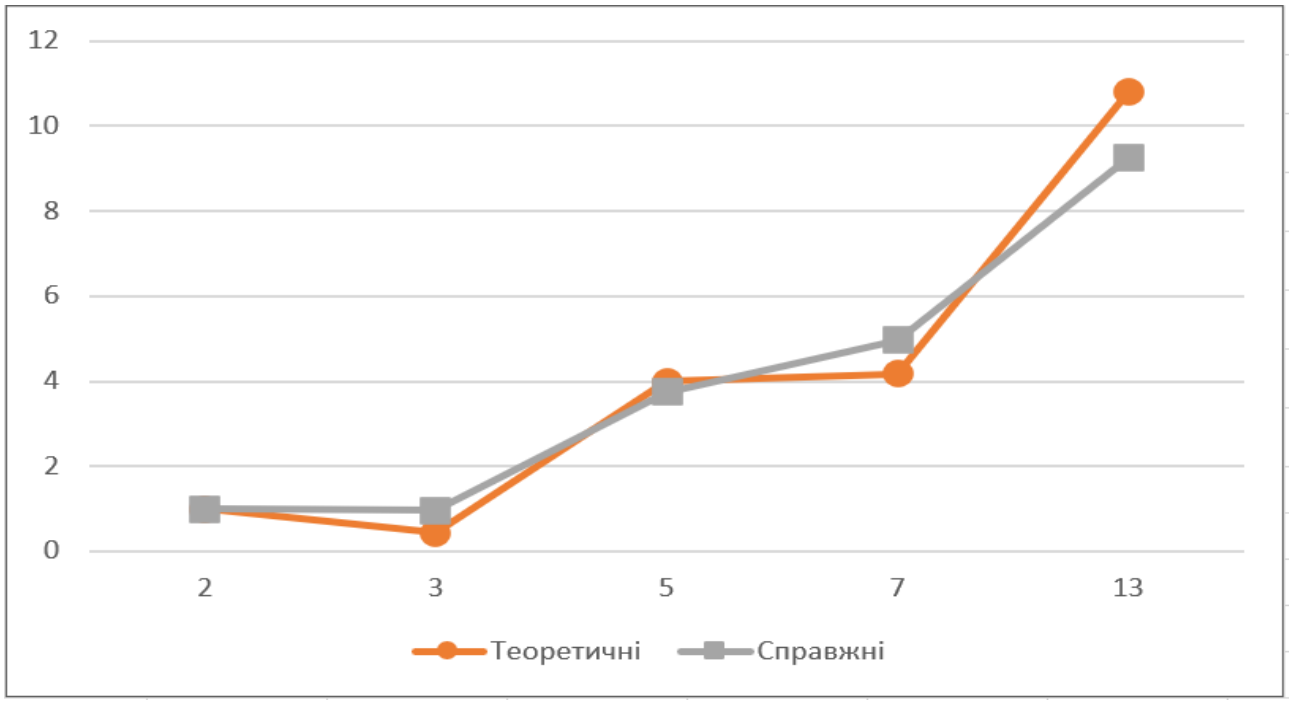


Рис. 4.12 Порівняння теоретичних $\frac{KT_{\text{mul}}}{\text{C}_{\text{o}}}$ та реальних $\frac{KR_{\text{mul}}}{\text{C}_{\text{o}}}$ коефіцієнтів апаратних витрат на реалізацію помножувачів елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ЧС

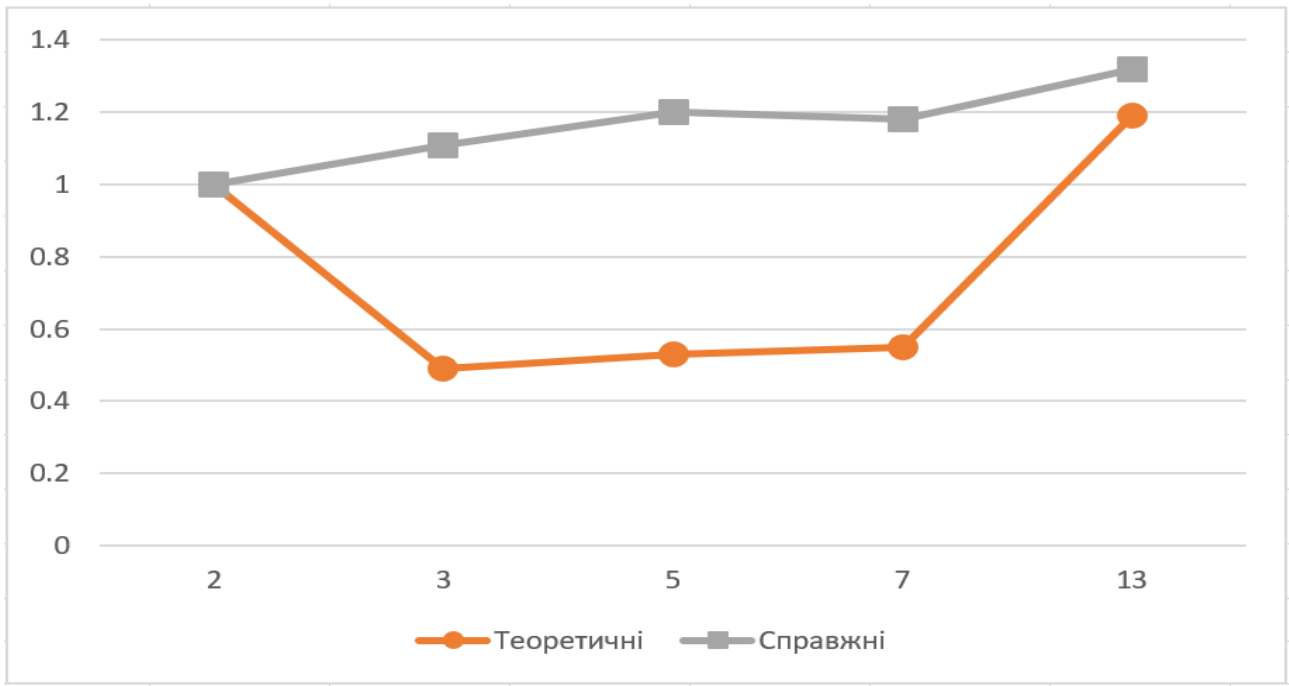


Рис. 4.13 Порівняння теоретичних $\frac{KT_{\text{mul}}}{\text{C}_{\text{o}}}$ та реальних $\frac{KR_{\text{mul}}}{\text{C}_{\text{o}}}$ коефіцієнтів апаратних витрат на реалізацію помножувачів елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ФВ

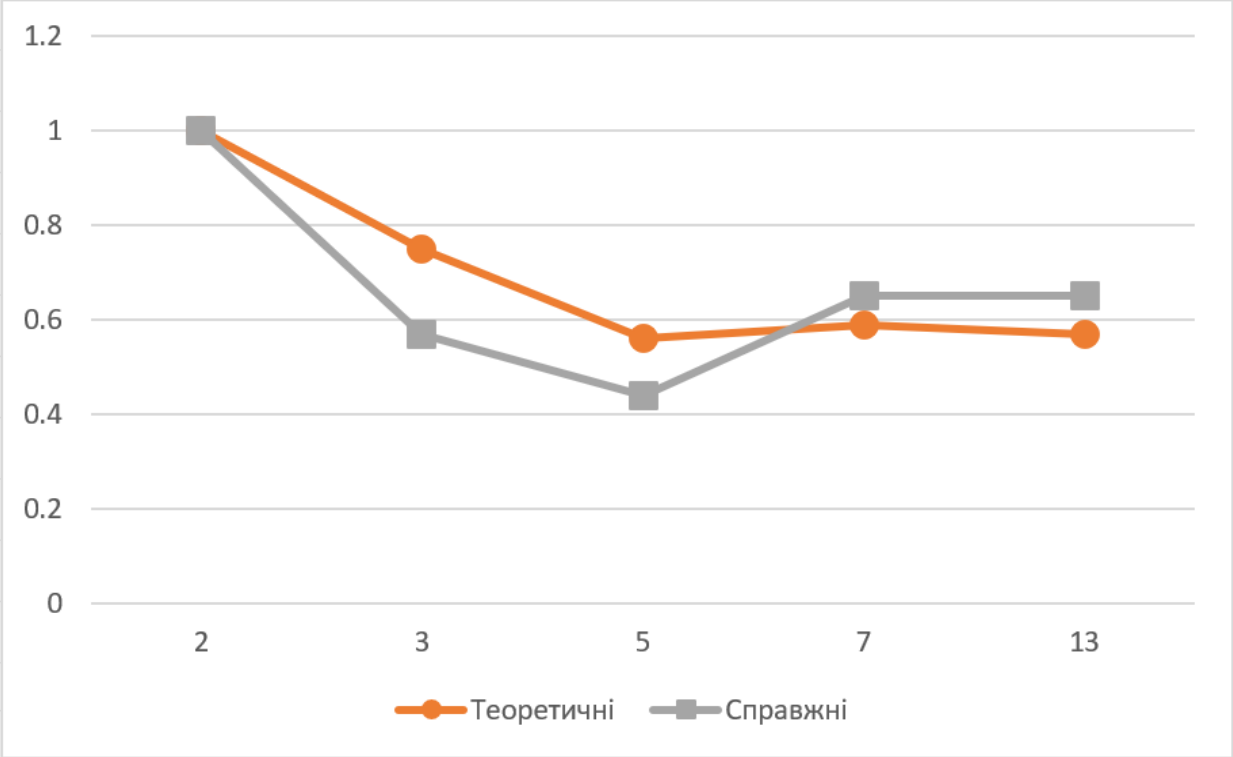


Рис. 4.14 Порівняння теоретичних $\frac{KT_{\text{mul}}}{\text{C}_{\text{o}}}$ та реальних $\frac{KR_{\text{mul}}}{\text{C}_{\text{o}}}$ коефіцієнтів апаратних витрат на реалізацію помножувачів елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ЛВ

На рис. 4.15 наведене порівняння теоретичних апаратних витрат на реалізацію помножувачів за 3 структурами МКГ: ЧС, ФВ, ЛВ. З рисунка видно, що структура МКГ ЧС має переваги при реалізації МКГ для розширених полів Галуа $GF(p^n)$ з характеристиками поля $p = 2$ та $p = 3$. Структура МКГ ФВ забезпечує менші апаратні витрати, при реалізації помножувачів розширених полів Галуа $GF(p^n)$ з характеристиками поля $p = 2$, $p = 3$, $p = 5$, $p = 7$. Апаратна складність помножувачів на основі МКГ ЛВ для елементів розширених полів Галуа $GF(p^n)$ має складність більшу ніж складність помножувачів з структурами МКГ ЧС та ФВ.

Графіки порівняння апаратної складності, на реалізацію помножувачів за 3 структурами МКГ: ЧС, ФВ, ЛВ наведено на рис. 4.16. З рисунка видно, що реальні результати збігаються з теоретичними. Також видно, що структура МКГ ЧС має переваги, при реалізації МКГ для розширених полів Галуа $GF(p^n)$ з характеристиками поля $p = 2$ та $p = 3$. Структура МКГ ФВ забезпечує менші

апаратні витрати, при реалізації помножувача розширених полів Галуа $GF(p^n)$ з характеристиками поля $p = 2$, $p = 3$, $p = 5$, $p = 7$. Апаратна складність помножувачів для елементів розширених полів Галуа $GF(p^n)$ з структурою МКГ ЛВ має складність більшу ніж апаратна складність помножувачів з структурами МКГ ЧС та ФВ. Якщо порівняти не двійкові розширені поля Галуа $GF(p^n)$, то з таблиці 4.5 видно, що найменшу апаратну складність буде мати поле $GF(3^n)$, на $\frac{1{,}2-1{,}11}{1{,}11} = 8$ %, більшу поле $GF(5^n)$ та на $\frac{1{,}18-1{,}11}{1{,}11} = 6$ % більшу поле $GF(7^n)$.

Абсолютну різницю теоретичних та реальних апаратних витрат на реалізацію помножувачів для розширених полів Галуа $GF(p^n)$ у LUT наведено на рис. 4.17.

На рис. 4.18. наведено відносну різницю теоретичних та реальних апаратних витрат на реалізацію помножувачів для розширених полів Галуа $GF(p^n)$ – у відсотках. З рис. 4.18 видно, що для помножувачів з структурою МКГ ЧС точність теоретичних обчислень – 55 %, для МКГ ФВ – 56 %, для МКГ ЛВ – 31 %.

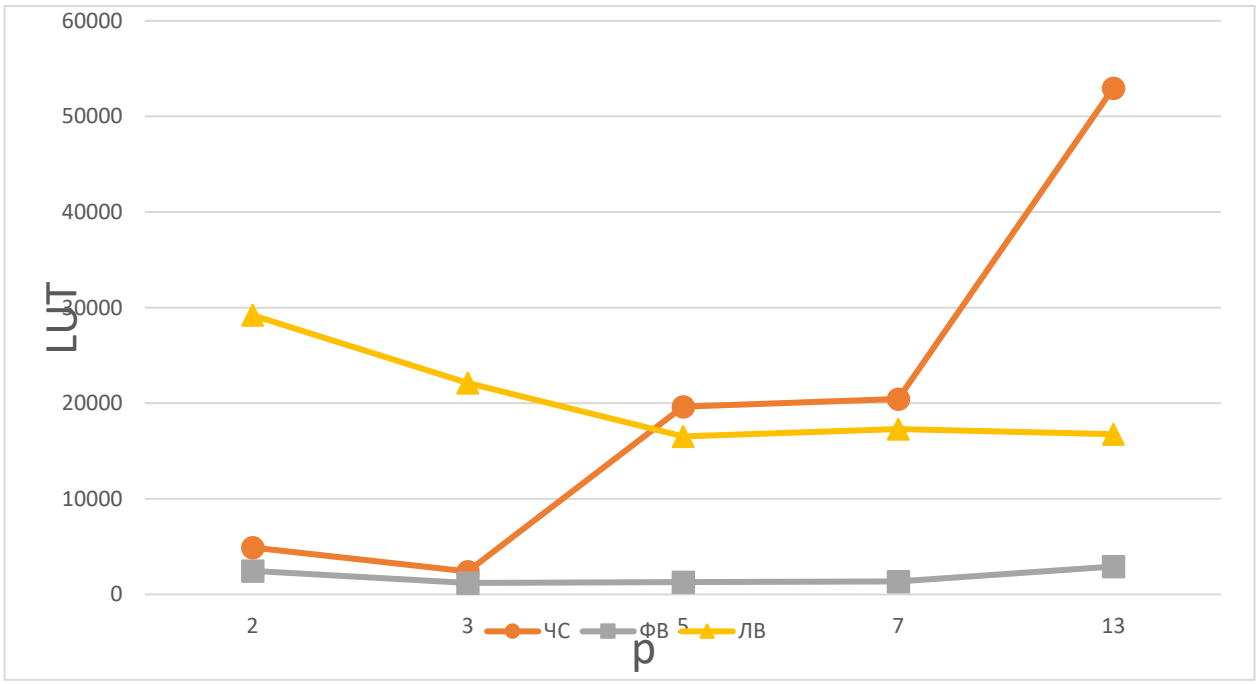


Рис. 4.15 Порівняння теоретичних апаратних витрат на реалізацію помножувачів за 3 структурами МКГ: ЧС, ФВ, ЛВ

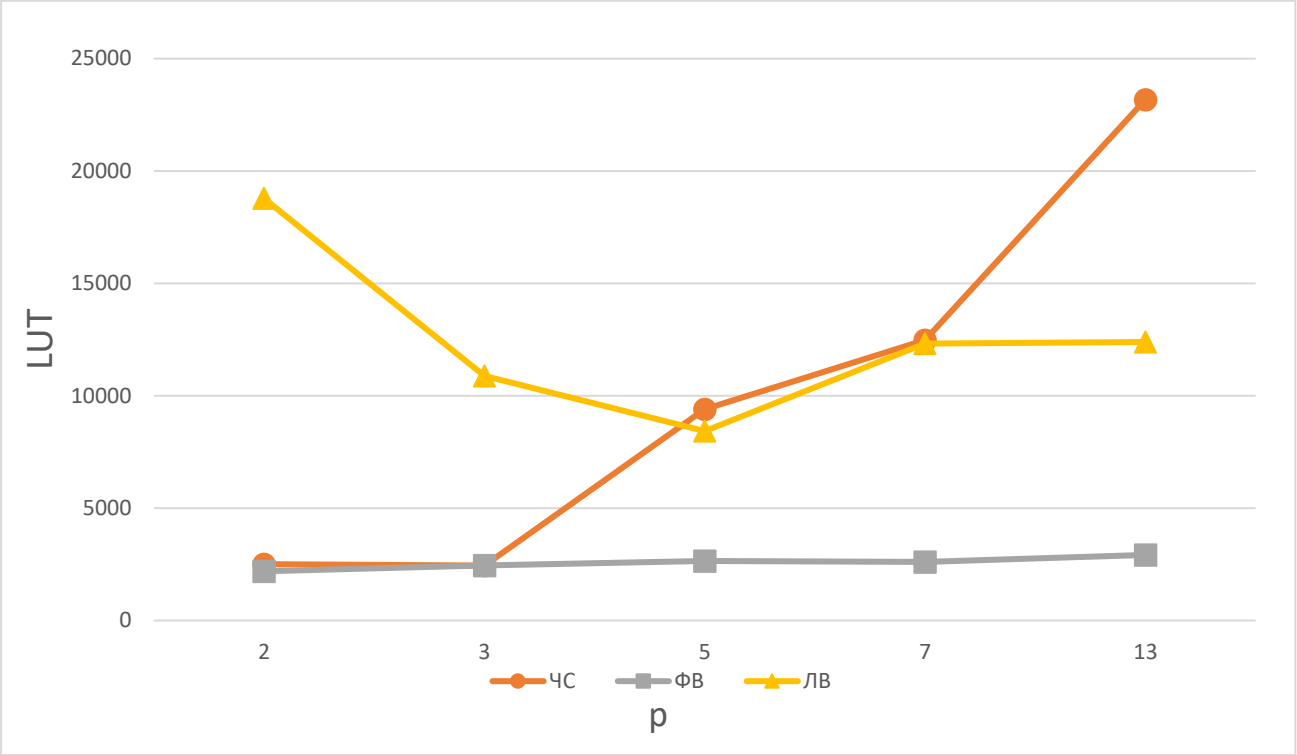


Рис. 4.16 Порівняння реальних апаратних витрат на реалізацію помножувачів за 3 структурами МКГ: ЧС, ФВ, ЛВ

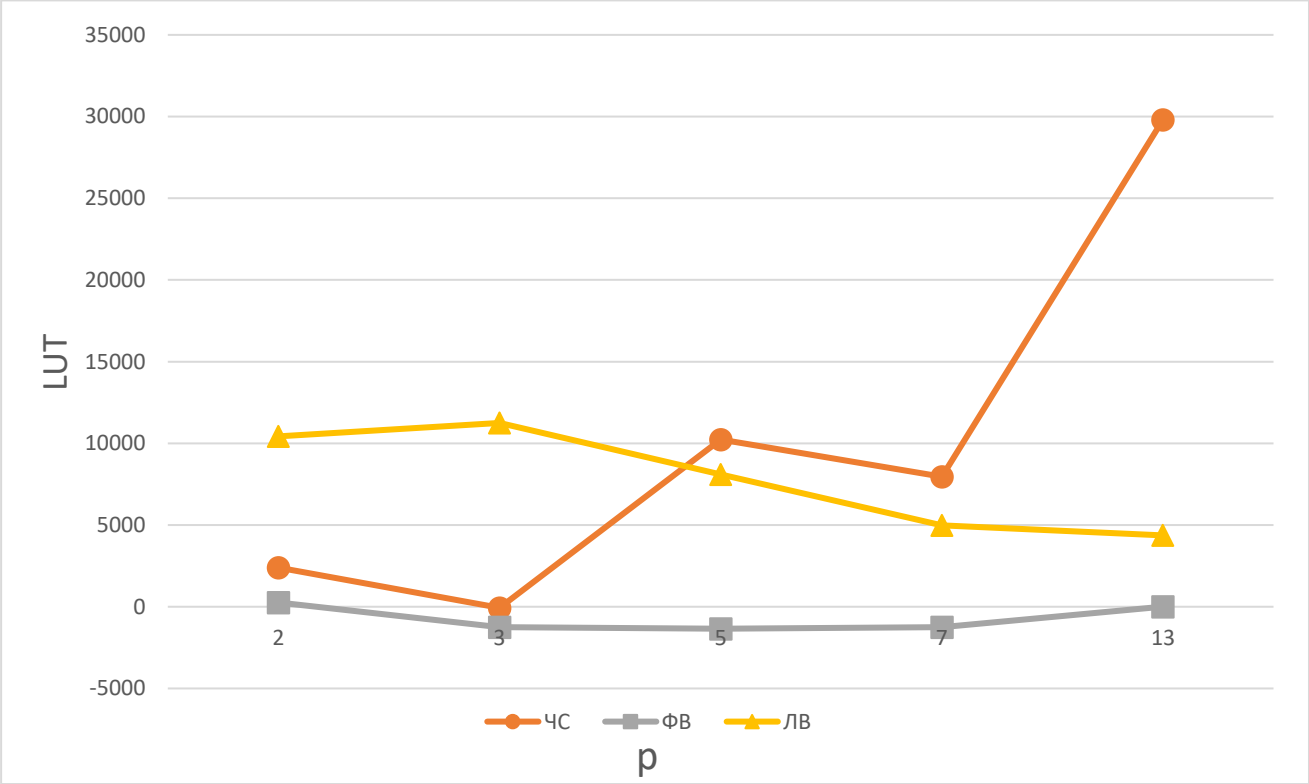


Рис. 4.17 Абсолютна різниця теоретичних та реальних значень апаратних витрат на реалізацію помножувачів за 3 структурами МКГ: ЧС, ФВ, ЛВ

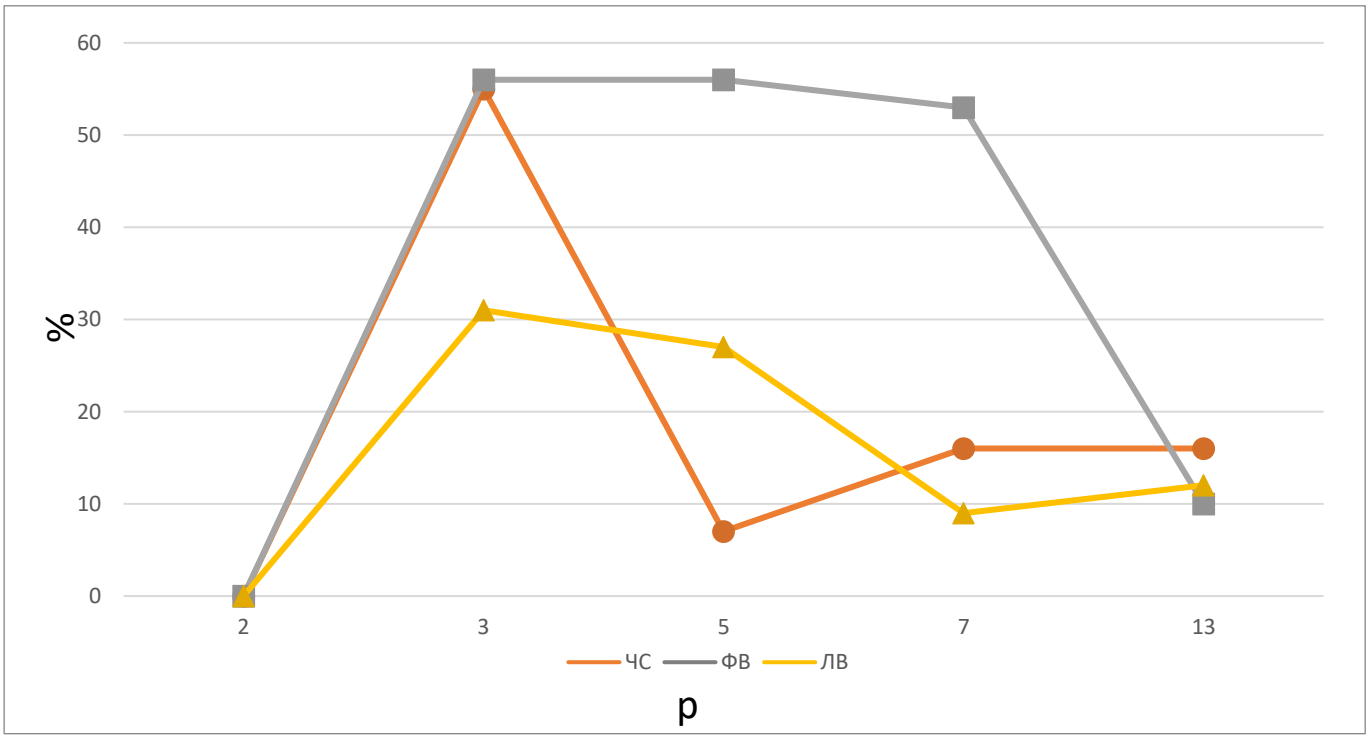


Рис. 4.18 Відносна різниця теоретичних та реальних значень апаратних витрат на реалізацію помножувачів за 3 структурами МКГ: ЧС, ФВ, ЛВ

У таблиці 4.8 наведено найбільші часові затримки, у наносекундах, множення елементів розширених полів Галуа $GF(p^n)$ на ПЛІС *Virtex UltraScale+ XCVU9P*. З таблиці видно, що найменші часові затримки для структури МКГ ЧС мають поля з характеристикою 3, з структурою ФВ – поля з характеристикою 2, з структурою ЛВ – поля з характеристикою 5.

На рис. 4.19 наведено відношення k часових витрат при програмній та апаратній реалізаціях множення елементів полів $GF(p^n)$ на ПЛІС *Virtex UltraScale+ XCVU9P*. З рис. 4.19 видно, що у порівнянні з двійковими полями найбільше відношення k часових витрат для структури МКГ ЧС мають поля з характеристикою 3 (в 1,8 разів більше), з структурою ФВ – поля з характеристикою 3 (в 1,27 разів більше), з структурою ЛВ – поля з характеристикою 3 (в 1,36 разів більше). Тобто, найкращим з огляду на криптографічну стійкість є поле $GF(3^n)$, яке в найгіршому випадку забезпечує в 1,27 разів більший внесок в криптографічну стійкість засобів КЗІ ніж поле $GF(2^m)$.

Таблиця 4.8

Найбільші часові затримки множення елементів розширених полів Галуа $GF(p^n)$ на ПЛІС Virtex UltraScale+ XCVU9P

Поле	Відносний час виконання множення у математичному пакеті *Maple*	Найбільша затримка апаратної реалізації множення нс.		
		МКГ ЧС	МКГ ФВ	МКГ ЛВ
$GF(2^{50})$	1.00	28.237	19.68	43.769
$GF(3^{32})$	1.46	22.935	22.697	47.345
$GF(5^{22})$	1.18	57.692	43.756	38.975
$GF(7^{18})$	0.84	29.123	36.263	37.429
$GF(13^{14})$	0.53	37.123	43.360	40.234

Таблиця 4.9

Відношення часових витрат k при програмній та апаратній реалізаціях множення елементів розширених полів Галуа $GF(p^n)$

Поле	Відношення часових витрат *k* при програмній та апаратній реалізаціях множення, при умові що коефіцієнт рахується окремо для кожної структури МКГ		
	МКГ ЧС	МКГ ФВ	МКГ ЛВ
$GF(2^{50})$	1	1	1
$GF(3^{32})$	1.797516	1.265929	1.349725
$GF(5^{22})$	0.577544	0.530725	1.325142
$GF(7^{18})$	0.814445	0.45587	0.982285
$GF(13^{14})$	0.403136	0.240554	0.576566

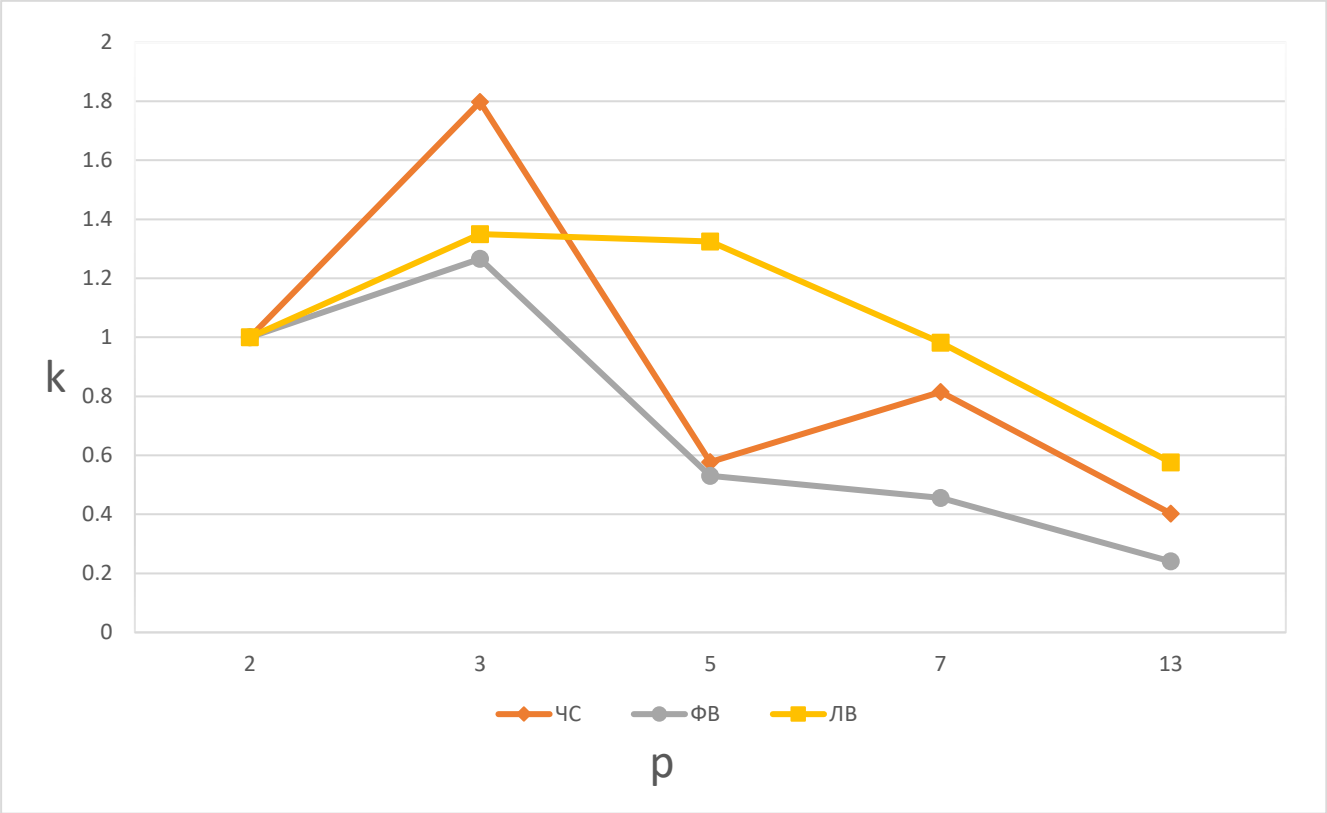


Рис. 4.19 Відношення часових витрат *k*, при програмній та апаратній реалізаціях множення елементів розширених полів Галуа $GF(p^n)$ на ПЛІС *Virtex UltraScale+ XCVU9P* за 3 структурами МКГ: ЧС, ФВ, ЛВ

У таблиці 4.10 наведено відношення часових витрат l при програмній та апаратній реалізаціях множення елементів розширених полів Галуа $GF(p^n)$ на ПЛІС *Virtex UltraScale+ XCVU9P,* при умові що коефіцієнт рахується відносно найбільшої затримки апаратної реалізації множення для МКГ ЧС. З таблиці 4.10 та графіку рис. 4.20 видно, що найкращим є поле $GF(3^n)$, яке на 27% краще ніж поле $GF(2^n)$. Більше відношення часових витрат l множення елементів розширених полів Галуа $GF(p^n)$ забезпечує більший внесок у криптографічну стійкість. Тобто, найкращим з огляду на криптографічну стійкість є поле $GF(3^n)$, яке в найгіршому випадку (структура МКГ ФВ) забезпечує в 1,27 разів більший внесок в криптографічну стійкість засобів КЗІ ніж поле $GF(2^m)$, тобто, найбільше ускладнить проведення криптоаналізу програмними методами, у порівнянні з іншими полями.

Таблиця 4.10

Відношення часових витрат l при програмній та апаратній реалізаціях множення елементів розширених полів Галуа $GF(p^n)$

Поле	Відношення часових витрат *l* при програмній та апаратній реалізаціях множення, при умові що коефіцієнт рахується спільно для всіх структур МКГ		
	МКГ ЧС	МКГ ФВ	МКГ ЛВ
$GF(2^{50})$	1	1.434807	0.645137
$GF(3^{32})$	1.797516	1.816364	0.870758
$GF(5^{22})$	0.577544	0.761488	0.854898
$GF(7^{18})$	0.814445	0.654085	0.633709
$GF(13^{14})$	0.403136	0.345148	0.371964

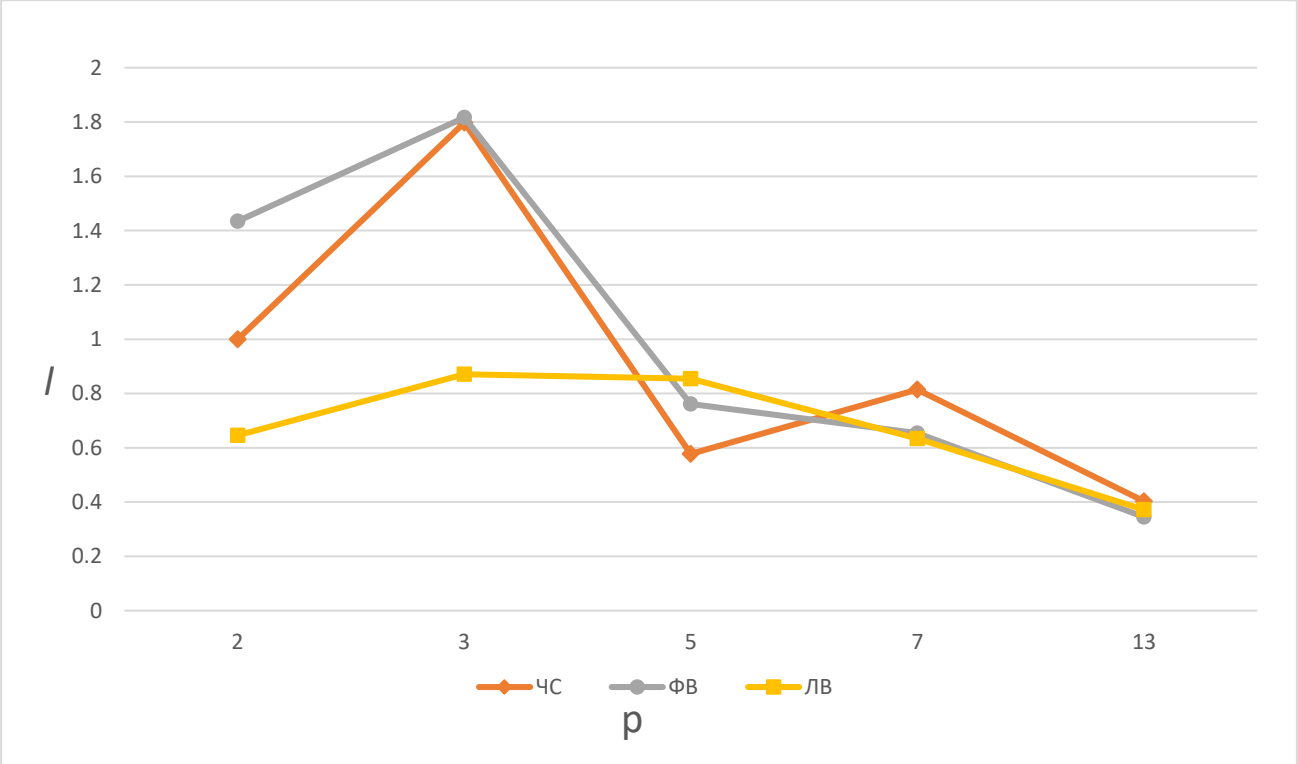


Рис. 4.20 Відношення часових витрат *l* при програмній та апаратній реалізаціях множення елементів розширених полів Галуа $GF(p^n)$ на ПЛІС *Virtex UltraScale+ XCVU9P* за 3 структурами МКГ: ЧС, ФВ, ЛВ

4.3. Впровадження результатів дисертаційної роботи

Наукові положення та висновки дисертації успішно використано під час виконання проєктних робіт у міжнародній компанії "JETSOFTPRO" (Україна, Польща, США) (Додаток В), та українській компанії ТзОВ “Кіберенергія” (Львів, Україна) (Додаток Г), та при проведенні держбюджетної науково-дослідної роботи ДБ/КІБЕР «Інтеграція методів і засобів вимірювання, автоматизації, опрацювання та захисту інформації в базисі кібер-фізичних систем» [228] (номер державної реєстрації 0115U000446), що також підтверджено відповідним Актом) (Додаток Б). Також результати дисертаційної роботи використано у Національному університеті «Львівська політехніка» Інституту комп’ютерних технологій, автоматики та метрології на кафедрі електронних обчислювальних машин при підготовці і викладанні курсів лекцій та лабораторних робіт навчальної дисципліни «Дослідження і проєктування комп’ютерних систем та мереж» (для освітньо-кваліфікаційного рівня «Магістр», спеціальність 123 «Комп’ютерна інженерія», спеціальностей «Комп’ютерні системи та мережі», «Кіберфізичні системи» та «Системне програмування»), що підтверджено відповідним Актом (Додаток А).

4.3.1 Впровадження результатів дисертаційної роботи у навчальний процес

У навчальному процесі використовуються запропоновані:

1) помножувачі елементів розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда;
2) *VHDL*-опис модифікованої комірки Гілда.

Ці результати відображені у методичних вказівках до лабораторних робіт “Проектування і моделювання елементів комп’ютерних систем та мереж” [**Error! Reference source not found.**].

4.3.2 Впровадження результатів дисертаційної роботи на виробництві

Основні результати дисертації використані міжнародною компанією "JETSOFTPRO" (Україна, Польща, США) (Додаток В) та українською компанії ТзОВ “Кіберенергія” (Львів, Україна) (Додаток Г).

Результати дисертаційної роботи, які дозволили зробити покращення у проектах міжнародної компанії "JETSOFTPRO":

1) методи створення помножувачів елементів у розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда;

2) методика оцінки апаратної складності помножувачів елементів у розширених полях Галуа $GF(p^n)$ на основі модифікованих комірок Гілда;

3) *VHDL*-описи помножувачів для розширених полів Галуа $GF(p^n)$;

4) метод перевірки розроблених помножувачів та елементів генератора помножувачів у розширених полях Галуа $GF(p^n)$.

Результати дисертаційної роботи, які дозволили зробити покращення у проектах української компанії ТзОВ “Кіберенергія”:

1) метод створення помножувачів елементів розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда на ПЛІС;

2) метод оцінювання апаратної складності на ПЛІС помножувачів елементів розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда;

3) метод оцінювання часової складності на ПЛІС помножувачів елементів розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда;

4) метод тестування генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$.

4.3.3 Створення та аналіз деталізованих структурних моделей операційних блоків для розширених полів Галуа, застосованих у системах КЗІ на базі ЕК, та інтеграція отриманих результатів у ДБ «Кібер».

Ціль держбюджетної наукової роботи ДБ/КІБЕР «Інтеграція методів і засобів вимірювання, автоматизації, опрацювання та захисту інформації в базисі кібер-фізичних систем» (із державним реєстраційним номером 0115U000446) полягала в розробці методик і технологій для вимірювання, автоматизації, обробки та захисту інформації, основаних на кібер-фізичних системах, включно з теорією створення таких систем і принципами їх діяльності, а також в їх практичній імплементації як модульної, адаптивної, реконфігурованої та масштабо-

ваної базової платформи. В рамках цієї платформи передбачається створення безпечного середовища для взаємодії між вимірювальними, обчислювальними, керуючими, комунікаційними та виконавчими компонентами. Проєкт Кібер зосереджений на вирішенні ряду задач, включаючи розробку принципів безпечного обміну, обробки та зберігання вимірювань та службової інформації. Особлива увага приділялася методам забезпечення конфіденційності, цілісності, автентичності даних, технічному та криптографічному захисті інформаційних потоків між елементами кібер-фізичних систем, а також контролю доступу до них. Розроблені методологічні принципи інформаційної та функціональної безпеки були висвітлені у монографії [103]. Результати, отримані в рамках виконання проєкту ДБ/КІБЕР, були опубліковані у відповідній монографії.

Деталі комп'ютера, на якому проводились вимірювання для більшості досліджених елементів [237]:

CPU : Intel(R) Core(TM) i5-3570 CPU.

frequency: 3.40GHz - 3.80 GHz.

memory: 16 GB.

Windows 10 , 64 bit.

4.4 Висновки до розділу 4

В 4 розділі наведено структурну схему криптопроцесора, описано основні принципи його роботи та виділено найбільш проблемний напрямок при проектуванні та розробці таких спеціалізованих систем. Наведено 3 структури побудови помножувачів елементів розширених полів Галуа $GF(p^n)$ та наведено графіки їх апаратних витрат. Результати дають можливість прогнозувати апаратні витрати помножувачів розширених полів Галуа $GF(p^n)$ з любою характеристикою поля p та будь-яким порядком поля n. Реконфігурування помножувачів елементів розширених полів Галуа $GF(p^n)$ дає змогу значно підвищити стійкість шифру до злому.

Також наведене порівняння теоретичних та реальних апаратних витрат на реалізацію помножувачів за 3 структурами МКГ: ЧС, ФВ, ЛВ. Результати є ду-

же близькими. З рисунків видно, що структура МКГ ЧС має переваги при реалізації МКГ для розширених полів Галуа $GF(p^n)$ з характеристиками поля $p = 2$ та $p = 3$. Структура МКГ ФВ забезпечує менші апаратні витрати, при реалізації помножувачів елементів розширених полів Галуа $GF(p^n)$ з характеристиками поля $p = 2$, $p = 3$, $p = 5$, $p = 7$. Найменшу апаратну складність серед не двійкових розширених полів Галуа $GF(p^n)$ має поле $GF(3^n)$, на 6% більшу апаратну складність має поле $GF(7^n)$ та на 7% поле $GF(5^n)$. Помножувачі з структурою МКГ ЛВ мають значно більшу апаратну складність ніж помножувачі з структурою ФВ, при умові, що $p \leq 13$.

За відношенням часових витрат при програмній та апаратній реалізаціях множення елементів розширених полів Галуа $GF(p^n)$ найкращим є поле $GF(3^n)$, яке на *27%* краще ніж поле $GF(2^n)$. Тобто, при реалізації на ПЛІС, поле $GF(3^n)$ буде мати щонайменше на 27 % більшу стійкість шифру до злому у порівнянні з іншими полями.

ВИСНОВКИ

У дисертації розв'язано важливе науково-технічне завдання створення реконфігурованих вузлів КЗІ, які оперують у КФС елементами розширених полів Галуа $GF(p^n)$, та у порівнянні з розширеними двійковими полями $GF(2^m)$ мають більшу криптографічну стійкість. При цьому $p^n \approx 2^m$, де $p > 2$ – просте число, конфігурована характеристика поля, n, m – порядок утворюючого поле полінома, $m \leq 1024$. Основним вузлом для аналізу обрано апаратний паралельний помножувач елементів таких розширених полів Галуа $GF(p^n)$, який побудовано на основі реконфігурованого вузла – МКГ, що дало можливість провести порівняльну оцінку помножувачів для різних розширених полів Галуа $GF(p^n)$ і визначити найкраще, за вказаними характеристиками, поле для реалізації помножувачів. При цьому у роботі досягнуто таких результатів:

1. Отримав подальший розвиток метод оцінювання часової складності множення елементів розширених полів Галуа $GF(p^n)$ апаратним способом, за яким помножувач складається з МКГ, що дало можливість визначити поле $GF(3^n)$, у якому відношення часів множення програмним та апаратним способами перевищує таке відношення в інших полях щонайменше в 1,27 раза, чим забезпечує найбільшу криптографічну стійкість засобів КЗІ при інших однакових умовах.

2. Отримав подальший розвиток метод оцінювання апаратної складності помножувачів елементів розширених недвійкових полів Галуа $GF(p^n)$, де $p > 2$, який на відміну від відомих розглядає помножувачі для полів з приблизно однаковим порядком p^n і які складаються з МКГ, що дало можливість визначити поле $GF(3^n)$, де помножувачі мають щонайменше на 6 % меншу апаратну складність, у порівнянні з помножувачами для інших недвійкових полів.

3. Вперше запропоновано метод створення для ПЛІС генераторів моделей (ядер) реконфігурованих паралельних помножувачів елементів розширених полів Галуа $GF(p^n)$ для вузлів КЗІ у КФС, за яким на відміну від відомих генерується 3 варіанти помножувачів з 3 структурами МКГ, що дало можливість ство-

рити описи моделей помножувачів та визначити реальну апаратну складність кожного із помножувачів та обрати найкращу реалізацію для кожного конкретного поля.

4. Вперше запропоновано метод тестування генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$, який на відміну від відомих використовує 2 різні еталони (2 різні математичні пакети) для перевірки згенерованих помножувачів на основі МКГ, що дало можливість підтвердити правильну роботу генераторів помножувачів та самих помножувачів для всіх варіантів їх реалізації.

5. Розроблено генератори ядер (генератори моделей) помножувачів елементів розширених полів Галуа $GF(p^n)$ такими, що порядок поля $p^n < 2^{1024}$. Генератори створюють моделі помножувачів на основі МКГ з трьома варіантами структури МКГ: ЧС, ФВ, ЛВ. Це дало можливість створити ряд ядер помножувачів елементів розширених полів Галуа $GF(p^n)$ з трьома структурами МКГ та провести їх імплементацію на ПЛІС, провести порівняння результатів імплементації помножувачів для полів $GF(2^{15})$, $GF(2^{50})$, $GF(2^{998})$, $GF(3^9)$, $GF(3^{32})$, $GF(3^9)$, $GF(5^6)$, $GF(5^{22})$, $GF(7^5)$, $GF(7^{18})$, $GF(13^3)$, $GF(13^{14})$, $GF(13^{270})$. Показано збіжність теоретичних та практичних результатів оцінювання апаратної та часової складностей помножувачів.

6. На основі розроблених і вдосконалених методів проведено аналіз апаратних складностей створених помножувачів елементів розширених полів Галуа $GF(p^n)$, $p > 2$, що дало можливість визначити поля, які найкраще підходять для реалізації помножувачів. Найменшу апаратну складність мають помножувачі для полів $GF(3^n)$, на 6% більшу – для полів $GF(7^n)$ та на 8% - для полів $GF(5^n)$. Проведено аналіз часових складностей помножувачів елементів розширених полів Галуа $GF(p^n)$, $p > 2$, за відношенням часових витрат при програмній та апаратній реалізаціях, найкращим є поле $GF(3^n)$, відношення часових витрат у якому в 1,27 разів більше ніж у полі $GF(2^m)$.

Результати роботи впроваджено:

1) в міжнародній компанії “JETSOFTPRO”;

2) в українській компанії ТзОВ “Кіберенергія”;

3) в держбюджетній науково-дослідній роботі ДБ/КІБЕР (номер державної реєстрації 0115U000446);

4) у навчальному процесі кафедри ЕОМ НУ ”ЛП”.

Отримані під час виконання дисертаційної роботи наукові результати показують досягнення поставленої в роботі мети підвищення криптографічної стійкості засобів КЗІ, які використовуються в складі КФС, шляхом розвитку методів та засобів створення реконфігурованих операційних пристроїв (а саме, помножувачів) для роботи з елементами розширених полів Галуа $GF(p^n)$ з характеристиками p та з порядками n утворюючого поле полінома такими, що $p^n \approx 2^m$, де $p > 2$, $m \leq 1024$ створюють методологічну базу для розробки вузлів КЗІ, які дозволяють підвищити надійність, достовірність та захищеність сучасних апаратних засобів КЗІ, які працюють з використанням ЕК та розширених полів Галуа $GF(p^n)$.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. R. Z. Mahmood and A. F. Fathil, "High Speed Parallel RC4 Key Searching Brute Force Attack Based on FPGA," 2019 International Conference on Advanced Science and Engineering (ICOASE), 2019, pp. 129-134, doi: 10.1109/ICOASE.2019.8723737.

2. ДСТУ 9041:2020. Інформаційні технології. КРИПТОГРАФІЧНИЙ ЗАХИСТ ІНФОРМАЦІЇ. Алгоритм шифрування коротких повідомлень, що ґрунтується на скручених еліптичних кривих Едвардса.

3. ДСТУ ISO/IEC 15946-5:2019 (ISO/IEC 15946-5:2017) "Інформаційні технології. Методи захисту. Криптографічні методи на основі еліптичних кривих. Частина 5. Генерування еліптичних кривих".

4. ДСТУ 7624:2014. Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення.

5. ДСТУ 7564:2014 Інформаційні технології. Криптографічний захист інформації. Функція хешування.

6. Жолубак, І. М., Глухов, В. С. Визначення розширеного поля Галуа GF(dm) з найменшою апаратною складністю помножувача [Текст]. / І. М. Жолубак, В. С. Глухов // Вісник Національного університету «Львівська політехніка» “Інформаційні системи та мережі”, № 854. Львів, 2016. С. 63 – 69.

7. Жолубак, І. М., Дослідження апаратної складності помножувачів розширених полів Галуа GF(dm). / Жолубак, І. М., Глухов, В. С. / Кіберфізичні системи: досягнення та виклики. – 2016 р. Матеріали II Наукового семінару, 21–22 червня 2016 року, Львів. С. 98 – 104.

8. І.М. Жолубак. Порівняння апаратних витрат помножувачів елементів розширених полів Галуа / В.С. Глухов, І.М. Жолубак / 17-а міжнародно науково-практична конференція «Сучасні інформаційні та електронні технології» Одеса, Україна, 23—27 травня 2016 р. С. 133 – 134.

9. Ivan Zholubak. The research of the binary codes program complication and application in Cyber Physical Systems. / Andrii Kostyk, Ivan Zholubak / 6th

International Youth Science Forum LITTERIS ET ARTIBUS 2016, Computer Science & Engineering (CSE-2016). November 24–26 2016, Lviv, Ukraine.

10. Zholubak I., Hlukhov V. Hardware complexity of multipliers of extended Galois field in FPGA. 7th International Youth Science Forum LITTERIS ET ARTIBUS 2017, Computer Science & Engineering (CSE-2017). Proceedings. Pp. 420 – 421. November 23–25 2017, Lviv, Ukraine.

11. І.М. Жолубак. Визначення розширеного поля Галуа GF(d^m) з найменшою апаратною складністю помножувача / І.М. Жолубак, В.С. Глухов. / *Інформаційні технології та комп'ютерне моделювання*: матеріали статей Міжнародної науково-практичної конференції, 23 – 28 травня 2016 року. - Івано-Франківськ. 2016. С. 80 - 81.

12. І. М. Жолубак, А. Т. Костик, В. С. Глухов. Особливості опрацювання елементів трійкових полів Галуа на сучасній елементній базі/ І. М. Жолубак, А. Т. Костик, В. С. Глухов // Вісник Національного університету "Львівська політехніка" "Комп'ютерні системи та мережі". – Львів: - 2015. - Вип. 830. - С. 27 - 33.

13. І. М. Жолубак, В. С. Глухов. Апаратні витрати помножувачів полів Галуа GF(dm) з великою основою/ І. М. Жолубак, В. С. Глухов // Вісник Національного університету «Львівська політехніка» "Комп'ютерні науки та інформаційні технології". – Львів, - 2017. – Вип. 864. - С. 77 – 82.

14. І. М. Жолубак, В. С. Глухов. Реалізація у ПЛІС помножувачів елементів полів Галуа високих порядків/ І. М. Жолубак, В. С. Глухов // Вісник Національного університету «Львівська політехніка» "Комп'ютерні системи та мережі". – Львів, - 2017. – Вип. 881. - С. 41 – 47.

15. Жолубак І. М., Вбудований контроль пристроїв для опрацювання елементів розширених полів Галуа / Р. М. Еліас, В. С. Глухов, М. Рахма, І. М. Жолубак / Вісник Національного університету «Львівська політехніка» "Комп'ютерні системи та мережі", № 905. Львів, 2018. С. 64-72

16. Ivan Zholubak. Features of multiplication execution of operations in binary and ternary Galois fields / Andrii Kostyk, Ivan Zholubak / 5th International Youth

Science Forum LITTERIS ET ARTIBUS 2015. November 26–28 2015. Lviv, Ukraine.

17. Жолубак І. М., Ємнісна складність та вбудований контроль пристроїв для опрацювання елементів розширених полів Галуа / Родріг Еліас, Валерій Глухов, Мохаммед Рахма, Іван Жолубак / Електротехнічні та комп`ютерні системи. – Одеса : – 2018. Вид-во Наука і техніка. 29(105), с. 95-102

18. Жолубак Іван. Принципи побудови та проектування операційних вузлів для полів Галуа, що використовуються в задачах криптографічному захисті інформації на основі еліптичних кривих / В.С. Глухов, І.М. Жолубак, Мохаммед Кадім Рахма Рахма / Кіберфізичні системи: багаторівнева організація та проектування [Текст]: монографія – А.О. Мельник та інші. За редакцією професора А. О. Мельника. Львів: «Магнолія 2006», 2019. 238 с. С. 58-131.

19. Жолубак І. М., Курман П. В. Система безконтактних платежів на основі технології NFC. Вісник Національного університету «Львівська політехніка» "*Комп'ютерні системи та мережі*", № 1. Львів, 2022. С. 28 – 37.

20. Жолубак І. М., Матвієць В. Ю. Трекер для сонячних електростанцій. Вісник Національного університету «Львівська політехніка» "*Комп'ютерні системи та мережі*", № 1. Львів, 2022. С. 37 – 46.

21. Bohdan Marii, Ivan Zholubak. Features of Development and Analysis of REST Systems. Advances in Cyber-Physical Systems. Volume 7. Number 2. Lviv Polytechnic National University. 2022. pp. 121 – 129, DOI: https://doi.org/10.23939/acps2022.02.121

22. Жолубак І. М., Аналіз алгоритмів множення в полях Галуа для криптографічного захисту інформації. Вісник Національного університету «Львівська політехніка» "*Інформаційні системи та мережі*", № 13. Львів, 2023. С. 338 – 349, DOI: https://doi.org/10.23939/sisn2023.13.338

23. Zholubak, I.M., Hlukhov, V.S. Galua Field Multipliers Core Generator. International Journal of Computer Network and Information Security, 2023, 3, 1-14 Published Online on June 8, 2023 by MECS Press (http://www.mecs-press.org/) DOI: 10.5815/ijcnis.2023.03.01

24. Zholubak, I.M., Hlukhov V.S., "Comparison of hardware complexity of multipliers GF(p^m)" *2023* 12th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications *(IDAACS)*, Dortmund, Germany, 2023, pp. 1-5, doi: 10.1109/IDAACS-SWS50031.2020.9297059, (Scopus Conference paper).

25. Zholubak, I.M., Hlukhov V.S., "Verification of Synthesized by the IP-core Generator Multipliers of Extended Galois Fields GF(p^n) Elements" 2013 13th International Conference Dependable Systems, Services and Technologies (DESSERT), Greece, Athens, 2023, pp. - , (Scopus Conference paper).

26. Zholubak, I.M., Hlukhov V.S., "Validation of Multipliers for Elements of Extended Galois Fields GF(p^n) and Multipliers IP-core Generator" 2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), Ukraine, Lviv, 2023, pp. - , (Scopus Conference paper).

27. Zholubak I. M., Galois Fields Elements Processing Units for Cryptographic Data Protection in Cyber-Physical Systems / V. Hlukhov, I. Zholubak, A. Kostyk, M. Rahma / Advances in Cyber-Physical Systems, Вид-во Національного університету Львівська політехніка. - Volume 2, Number 2, 2017. – pp. 47- 53.

28. Zholubak I. M., FPGA cores for fast multiplicative inverse calculation in Galois Fields / Rodrigue Elias, Valerii Hlukhov, Mohammed Rahma, Ivan Zholubak. 80 80 101 25 119 153 Електротехнічні та комп`ютерні системи. – Одеса : – 2018. Вид-во Наука і техніка. 27(103), с. 227-233

29. Ivan Zholubak. Automation System for Configuration of Cryptographic Data Protection Unit Model / Ivan Zholubak, Mohammed Kadhim Rahma and Valeriy Hlukhov / Proceedings of 4th International Workshop on Theory of Reliability and Markov Modeling for Information Technologies (WS TheRMIT 2018, in frameworks of the 14th International Conference ICTERI2018). May 14, 2018, Kyiv, pp. 669-679.

30. Ivan Zholubak. Devices for Multiplicative Inverse Calculation in Binary Galois Fields / Valeriy Hlukhov, Mohammed Rahma and Ivan Zholubak. / Proceedings of

9th International IEEE Conference Dependable Systems, Services and Technologies DESSERT'2018. Kyiv, May 24-27, pp. 275-278.

31. Ivan Zholubak, Mohamed Rahma and Valeriy Hlukhov. Automation system program models configuration of cryptography cells in cyber-phisical systems. 14th International Conference on ICT in Education, Research, and Industrial Applications ICTERI 2018. Kyiv, Ukraine, May 14-17, 2018. Proceedings of the 14th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops. Pp. 669-679.

32. Ivan Zholubak. Hardware components for post-quantum elliptic curves cryptography / Rodrigue Elias, Valerii Hlukhov, Mohammed Rahma, Ivan Zholubak. / Proceedings of International Conference "Advanced Computer Information Technologies", June 1-3, 2018 in Ceske Budejovice, Czech Republic, pp. 236-239.

33. Мельник А. О., Мельник В. А. Персональні суперкомп'ютери: архітектура, проектування, застосування. Монографія. Львів: Видавництво Львівської політехніки, 2013. 516 с.

34. Anatoliy Melnyk, Viktor Melnyk, Heterogeneous computing: from the ASIC-based hardware accelerators to the reconfigurable and self-configurable computer systems, in: Proceedings of the 6-th International Conference on Advanced Computer Systems and Networks, ACSN-2013, Lviv, 2013, pp. 9-12

35. Anatoliy Melnyk, Viktor Melnyk, Self-Improvable Computer System Model and Architecture Based on Reconfigurable Hardware, Automatic Design and Synthesis Tools and Artificial Intelligence Technologies, The Fourth International Workshop on Computer Modeling and Intelligent Systems, April 27, 2021, Zaporizhzhia, Ukraine, pp 1 – 12

36. G. D'Andrea and L. Pomante, "Design for ReConfigurability: An Electronic System Level Methodology to Exploit Reconfigurable Platforms," *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, 2020, pp. 355-356, doi: 10.1109/FPL50879.2020.00066.

37. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Introduction to Algorithms. Fourth Edition, Publisher: The MIT Press, 2022, 1312p.

38. K. Lata, S. Chhabra and S. Saini, "Hardware–Software Co-Design Framework for Data Encryption in Image Processing Systems for the Internet of Things Environment," in *IEEE Consumer Electronics Magazine*, vol. 11, no. 4, pp. 92-97, 1 July 2022, doi: 10.1109/MCE.2021.3115999.

39. Robert Sedgewick, Kevin Wayne, Algorithms (4th Edition), Publisher : Addison-Wesley Professional, 2022, 976p.

40. Huang M. Reconfiguration and Communication-Aware Task Scheduling for High-Performance Reconfigurable Computing / M. Huang, V.K. Narayana, H. Simmler, O. Serres, T. El-Ghazawi // Transactions on Reconfigurable Technology and Systems (TRETS). – US, NY, New York, ACM, 2010. – Vol. 3, № 4. – P. 20.1-20.25.

41. Мельник А. О. Кіберфізичні системи: проблеми створення та напрями розвитку / А. О. Мельник // Вісник Національного університету "Львівська політехніка". – 2014. – № 806 : Комп'ютерні системи та мережі. – С. 154– 161. – Бібліографія: 31 назва.

42. Дудикевич В. Б. Парадигма та концепція побудови багаторівневої комплексної системи безпеки кіберфізичних систем / В. Б. Дудикевич, В. М. Максимович, Г. В. Микитин // Вісник Національного університету «Львівська політехніка». Серія: Автоматика, вимірювання та керування : збірник наукових праць. – 2015. – № 821. – С. 3–7. – Бібліографія: 8 назв.

43. S. Lata Tripathi , & S. Dwivedi (Eds.). (2022). Electronic Devices and Circuit Design: Challenges and Applications in the Internet of Things (1st ed.). Apple Academic Press . https://doi.org/10.1201/9781003145776 . ISBN: 9781771889933

44. Sinitò D., Santarcangelo V., Stanco F., Giacalone M.,Industry 4.0: Machinery integration with supply chain and logistics in compliance with Italian regulations,(2023) MethodsX, 11, DOI: 10.1016/j.mex.2023.102269

45. W. Xie, S. Lu, L. Zhang, X. He, R. Gu and Y. Lu, "Industrial Internet identification code compatibility method for power industry," 2023 IEEE 6th

Information Technology,Networking,Electronic and Automation Control Conference (ITNEC), Chongqing, China, 2023, pp. 1168-1171, doi: 10.1109/ITNEC56291.2023.10082435.

46. Prabhakara Rao T., Satyanarayana Murthy B., Extended group-based verification approach for secure M2M communications, (2023) International Journal of Information Technology (Singapore), 15 (5), pp. 2479 - 2488, DOI: 10.1007/s41870-023-01284-w

47. G. F. Pittalà and W. Cerroni, "Intelligent Service Provisioning in Fog Computing," 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, 2023, pp. 354-357, doi: 10.1109/NetSoft57336.2023.10175416.

48. Kris Jamsa, Cloud Computing: SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and More, Publisher : Jones & Bartlett Learning; 2nd edition (2022), 324p.

49. Aliero, M. S., Qureshi, K. N., Pasha, M. F., Ghani, I., & Yauri, R. A. (2021). Systematic mapping study on energy optimization solutions in smart building structure: Opportunities and challenges. Wireless Personal Communications, 119(3), 2017–2053. https://doi.org/10.1007/s11277-021-08316-3

50. Pedro H. J. Nardelli, "Dynamics of Cyber-Physical Systems," in Cyber-physical Systems: Theory, Methodology, and Applications , IEEE, 2022, pp.155-175, doi: 10.1002/9781119785194.ch8.

51. Chen-Ching Liu; Juan C. Bedoya; Nitasha Sahani; Alexandru Stefanov; Jennifer Appiah-Kubi; Chih-Che Sun; Jin Young Lee; Ruoxi Zhu, Cyber-Physical System Security of Distribution Systems , now, 2021.

52. Pedro H. J. Nardelli, "The Three Layers of Cyber-Physical Systems," in Cyber-physical Systems: Theory, Methodology, and Applications , IEEE, 2022, pp.131-154, doi: 10.1002/9781119785194.ch7.

53. Anatoliy Melnyk. Cyber-physical systems multilayer platform and research framework. Advances in Cyber-Physical Systems. Volume 1. Number 1. Lviv Polytechnic National University. 2016. pp. 1 – 6.

54. Christian Sonntag; Sebastian Engell; Tariq Samad, "Challenges and Potential for EU–US Collaboration at the Intersection of the Internet of Things and Cyber-physical Systems," in ICT Policy, Research, and Innovation: Perspectives and Prospects for EU-US Collaboration , IEEE, 2020, pp.111-144, doi: 10.1002/9781119632481.ch5.

55. Pedro H. J. Nardelli, "Introduction," in Cyber-physical Systems: Theory, Methodology, and Applications , IEEE, 2022, pp.1-17, doi: 10.1002/9781119785194.ch1.

56. Li, Y., Huang, Gq., Wang, Cz. et al. Analysis framework of network security situational awareness and comparison of implementation methods. J Wireless Com Network 2019, 205 (2019). https://doi.org/10.1186/s13638-019-1506-1

57. Muhammad Faheem Mushtaq, Sapiee Jamel, Abdulkadir Hassan Disina, Zahraddeen A. Pindar, Nur Shafinaz Ahmad Shakir, Mustafa Mat Deris, A Survey on the Cryptographic Encryption Algorithms, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 8, No. 11, 2017

58. Bansal, B., Jenipher, V.N., Jain, R., Dilip, R., Kumbhkar, M., Pramanik, S., Roy, S. and Gupta, A. (2022). Big Data Architecture for Network Security. In Cyber Security and Network Security (eds S. Pramanik, D. Samanta, M. Vinay and A. Guha). *https://doi.org/10.1002/9781119812555.ch11*

59. Obaida M. Al-hazaimeh, Moyawiah A. Al-Shannaq, Mohammed J. Bawaneh1, Khalid M.O. Nahar, “Analytical Approach for Data Encryption Standard Algorithm”, Short Paper—Paper Formatting for online-journals.org, Vol. 1, No.1, 2022

60. A. S. Jamil and A. M. S. Rahma, "Cyber Security for Medical Image Encryption using Circular Blockchain Technology Based on Modify DES Algorithm," International Journal of Online & Biomedical Engineering, vol. 19, 2023.

61. O. C. Abikoye, A. D. Haruna, A. Abubakar, N. O. Akande, and E. O. Asani, "Modified advanced encryption standard algorithm for information security," Symmetry, vol. 11, p. 1484, 2019.

62. Widiyanto WW, Iskandar D, Wulandari S, Susena E, Susanto E. Implementation Security Digital Signature Using Rivest Shamir Adleman (RSA) Algorithm As A Letter Validation And Distribution Validation System. In2022 International

Interdisciplinary Humanitarian Conference for Sustainability (IIHC) 2022 Nov 18 (pp. 599-605). IEEE.

63. Kuang R, Perepechaenko M, Toth R, Barbeau M. Benchmark Performance of a New Quantum-Safe Multivariate Polynomial Digital Signature Algorithm. In2022 IEEE International Conference on Quantum Computing and Engineering (QCE) 2022 Sep 18 (pp. 454-464). IEEE.

64. M. A. Dar, A. Askar, D. Alyahya, and S. A. Bhat, "Lightweight and Secure Elliptical Curve Cryptography (ECC) Key Exchange for Mobile Phones," International Journal of Interactive Mobile Technologies, vol. 15, 2021.

65. Z. Zhang, J. Liu, S. Zhang, H. Zhu and B. Zhang, "Dynamic Event-triggered Scheme and Output Feedback Control for CPS under Multiple Cyber Attacks," 2021 IEEE 10th Data Driven Control and Learning Systems Conference (DDCLS), Suzhou, China, 2021, pp. 299-303, doi: 10.1109/DDCLS52934.2021.9455613.

66. 2019, Volgograd, Russia, September 16–19, 2019, Proceedings, Part I 3, pp. 382-391. Springer International Publishing, 2019.

67. C. Sun and L. Zhang, "Design and Modeling of Intelligent Home Security Monitoring System Based on CPS," 2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2021, pp. 186-189, doi: 10.1109/ICSESS52187.2021.9522189.

68. Химич ІГ, Гой ВВ. ЕЛЕКТРОННИЙ ЦИФРОВИЙ ПІДПИС–ЗАХИСТ ВІД ПІДРОБКИ ЕЛЕКТРОННОГО ДОКУМЕНТА. Цифрове суспільство: фінанси, економіка, управління: матеріали Між-народної науково-практичної конференції. Дніпро: Університет митної справи та фінансів, 2020. 315 с. Матеріали Міжнародної науково-практичної конференції, які включе-ні до збірника, присвячено актуальним проблемам розвитку фінансової си. 2020:117.

69. Блинков, Володимир Геннадійович. "Цифрова реалізація ідентифікаційних документів на мобільних пристроях з гарантуванням автентичності." Master's thesis, КПІ ім. Ігоря Сікорського, 2019.

70. Реута, Гліб Васильович. "Забезпечення цілісності даних з використанням цифрових підписів і сертифікатів." Bachelor's thesis, Київ, 2023.

71. Gorbenko, I. D., and A. A. Zamula. "Науковий підхід до ймовірнісної оцінки захищеності інформації від нав'язування хибних повідомлень у телекомунікаційних системах." Radiotekhnika 208 (2022): 7-15.

72. Alagic G, Alperin-Sheriff J, Apon D, Cooper D, Dang Q, Kelsey J, Liu YK, Miller C, Moody D, Peralta R, Perlner R. Status report on the second round of the NIST post-quantum cryptography standardization process. US Department of Commerce, NIST. 2020 Jul 22;2.

73. Фізулі-кизи, Саміра Султанова. "ОГЛЯД ТА ПОРІВНЯННЯ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ НА ОСНОВІ ЕЛІПТИЧНИХ КРИВИХ." Інформаційні моделюючі технології, системи та комплекси (ІМТСК-2023): IV міжнародна науково-практична конференція. 25-26 травня 2023 р., Черкаси, Україна.– Черкаси: Черкаський національний університет імені Богдана Хмельницького, 2023.–153 с. В матеріалах конференції відображені результати теоретичних та (2023): 76.

74. J. VenkataGiri and A. Murty, "Elliptical Curve Cryptography Design Principles," 2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT), Bangalore, India, 2021, pp. 889-893, doi: 10.1109/RTEICT52294.2021.9573662.

75. D. Jancarczyk, V. Rudnytskyi, R. Breus, M. Pustovit, O. Veselska and R. Ziubina, "Two-Operand Operations of Strict Stable Cryptographic Coding with Different Operands' Bits," 2020 IEEE 5th International Symposium on Smart and Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS), Dortmund, Germany, 2020, pp. 1-8, doi: 10.1109/IDAACS-SWS50031.2020.9297067.

76. ДСТУ ISO/IEC 7498-1:2004. Інформаційні технології. Взаємозв'язок відкритих систем. Базова еталонна модель. Частина 1. Базова модель (ISO/IEC 7498-1:1994).

77. Закон України «Про електронний ЕЦП» від 22.05.2003 № 852-IV (Відомості Верховної Ради (ВВР), 2003, N 36, ст.276).

78. ДСТУ ISO/IEC 13888-1:2015. Інформаційні технології; Методи захисту. Неспростовність. Частина 1. Загальні положення (ISO/IEC 13888-1:2009, IDT).

79. ДСТУ ISO/IEC 15946-1:2015 Інформаційні технології. Методи захисту. Криптографічні методи, що грунтуються на еліптичних кривих. Частина 1. Загальні положення.

80. Rahma, Mohammed Kadhim. Galois Field Operational unit For Elliptic Curve Cryptography Digital Signature. V Міжнародний молодіжний науковий форум “Litteris et Artibus”. 26–28 листопада, 2015. Україна, Львів. pp. 66-71.

81. IEEE 1363-2000. Standard Specifications for Public-Key Cryptography. Copyright © 2000 IEEE. All rights reserved.

82. ДСТУ 4145-2002. Інформаційні технології. Криптографічний захист інформації. ЕЦП, що ґрунтується на еліптичних кривих. Формування та перевірян-ня. Київ. 2003.

83. Николайчук Я. М. Коди поля Галуа : теорія та застосування [Текст] : монографія / за ред. Я. М. Николайчука. – Тернопіль : ТзОВ "Тернграф", 2012. - 392 с.

84. Я. М. Николайчук, Н. Я. Возна, В. М. Грига, Б. Б. Круліковський, А. Я. Давлетова. Високопродуктивні матричні та потокові перемножувачі цифрових даних. Математичне та комп'ютерне моделювання. Серія: Технічні науки: зб. наук. пр. — Кам’янець-Подільський: Кам'янець-Подільськ. нац. унт, 2019. — Вип. 19. — С. 101-107

85. Я. М. Николайчук, О. І. Волинський, П. В. Гуменний, Т. І. Пастух. Методи міжбазисних перетворень багаторозрядних кодів теоретико-числових базисів Радемахера–Крестенсона. Математичне та комп'ютерне моделювання. Серія: Технічні науки. Випуск 15. Інститут кібернетики ім. В. М. Глушкова НАН України. 2017. С. 143 – 149.

86. Николайчук, Я. М. Метод факторизации многоразрядных чисел на основе свойств квадратичности вычетов в системе остаточных классов / Я. Н. Николайчук, С. В. Ивасьев, И. З. Якименко, М. Н. Касянчук // Вестник Брестского

государственного технического университета. – 2015. – № 5(95): Физика, математика, информатика. – С. 42–45.

87. Глухов В.С. Особливості виконання операцій над матрицями в полях Галуа // Вісник Національного університету “Львівська політехніка” “Комп’ютерні системи проектування. Теорія і практика”. Вип. 564. Львів, 2006. С.35-39.

88. Глухов В.С. Обчислювальний пристрій для операцій над еліптичними кривими // Вісник Національного університету “Львівська політехніка” “Комп`ютерні системи та мережі”. № 573. Львів, 2006. С.54-61.

89. Глухов В.С. Порівняння поліноміального та нормального базисів представлення елементів полів Галуа // Вісник Національного університету “Львівська політехніка” “Комп’ютерні системи проектування. Теорія і практика”. №591, с.22–27. Львів, 2007р.

90. В.С. Глухов. Вдосконалення алгоритму обчислення оберненого елемента GF (2t) в нормальному базисі // Вісник Національного університету “Львівська політехніка” “Комп`ютерні системи та мережі”. №603. Львів, 2007. С.20-26.

91. Глухов В.С. Багаторівнева організація операційного пристрою для роботи з елементами поля Галуа, представленими у нормальній формі. Збірник матеріалів міжвузівської науково-технічної конференція науково-педагогічних працівників «Проблеми та перспективи розвитку економіки і підприємництва та комп’ютерних технологій в Україні» . - Львів: Ліга-Прес. 2007р.

92. В.С.Глухов. Оцінка апаратних витрат на реалізацію багаторівневої комп’ютерної системи // Вісник Національного університету «Львівська політехніка» «Комп’ютерні науки та інформаційні технології» № 629. Львів, 2008. С.13-20.

93. Глухов В.С. Вибір багатоядерних структур для пристроїв обробки ЕЦП // Вісник Національного університету “Львівська політехніка” “Комп`ютерні системи та мережі”. № 658. Львів, 2009. С.35 – 39.

94. Глухов В.С. Вбудований контроль множення в гаусівському нормальному базисі типу 2 полів Галуа GF(2m). Науково-технічний журнал «Радіоелектронні і комп’ютерні системи 6(47). Національний аерокосмічний університет ім.

М.Є. Жуковського «Харківський авіаційний інститут». Харків. «ХАІ». 2010. С. 255 – 259.

95. Глухов В.С. Оцінювання апаратних витрат на реалізацію багаторівневої комп'ютерної системи з врахуванням закону Амдаля // Вісник Національного університету «Львівська політехніка» «Комп'ютерні науки та інформаційні технології» № 663. Львів, 2010. С.17 - 23.

96. Глухов В.С. Особливості виконання операцій у простих полях Галуа GF(p) у сучасних засобах захисту інформації // Вісник Національного університету "Львівська політехніка" "Комп'ютерні системи та мережі", № 717. Львів, 2011. С.3 - 9.

97. В.С.Глухов, Т.С.Берко. Перевірка пристроїв для обробки ЕЦП, що ґрунтуються на еліптичних кривих / Науково-соціальний часопис "Технічні вісті". Орган Українського інженерного товариства у Львові, 2007/1(25), 2(26), с. 53-57.

98. Глухов В. С., Глухова О. В. Результати оцінювання структурної складності помножувачів елементів полів Галуа [Текст] / В. С. Глухов, О. В. Глухова // Вісник Національного університету "Львівська політехніка" "Комп'ютерні системи та мережі". – Львів: - 2013. - Вип. 773. - С. 27 - 32.

99. В. С. Глухов, В. А. Голембо. Методичні вказівки до курсової роботи "Арифметичні та логічні основи комп'ютерних технологій" з дисципліни "Комп'ютерна логіка". Львів: Видавництво Еаціонального університету "Львівська політехніка", 2014.

100. Глухов В. С., Еліас Р. М., Мельник А. О. Особливості реалізації на ПЛІС секційних помножувачів елементів полів Галуа GF(2m) з надвеликим степенем [Текст] / В.С. Глухов., Р.М. Еліас, А.О. Мельник // "Комп'ютерно-інтегровані технології:освіта, наука, виробництво" - науковий журнал, Луцький національний технічний університет. – Луцьк: 2013. - № 12. - С. 103 – 106.

101. Глухов В.С., Жолубак І.М., Костик А.Т, Рахма М.К.Р. Проектування і моделювання елементів комп'ютерних систем та мереж. Методичні вказівки до лабораторних робіт з дисципліни"Дослідження і проектування комп'ютерних систем та мереж" для студентів освітньо-кваліфікаційного рівня «Магістр»,

спеціальність 123 «Комп'ютерна інженерія», Поліграфічний центр Видавництва Національного університету "Львівська політехніка". Львів. 2019. 118 с.

102. В. С. Глухов, А. Т. Костик. Використання сучасних ПЛІС для опрацювання елементів полів Галуа GF(p^q). Дев'ята конференція ХУ ПС ім. І. Кожедуба, 17 – 18 квітня, 2013 року. Харків. 2013. С . 178.

103. Глухов В.С., Ногаль М.В. Спеціалізований однорозрядний процесор для захисту інформації в гарантоздатних системах. Науково-технічний журнал «Радіоелектронні і комп'ютерні системи 5 (32). Національний аерокосмічний університет ім. М.Є. Жуковського «Харківський авіаційний інститут». Харків. «ХАІ». 2008. С. 104-109.

104. Кочубинский А.И. Эллиптические кривые в криптографии. //Безопасность информации. – 2, - 2000, с. 18 – 31. www.bitis.com.ua:8080/downloads/elliptica.doc

105. Ковтун М. Г. Застосування кривих Едвардса для захищеної реалізації механізмів електронного цифрового підпису згідно з *ДСТУ 4145-2002* // *Системи обробки інформації* : зб. наук. праць. — *Харків*, 2017. — Вип. 5. — № 151. — С. 130-137. — *ISSN 1681-7710*. — 025603868. *Архівовано* з джерела 22 січня 2018.

106. Сліпі мультипідписи на основі стандартів ДСТУ 4145-2002 та ГОСТ Р 34.10-2001 / А.И. Кочубинский, Н.А. Молдовян, А.М. Фаль // Доповіді НАН України : зб. наук. праць. — *Київ* : Видавничий дім "Академперіодика" НАН України, 2012. — № 3. — С. 38-44. — *ISSN 1025-6415*. *Архівовано* з джерела 21 січня 2018.

107. ДСТУ ISO/IEC 15946-3:2006 Інформаційні технології. Методи захисту. Криптографічні методи, що грунтуються на еліптичних кривих. Частина 3. Установлення ключів.

108. Мельник А. О., Коркішко Т. А. Система підтримки виконання алгоритмів криптографічного захисту інформації на основі програмованого процесора та криптографічних акселераторів // Вісник Державного університету "Львівська політехніка" № 385 «Комп'ютерні системи та мережі». Львів. Видавництво Державного університету «Львівська політехніка». 2000. С. 77 – 80.

109. Кудін, Антон, Володимир Ткач, and Світлана Носок. "Проблеми побудови аксіоматики сучасної криптографії: інформаційний, обчислювальний та інформаційно-обчислювальний підходи." ФІЗИКО-МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ 36 (2023): 137-142.

110. Горбенко І. Д., Гріненко Т. О. Захист інформації в інформаційнотелекомунікаційних системах : Навч. посіб. для студ. спец. “Комп'ютерні науки”, “Комп'ютена інженерія”, “Прикладна математика”, “Інформаційна безпека” вищ. навч. закл. / Харківський національний ун-т радіоелектроніки. - Х. : ХНУРЕ, 2004. - Бібліогр.: с. 364-368. Ч. 1 : Криптографічний захист інформації. - 368с. : рис. - ISBN 966-659-081-6.

111. Шапочка Н. В., Горбенко І. Д. Обґрунтування та визначення вимог до засобів криптографічного захисту інформації. Сборник трудов второй международной студенческой научно-технической конференции «Информатика и компьютерные технологии 2006» 13 декабря 2006 года. ДонНТУ. Донецк 2006 р.

112. Г. В. Кузнецов, В. В. Фомичов, С. О. Сушко, Л. Я. Фомичова. Математичні основи криптографії. Навч. посібник. Дніпропетровськ. Національний гірничий університет, 2004 - Ч. 1. - 391 с.

113. ENCYCLOPEDIA OF CRYPTOGRAPHY AND SECURITY. Editor-in-chief Henk C.A. van Tilborg. Eindhoven University of Technology. The Netherlands. © 2005 Springer Science+Business Media, Inc.

114. М. В. Черкаський, “Складність апаратно-програмних комп'ютерних засобів”, Contemporary computing in Ukraine, 2000, С. 58 – 67.

115. М. В. Черкаський, А. О. Саченко, "ПАРАМЕТРИЧНІ МОДЕЛІ АЛГОРИТМІВ", Електротехнічні та комп’ютерні системи №04(80), 2011. С. 162-167.

116. Черкаський М.В. SH-модель алгоритму // Вісник Національного університету “Львівська політехніка” № 433. Видавництво Національного університету «Львівська політехніка». 2001. С.127 – 134.

117. Черкаський М.В., Хусейн Халід Мурад. Універсальна SH-модель // Вісник Національного університету “Львівська політехніка” № 523 «Комп’ютерні сис-

теми та мережі». Львів. Видавництво Національного університету «Львівська політехніка». 2004. С.150 – 154 .

118. V. Grozov, A. Guirik, M. Budko and M. Budko, "Cryptographic Strength Study of the Pseudorandom Sequences Generator Based on the Blender Algorithm," 2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Valencia, Spain, 2022, pp. 191-195, doi: 10.1109/ICUMT57764.2022.9943350.

119. Z. Cai, Y. Si, J. Zhang and J. Wang, "Design and Implementation of Image Copyright Protection System Based on Chinese Cryptographic Algorithms," 2022 7th International Conference on Signal and Image Processing (ICSIP), Suzhou, China, 2022, pp. 395-399, doi: 10.1109/ICSIP55141.2022.9886877.

120. О. В. Потій, К. В. Ісірова. Аналіз вимог та моделей безпеки для постквантової криптографії // Математичне та комп'ютерне моделювання. Серія: Технічні науки. Випуск 15. 2017. С. 192 – 197.

121. M. -J. O. Saarinen, "WiP: Applicability of ISO Standard Side-Channel Leakage Tests to NIST Post-Quantum Cryptography," *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 2022, pp. 69-72, doi: 10.1109/HOST54066.2022.9839849.

122. U. Banerjee, S. Das and A. P. Chandrakasan, "Accelerating Post-Quantum Cryptography using an Energy-Efficient TLS Crypto-Processor," *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Seville, Spain, 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9180550.

123. P. He, U. Guin and J. Xie, "Novel Low-Complexity Polynomial Multiplication Over Hybrid Fields for Efficient Implementation of Binary Ring-LWE Post-Quantum Cryptography," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 383-394, June 2021, doi: 10.1109/JETCAS.2021.3075456.

124. D. Muraleedharan and S. K. Daniel, "An efficient IP core of Consultative Committee for Space Data Systems (CCSDS) Recommended Authenticated Cryptographic Algorithm," *2020 8th International Symposium on Digital Forensics*

and Security (ISDFS), Beirut, Lebanon, 2020, pp. 1-6, doi: 10.1109/ISDFS49300.2020.9116306.

125. Valerii Hlukhov. Implementing Quantum Fourier Transform in a Digital Quantum Coprocessor. Advances in Cyber-Physical Systems. Volume 4. Number 1. Lviv Polytechnic National University. 2019. pp. 6 - 13.

126. V. Hlukhov, "Digital Qubits for FPGA-based Homogenous Quantum Coprocessor," *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Cracow, Poland, 2021, pp. 318-322, doi: 10.1109/IDAACS53288.2021.9660970.

127. V. Hlukhov, "FPGA Based Digital Quantum Computer Verification," *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Kyiv, Ukraine, 2020, pp. 178-182, doi: 10.1109/DESSERT50317.2020.9125077.

128. Valerii Hlukhov, Bohdan Havano. Principles of Digital Quantum Coprocessor Based on a FPGA, which Operates under the Control of a Classical Computer. Advanced Computer Information Technologies Acit 2019. June 5 - 7, 2019. International Conference. Ceske Budejovice, Czech Republic. Conference Proceedings, pp. 191 – 194.

129. Nykolaychuk, Y.M., Yakymenko, I.Z., Vozna, N.Y., & Kasianchuk, M.M. (2022). Residue Number System Asymmetric Cryptoalgorithms. Cybernetics and Systems Analysis, 58(4), 611–618.

130. M. A. Mehrabi, C. Doche and A. Jolfaei, "Elliptic Curve Cryptography Point Multiplication Core for Hardware Security Module," in *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1707-1718, 1 Nov. 2020, doi: 10.1109/TC.2020.3013266.

131. S. Mondal and S. Patkar, "Hardware-software hybrid implementation of non-deterministic ECC over Curve-25519 for resource constrained devices," *2021 Asian Conference on Innovation in Technology (ASIANCON)*, PUNE, India, 2021, pp. 1-8, doi: 10.1109/ASIANCON51346.2021.9544627.

132. AMERICAN NATIONAL STANDARD X9.62-1998. Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm.

133. BSI TR-02102-1. Technical Guideline – Cryptographic Algorithms and Key Lengths. Version: 2023-01. Federal Office for Information Security 2023.

134. S. Chhabra and K. Lata, "Key-based Obfuscation using HT-like Trigger Circuit for 128-bit AES Hardware IP Core," *2021 IEEE 34th International System-on-Chip Conference (SOCC)*, Las Vegas, NV, USA, 2021, pp. 164-169, doi: 10.1109/SOCC52499.2021.9739619.

135. ISO/IEC 14888-3:1998. Information technology -- Security techniques -- Digital signatures with appendix -- Part 3: Certificate-based mechanisms.

136. KippsDeSanto A&D and Government Services 2018 M&A Survey. https://kippsdesanto.com › uploads › 2018/07

137. M. Yoshida, S. Morioka and S. Obana, "On Cryptographic Algorithms and Key Length for Delayed Disclosure Authentication of GNSS," *2022 International Conference on Localization and GNSS (ICL-GNSS)*, Tampere, Finland, 2022, pp. 1-6, doi: 10.1109/ICL-GNSS54081.2022.9797034.

138. Глухов В.С., Еліас Р.М., Рахма М.К.Р. Часова складність орієнтованих на виконання криптографічних перетворень в складі кіберфізичних систем помножувачів на основі модифікованих комірок Гілда / Матеріали другого наукового семінару Кібер-фізичні системи: досягнення та виклики, Львів, Національний університет «Львівська політехніка», 21-22 червня 2016 р. С. 36-42

139. Р. Еліас, М. Рахма, В.С. Глухов. Часова складність помножувачів для полів Галуа. Програма 2-ої Міжнародної науково-технічної конференції «Електротехнічні і комп'ютерні системи: теорія і практика (Елтекс 2016)» м. Одесса, 26–28 червня 2016 р.

140. Р. Еліас, М. Рахма, В.С. Глухов. Часова складність помножувачів для полів Галуа / Електротехнічні та комп’ютерні системи. – Одеса: – 2016. Вид-во Наука і техніка. – № 22 (98). – С. 323-327

141. Rahma, Mohammed Kadhim. Time complexity of multipliers for Galois fields / Mohammed Kadhim Rahma, Valeriy S.Hlukhov / INTERNATIONAL YOUTH

SCIENCE FORUM ”LITTERIS ET ARTIBUS”, 24-26 NOVEMBER 2016, LVIV, UKRAINE. Proceedings, pp. 52-53.

142. В. С. Глухов, Р. Еліас, М. Рахма. Аналіз можливості побудови багатосекційних помножувачів елементів полів Галуа для нормального та поліноміального базисів / Матеріали третього наукового семінару Кіберфізичні системи: досягнення та виклики, Львів, Національний університет «Львівська політехніка», 13-14 червня 2017 р. С. 38-47.

143. Р. Еліас, М. Рахма, В. Глухов. Зменшення структурної складності багатосекційних помножувачів елементів полів Галуа. Міжнародна науковотехнічна конференція «Електротехнічні і комп'ютерні системи: теорія і практика (ЕЛТЕКС-2017)». Одеса, Одеський національний політехнічний університет. 26 – 28 червня 2017 р.

144. Рахма, М. Структурна складність помножувачів елементів полів Галуа у нормальному та поліноміальному базисах / Р. Еліас, М. Рахма, В. Глухов / Електротехнічні та комп’ютерні системи. – Одеса: – 2017. Вид-во Наука і техніка. - № 25 (101). – С. 332-340.

145. J. Aberg, Y. M. Shtarkov and B. J. M. Smeets, "Estimation of escape probabilities for PPM based on universal source coding theory," Proceedings of IEEE International Symposium on Information Theory, 1997, pp. 65-, doi: 10.1109/ISIT.1997.612980.

146. S. Viorica and A. Victor, "Multidimensional Digital Signal Processing for Printed Circuit Boards Testing," 2020 International Conference on Development and Application Systems (DAS), 2020, pp. 60-63, doi: 10.1109/DAS49615.2020.9108971.

147. S. Yu and Q. Huang, "Hard Reliability-Based Ordered Statistic Decoding and Its Application to McEliece Public Key Cryptosystem," in *IEEE Communications Letters*, vol. 26, no. 3, pp. 490-494, March 2022, doi: 10.1109/LCOMM.2021.3137529.

148. K. Pavani and P. Sriramya, "Enhancing Public Key Cryptography using RSA, RSA-CRT and N-Prime RSA with Multiple Keys," *2021 Third International*

Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), Tirunelveli, India, 2021, pp. 1-6, doi: 10.1109/ICICV50876.2021.9388621.

149. J. Xie, C. -Y. Lee, P. K. Meher and Z. -H. Mao, "Novel Bit-Parallel and Digit-Serial Systolic Finite Field Multipliers Over $GF(2^m)$ Based on Reordered Normal Basis," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 9, pp. 2119-2130, Sept. 2019, doi: 10.1109/TVLSI.2019.2918836.

150. A. Reyhani-Masoleh, H. El-Razouk and A. Monfared, "New Multiplicative Inverse Architectures Using Gaussian Normal Basis," in *IEEE Transactions on Computers*, vol. 68, no. 7, pp. 991-1006, 1 July 2019, doi: 10.1109/TC.2018.2859941.

151. M. Kalaiarasi, V. R. Venkatasubramani and S. Rajaram, "A Parallel Quad Itoh-Tsujii Multiplicative Inversion Algorithm for FPGA Platforms," *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*, Guwahati, India, 2020, pp. 31-35, doi: 10.1109/ISEA-ISAP49340.2020.234996.

152. Y. Xie et al., "A Dual-Core High-Performance Processor for Elliptic Curve Cryptography in GF(p) Over Generic Weierstrass Curves," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 11, pp. 4523-4527, Nov. 2022, doi: 10.1109/TCSII.2022.3194019.

153. H. K. Balupala, K. Rahul and S. Yachareni, "Galois Field Arithmetic Operations using Xilinx FPGAs in Cryptography," 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), 2021, pp. 1-6, doi: 10.1109/IEMTRONICS52119.2021.9422551.

154. H. Inumarty and M. A. Basiri M., "Reconfigurable Hardware Design for Polynomial Galois Field Arithmetic Operations," 2020 24th International Symposium on VLSI Design and Test (VDAT), 2020, pp. 1-5, doi: 10.1109/VDAT50263.2020.9190485.

155. K. K. Gupt, M. Kshirsagar, J. P. Sullivan and C. Ryan, "Automatic Test Case Generation for Vulnerability Analysis of Galois Field Arithmetic Circuits," 2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP), 2021, pp. 32-37, doi: 10.1109/CSP51677.2021.9357567

156. C. -Y. Lee and J. Xie, "Efficient Scalable Three Operand Multiplier Over GF(2^m) Based on Novel Decomposition Strategy," 2019 IEEE 37th International Conference on Computer Design (ICCD), 2019, pp. 29-37, doi: 10.1109/ICCD46524.2019.00013.

157. J. L. Imaña, "LFSR-Based Bit-Serial $GF(2^m)$ G F (2 m) Multipliers Using Irreducible Trinomials," in IEEE Transactions on Computers, vol. 70, no. 1, pp. 156-162, 1 Jan. 2021, doi: 10.1109/TC.2020.2980259.

158. C. Fei, F. Zhou, N. Wu, F. Ge, J. Wen and P. Qin, "A Scalable Bit-Parallel Word-Serial Multiplier with Fault Detection on GF(2^m)," 2020 IEEE 20th International Conference on Communication Technology (ICCT), 2020, pp. 1660-1664, doi: 10.1109/ICCT50939.2020.9295940.

159. I. Yavuz, S. B. Ö. Yalçin and Ç. K. Koç, "FPGA Implementation of an Elliptic Curve Cryptosystem over GF(3^m)," 2008 International Conference on Reconfigurable Computing and FPGAs, 2008, pp. 397-402, doi: 10.1109/ReConFig.2008.66.

160. H. Zeng, W. Li, T. Chen and L. Nan, "An Efficient Module Arithmetic Logic Unit in Dual Field for Internet of Things Applications," *2021 IEEE 14th International Conference on ASIC (ASICON)*, Kunming, China, 2021, pp. 1-4, doi: 10.1109/ASICON52560.2021.9620531.

161. C. Qin, C. Jiang, Q. Mo, H. Yao and C. -C. Chang, "Reversible Data Hiding in Encrypted Image via Secret Sharing Based on GF(p) and GF(2^8)," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 32, no. 4, pp. 1928-1941, April 2022, doi: 10.1109/TCSVT.2021.3091319.

162. V. Trujillo-Olaya and J. Velasco-Medina, "Half-Matrix Normal Basis Multiplier Over GF(p^{m})," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 64, no. 4, pp. 879-891, April 2017, doi: 10.1109/TCSI.2016.2626375.

163. B. Alhazmi and F. Gebali, "Fast Large Integer Modular Addition in GF(p) Using Novel Attribute-Based Representation," in IEEE Access, vol. 7, pp. 58704-58719, 2019, doi: 10.1109/ACCESS.2019.2914641.

164. Y. Liu, L. Wang, H. Wu and S. Liu, "Performance of Lossy P-LDPC Codes over GF(2)," 2020 14th International Conference on Signal Processing and Communication Systems (ICSPCS), 2020, pp. 1-5, doi: 10.1109/ICSPCS50536.2020.9310025.

165. J. Zhang, B. Bai, D. Deng, M. Zhu, H. Xu and M. Guan, "Non-Uniform Spatially-Coupled LDPC Codes over GF(2m)," 2018 IEEE International Symposium on Information Theory (ISIT), 2018, pp. 816-820, doi: 10.1109/ISIT.2018.8437900.

166. D. Basu Roy and D. Mukhopadhyay, "High-Speed Implementation of ECC Scalar Multiplication in GF(p) for Generic Montgomery Curves," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 7, pp. 1587-1600, July 2019, doi: 10.1109/TVLSI.2019.2905899.

167. R. Wu *et al*., "Efficient High-Radix GF(p) Montgomery Modular Multiplication Via Deep Use Of Multipliers," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 12, pp. 5099-5103, Dec. 2022, doi: 10.1109/TCSII.2022.3197314.

168. S. R. Pillutla, B. Sandeep Vankeswaram, T. Velaga, S. S. Lellapalli and V. Abhinav Vadlamudi, "Implementation of Low Complexity Finite Field GF (2m) Multiplier Using Irreducible Primitive Polynomials," 2022 Second International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), 2022, pp. 1-6, doi: 10.1109/ICAECT54875.2022.9807862.

169. Y. G. Desale and V.V. Ingale, "Design of Power Efficient Bit Serial Finite Field GF(2m) Multiplier," 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), 2019, pp. 1-4, doi: 10.1109/I2CT45611.2019.9033682.

170. J. Xie, C. -Y. Lee, P. K. Meher and Z. -H. Mao, "Novel Bit-Parallel and Digit-Serial Systolic Finite Field Multipliers Over $GF(2^m)$ Based on Reordered Normal Basis," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 9, pp. 2119-2130, Sept. 2019, doi: 10.1109/TVLSI.2019.2918836.

171. H. El-Razouk, K. Kotha and M. Puligunta, "Novel $GF\left(2^{m}\right)$GF2m Digit-Serial PISO Multipliers for the Self-Dual

Gaussian Normal Bases," in IEEE Transactions on Computers, vol. 70, no. 10, pp. 1732-1746, 1 Oct. 2021, doi: 10.1109/TC.2020.3023131.

172. T. J. Grale and E. E. Swartzlander, "Parallel GF(2n) Modular Squarers," 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), 2019, pp. 872-875, doi: 10.1109/MWSCAS.2019.8884794.

173. X. Wang, N. Wu, F. Zhou and F. Ge, "Efficient Configurable Digit-Serial Multiplier Based on Improved Karatsuba Algorithm over GF(2m)," 2022 IEEE 22nd International Conference on Communication Technology (ICCT), Nanjing, China, 2022, pp. 1531-1535, doi: 10.1109/ICCT56141.2022.10072558

174. A. Ibrahim, "Unified and Scalable Digit-Serial Systolic Array for Multiplication and Division Over GF (2m)," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1546-1549, July 2020, doi: 10.1109/TCAD.2019.2917843.

175. S. Mohaghegh, G. Yemişçioglu and A. Muhtaroğlu, "Low-Power and Area-Efficient Finite Field Multiplier Architecture Based on Irreducible All-One Polynomials," *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Seville, Spain, 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9181179.

176. S. R. Pillutla, B. Sandeep Vankeswaram, T. Velaga, S. S. Lellapalli and V. Abhinav Vadlamudi, "Implementation of Low Complexity Finite Field GF (2m) Multiplier Using Irreducible Primitive Polynomials," *2022 Second International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*, Bhilai, India, 2022, pp. 1-6, doi: 10.1109/ICAECT54875.2022.9807862.

177. C. Fei, F. Zhou, N. Wu, F. Ge, J. Wen and P. Qin, "A Scalable Bit-Parallel Word-Serial Multiplier with Fault Detection on GF(2^m)," *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, Nanning, China, 2020, pp. 1660-1664, doi: 10.1109/ICCT50939.2020.9295940.

178. J. L. Imaña, "Low-Delay FPGA-Based Implementation of Finite Field Multipliers," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 8, pp. 2952-2956, Aug. 2021, doi: 10.1109/TCSII.2021.3071188.

179. S. Kishimoto, D. Wu and S. Ohnuki, "Analysis of Time-Varying Fields Using Fast Inverse Laplace Transform," *2021 International Applied Computational Electromagnetics Society Symposium (ACES)*, Hamilton, ON, Canada, 2021, pp. 1-2.

180. J. Xie, C. -Y. Lee, P. K. Meher and Z. -H. Mao, "Novel Bit-Parallel and Digit-Serial Systolic Finite Field Multipliers Over GF(2m) Based on Reordered Normal Basis," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 9, pp. 2119-2130, Sept. 2019, doi: 10.1109/TVLSI.2019.2918836.

181. S. Deshpande, S. M. d. Pozo, V. Mateu, M. Manzano, N. Aaraj and J. Szefer, "Modular Inverse for Integers using Fast Constant Time GCD Algorithm and its Applications," *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, Dresden, Germany, 2021, pp. 122-129, doi: 10.1109/FPL53798.2021.00028.

182. Rahma, M. Computing Square Roots and Solve Equations of ECC over Galois Fields /M. Rahma, V. Hlukhov / International Youth Science Forum ”Litteris Et Artibus”, November 23-25, 2017, Lviv, Ukraine, pp. 437-440.

183. Ren, Bo, Shibin Ma, Biao Hou, Danfeng Hong, Jocelyn Chanussot, Jianlong Wang, and Licheng Jiao. "A dual-stream high resolution network: Deep fusion of GF-2 and GF-3 data for land cover classification." International Journal of Applied Earth Observation and Geoinformation 112 (2022): 102896.

184. Y. G. Desale and V.V. Ingale, "Design of Power Efficient Bit Serial Finite Field GF(2m) Multiplier," 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Bombay, India, 2019, pp. 1-4, doi: 10.1109/I2CT45611.2019.9033682.

185. R. Wu et al., "Efficient High-Radix GF(p) Montgomery Modular Multiplication Via Deep Use Of Multipliers," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 12, pp. 5099-5103, Dec. 2022, doi: 10.1109/TCSII.2022.3197314.

186. H. El-Razouk, "Input-Latency Free Versatile Bit-Serial GF(2m) Polynomial Basis Multiplication," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 5, pp. 589-602, May 2022, doi: 10.1109/TVLSI.2022.3155611.

187. H. El-Razouk, K. Kotha and M. Puligunta, "Novel $GF\left(2^{m}\right)$GF2m Digit-Serial PISO Multipliers for the Self-Dual Gaussian Normal Bases," in IEEE Transactions on Computers, vol. 70, no. 10, pp. 1732-1746, 1 Oct. 2021, doi: 10.1109/TC.2020.3023131.

188. X. Wang, N. Wu, F. Zhou and F. Ge, "Efficient Configurable Digit-Serial Multiplier Based on Improved Karatsuba Algorithm over GF(2m)," 2022 IEEE 22nd International Conference on Communication Technology (ICCT), Nanjing, China, 2022, pp. 1531-1535, doi: 10.1109/ICCT56141.2022.10072558.

189. E. Morancho, "A Vector Implementation of Gaussian Elimination over GF(2): Exploring the Design-Space of Strassen's Algorithm as a Case Study," 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Turku, Finland, 2015, pp. 111-118, doi: 10.1109/PDP.2015.24.

190. S. Yang, W. Yuechuan and Z. Mingshu, "Research and Design of Dedicated Instruction for Reconfigurable Matrix Multiplication of VLIW Processor," 2016 International Conference on Intelligent Networking and Collaborative Systems (INCoS), Ostrava, Czech Republic, 2016, pp. 324-327, doi: 10.1109/INCoS.2016.18.

191. S. Chen, L. Huang, J. Liu, C. Liu and Y. Fan, "An Error-Surface-Based Fractional Motion Estimation Algorithm and Hardware Implementation for VVC," 2023 IEEE International Symposium on Circuits and Systems (ISCAS), Monterey, CA, USA, 2023, pp. 1-5, doi: 10.1109/ISCAS46773.2023.10182170.

192. M. Gurtner, P. Zips, A. Trachte and A. Kugi, "Application of Monte Carlo Method for Probabilistic Robust Control Performance Assessment," 2022 6th International Conference on System Reliability and Safety (ICSRS), Venice, Italy, 2022, pp. 163-168, doi: 10.1109/ICSRS56243.2022.10067384.

193. S. E. Mironov, O. I. Bureneva and A. D. Milakin, "Analysis of Multiplier Architectures for Neural Networks Hardware Implementation," *2022 III International Conference on Neural Networks and Neurotechnologies (NeuroNT)*, Saint Petersburg, Russian Federation, 2022, pp. 32-35, doi: 10.1109/NeuroNT55429.2022.9805564.

194. O. V. Thanh, N. Gillis and F. Lecron, "Bounded Simplex-Structured Matrix Factorization," *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Singapore, Singapore, 2022, pp. 9062-9066, doi: 10.1109/ICASSP43922.2022.9747124.

195. M. Bednara, Michael Daldrup, Joachim von zur Gathen, J¨urgenTeich, Jamshid Shokrollahi, “Reconfigurable Implementation of Elliptic Curve Crypto Algorithms,” in 9th Reconfigurable Architecture Workshop (RAW 2002), 2002.

196. Мельник А. О. Хамелеон - система високорівневого синтезу спеціалізованих процесорів / А. О. Мельник, А. М. Сало, В. Клименко, Л. Цигилик, А. Юрчук // Радіоелектронні і комп’ютерні системи. - 2009. - № 5. - С. 189–194. - Режим доступу: http://nbuv.gov.ua/UJRN/recs_2009_5_37.

197. Мельник В. А. Дослідження характеристик засобів генерування програмних моделей спеціалізованизпроцесорів у самоконфігурованій комп’ютерній системі / Мельник В. А. // Вісник Національного університету «Львівська політехніка» “Комп’ютерні науки та інформаційні технології”. – Львів, - 2012. – Вип. 744. - С. 86 – 93.

198. M. M. Ayub, H. Ahmadzay, J. Eckmüller and F. Kreupl, "Electronic System Level Power and Performance Analysis for Multi-Processor-System-on-Chip," *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, 2019, pp. 1-2, doi: 10.1109/ReConFig48160.2019.8994783.

199. S. K. Selvan, "Verilog-HDL Based Implementation of Sum of Absolute Difference Architecture using Adder Compressors," *2021 IEEE 2nd International Conference on Applied Electromagnetics, Signal Processing, & Communication (AESPC)*, Bhubaneswar, India, 2021, pp. 1-5, doi: 10.1109/AESPC52704.2021.9708520.

200. O. B. Tariq *et al.*, "High-Level Annotation of Routing Congestion for Xilinx Vivado HLS Designs," in *IEEE Access*, vol. 9, pp. 54286-54297, 2021, doi: 10.1109/ACCESS.2021.3067453.

201. J. Hu et al., "Formal Verification of GCSE in the Scheduling of High-level Synthesis: Work-in-Progress," 2020 International Conference on Hardware/Software

Codesign and System Synthesis (CODES+ISSS), 2020, pp. 1-2, doi: 10.1109/CODESISSS51650.2020.9244039.

202. C. Wu, J. Xie and K. Wang, "Highly Efficient Modulo Loop Pipeline For High Level Synthesis," 2021 IEEE 14th International Conference on ASIC (ASICON), 2021, pp. 1-4, doi: 10.1109/ASICON52560.2021.9620276.

203. R. Nozaki, H. Nishikawa, I. Taniguchi and H. Tomiyama, "Function-Level Module Sharing in High-Level Synthesis," 2019 International SoC Design Conference (ISOCC), 2019, pp. 50-51, doi: 10.1109/ISOCC47750.2019.9078522.

204. Rahma, Mohammed Kadhim. Time complexity of multipliers for Galois fields / Mohammed Kadhim Rahma, Valeriy S.Hlukhov / INTERNATIONAL YOUTH SCIENCE FORUM ”LITTERIS ET ARTIBUS”, 24-26 NOVEMBER 2016, LVIV, UKRAINE. Proceedings, pp. 52-53.

205. ДСТУ ISO/IEC 13888-1:2015. Інформаційні технології; Методи захисту. Неспростовність. Частина 1. Загальні положення (ISO/IEC 13888-1:2009, IDT).

206. ДСТУ ISO/IEC 15946-1:2015 Інформаційні технології. Методи захисту. Криптографічні методи, що грунтуються на еліптичних кривих. Частина 1. Загальні положення.

207. Nikishova, Arina, Ekaterina Vitenburg, Mikhail Umnitsyn, and Tatiana Omelchenko. "Cryptographic Protection of Data Transmission Channel." In Creativity in Intelligent Technologies and Data Science: Third Conference, CIT&DS

208. G. D'Andrea and L. Pomante, "Design for ReConfigurability: An Electronic System Level Methodology to Exploit Reconfigurable Platforms," *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, 2020, pp. 355-356, doi: 10.1109/FPL50879.2020.00066.

209. Глухов В.С. Система команд криптографічного процесора // Вісник Національного університету “Львівська політехніка” “Комп`ютерні системи та мережі”. Вип. 523. Львів, 2004.

210. Глухов, В.С., І.М. Жолубак, А.Т. Костик, and М.К.Р. Рахма. Проектування і моделювання елементів комп’ютерних систем та мереж: З дисципліни "Дослідження і проектування комп’ютерних систем та мереж" для студентів освітньо-

кваліфікаційного рівня "Магістр", спеціальність 123 "Комп’ютерна інженерія", спеціалізації "Комп’ютерні системи та мережі", "Кіберфізичні системи" та "Системне програмування". Видавництво Національного університету "Львівська політехніка", 2019. Beran, Adam. "Edwards curves and elliptic function fields.", Department: Department of Algebra, (2023).

211. Sowjanya, K., Mou Dasgupta, and Sangram Ray. "A lightweight key management scheme for key-escrow-free ECC-based CP-ABE for IoT healthcare systems." Journal of Systems Architecture 117 (2021): 102108.

212. Sethi, Purna Chandra, Neelima Sahu, and Prafulla Kumar Behera. "Group security using ECC." International Journal of Information Technology (2022): 1-9.

213. С. Гудман, С. Хидетниеми. Введение в разработку и анализ алгоритмов. Издательство «Мир». Москва. 1981.

214. Домарев В. В. Безопасность информационных технологий. Системный подход. Издательство ТИД “ДС”, 2004 г., 992 стр. ISBN 966-7992-36-5

215. Глухов В.С., Еліас Р. Вбудований контроль спецпроцесорів для обробленя цифрових підписів // Вісник Національного університету "Львівська політехніка". Комп’ютерні науки та інформаційні технології. 2009. № 686. С. 64-69.

216. Sergi Granell Escalfet, ACCELERATING HALIDE ON AN FPGA: Accelerating Halide on an FPGA by using CIRCT and Calyx as an intermediate step to go from a high-level and software-centric IRs down to RTL, 2023, 155p.

217. T. Soni, A. Kumar and M. K. Panda, "Modified Efficient Parallel Distributed Arithmetic based FIR Filter Architecture for ASIC and FPGA," 2023 10th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 2023, pp. 860-865, doi: 10.1109/SPIN57001.2023.10116765.

218. **Тестування і діагностика програмно-апаратних засобів:** методичні вказівки до лабораторних робіт № 1, № 2, № 3 для студентів першого (бакалаврського) рівня вищої освіти спеціальності 123 “Комп’ютерна інженерія” / Укл.: В. С. Глухов, І. М. Жолубак – Львів: Видавництво Національного університету “Львівська політехніка”, 2019. – 21 с.

219. D. -G. Lim, S. -M. Yu, J. -h. Myung and J. Song, "Fabricating Test Module of Parts For EV With FPGA," 2021 36th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Jeju, Korea (South), 2021, pp. 1-3, doi: 10.1109/ITC-CSCC52171.2021.9501452.

220. P. Schulz and L. Y. Ungar, "FPGA-Based Digital Twin Approach for Design and Test," 2023 IEEE AUTOTESTCON, National Harbor, MD, USA, 2023, pp. 1-5, doi: 10.1109/AUTOTESTCON47464.2023.10296392.

221. Borodzhieva A. Project-Based Learning for Teaching “Diffie-Hellman Algorithm, Elgamal Variant”. In2022 29th International Conference on Systems, Signals and Image Processing (IWSSIP) 2022 Jun 1 (pp. 1-5). IEEE.

222. Pang, Xiao-Ling, Ai-Lin Yang, Chao-Ni Zhang, Jian-Peng Dou, Hang Li, Jun Gao, and Xian-Min Jin. "Hacking quantum key distribution via injection locking." Physical Review Applied 13, no. 3 (2020): 034008.

223. Kaihua Qin, Liyi Zhou, Benjamin Livshits, Arthur Gervais, “Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit”, 2020, doi: https://doi.org/10.48550/arXiv.2003.03810

224. E. Yoshiya, T. Nakanishi and T. Isshiki, "RTL Design Framework for Embedded Processor by using C++ Description," 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021, pp. 1208-1211, doi: 10.23919/DATE51398.2021.9473942.

225. Chéour, Rym, Sabrine Khriji, and Olfa Kanoun. "Microcontrollers for IoT: optimizations, computing paradigms, and future directions." 2020 IEEE 6th World Forum on Internet of Things (WF-IoT). IEEE, 2020.

226. Yi Qian; Feng Ye; Hsiao-Hwa Chen, "Mathematical Background," in Security in Wireless Communication Networks , IEEE, 2022, pp.27-49, doi: 10.1002/9781119244400.ch3.

227. ДСТУ ISO-IEC 10118-1-2003. Інформаційні технології; Методи захисту. Геш-функції / А. Анісімов (пер.і наук.-техн.ред.). - Офіц. вид - К. : Держспожи-встандарт України, 2004.

228. Звіт про науково-дослідну роботу ДБ/КІБЕР (№ держреєстрації 0115U000446) «Інтеграція методів і засобів вимірювання, автоматизації, опрацювання та захисту інформації в базисі кіберфізичних систем», розділ 9 «Захищений обмін, опрацювання та зберігання вимірювальної та службової інформації».

229. Энциклопедия кибернетики. Главная редакция украинской советской 30 148 энциклопедии. Киев – 1975.

230. H. Inumarty and M. A. Basiri M., "Reconfigurable Hardware Design for Polynomial Galois Field Arithmetic Operations," *2020 24th International Symposium on VLSI Design and Test (VDAT)*, Bhubaneswar, India, 2020, pp. 1-5, doi: 10.1109/VDAT50263.2020.9190485.

231. R. K. K. Ajeena, "The Graph and its Role for Speeding up the Elliptic Scalar Multiplication Algorithms," *2019 2nd International Conference on Engineering Technology and its Applications (IICETA)*, Al-Najef, Iraq, 2019, pp. 227-228, doi: 10.1109/IICETA47481.2019.9012974.

232. Koblitz, N. A riddle wrapped in an enigma [Electronic resource] / N. Koblitz, A. J. Menezes // ePrint Archive. – 2016. – P. 1–21. – Available at: *http://eprint.iacr.org/2015/1018.pdf*

233. Z. Zhang, S. Gu, S. Li, Y. Yang and Q. Zhang, "Multi-hop Coflow Routing for LEO Distributed Computation Satellite Networks," 2022 IEEE 96th Vehicular Technology Conference (VTC2022-Fall), London, United Kingdom, 2022, pp. 1-5, doi: 10.1109/VTC2022-Fall57202.2022.10012814.

234. Maurer, Peter M. Primitive Polynomials for the Field GF(3). Dept. of Computer Science, Baylor University, Waco, Texas 76798. https://baylorir.tdl.org/handle/2104/8793 29.08.2019

235. C. Fu, P. Li, Z. Bai and Y. Wang, "Helical Long Period Fiber Grating Inscribed in Elliptical Core Polarization-Maintaining Fiber," in *IEEE Access*, vol. 9, pp. 59378-59382, 2021, doi: 10.1109/ACCESS.2021.3073735.

236. J. Xie, C. -Y. Lee and P. K. Meher, "Low-Complexity Systolic Multiplier for GF(2m) using Toeplitz Matrix-Vector Product Method," *2019 IEEE International*

Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 2019, pp. 1-5, doi: 10.1109/ISCAS.2019.8702298.

237. M. Rathor and A. Sengupta, "Signature Biometric based Authentication of IP Cores for Secure Electronic Systems," *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*, Jaipur, India, 2021, pp. 384-388, doi: 10.1109/iSES52644.2021.00094.

238. Signature-based Approach to Detecting Malicious Outgoing Traffic,Petliak, N., Klots, Y., Titova, V., Cheshun, V., Boyarchuk, A.,CEUR Workshop Proceedings, 2023, 3373, pp. 486–506.

239. Андрощук, О.С., Кльоц, Ю.П., Орленко, В.С., Чешун, В.М., Коротун, Т.М. (2021). Функціональна реалізація генератора криптоключів з джерелами ентропії для мобільного банкінгу [Functional Realization of Cryptographic Keys Generator with Entropy Sources for Mobile Banking]. Хмельницький національний університет, Заклад вищої освіти “Міжнародний науково-технічний університет імені академіка Юрія Бугая”. №1, 2021 (293), Сторінки: 7-11. DOI: *https://www.doi.org/10.31891/2307-5732-2021-293-1-7-11*. Рецензія: 07.01.2021 р. Надрукована: 10.03.2021 р.

240. Yakymenko, I., Kasianchuk, M., Shylinska, I., Yatskiv, V., & Karpinski, M. (2022). Polynomial Rabin Cryptosystem Based on the Operation of Addition. Proceedings - International Conference on Advanced Computer Information Technologies, ACIT, pp. 345–350.

241. S. Chhabra, V. Dhanwani, V. K. Dhaka and K. Lata, "Design and Analysis of Secure One-way Functions for the Protection of Symmetric Key Cryptosystems," *2020 24th International Symposium on VLSI Design and Test (VDAT)*, Bhubaneswar, India, 2020, pp. 1-6, doi: 10.1109/VDAT50263.2020.9190432.

242. A. Alharam, Y. Alqassab, R. Senan, M. Almalki and W. Elmedany, "Reconfigurable Cyber-Security Architecture for Small Satellite with Low Complexity and Power," *2022 International Conference on Innovation and*

Intelligence for Informatics, Computing, and Technologies (3ICT), Sakheer, Bahrain, 2022, pp. 245-249, doi: 10.1109/3ICT56508.2022.9990704.

243. Y. Song, X. Hu, W. Wang, J. Tian and Z. Wang, "High-Speed and Scalable FPGA Implementation of the Key Generation for the Leighton-Micali Signature Protocol," *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, Daegu, Korea, 2021, pp. 1-5, doi: 10.1109/ISCAS51556.2021.9401177.

244. A. Sideris, T. Sanida and M. Dasygenis, "High Throughput Pipelined Implementation of the SHA-3 Cryptoprocessor," *2020 32nd International Conference on Microelectronics (ICM)*, Aqaba, Jordan, 2020, pp. 1-4, doi: 10.1109/ICM50269.2020.9331803.

245. R. Choudhury, S. R. Ahamed and P. Guha, "Hardware Implementation of Low Complexity High-speed Perceptron Block," *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, Austin, TX, USA, 2022, pp. 26-30, doi: 10.1109/ISCAS48785.2022.9937758.

246. I. I. Livshitz, P. A. Lontsikh, N. P. Lontsikh and G. Alexey E, "Methodology for Ensuring EDI Security," *2022 International Conference on Quality Management, Transport and Information Security, Information Technologies (IT&QM&IS)*, Saint Petersburg, Russian Federation, 2022, pp. 170-172, doi: 10.1109/ITQMIS56172.2022.9976565.

247. Y. Tang, T. Townsend, H. Deng, Y. Liu, R. Zhang and J. Chen, "A Highly Linear FPGA-Based TDC and a Low-Power Multichannel Readout ASIC With a Shared SAR ADC for SiPM Detectors," in *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, pp. 2286-2293, Aug. 2021, doi: 10.1109/TNS.2021.3096162. http://www.onsemi.ru.com/PowerSolutions/content.do?id=16788

248. Review Acer Aspire 4830TG Notebook. https://ark.intel.com/content/www/us/en/ark/ products/65520/intel-core-i53570k-processor-6m-cache-up-to-3-80-ghz.html

249. M. Chouhan, A. S. Raghuvanshi and D. Muchahary, "FPGA Implementation of High Performance and Energy Efficient Radix-4 based FFT," *2022 2nd Asian*

Conference on Innovation in Technology (ASIANCON), Ravet, India, 2022, pp. 1-5, doi: 10.1109/ASIANCON55314.2022.9908613.

250. UG116 (v10.8.1). Device Reliability Report. June 29, 2022

251. Xilinx Intellectual Property (IP) cores address requirements for DSP, Embedded, and Connectivity designs. http://www.xilinx.com/products/intellectualproperty/index.htm

252. http://ru.wikipedia.org/wiki/IP-cores. 01:33, 19 травня 2019.

253. A. Bakhtiyor, A. Orif, B. Ilkhom and K. Zarif, "Differential Collisions in SHA-1," *2020 International Conference on Information Science and Communications Technologies (ICISCT)*, Tashkent, Uzbekistan, 2020, pp. 1-5, doi: 10.1109/ICISCT50599.2020.9351441.

254. V. D. Phan, H. L. Pham, T. H. Tran and Y. Nakashima, "High Performance Multicore SHA-256 Accelerator using Fully Parallel Computation and Local Memory," *2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Tokyo, Japan, 2021, pp. 1-3, doi: 10.1109/COOLCHIPS52128.2021.9410349.

255. I. Neforawati and D. Arnaldy, "Message Digest 5 (MD-5) Decryption Application using Python-Based Dictionary Attack Technique," *2021 4th International Conference of Computer and Informatics Engineering (IC2IE)*, Depok, Indonesia, 2021, pp. 424-428, doi: 10.1109/IC2IE53219.2021.9649020.

256. T. J. Callahan, J. R. Hauser and J. Wawrzynek, "The Garp architecture and C compiler," in Computer, vol. 33, no. 4, pp. 62-69, April 2000, doi: 10.1109/2.839323.

257. Baumgarte, V., Ehlers, G., May, F. et al. PACT XPP—A Self-Reconfigurable Data Processing Architecture. The Journal of Supercomputing 26, 167–184 (2003). *https://doi.org/10.1023/A:1024499601571*

258. R. Kapila, T. Ragunathan, S. Saleti, T. J. Lakshmi and M. W. Ahmad, "Heart Disease Prediction Using Novel Quine McCluskey Binary Classifier (QMBC)," in IEEE Access, vol. 11, pp. 64324-64347, 2023, doi: 10.1109/ACCESS.2023.3289584.

ДОДАТОК А.

«ЗАТВЕРДЖУЮ»
проректор з науково-педагогічної роботи Національного університету "Львівська політехніка"
Олег ДАВИДЧАК
» 2023 р.

АКТ
про впровадження в навчальний процес результатів дисертаційної роботи на здобуття наукового ступеня кандидата технічних наук Жолубака Івана Михайловича

Цей акт складений про те, що результати дисертаційної роботи Жолубака Івана Михайловича на тему «Методи та засоби створення реконфігурованих вузлів криптографічного захисту інформації для кіберфізичних систем», представленої на здобуття наукового ступеня кандидата технічних наук, впроваджено у навчальний процес кафедри «Електронних обчислювальних машин» Національного університету "Львівська політехніка". Матеріали дисертаційного дослідження використовуються під час викладання дисципліни «Дослідження і проектування комп'ютерних систем та мереж».

Зокрема, у навчальному процесі використовуються запропоновані Жолубаком Іваном Михайловичем:

- модифіковані комірки Гілда;
- помножувачі розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда.

Згадані результати дисертаційних досліджень викладено у Методичних вказівках до лабораторних робіт "Проектування і моделювання елементів комп'ютерних систем та мереж" з дисципліни "Дослідження і проектування комп'ютерних систем та мереж" для студентів освітньо-кваліфікаційного рівня «Магістр», спеціальність 123 «Комп'ютерна інженерія», спеціалізації «Комп'ютерні системи та мережі», «Кіберфізичні системи» та «Системне програмування» / укл.: Глухов В.С., Жолубак І.М., Костик А.Т, Рахма М.К.Р. - Львів: видавництво Національного університету "Львівська політехніка", 2019, 25 с.

Завідувач кафедри електронних обчислювальних машин, д.т.н., професор — Роман ДУНЕЦЬ

Науковий керівник професор кафедри електронних обчислювальних машин, д.т.н., професор — Валерій ГЛУХОВ

ДОДАТОК Б.

«ЗАТВЕРДЖУЮ»
Проректор з наукової роботи
Національного університету
"Львівська політехніка"
_____ Іван ДЕМИДОВ
"20" квітня 2023 р.

АКТ

про використання результатів дисертаційної роботи Жолубака Івана Михайловича «Методи та засоби створення реконфігурованих вузлів криптографічного захисту інформації для кіберфізичних систем», представленої на здобуття наукового ступеня кандидата технічних наук, при виконанні науково-дослідної роботи ДБ/КІБЕР Національного університету "Львівська політехніка"

Комісія у складі: голови – начальника науково-дослідної частини (НДЧ), д.т.н., ст.досл. Романа НЕБЕСНОГО, завідувача відділу науково-організаційного супроводу наукових досліджень, к.т.н. Галини ЛАЗЬКО, завідувача кафедри електронних обчислювальних машин, д.т.н. професора Романа ДУНЦЯ, та в.о. заступника начальника планово-фінансового відділу Ірини ФАСТ цим актом підтверджують, що результати дисертаційної роботи Жолубака Івана Михайловича «Методи та засоби створення реконфігурованих вузлів криптографічного захисту інформації для кіберфізичних систем» використано при виконанні науково-дослідної роботи ДБ/КІБЕР «Інтеграція методів і засобів вимірювання, автоматизації, опрацювання та захисту інформації в базисі кібер-фізичних систем» (номер державної реєстрації 0115U000446).

У результаті досліджень, виконаних Жолубаком Іваном Михайловичем:

розроблено та наведено методику оцінювання апаратної складності помножувачів розширених полів Галуа $GF(p^n)$;

запропоновано та проаналізовано 3 структури побудови МКГ для помножувачів розширених полів Галуа $GF(p^n)$;

визначено найкращі для використання розширені поля Галуа $GF(p^n)$;

створено та апробовано технологічні засоби (генератори ядер) для проектування помножувачів для розширених полів Галуа $GF(p^n)$, що використовуються при криптографічному захисті інформації.

Голова комісії
начальник НДЧ,
д.т.н., ст.досл. _____ Роман НЕБЕСНИЙ

Члени комісії:
завідувач відділу науково-організаційного
супроводу наукових досліджень, к.т.н. _____ Галина ЛАЗЬКО
в.о. заступника начальника
планово-фінансового відділу _____ Ірина ФАСТ
зав. каф. ЕОМ,
д.т.н., професор _____ Роман ДУНЕЦЬ

ДОДАТОК В.

JETSOFTPRO
Agility. Performance. Scale.

Lviv, Ukraine
Wroclaw, Poland
New York, USA
+38 (093) 564 65 77
contactus@jetsoftpro.com

jetsoftpro.com

To whom it may concern

Re. no. 09-01/04

Date: 09/04/2024

Re: The results of the Candidate of Technical Sciences Thesis (Methods and Means of Creating Reconfigurable Nodes for Cryptographic Information Protection in Cyber-Physical Systems) of Mr. Ivan Zholubak.

The main results of the thesis (Methods and Means of Creating Reconfigurable Nodes for Cryptographic Information Protection in Cyber-Physical Systems) of Mr. Ivan Zholubak that is applied to get a Candidate of Technical Sciences degree is used within company line of cybersecurity outsourcing services and products usage within R&D security prototyping and cybersecurity projects implementations at "JETSOFTPRO UKRAINE" LLC.

Here are the results of the dissertation work that allowed us to improve the performance of our projects:

1) methods for creating multipliers of elements in extended Galois fields GF(p^n) based on modified Guild cells;

2) method for assessing the hardware complexity of multipliers of elements in extended Galois fields GF(p^n) based on modified Guild cells;

3) method for verifying the developed multipliers and multiplier generator elements in extended Galois fields GF(p^n);

4) VHDL-descriptions of multipliers for extended Galois fields GF(p^n).

Executive Manager

Serhiy Kharytonov

ДОДАТОК Г.

АКТ
про впровадження на виробництві результатів дисертаційної роботи на здобуття наукового ступеня кандидата технічних наук
Жолубака Івана Михайловича

Результати дисертаційної роботи Жолубака Івана Михайловича на тему «Методи та засоби створення реконфігурованих вузлів криптографічного захисту інформації для кіберфізичних систем», представленої на здобуття наукового ступеня кандидата технічних наук, впроваджено у компанії ТзОВ "КІБЕРЕНЕРГІЯ", а саме:

1) метод створення помножувачів елементів розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда на ПЛІС;
2) метод оцінювання апаратної складності на ПЛІС помножувачів елементів розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда;
3) метод оцінювання часової складності на ПЛІС помножувачів елементів розширених полів Галуа $GF(p^n)$ на основі модифікованих комірок Гілда;
4) метод тестування генераторів ядер помножувачів елементів розширених полів Галуа $GF(p^n)$.

Директор компанії, к.т.н Андрій САЛО

ДОДАТОК Д. ПРОГРАМНИЙ КОД ГЕНЕРАТОРА ПОМНОЖУВАЧІВ ЕЛЕМЕНТІВ РОЗШИРЕНИХ ПОЛІВ ГАЛУА *GF(p^n)* З СТРУКТУРОЮ ЧС

ДОДАТОК Д.1 Заголовочний файл класу, який створює *VHDL*-опис вузла *F*

create_f.h

```
1. #ifndef CREATE_F_H
2. #define CREATE_F_H
3. #include "BaseClass.h"
4.
5. class createF : public BaseClass
6. {
7. public:
8.      createF(int basis, std::string fileN);
9.      int creaF();
10. private:
11.     int Basis;
12.     int Count_zminna;
13.     std::string fileName;
14. };
15.
16. #endif // CREATE_F_H
```

ДОДАТОК Д.2 Файл опису класу, який створює *VHDL*-опис вузла *F*

create_f.cpp

```
1. #include <iostream>
2. #include <fstream>
3. #include <io.h>
4. #include <math.h>
5. #include <string>
6. #include "create_f.h"
7.
8. createF::createF(int basis, std::string fileN)
9. {
10.     Basis = basis;
11.     fileName = fileN;
12. }
13.
14. int createF::creaF()
15. {
16.     if (Basis > 998)
17.     {
18.             std::cout << "The basis is too large" << std::endl;
19.             return 1;
20.     }
21.     int Count_zminna = ceil((log(Basis)) / (log(2)));
22.
23.     std::ifstream *fout_F = new std::ifstream [Count_zminna];
24.     std::ifstream *fout_Fi = new std::ifstream [Count_zminna];
25.     //char mass[len];
26.     //int Count_zminna = 5;
27.     for (int i = 0; i < Count_zminna; i++)
28.     {
```

```
29.             std::string str4 = "D://Resourse_min_F" + std::to_string(i) +
".txt";
30.             std::string str5 = str4;
31.             fout_F[i].open(str5, std::ios::in | std::ios::binary |
std::ios::ate);
32.     }
33.     for (int j = 0; j < Count_zminna; j++)
34.     {
35.             std::string str4 = "D://Resourse_min_F" + std::to_string(j) +
".txt";
36.             std::string str5 = str4;
37.             fout_Fi[j].open(str5, std::ios::in | std::ios::binary);
38.     }
39.     int *filesize = new int [Count_zminna];
40.     char **A = new char *[Count_zminna];
41.     for (int i = 0; i < Count_zminna; i++)
42.     {
43.             filesize[i] = fout_F[i].tellg();
44.             A[i] = new char[filesize[i]];
45.     }
46.     for (int i = 0; i < Count_zminna; i++)
47.     {
48.             fout_F[i].close();
49.     }
50.
51.     for (int i = 0; i < Count_zminna; i++)
52.     {
53.         if (!fout_Fi[i]) return 1;
54.
55.         fout_Fi[i].getline(A[i], filesize[i] + 1);
56.     }
57.     for (int i = 0; i < Count_zminna; i++)
58.     {
59.         fout_Fi[i].close();
60.            fout_F[i].close();
61.     }
62.
63.     std::ofstream fout_f;
64.     fout_f.open("D://" + fileName + ".txt", std::ios::out |
std::ios::trunc);
65.     fout_f << "library IEEE;" << std::endl;
66.     fout_f << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
67.     fout_f << std::endl;
68.     fout_f << "entity " << fileName << " is" << std::endl;
69.     fout_f << "       port(" << std::endl;
70.     if (Count_zminna > 1)
71.     {
72.             fout_f << "          A : in STD_LOGIC_VECTOR(" << Count_zminna -
1 << " downto 0);" << std::endl;
73.             fout_f << "          B : out STD_LOGIC_VECTOR(" << Count_zminna -
1 << " downto 0)" << std::endl;
74.     }
75.     else
76.     {
77.             fout_f << "          A : in STD_LOGIC;" << std::endl;
78.         fout_f << "          B : out STD_LOGIC" << std::endl;
79.     }
80.     fout_f << "          );" << std::endl;
81.     fout_f << "end " << fileName << ";" <<std::endl;
82.     fout_f << std::endl;
83.     fout_f << std::endl;
84.     fout_f << "architecture " << fileName << " of " << fileName << " is" <<
std::endl;
```

```
85.     fout_f << "begin" << std::endl;
86.     fout_f << "         process(A)" << std::endl;
87.     fout_f << "       begin" << std::endl;
88.     if (Count_zminna > 1)
89.     {
90.             for (int i = 0; i < Count_zminna; i++)
91.             {
92.                     fout_f << "          B(" << i << ") <= " << A[i] << ";"
<< std::endl;
93.             }
94.     }
95.     else
96.     {
97.             for (int i = 0; i < Count_zminna; i++)
98.             {
99.              fout_f << "          B <= " << A[i] << ";" << std::endl;
100.            }
101.    }
102.    fout_f << "       end process;" << std::endl;
103.    fout_f << std::endl;
104.    fout_f << std::endl;
105.    fout_f << "end " << fileName << ";" << std::endl;
106.
107.    gotoxy(0, 14);
108.    for (int x = 0; x < 120; x++)
109.    {
110.            std::cout << " ";
111.    }
112.    gotoxy(0, 14);
113.
114.    std::cout << "F is finished" << std::endl;
115.    fout_f.close();
116.    delete[] A;
117.
118.    return 0;
119. }
```

ДОДАТОК Д.3 Заголовочний файл класу, який створює *VHDL*-опис вузла *MGC*

create_mgc.h

```
1. #ifndef CREATE_MGC_H
2. #define CREATE_MGC_H
3. #include "BaseClass.h"
4. #include <iostream>
5.
6. class createMGC : public BaseClass
7. {
8. public:
9.      createMGC(int basis, std::string fileN);
10.     int creaMGC();
11. private:
12.     int Basis;
13.     int Count_zminna;
14.     std::string fileName;
15. };
16.
17. #endif // CREATE_MGC_H
18.
```

ДОДАТОК Д.4 Файл опису класу, який створює *VHDL*-опис вузла *MGC*

create_mgc.cpp

```
1. #include <iostream>
2. #include <fstream>
3. #include <io.h>
4. #include <math.h>
5. #include "create_mgc.h"
6. #include "windows.h"
7. #include "string"
8.
9. createMGC::createMGC(int basis, std::string fileN)
10. {
11.     Basis = basis;
12.     fileName = fileN;
13. }
14.
15. int createMGC::creaMGC()
16. {
17.     if (Basis > 998)
18.     {
19.             std::cout << "The basis is too large" << std::endl;
20.             return 1;
21.     }
22.     const int Count_zminna = ceil((log(Basis)) / (log(2)));
23.
24.     std::ifstream *fout_Q = new std::ifstream [Count_zminna];
25.     std::ifstream *fout_Qi = new std::ifstream [Count_zminna];
26.     //char mass[len];
27.     //int Count_zminna = 5;
28.     for (int i = 0; i < Count_zminna; i++)
29.     {
30.             std::string str4 = "D://Resourse_min_Q" + std::to_string(i) +
".txt";
31.             std::string str5 = str4;
32.             fout_Q[i].open(str5, std::ios::in | std::ios::binary |
std::ios::ate);
33.     }
34.     for (int j = 0; j < Count_zminna; j++)
35.     {
36.             std::string str4 = "D://Resourse_min_Q" + std::to_string(j) +
".txt";
37.             std::string str5 = str4;
38.             fout_Qi[j].open(str5, std::ios::in | std::ios::binary);
39.     }
40.     int *filesize = new int [Count_zminna];
41.     char **A = new char *[Count_zminna];
42.     for (int i = 0; i < Count_zminna; i++)
43.     {
44.             filesize[i] = fout_Q[i].tellg();
45.             A[i] = new char[filesize[i]];
46.     }
47.     for (int i = 0; i < Count_zminna; i++)
48.     {
49.             fout_Q[i].close();
50.     }
51.     //fout_Q[0].close();
52.     //ifstream fs1("D://Resourse_Q0.txt", ios::in | ios::binary);
53.     for (int i = 0; i < Count_zminna; i++)
54.     {
55.         if (!fout_Qi[i]) return 1;
56.         fout_Qi[i].getline(A[i], filesize[i] + 1);
57.     }
58.
```

```
59.     for (int i = 0; i < Count_zminna; i++)
60.     {
61.         fout_Qi[i].close();
62.     }
63.
64.     std::ofstream fout_MGC;
65.     fout_MGC.open("D://" + fileName + ".txt", std::ios::out |
std::ios::trunc);
66.     fout_MGC << "library IEEE;" << std::endl;
67.     fout_MGC << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
68.     fout_MGC << std::endl;
69.     fout_MGC << "entity " << fileName << " is" << std::endl;
70.     fout_MGC << "       port(" << std::endl;
71.
72.     if (Count_zminna > 1)
73.     {
74.             //std::cout << "          A : in STD_LOGIC_VECTOR(" <<
Count_zminna - 1 << " downto 0);" << std::endl;
75.             fout_MGC << "          A : in STD_LOGIC_VECTOR(" << Count_zminna
- 1 << " downto 0);" << std::endl;
76.             //std::cout << "          B : in STD_LOGIC_VECTOR(" <<
Count_zminna - 1 << " downto 0);" << std::endl;
77.             fout_MGC << "          B : in STD_LOGIC_VECTOR(" << Count_zminna
- 1 << " downto 0);" << std::endl;
78.             //std::cout << "          C : in STD_LOGIC_VECTOR(" <<
Count_zminna - 1 << " downto 0);" << std::endl;
79.             fout_MGC << "          C : in STD_LOGIC_VECTOR(" << Count_zminna
- 1 << " downto 0);" << std::endl;
80.             //std::cout << "          S : out STD_LOGIC_VECTOR(" <<
Count_zminna - 1 << " downto 0);" << std::endl;
81.             fout_MGC << "          S : out STD_LOGIC_VECTOR(" << Count_zminna
- 1 << " downto 0)" << std::endl;
82.     }
83.     else
84.     {
85.             fout_MGC << "          A : in STD_LOGIC;" << std::endl;
86.             fout_MGC << "          B : in STD_LOGIC;" << std::endl;
87.             fout_MGC << "          C : in STD_LOGIC;" << std::endl;
88.         fout_MGC << "          S : out STD_LOGIC" << std::endl;
89.     }
90.
91.     fout_MGC << "          );" << std::endl;
92.     fout_MGC << "end " << fileName << ";" << std::endl;
93.     fout_MGC << std::endl;
94.     fout_MGC << std::endl;
95.     fout_MGC << "architecture " << fileName << " of " << fileName << " is"
<< std::endl;
96.     fout_MGC << "begin" << std::endl;
97.     fout_MGC << "         process(A, B, C)" << std::endl;
98.     fout_MGC << "       begin" << std::endl;
99.     if (Count_zminna > 1)
100.    {
101.            for (int i = 0; i < Count_zminna; i++)
102.            {
103.                    fout_MGC << "          S(" << i << ") <= " << A[i] << ';'
<< std::endl;
104.            }
105.    }
106.    else
107.    {
108.            for (int i = 0; i < Count_zminna; i++)
109.            {
```

```
110.                    fout_MGC << "          S <= " << A[i] << ";" <<
std::endl;
111.            }
112.    }
113.    fout_MGC << "       end process;" << std::endl;
114.    fout_MGC << std::endl;
115.    fout_MGC << std::endl;
116.    fout_MGC << "end " << fileName << ";" << std::endl;
117.
118.
119.    gotoxy(0, 12);
120.    for (int x = 0; x < 120; x++)
121.    {
122.            std::cout << " ";
123.    }
124.    gotoxy(0, 12);
125.
126.    std::cout << "MGC is finished" << std::endl;
127.    fout_MGC.close();
128.    delete[] A;
129.
130.    return 0;
131. }
132.
```

ДОДАТОК Д.5 Заголовочний файл класу, який створює *VHDL*-опис помножувача

create_matrix.h

```
1. #ifndef CREATING_MATRIX_H
2. #define CREATING_MATRIX_H
3. #include <iostream>
4. #include "mgc_element.h"
5. #include "f_element.h"
6. #include "BaseClass.h"
7.
8. class CreateMatrix : public BaseClass
9. {
10. public:
11.     CreateMatrix(int mbase, int mlarge, std::string mF, std::string mMGC,
std::string mMultiplier);
12.     void fillMatrix();
13.     void printFile();
14.     int wherex();
15.     int wherey();
16.     ~CreateMatrix();
17. private:
18.     std::string strF;
19.     std::string strMGC;
20.     std::string strMultiplier;
21.     int busCount;
22.     int base;
23.     int large;
24.     MGC** MG;
25.     felement* F;
26. };
27.
28. #endif // CREATING_MATRIX_H
29.
```

ДОДАТОК Д.6 Файл опису класу, який створює *VHDL*-опис помножува-

ча

create_matrix.cpp

```
1. #include "creating_matrix.h"
2. #include "mgc_element.h"
3. #include "f_element.h"
4. #include <iostream>
5. #include <fstream>
6. #include <math.h>
7. #include "windows.h"
8. #include <stdio.h>
9. #include <string>
10.
11.
12. CreateMatrix::CreateMatrix(int mbase, int mlarge, std::string mF,
std::string mMGC, std::string mMultiplier)
13. {
14.     strF = mF;
15.     strMGC = mMGC;
16.     strMultiplier = mMultiplier;
17.     base = mbase;
18.     busCount = 0;
19.     large = mlarge;
20.     //exit(1);
21.     int matrix_size = (large * 2) - 1;
22.     MG = new MGC *[matrix_size];
23.     for (int i = 0; i < matrix_size; i++)
24.             MG[i] = new MGC[matrix_size];
25.     F = new felement[large - 1];
26.     for (int i = 0; i < large - 1; i++)
27.     {
28.             F[i].initialF(base, large);
29.     }
30.     //MGC MG[matrix_size][matrix_size];
31.     std::cout << "Dates are processing. Wait for finishing" << std::endl;
32. }
33.
34. CreateMatrix::~CreateMatrix()
35. {
36.     int matrix_size = (large * 2) - 1;
37.     for (int count = 0; count < matrix_size; count++)
38.             delete[] MG[count];
39.     delete[]F;
40. }
41.
42. void CreateMatrix::fillMatrix()
43. {
44.     int countProcessing;
45.     int matrix_size = (large * 2) - 1;
46.     //int busCount = 0;
47.     int m = large, k = m - 1;
48.     int p = 0;
49.     //Cicle for setting elements of array full or empty
50.     for (int j = 0; j <= 2 * m - 2; j++)
51.     {
52.             for (int i = k; i <= 2 * m - 2 - p; i++)
53.             {
54.                     MG[j][i].MGCSetFull();
55.             }
56.             if (j < m - 1)
57.             {
58.                     k--;
```

```
59.                     p++;
60.             }
61.             else {
62.                     k++;
63.                     p--;
64.             }
65.     }
66.     k = m - 1;
67.     p = 0;
68.     //Cicle for setting in input C elements L
69.     for (int j = 0; j < matrix_size; j++)
70.     {
71.             for (int i = 0; i < matrix_size; i++)
72.             {
73.                     if ((i + j == large - 1) || (j == 0 && i >= large - 1))
74.                     {
75.                             MG[j][i].AddC(-5);
76.                     }
77.             }
78.     }
79.     //Cicle for setting A and B on elements of array
80.     int elem;
81.     for (int j = 0; j <= 2 * m - 2; j++)
82.     {
83.             elem = large - 1;
84.             for (int i = k; i <= 2 * m - 2 - p; i++)
85.             {
86.                     if (j <= large - 1)
87.                     {
88.                             MG[j][i].AddA(j);
89.                             MG[j][i].AddB(elem);
90.                     }
91.                     else
92.                     {
93.                             MG[j][i].AddA(elem);
94.                     }
95.                     elem--;
96.             }
97.             //std::cout << std::endl;
98.             if (j < m - 1)
99.             {
100.                    k--;
101.                    p++;
102.            }
103.            else {
104.                    k++;
105.                    p--;
106.            }
107.    }
108.    k = m - 1;
109.    p = 0;
110.    int vel;
111.    k = m - 1;
112.    p = 0;
113.    //Cicle for add adreses amonth elements that create koeficient on that
we multiply polynom
114.    int aa = large - 2;
115.    int b = -1;
116.    for (int j = 0; j <= 2 * m - 2; j++)
117.    {
118.            int r = 0;
119.            vel = -1;
120.            for (int i = k; i <= 2 * m - 2 - p; i++)
```

```
121.            {
122.                    //aa++;
123.                    //b++;
124.                    //int r = 0;
125.                    if (j > large - 1)
126.                    {
127.                            if (r == 0)
128.                            {
129.                                    aa++;
130.                                    b++;
131.                            }
132.                            vel++;
133.                            if (vel == 0)
134.                            {
135.                                    busCount++;
136.                            }
137.                            MG[j][i].Addb(&MG[aa][b]);
138.                            MG[aa][b].Adds(&MG[j][i]);
139.
140.                            MG[j][i].AddBbus(busCount - 1);
141.                            MG[aa][b].AddSbus(busCount - 1);
142.                            r++;
143.                    }
144.            }
145.            if (j < m - 1)
146.            {
147.                    k--;
148.                    p++;
149.            }
150.            else {
151.                    k++;
152.                    p--;
153.            }
154.    }
155.    k = m - 1;
156.    p = 0;
157.    //Circuit for adding result elements R
158.    int val = large;
159.    for (int i = 0; i <= (2 * large) - 2; i++)
160.    {
161.            if (MG[(2 * large) - 2][i].returnMGCempty() == true)
162.            {
163.                    MG[(2 * large) - 2][i].AddS(val - 1);
164.                    val--;
165.            }
166.    }
167.    //Circle for adding connections between elements in array
168.    k = m - 1;
169.    p = 0;
170.    //int busCount = 0;
171.    for (int j = 0; j <= 2 * m - 2; j++)
172.    {
173.            for (int i = k; i <= 2 * m - 2 - p; i++)
174.            {
175.                    if (MG[j][i].ReturnS() == -1)
176.                    {
177.                            for (int s = j + 1; s <= (2 * large) - 2; s++)
178.                            {
179.                                    if (MG[s][i].returnMGCempty() == true)
180.                                    {
181.                                            busCount++;
182.                                            MG[j][i].Adds(&MG[s][i]);
183.                                            MG[j][i].AddSbus(busCount - 1);
```

```
184.                                            MG[s][i].Addc(&MG[j][i]);
185.                                            MG[s][i].AddCbus(busCount - 1);
186.                                            break;
187.                                    }
188.                            }
189.                    }
190.            }
191.            if (j < m - 1)
192.            {
193.                    k--;
194.                    p++;
195.            }
196.            else {
197.                    k++;
198.                    p--;
199.            }
200.    }
201.    //Circle for adding elements F to MG
202.    int countInputBits = ceil((log(base)) / (log(2)));
203.    if (countInputBits > 1)
204.    {
205.            int i = 0;
206.            for (int j = large - 1; j < (2 * large) - 2; j++)
207.            {
208.                    MG[j][i].AddSbus(busCount);
209.                    F[i].AddAbus(busCount);
210.                    int l = MG[j + 1][i + 1].ReturnBbus();
211.                    F[i].AddBbus(MG[j + 1][i + 1].ReturnBbus());
212.                    busCount++;
213.                    i++;
214.            }
215.    }
216. }
217.
218. void CreateMatrix::printFile()
219. {
220.    int countProcessing = 0;
221.    int U = 0;
222.    int countInputBits = ceil((log(base)) / (log(2)));
223.    std::ofstream fout;
224.    //std::string str = "D://" + strMultiplier + ".txt";
225.    fout.open("D://" + strMultiplier + ".txt");
226.    if (countInputBits >1)
227.    {
228.            fout << "library IEEE;" << std::endl;
229.            fout << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
230.            fout << std::endl;
231.            fout << "entity " << strMultiplier << " is" << std::endl;
232.            fout << "       port(" << std::endl;
233.            fout << "          A : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
234.            fout << "          B : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
235.            fout << "          p : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
236.            fout << "          L : in STD_LOGIC_VECTOR(" << countInputBits -
1 << " downto 0);" << std::endl;
237.            fout << "          R : out STD_LOGIC_VECTOR(" << (countInputBits
* large) - 1 << " downto 0)" << std::endl;
238.            fout << "          );" << std::endl;
239.            fout << "end " << strMultiplier << ";" << std::endl;
240.            fout << std::endl;
241.            fout << std::endl;
```

```
242.            fout << "architecture " + strMultiplier + " of " + strMultiplier
+ " is" << std::endl;
243.            fout << std::endl;
244.            fout << std::endl;
245.            fout << "component " + strMGC << std::endl;
246.            fout << "  port (" << std::endl;
247.            fout << "       A : in STD_LOGIC_VECTOR(" << countInputBits - 1
<< " downto 0);" << std::endl;
248.            fout << "       B : in STD_LOGIC_VECTOR(" << countInputBits - 1
<< " downto 0);" << std::endl;
249.            fout << "       C : in STD_LOGIC_VECTOR(" << countInputBits - 1
<< " downto 0);" << std::endl;
250.            fout << "       S : out STD_LOGIC_VECTOR(" << countInputBits - 1
<< " downto 0)" << std::endl;
251.            fout << "       );" << std::endl;
252.            fout << "end component;" << std::endl;
253.            fout << std::endl;
254.            fout << std::endl;
255.            fout << "component " + strF << std::endl;
256.            fout << "  port (" << std::endl;
257.            fout << "       A : in STD_LOGIC_VECTOR(" << countInputBits - 1
<< " downto 0);" << std::endl;
258.            fout << "       B : out STD_LOGIC_VECTOR(" << countInputBits - 1
<< " downto 0)" << std::endl;
259.            fout << "       );" << std::endl;
260.            fout << "end component;" << std::endl;
261.            fout << std::endl;
262.            fout << std::endl;
263.            for (int i = 0; i < busCount; i++)
264.            {
265.                    fout << "signal BUS" << i << " : STD_LOGIC_VECTOR (" <<
countInputBits - 1 << " downto 0);" << std::endl;
266.            }
267.            fout << std::endl;
268.            fout << std::endl;
269.            fout << "begin" << std::endl;
270.            fout << std::endl;
271.            fout << std::endl;
272.            for (int j = 0; j <= 2 * large - 2; j++)
273.            {
274.                    for (int i = 0; i <= 2 * large - 2; i++)
275.                    {
276.                            if (j < large)
277.                            {
278.                                    if (MG[j][i].returnMGCempty())
279.                                    {
280.                                            fout << "U" << U << " : " + strMGC
<< std::endl;
281.                                            fout << "  port map(" <<
std::endl;
282.                                            for (int k = 0; k <
countInputBits; k++)
283.                                            {
284.                                                    fout << "       A(" << k
<< ") =>" << "A(" << (MG[j][i].ReturnA() * countInputBits) + k << ")," <<
std::endl;
285.                                            }
286.                                            for (int k = 0; k <
countInputBits; k++)
287.                                            {
288.                                                    fout << "       B(" << k
<< ") =>" << "B(" << (MG[j][i].ReturnB() * countInputBits) + k << ")," <<
std::endl;
```

```
289.                                                    }
290.                                                    if (MG[j][i].ReturnC() == -5)
291.                                                    {
292.                                                            fout << "       C => L,"
<< std::endl;
293.                                                    }
294.                                                    else
295.                                                    {
296.                                                            fout << "       C" << " =>
" << "BUS" << MG[j][i].ReturnCbus() << "," << std::endl;
297.                                                    }
298.                                                    fout << "       S" << " => " <<
"BUS" << MG[j][i].ReturnSbus() << std::endl;
299.                                                    fout << "  );" << std::endl;
300.                                                    fout << std::endl;
301.                                                    U++;
302.                                            }
303.                                    }
304.                                    else
305.                                    {
306.                                            if (MG[j][i].returnMGCempty())
307.                                            {
308.                                                    fout << "U" << U << " : " + strMGC
<< std::endl;
309.                                                    fout << "  port map(" <<
std::endl;
310.                                                    for (int k = 0; k <
countInputBits; k++)
311.                                                    {
312.                                                            fout << "       A(" << k
<< ") =>" << "p(" << (MG[j][i].ReturnA() * countInputBits) + k << ")," <<
std::endl;
313.                                                    }
314.                                                    fout << "       B" << " => " <<
"BUS" << MG[j][i].ReturnBbus() << "," << std::endl;
315.                                                    if (MG[j][i].ReturnC() != -5)
316.                                                    {
317.                                                            fout << "       C" << " =>
" << "BUS" << MG[j][i].ReturnCbus() << "," << std::endl;
318.                                                    }
319.                                                    if (MG[j][i].ReturnS() == -1)
320.                                                    {
321.                                                            fout << "       S" << " =>
" << "BUS" << MG[j][i].ReturnSbus() << std::endl;
322.                                                            fout << "  );" <<
std::endl;
323.                                                    }
324.                                                    else
325.                                                    {
326.                                                            for (int k = 0; k <
countInputBits; k++)
327.                                                            {
328.                                                                    if (k ==
countInputBits - 1)
329.                                                                    {
330.                                                                            fout << "
S(" << k << ") =>" << "R(" << (MG[j][i].ReturnS() * countInputBits) + k << ")"
<< std::endl;
331.                                                                    }
332.                                                                    else
333.                                                                    {
```

```
334.                                                                    fout << "
S(" << k << ") =>" << "R(" << (MG[j][i].ReturnS() * countInputBits) + k << ")"
<< "," << std::endl;
335.                                                                }
336.                                                        }
337.                                                        fout << "  );" <<
std::endl;
338.                                                }
339.                                                fout << std::endl;
340.                                                U++;
341.                                        }
342.                                }
343.                        }
344.                        if (countProcessing < 120)
345.                        {
346.                                std::cout << char(221);
347.                                countProcessing++;
348.                        }
349.                        else
350.                        {
351.                                gotoxy(0, 10);
352.                                countProcessing = 0;
353.                                for (int x = 0; x < 120; x++)
354.                                {
355.                                        std::cout << " ";
356.                                }
357.                                gotoxy(0, 10);
358.                        }
359.                }
360.                for (int i = 0; i < large - 1; i++)
361.                {
362.                        fout << "U" << U << " : " + strF << std::endl;
363.                        fout << "  port map(" << std::endl;
364.                        fout << "       A" << " => " << "BUS" <<
F[i].ReturnAbus() << "," << std::endl;
365.                        fout << "       B" << " => " << "BUS" <<
F[i].ReturnBbus() << std::endl;
366.                        fout << "  );" << std::endl;
367.                        fout << std::endl;
368.                        U++;
369.                }
370.                fout << std::endl;
371.                fout << std::endl;
372.                fout << "end " + strMultiplier + ";";
373.        }
374.        else if (countInputBits = 1)
375.        {
376.                fout << "library IEEE;" << std::endl;
377.                fout << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
378.                fout << std::endl;
379.                fout << "entity " << strMultiplier << " is" << std::endl;
380.                fout << "       port(" << std::endl;
381.                fout << "          A : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
382.                fout << "          B : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
383.                fout << "          L : in STD_LOGIC"  << ";" << std::endl;
384.                fout << "          p : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
385.                fout << "          R : out STD_LOGIC_VECTOR(" << (countInputBits
* large) - 1 << " downto 0)" << std::endl;
386.                fout << "          );" << std::endl;
387.                fout << "end " << strMultiplier << ";" << std::endl;
```

```
388.           fout << std::endl;
389.           fout << std::endl;
390.         fout << "architecture " << strMultiplier << " of " << strMultiplier
<< " is" << std::endl;
391.           fout << std::endl;
392.           fout << std::endl;
393.           fout << "component " + strMGC << std::endl;
394.           fout << "  port (" << std::endl;
395.           fout << "       A : in STD_LOGIC;" << std::endl;
396.           fout << "       B : in STD_LOGIC;" << std::endl;
397.           fout << "       C : in STD_LOGIC;" << std::endl;
398.           fout << "       S : out STD_LOGIC" << std::endl;
399.           fout << "       );" << std::endl;
400.           fout << "end component;" << std::endl;
401.           fout << std::endl;
402.           fout << std::endl;
403.           for (int i = 0; i < busCount; i++)
404.           {
405.                   fout << "signal BUS" << i << " : STD_LOGIC;" <<
std::endl;
406.           }
407.           fout << std::endl;
408.           fout << std::endl;
409.           fout << "begin" << std::endl;
410.           fout << std::endl;
411.           fout << std::endl;
412.           for (int j = 0; j <= 2 * large - 2; j++)
413.           {
414.                   for (int i = 0; i <= 2 * large - 2; i++)
415.                   {
416.                           if (j < large)
417.                           {
418.                                   if (MG[j][i].returnMGCempty())
419.                                   {
420.                                           fout << "U" << U << " : " <<
strMGC << std::endl;
421.                                           fout << "  port map(" <<
std::endl;
422.                                           for (int k = 0; k <
countInputBits; k++)
423.                                           {
424.                                                   fout << "       A =>" <<
"A(" << (MG[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
425.                                           }
426.                                           for (int k = 0; k <
countInputBits; k++)
427.                                           {
428.                                                   fout << "       B =>" <<
"B(" << (MG[j][i].ReturnB() * countInputBits) + k << ")," << std::endl;
429.                                           }
430.                                           if (MG[j][i].ReturnC() == -5)
431.                                           {
432.                                                   fout << "       C => L,"
<< std::endl;
433.                                           }
434.                                           else
435.                                           {
436.                                                   fout << "       C" << " =>
" << "BUS" << MG[j][i].ReturnCbus() << "," << std::endl;
437.                                           }
438.                                           fout << "       S" << " => " <<
"BUS" << MG[j][i].ReturnSbus() << std::endl;
439.                                           fout << "  );" << std::endl;
```

```
440.                                            fout << std::endl;
441.                                            U++;
442.                                    }
443.                            }
444.                            else
445.                            {
446.                                    if (MG[j][i].returnMGCempty())
447.                                    {
448.                                            fout << "U" << U << " : " <<
strMGC << std::endl;
449.                                            fout << "  port map(" <<
std::endl;
450.                                            for (int k = 0; k <
countInputBits; k++)
451.                                            {
452.                                                    fout << "       A =>" <<
"p(" << (MG[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
453.                                            }
454.                                            fout << "       B" << " => " <<
"BUS" << MG[j][i].ReturnBbus() << "," << std::endl;
455.                                            if (MG[j][i].ReturnC() != -5)
456.                                            {
457.                                                    fout << "       C" << " =>
" << "BUS" << MG[j][i].ReturnCbus() << "," << std::endl;
458.                                            }
459.                                            if (MG[j][i].ReturnS() == -1)
460.                                            {
461.                                                    fout << "       S" << " =>
" << "BUS" << MG[j][i].ReturnSbus() << std::endl;
462.                                                    fout << "  );" <<
std::endl;
463.                                            }
464.                                            else
465.                                            {
466.                                                    for (int k = 0; k <
countInputBits; k++)
467.                                                    {
468.                                                            if (k ==
countInputBits - 1)
469.                                                            {
470.                                                                    fout << "
S =>" << "R(" << (MG[j][i].ReturnS() * countInputBits) + k << ")" << std::endl;
471.                                                            }
472.                                                            else
473.                                                            {
474.                                                                    fout << "
S =>" << "R(" << (MG[j][i].ReturnS() * countInputBits) + k << ")," << std::endl;
475.                                                            }
476.                                                    }
477.                                                    fout << "  );" <<
std::endl;
478.                                            }
479.                                            fout << std::endl;
480.                                            U++;
481.                                    }
482.                            }
483.                    }
484.                    if (countProcessing < 120)
485.                    {
486.                            std::cout << char(221);
487.                            countProcessing++;
488.                    }
489.                    else
```

```
490.                    {
491.                            gotoxy(0, 10);
492.                            countProcessing = 0;
493.                            for (int x = 0; x < 120; x++)
494.                            {
495.                                    std::cout << " ";
496.                            }
497.                            gotoxy(0, 10);
498.                    }
499.            }
500.            fout << std::endl;
501.            fout << std::endl;
502.            fout << "end " + strMultiplier + ";";
503.    }
504.    gotoxy(0, 10);
505.    //countProcessing = 0;
506.    for (int x = 0; x < 120; x++)
507.    {
508.            std::cout << " ";
509.    }
510.    gotoxy(0, 10);
511.    std::cout << "Processing is finished" << std::endl;
512. }
```

ДОДАТОК Д.7 Заголовочний файл класу, який містить опис вузла *F*

f_element.h

```
1. #ifndef F_ELEMENT_H
2. #define F_ELEMENT_H
3.
4. class felement
5. {
6. public:
7.      felement();
8.      void initialF(int mbase, int mlarge);
9.      void AddA(int p);
10.     void AddB(int p);
11.     void AddAbus(int p);
12.     void AddBbus(int p);
13.     int ReturnA();
14.     int ReturnB();
15.     int ReturnAbus();
16.     int ReturnBbus();
17. private:
18.     int countInputBits;
19.     int base;
20.     int large;
21.     int A, B;
22.     int Abus, Bbus;
23. };
24.
25. #endif // F_ELEMENT_H
```

ДОДАТОК Д.8 Файл опису класу, який містить опис вузла *F*

f_element.cpp

```
1. #include "f_element.h"
2. #include "math.h"
3.
4.
5. felement::felement()
6. {
```

```
7.      //countInputBits = ceil((log(mbase))/(log(2)));
8. }
9.
10. void felement::initialF(int mbase, int mlarge)
11. {
12.     base = mbase;
13.     large = mlarge;
14.     countInputBits = ceil((log(mbase)) / (log(2)));
15. }
16.
17. void felement::AddA(int p)
18. {
19.     A = p;
20. }
21.
22. void felement::AddB(int p)
23. {
24.     B = p;
25. }
26.
27. void felement::AddAbus(int p)
28. {
29.     Abus = p;
30. }
31.
32. void felement::AddBbus(int p)
33. {
34.     Bbus = p;
35. }
36.
37. int felement::ReturnA()
38. {
39.     return A;
40. }
41.
42. int felement::ReturnB()
43. {
44.     return B;
45. }
46.
47. int felement::ReturnAbus()
48. {
49.     return Abus;
50. }
51.
52. int felement::ReturnBbus()
53. {
54.     return Bbus;
55. }
```

ДОДАТОК Д.9 Заголовочний файл класу, який створює булеві функції для вузла *F*

generate_functions_for_f.h

```
1. #ifndef GENERATE_FUNCTIONS_FOR_F_H
2. #define GENERATE_FUNCTIONS_FOR_F_H
3. #include "BaseClass.h"
4.
5. class generateFunctionsForF : public BaseClass
6. {
7. public:
8.      generateFunctionsForF(int basis);
```

```
9.      int generateFunction();
10. private:
11.     int Basis;
12.     int Count_zminna;
13. };
14.
15. #endif // GENERATE_FUNCTIONS_FOR_F_H
```

ДОДАТОК Д.10 Файл опису класу, який створює булеві функції для вузла *F*

generate_functions_for_f.cpp

```
1. #include "generate_functions_for_f.h"
2. #include <fstream>
3. #include <iostream>
4. #include <math.h>
5. #include <string.h>
6. #include <bitset>
7. #include <windows.h>
8.
9. generateFunctionsForF::generateFunctionsForF(int basis)
10. {
11.     Basis = basis;
12. }
13.
14. int generateFunctionsForF::generateFunction()
15. {
16.     gotoxy(0, 14);
17.     if (Basis > 998)
18.     {
19.             std::cout << "The basis is too large" << std::endl;
20.             return 1;
21.     }
22.     Count_zminna = ceil((log(Basis)) / (log(2)));
23.     std::ofstream *fout_F = new std::ofstream [Count_zminna];
24.     for (int i = 0; i < Count_zminna; i++)
25.     {
26.             std::string str4 = "D://Resourse_F" + std::to_string(i) + ".txt";
27.             std::string str5 = str4;
28.             fout_F[i].open(str5, std::ofstream::out | std::ofstream::trunc);
29.     }
30.     std::bitset<10> A;
31.     //std::bitset<10> B;
32.     //std::bitset<10> C;
33.     int res;
34.     bool *oor = new bool [Count_zminna];
35.     for (int m = 0; m < Count_zminna; m++)
36.     {
37.             oor[m] = false;
38.     }
39.     int countProcessing = 0;
40.     for (int i = 0; i < Basis; i++)
41.     {
42.             A = i;
43.             if (Basis == 0)
44.             {
45.                     res = 0;
46.             }
47.             else
48.             {
49.                     res = (Basis - i);
50.             }
```

```
51.             //std::cout << std::hex << int(res) << std::endl;
52.             if (res != 0)
53.             {
54.                     for (int r = 0; r < Count_zminna; r++)
55.                     {
56.                             //bool oor = false;
57.                             int YangerBit = (res >> r) & 0x01;
58.                             if (YangerBit == 1)
59.                             {
60.                                     if (oor[r] == true)
61.                                     {
62.                                             fout_F[r] << " or ";
63.                                     }
64.                                     for (int s = Count_zminna - 1; s >= 0; s--
)
65.                                     {
66.                                             oor[r] = true;
67.                                             if (s != Count_zminna - 1)
68.                                             {
69.                                                     fout_F[r] << " and ";
70.                                             }
71.                                             if (s == (Count_zminna - 1))
72.                                             {
73.                                                     fout_F[r] << "( ";
74.                                             }
75.
76.
77.                                             if (A[s] == 0)
78.                                             {
79.                                                     fout_F[r] << "not ";
80.                                                     std::string str6 = "A(" +
std::to_string(s) + ")";
81.                                                     std::string str7 = str6;
82.                                                     fout_F[r] << str7;
83.                                             }
84.                                             if (A[s] == 1)
85.                                             {
86.                                                     std::string str6 = "A(" +
std::to_string(s) + ")";
87.                                                     std::string str7 = str6;
88.                                                     fout_F[r] << str7;
89.                                             }
90.                                     }
91.                                     fout_F[r] << " )";
92.                             }
93.                     }
94.                     if (countProcessing < 120)
95.                     {
96.                             std::cout << char(221);
97.                             countProcessing++;
98.                     }
99.                     else
100.                    {
101.                            gotoxy(0, 14);
102.                            countProcessing = 0;
103.                            for (int x = 0; x < 120; x++)
104.                            {
105.                                    std::cout << " ";
106.                            }
107.                            gotoxy(0, 14);
108.                    }
109.
110.            }
```

```
111.    }
112.
113.
114.    for (int l = 0; l < Count_zminna; l++)
115.    {
116.            fout_F[l].close();
117.    }
118.    return 0;
119. }
```

ДОДАТОК Д.11 Заголовочний файл класу, який створює булеві функції для вузла *MGC*

generate_functions_for_mgc.h

```
1. #ifndef GENERATE_FUNCTIONS_FOR_MGC_H
2. #define GENERATE_FUNCTIONS_FOR_MGC_H
3. #include "BaseClass.h"
4.
5. class generateFunctionsForMGC : public BaseClass
6. {
7. public:
8.      generateFunctionsForMGC(int basis);
9.      int generateFunction();
10. private:
11.     int Basis;
12.     //int Count_zminna;
13. };
14.
15. #endif // GENERATE_FUNCTIONS_FOR_MGC_H
```

ДОДАТОК Д.12 Файл опису класу, який створює булеві функції для вузла *MGC*

generate_functions_for_mgc.cpp

```
1. #include "generate_functions_for_mgc.h"
2. #include <fstream>
3. #include <iostream>
4. #include <math.h>
5. #include <bitset>
6. #include <windows.h>
7. #include <string.h>
8.
9. generateFunctionsForMGC::generateFunctionsForMGC(int basis)
10. {
11.     Basis = basis;
12. }
13.
14. int generateFunctionsForMGC::generateFunction()
15. {
16.     gotoxy(0, 12);
17.     if (Basis > 998)
18.     {
19.             std::cout << "The basis is too large" << std::endl;
20.             return 1;
21.     }
22.     const int Count_zminna = ceil((log(Basis)) / (log(2)));
23.     std::ofstream *fout_Q = new std::ofstream [Count_zminna];
24.     for (int i = 0; i < Count_zminna; i++)
25.     {
26.             std::string str4 = "D://Resourse_Q" + std::to_string(i) + ".txt";
```

```
27.             std::string str5 = str4;
28.             fout_Q[i].open(str5, std::ofstream::out | std::ofstream::trunc);
29.     }
30.     std::bitset<10> A;
31.     std::bitset<10> B;
32.     std::bitset<10> C;
33.     int res;
34.     bool *oor = new bool [Count_zminna];
35.     for (int m = 0; m < Count_zminna; m++)
36.     {
37.             oor[m] = false;
38.     }
39.     int countProcessing = 0;
40.     for (int i = 0; i < Basis; i++)
41.     {
42.             A = i;
43.             for (int j = 0; j < Basis; j++)
44.             {
45.                     B = j;
46.                     for (int k = 0; k < Basis; k++)
47.                     {
48.                             C = k;
49.                             res = (((j * k) % Basis) + i) % Basis;
50.                             //std::cout << std::hex << int(res) << std::endl;
51.                             if (res != 0)
52.                             {
53.                                     for (int r = 0; r < Count_zminna; r++)
54.                                     {
55.                                             //bool oor = false;
56.                                             int YangerBit = (res >> r) & 0x01;
57.                                             if (YangerBit == 1)
58.                                             {
59.                                                     if (oor[r] == true)
60.                                                     {
61.                                                             fout_Q[r] << " or
";
62.                                                     }
63.                                                     for (int s = Count_zminna
- 1; s >= 0; s--)
64.                                                     {
65.                                                             oor[r] = true;
66.                                                             if (s !=
Count_zminna - 1)
67.                                                             {
68.                                                                     fout_Q[r]
<< " and ";
69.                                                             }
70.                                                             if (s ==
(Count_zminna - 1))
71.                                                             {
72.                                                                     fout_Q[r]
<< "( ";
73.                                                             }
74.
75.
76.                                                             if (A[s] == 0)
77.                                                             {
78.                                                                     fout_Q[r]
<< "not ";
79.                                                                     std::string
str6 = "A(" + std::to_string(s) + ")";
80.                                                                     std::string
str7 = str6;
```

```
81.                                                                     fout_Q[r]
<< str7;
82.                                                             }
83.                                                             if (A[s] == 1)
84.                                                             {
85.                                                                     std::string
str6 = "A(" + std::to_string(s) + ")";
86.                                                                     std::string
str7 = str6;
87.                                                                     fout_Q[r]
<< str7;
88.                                                             }
89.                                                     }
90.                                                     for (int s = Count_zminna
- 1; s >= 0; s--)
91.                                                     {
92.                                                             fout_Q[r] << " and
";
93.                                                             if (B[s] == 0)
94.                                                             {
95.                                                                     fout_Q[r]
<< "not ";
96.                                                                     std::string
str6 = "B(" + std::to_string(s) + ")";
97.                                                                     std::string
str7 = str6;
98.                                                                     fout_Q[r]
<< str7;
99.                                                             }
100.                                                            if (B[s] == 1)
101.                                                            {
102.                                                                    std::string
str6 = "B(" + std::to_string(s) + ")";
103.                                                                    std::string
str7 = str6;
104.                                                                    fout_Q[r]
<< str7;
105.                                                            }
106.                                                    }
107.                                                    for (int s = Count_zminna
- 1; s >= 0; s--)
108.                                                    {
109.                                                            fout_Q[r] << " and
";
110.                                                            if (C[s] == 0)
111.                                                            {
112.                                                                    fout_Q[r]
<< "not ";
113.                                                                    std::string
str6 = "C(" + std::to_string(s) + ")";
114.                                                                    std::string
str7 = str6;
115.                                                                    fout_Q[r]
<< str7;
116.                                                            }
117.                                                            if (C[s] == 1)
118.                                                            {
119.                                                                    std::string
str6 = "C(" + std::to_string(s) + ")";
120.                                                                    std::string
str7 = str6;
121.                                                                    fout_Q[r]
<< str7;
```

```
122.                                                                    }
123.                                                                    if (s == 0)
124.                                                                    {
125.                                                                            fout_Q[r]
<< " )";
126.                                                                    }
127.                                                            }
128.                                                    }
129.                                                    if (countProcessing < 120)
130.                                                    {
131.                                                            std::cout << char(221);
132.                                                            countProcessing++;
133.                                                    }
134.                                                    else
135.                                                    {
136.                                                            gotoxy(0, 12);
137.                                                            countProcessing = 0;
138.                                                            for (int x = 0; x < 120;
x++)
139.                                                            {
140.                                                                    std::cout << " ";
141.                                                            }
142.                                                            gotoxy(0, 12);
143.                                                    }
144.                                            }
145.                                    }
146.                            }
147.                    }
148.    }
149.
150.    for (int l = 0; l < Count_zminna; l++)
151.    {
152.            fout_Q[l].close();
153.    }
154.    return 0;
155. }
```

ДОДАТОК Д.13 Заголовочний файл класу, який реалізує інтерфейс для роботи з булевими функціями

logical_expr.h

```
1. #ifndef LOGICAL_EXPRESSION_HPP
2. #define LOGICAL_EXPRESSION_HPP
3. #include <iostream>
4. #include <string>
5. #include <sstream>
6. #include <utility>
7. #include <iterator>
8. #include <algorithm>
9. #include <stdexcept>
10. #include <cmath>
11. #include <boost/regex.hpp>
12. #include <boost/format.hpp>
13. #include <boost/tokenizer.hpp>
14. #include <boost/call_traits.hpp>
15. #include <boost/algorithm/string.hpp>
16. #include <boost/dynamic_bitset.hpp>
17. #include <boost/io/ios_state.hpp>
18. #include <boost/optional.hpp>
19.
20. namespace logical_expr {
21.     using namespace std;
```

```
22.
23.     // Don't care
24.     /*thread_local*/static const boost::optional<bool> dont_care =
boost::none;
25.     // static const boost::logic::tribool dont_care = indeterminated; better
than optional<bool>
26.
27.     // expr_mode is not used now.
28.     enum expr_mode { alphabet_expr, verilog_expr, truth_table };
29.
30.     // argument generating iterator for logical_function
31.     class arg_gen_iterator {
32.     public:
33.             typedef boost::dynamic_bitset<> value_type;
34.             typedef arg_gen_iterator this_type;
35.             arg_gen_iterator(int width, int val)
36.                     : width_(width), current_val_(val), value_(width, val) {}
37.             this_type& operator++() {
38.                     value_type tmp(width_, current_val_ + 1);
39.                     value_.swap(tmp);   // never throw any exceptions
40.                     ++current_val_;
41.                     return *this;
42.             }
43.             this_type operator++(int)
44.             {
45.                     this_type before = *this; ++*this; return before;
46.             }
47.             this_type& operator--() {
48.                     value_type tmp(width_, current_val_ - 1);
49.                     value_.swap(tmp);
50.                     --current_val_;
51.                     return *this;
52.             }
53.             this_type operator--(int)
54.             {
55.                     this_type before = *this; --*this; return before;
56.             }
57.             bool operator<(const this_type &it) const
58.             {
59.                     return (it.width_ == width_ && current_val_ <
it.current_val_);
60.             }
61.             bool operator>(const this_type &it) const
62.             {
63.                     return (it.width_ == width_ && current_val_ >
it.current_val_);
64.             }
65.             bool operator==(const this_type &it) const
66.             {
67.                     return (it.width_ == width_ && current_val_ ==
it.current_val_);
68.             }
69.             bool operator!=(const this_type &it) const
70.             {
71.                     return !(*this == it);
72.             }
73.             const value_type& operator*() const { return value_; }
74.     private:
75.             const int width_;
76.             int current_val_;
77.             value_type value_;
78.     };
79.
```

```
80.     // Argument generator for logical_function using Iterator (default:
arg_gen_iterator)
81.     template<typename Iterator = arg_gen_iterator>
82.     class arg_generator {
83.     public:
84.             arg_generator(int nbegin, int nend, int width)
85.                     : begin_(width, nbegin), end_(width, nend) {}
86.             const Iterator& begin() const { return begin_; }
87.             const Iterator&  end()  const { return end_; }
88.     private:
89.             const Iterator begin_, end_;
90.     };
91.
92.
93.     //
94.     // * String to be parsed has to be in the following form:
95.     // * ${Function-Name}(Variables-divided-by-',' ...) = ${TERMS} + ...
96.     // * White spaces will be ignored
97.     // * Default character to invert a variable is '~' (first template
parameter)
98.     // See README for more information about parsing
99.     // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
100.    template<char inverter = '~', bool escape = true, expr_mode mode =
alphabet_expr>
101.    class function_parser {
102.    public:
103.            typedef std::pair<string, vector<string>> result_type;
104.            function_parser() {}
105.            explicit function_parser(const string &expr, const char first) :
expr_(expr), first_char_(first) {}
106.            ~function_parser() {}
107.
108.            void set_expression(const string &expr) { expr_ = expr; }
109.            const string& get_expression() const { return expr_; }
110.            const string& function_name() const { return func_name_; }
111.
112.            result_type parse() {
113.                    auto untokenized = scanner();
114.                    auto token = tokenizer(untokenized);
115.                    boost::optional<char> undecl =
use_undeclared_vars(token.second, token.first);
116.                    if (undecl)
117.                            throw std::runtime_error(
118.                            (boost::format("expr: Using undeclared variable,
%c") % *undecl).str()
119.                            );
120.                    if (!is_sequence(token.first, first_char_))
121.                            throw std::runtime_error("expr: used variables
are not sequence");
122.                    return std::move(token);
123.            }
124.
125.            vector<string> scanner() {
126.                    if (expr_.empty())
127.                            throw std::runtime_error("expr: Expression is
empty, aborted");
128.                    string expr_with_nospaces = boost::regex_replace(expr_,
boost::regex("\\s"), "");
129.                    boost::regex reg(
130.                            (boost::format("([A-Za-z_-]+)\\((((\\s*[A-Za-
z],)*)([A-Za-z]))\\)=(((%1%?[A-Za-z])+\\+)*((%1%?[A-Za-z])+))$")
131.                                    % (escape ? string{ '\\', inverter } :
string{ inverter })).str(),
```

```
132.                            boost::regex::perl
133.                    );
134.                    boost::smatch result;
135.                    if (!boost::regex_match(expr_with_nospaces, result, reg))
136.                            throw std::runtime_error("expr: Input string does
not match the correct form");
137.                    func_name_ = result[1];
138.                    //                              decl-vars  decl-terms
139.                    return vector<string>{result[2], result[6]};
140.            }
141.
142.            static result_type tokenizer(const vector<string> &untokenized) {
143.                    std::vector<string> terms;
144.                    typedef boost::char_separator<char> char_separator;
145.                    boost::tokenizer<char_separator>
146.                            var_tokenizer(untokenized[0],
char_separator(",")), term_tokenizer(untokenized[1], char_separator("+"));
147.                    ostringstream oss;
148.                    for (auto token : var_tokenizer)   oss << token;
149.                    for (auto token : term_tokenizer)
terms.push_back(token);
150.                    return std::make_pair(oss.str(), terms);
151.            }
152.
153.            static bool is_sequence(const string &vars, char first_char) {
154.                    if (vars[0] != first_char)
155.                            throw std::runtime_error("expr: declare terms
which starts with not specified char");
156.                    auto previous = vars.begin();
157.                    for (auto it = ++vars.begin(); it != vars.end(); ++it) {
158.                            if (*it != static_cast<char>(*previous + 1))
159.                                    return false;
160.                            previous = it;
161.                    }
162.                    return true;
163.            }
164.
165.            static boost::optional<char> use_undeclared_vars(const
vector<string> &terms, const string &vars) {
166.                    for (auto term : terms)
167.                            for (auto used_var : term)
168.                                    if (used_var != inverter &&
vars.find(used_var, 0) == string::npos)
169.                                            return (used_var);
170.                    return boost::none;
171.            }
172.
173.    private:
174.            string expr_, func_name_;
175.            char first_char_;
176.    };
177.
178.
179.    //
180.    // Term properties
181.    //
182.    // Type Requirements:
183.    //  [*] typedef value type
184.    //  [*] Have set() and get() member functions
185.    //  [*] Default constructible
186.    //  [*] Has swap() member fuction that never throws any exceptions for
expcetion-safe
187.    //
```

```
188.
189.    // Term Property template class for POD types
190.    template<typename T, T DefaultValue>
191.    class term_property_pod {
192.    public:
193.            typedef T value_type;
194.            term_property_pod() : value_(DefaultValue) {}
195.            typename boost::call_traits<T>::param_type get() const { return
value_; }
196.            void set(typename boost::call_traits<T>::param_type value) {
value_ = value; }
197.            void swap(const term_property_pod<T, DefaultValue> &property)
noexcept(true)
198.            {
199.                    std::swap(value_, property.value_);
200.            }
201.    private:
202.            value_type value_;
203.    };
204.
205.    typedef term_property_pod<int, 0> term_no_property;
206.    typedef term_no_property term_dummy;
207.    typedef term_property_pod<char, ' '> term_name;
208.    typedef term_property_pod<bool, false> term_mark;
209.    typedef term_property_pod<unsigned int, 0> term_number;
210.    //
211.
212.    //
213.    // class: logical term
214.    //
215.    template<typename Property_ = term_dummy>
216.    class logical_term {
217.    public:
218.            typedef boost::optional<bool> value_type;
219.            typedef boost::dynamic_bitset<> arg_type;
220.            typedef logical_term<Property_> this_type;
221.            typedef std::size_t size_t;
222.            typedef Property_ property_type;
223.
224.            logical_term() {}
225.            logical_term(int bitsize, const value_type &init =
logical_expr::dont_care)
226.                    : term_(bitsize, init) {}
227.            template<typename Property>
228.            explicit logical_term(const logical_term<Property> &term)
229.            {
230.                    construct_from(term);
231.            }
232.            explicit logical_term(const arg_type &arg) {
233.                    for (int i = arg.size() - 1; 0 <= i; --i)
234.                            term_.push_back(arg[i]);
235.            }
236.
237.            template<typename Property>
238.            void construct_from(const logical_term<Property> &term)
239.            {
240.                    term_ = term.term_;
241.            }
242.
243.            template<typename Property>
244.            void swap(logical_term<Property> &term) noexcept(true) {
245.                    term_.swap(term.term_);
246.                    property_.swap(term.property_);
```

```
247.            }
248.
249.            size_t size() const
250.            {
251.                    return term_.size();
252.            }
253.
254.            bool size_check(const arg_type &arg) const
255.            {
256.                    return (size() == arg.size());
257.            }
258.
259.            bool is_same(const this_type &term) const
260.            {
261.                    return (term.term_ == term_);
262.            }
263.
264.            template<typename Property>
265.            bool size_check(const logical_term<Property> &term) const
266.            {
267.                    return (size() == term.size());
268.            }
269.
270.            size_t num_of_value(bool value) const {
271.                    return static_cast<size_t>(std::count_if(term_.begin(),
term_.end(),
272.                            [value](const value_type &b) { return (b !=
dont_care && *b == value); }));
273.            }
274.
275.            size_t diff_size(const this_type &term) const {
276.                    if (!size_check(term))
277.                            throw std::runtime_error(size_error_msg);
278.                    size_t diff_count = 0;
279.                    for (int i = 0; i < size(); ++i)
280.                            if (term_[i] != term[i])
281.                                    ++diff_count;
282.                    return diff_count;
283.            }
284.
285.            bool calculate(const arg_type &arg) const {
286.                    if (!size_check(arg))
287.                            throw std::runtime_error(size_error_msg);
288.                    bool ret = true;
289.                    for (int i = 0; i < term_.size(); ++i)
290.                            ret = ret && (term_[i] == dont_care ? true :
arg[term_.size() - 1 - i] == term_[i]);
291.                    return ret;
292.            }
293.
294.            bool operator()(const arg_type &arg) const
295.            {
296.                    return calculate(arg);
297.            }
298.
299.            value_type& operator[](int index)
300.            {
301.                    return term_[index];
302.            }
303.            const value_type& operator[](int index) const
304.            {
305.                    return term_[index];
306.            }
```

```
307.
308.            template<typename Property>
309.            bool operator==(const logical_term<Property> &term) const {
310.                    if (!size_check(term)) return false;
311.                    arg_generator<> gen(0, std::pow(2, size()), size());
312.                    for (auto it = gen.begin(); it != gen.end(); ++it)
313.                            if (calculate(*it) != term.calculate(*it))
314.                                    return false;
315.                    return true;
316.            }
317.
318.            template<typename Property>
319.            friend typename logical_term<Property>::property_type::value_type
320.                    property_get(const logical_term<Property>& term);
321.
322.            template<typename Property>
323.            friend void property_set(logical_term<Property>& term,
324.                    const typename
logical_term<Property>::property_type::value_type &arg);
325.
326.            template<typename Property>
327.            friend std::ostream& operator<<(std::ostream &os, const
logical_expr::logical_term<Property> &bf) {
328.                    boost::io::ios_flags_saver ifs(os);
329.                    for (auto b : bf.term_) {
330.                            if (b) os << std::noboolalpha << *b;
331.                            else    os << 'x';
332.                    }
333.                    return os;
334.            }
335.
336.    private:
337.            static const std::string size_error_msg;
338.            vector<value_type> term_;
339.            property_type property_;
340.    };
341.    template<typename Property>
342.    const string logical_term<Property>::size_error_msg = "target two
operands are not same size";
343.
344.
345.    // Create a logical_term with Property parsed from expr
346.    template<typename Property = term_no_property, char Inverter = '^'>
347.    logical_term<Property>
348.            parse_logical_term(const string &expr, int bitsize, char const
first_char = 'A') {
349.            logical_term<Property> term(bitsize);
350.            for (auto it = expr.begin(); it != expr.end(); ++it) {
351.                    bool value = true;
352.                    if (*it == Inverter) {
353.                            ++it; value = false;
354.                    }
355.                    term[*it - first_char] = value;
356.            }
357.            return term;
358.    }
359.
360.    //
361.    // Setter and getter functions of term property
362.    //
363.    template<typename Property>
364.    typename logical_term<Property>::property_type::value_type
365.            property_get(const logical_term<Property>& term)
```

```
366.    {
367.            return term.property_.get();
368.    }
369.
370.    template<typename Property>
371.    void property_set(logical_term<Property>& term,
372.            const typename logical_term<Property>::property_type::value_type
&arg)
373.    {
374.            term.property_.set(arg);
375.    }
376.
377.
378.    //
379.    // Minimize the different 1bit of term a and b
380.    //
381.    template<typename Property>
382.    logical_term<Property> onebit_minimize(const logical_term<Property> &a,
const logical_term<Property> &b)
383.    {
384.            if (a.size() != b.size() || 1 < a.diff_size(b))
385.                    throw std::runtime_error("tried to minimize a term which
has more than 1bit different bits");
386.            logical_term<Property> term(a);
387.            for (int i = 0; i < term.size(); ++i)
388.                    if (term[i] != b[i])
389.                            term[i] = dont_care;
390.            return std::move(term);
391.    }
392.
393.    // Return minimized term which has pval as its property value
394.    template<typename Property>
395.    logical_term<Property> onebit_minimize(
396.            const logical_term<Property> &a,
397.            const logical_term<Property> &b,
398.            const typename logical_term<Property>::property_type::value_type
&pval
399.    )
400.    {
401.            logical_term<Property> term = onebit_minimize(a, b);
402.            property_set(term, pval);
403.            return std::move(term);
404.    }
405.
406.
407.    //
408.    // class: logical function
409.    //
410.    template<typename TermType>
411.    class logical_function {
412.    public:
413.            typedef std::size_t size_t;
414.            typedef TermType value_type;
415.            typedef boost::dynamic_bitset<> arg_type;
416.            typedef typename vector<value_type>::iterator iterator;
417.            typedef typename vector<value_type>::const_iterator
const_iterator;
418.            typedef logical_function<TermType> this_type;
419.
420.            logical_function() {}
421.            explicit logical_function(const TermType &term) { add(term); }
422.            ~logical_function() {}
423.
```

```
424.            iterator begin() { return func_.begin(); }
425.            const_iterator begin() const { return func_.begin(); }
426.            iterator end() { return func_.end(); }
427.            const_iterator end()   const { return func_.end(); }
428.
429.            void swap(logical_function<TermType> &func) noexcept(true)
430.            {
431.                    func_.swap(func.func_);
432.            }
433.
434.            int size() const
435.            {
436.                    return func_.size();
437.            }
438.            size_t term_size() const {
439.                    if (func_.empty()) return 0;
440.                    return func_[0].size();
441.            }
442.            void add(const TermType &term)
443.            {
444.                    func_.push_back(term);
445.            }
446.
447.            void add(const this_type &func) {
448.                    vector<TermType> tmp(func_);
449.                    for (auto term : func)
450.                            tmp.push_back(term);
451.                    func_ = std::move(tmp);
452.            }
453.
454.            void clear()
455.            {
456.                    func_.clear();
457.            }
458.
459.            bool calculate(const arg_type &arg) const {
460.                    bool ret = false;
461.                    for (auto term : func_)
462.                            ret = ret || term(arg);
463.                    return ret;
464.            }
465.
466.            bool is_same(const this_type &func) const {
467.                    if (size() != func.size())
468.                            return false;
469.                    for (const value_type &term : func_)
470.                            if (std::find_if(func.begin(), func.end(),
471.                                    [&](const value_type &t) { return
t.is_same(term); })
472.                                    == func.end())
473.                                    return false;
474.                    return true;
475.            }
476.
477.            bool operator()(const arg_type &arg) const
478.            {
479.                    return calculate(arg);
480.            }
481.            value_type& operator[](int index)
482.            {
483.                    return func_[index];
484.            }
485.            const value_type& operator[](int index) const
```

```
486.            {
487.                    return func_[index];
488.            }
489.
490.            // The expression like "term + term = func" is not allowed
491.            const logical_function operator+(const TermType &term) {
492.                    logical_function ret(*this);
493.                    ret += term;
494.                    return ret;
495.            }
496.
497.            const logical_function operator+(const logical_function<TermType>
&func) {
498.                    logical_function ret(*this);
499.                    ret += func;
500.                    return ret;
501.            }
502.
503.            logical_function& operator+=(const TermType &term)
504.            {
505.                    add(term); return *this;
506.            }
507.            logical_function& operator+=(const logical_function &func)
508.            {
509.                    add(func); return *this;
510.            }
511.            friend ostream& operator<<(ostream &os, const logical_function
&bf) {
512.                    for (auto term : bf.func_)
513.                            os << term << " ";
514.                    return os;
515.            }
516.
517.            template<typename Property>
518.            bool operator==(const logical_function<logical_term<Property>>
&func) const {
519.                    arg_generator<> gen(0, std::pow(2, func.term_size()),
func.term_size());
520.                    for (auto it = gen.begin(); it != gen.end(); ++it)
521.                            if (calculate(*it) != func.calculate(*it))
522.                                    return false;
523.                    return true;
524.            }
525.
526.    private:
527.            vector<TermType> func_;
528.    };
529.
530.
531. }   // namespace logical_expr
532.
533.
534. template<typename Property>
535. logical_expr::logical_function<logical_expr::logical_term<Property>>
operator+
536. (const logical_expr::logical_term<Property> &first, const
logical_expr::logical_term<Property> &second) {
537.    logical_expr::logical_function<logical_expr::logical_term<Property>>
ret(first);
538.    ret += second;
539.    return ret;
540. }
541.
```

```
542. #endif  // LOGICAL_EXPRESSION_HPP
```

ДОДАТОК Д.14 Заголовочний файл класу, який описує вузол *MGC*

mgc_element.h

```
1. #ifndef MGC_ELEMENT_H
2. #define MGC_ELEMENT_H
3.
4. class MGC
5. {
6. public:
7.      MGC();
8.      void MGCSetFull();
9.      int returnMGCempty();
10.     void AddC(int p);
11.     void AddA(int p);
12.     void AddB(int p);
13.     void AddS(int p);
14.     void Addc(MGC *p);
15.     void Adda(MGC *p);
16.     void Addb(MGC *p);
17.     void Adds(MGC *p);
18.     void AddCbus(int p);
19.     void AddAbus(int p);
20.     void AddBbus(int p);
21.     void AddSbus(int p);
22.     int ReturnC();
23.     int ReturnA();
24.     int ReturnB();
25.     int ReturnS();
26.     int ReturnCbus();
27.     int ReturnAbus();
28.     int ReturnBbus();
29.     int ReturnSbus();
30. private:
31.     bool elem;
32.     int A, B, C, S;
33.     MGC *a, *b, *c, *s;
34.     int Abus, Bbus, Cbus, Sbus;
35. };
36.
37. #endif // MGC_ELEMENT_H
```

ДОДАТОК Д.15 Файл опису класу, який описує вузол *MGC*

mgc_element.cpp

```
1. #include "mgc_element.h"
2. #include <iostream>
3.
4. MGC::MGC()
5. {
6.      elem = false;
7.      a = b = c = s = NULL;
8.      A = B = C = S = -1;
9.      Abus = Bbus = Cbus = Sbus = -1;
10. }
11.
12. int MGC::returnMGCempty()
13. {
14.     return (int)elem;
15. }
16.
```

```
17. void MGC::MGCSetFull()
18. {
19.     elem = true;
20. }
21.
22. void MGC::AddC(int p)
23. {
24.     C = p;
25. }
26.
27. void MGC::AddA(int p)
28. {
29.     A = p;
30. }
31.
32. void MGC::AddB(int p)
33. {
34.     B = p;
35. }
36.
37. void MGC::AddS(int p)
38. {
39.     S = p;
40. }
41.
42. void MGC::Adda(MGC *p)
43. {
44.     a = p;
45. }
46.
47. void MGC::Addb(MGC *p)
48. {
49.     b = p;
50. }
51.
52. void MGC::Addc(MGC *p)
53. {
54.     c = p;
55. }
56.
57. void MGC::Adds(MGC *p)
58. {
59.     s = p;
60. }
61.
62. void MGC::AddAbus(int p)
63. {
64.     Abus = p;
65. }
66.
67. void MGC::AddBbus(int p)
68. {
69.     Bbus = p;
70. }
71.
72. void MGC::AddCbus(int p)
73. {
74.     Cbus = p;
75. }
76.
77. void MGC::AddSbus(int p)
78. {
79.     Sbus = p;
```

```
80. }
81.
82. int MGC::ReturnA()
83. {
84.     return A;
85. }
86.
87. int MGC::ReturnB()
88. {
89.     return B;
90. }
91.
92. int MGC::ReturnC()
93. {
94.     return C;
95. }
96.
97. int MGC::ReturnS()
98. {
99.     return S;
100. }
101.
102. int MGC::ReturnAbus()
103. {
104.    return Abus;
105. }
106.
107. int MGC::ReturnBbus()
108. {
109.    return Bbus;
110. }
111.
112. int MGC::ReturnCbus()
113. {
114.    return Cbus;
115. }
116.
117. int MGC::ReturnSbus()
118. {
119.    return Sbus;
120. }
```

ДОДАТОК Д.16 Заголовочний файл класу, який мінімізує булеві функції методом Квайна–Мак-Класкі–Петрика

quine_mccluskey.h

```
1. #ifndef QUINE_MCCLUSKEY_HPP
2. #define QUINE_MCCLUSKEY_HPP
3. #include <iostream>
4. #include <set>
5. #include <algorithm>
6. #include <stdexcept>
7. #include <cmath>
8. #include "logical_expr.h"
9.
10.
11. namespace quine_mccluskey {
12.
13.     using namespace std;
14.     using namespace logical_expr;
15.
16.     //
```

```
17.     // logical function simplifier
18.     //
19.     // How to simplify:
20.     //  [*] In the case which simplifier is constructed with
logical_function
21.     //      In this case, constructor will prepare to simplify a function
22.     //      1. compress_table()     // Compress the compression table
23.     //      2. simplify()           // simplify the function and get
simplified
24.     //  [*] In the case which simplifier is default-constructed
25.     //      1. set_function()       // set a target function
26.     //      2. make_std_spf()       // make a standard sum of products form
27.     //      3. make_min_table()     // create a compression table
28.     //      4. same as the case above
29.     //
30.     class simplifier {
31.     public:
32.
33.             typedef term_mark property_type;
34.             typedef logical_term<property_type> term_type;
35.             typedef vector<term_type> set_type;
36.             typedef vector<set_type> table_type;
37.
38.             simplifier() : min_level_(0)
39.             {
40.                     add_table(table_type()); make_min_table();
41.             }
42.             explicit simplifier(const logical_function<term_type> &function)
: min_level_(0), func_(function)
43.             {
44.                     add_table(table_type()); make_std_spf();
make_min_table();
45.             }
46.             ~simplifier() {}
47.
48.             void set_function(const logical_function<term_type> &func) {
func_ = func; }
49.             int get_current_level() const { return min_level_; }
50.             const logical_function<term_type>& get_std_spf() const { return
stdspf_; }
51.             const set_type& get_prime_implicants() const { return prime_imp;
}
52.
53.             // Make standard sum of products form
54.             const logical_function<term_type>& make_std_spf();
55.             const table_type& make_min_table();
56.             void compress_table(bool printable = false);
57.             const vector<logical_function<term_type>>& simplify();
58.
59.     private:
60.             void add_table(const table_type& table);
61.             void clear_table();
62.             template<typename T>
63.             static void make_unique(vector<T> &vec);
64.
65.             // compress compression table
66.             // return true while trying to compress
67.             // return false if compression finished
68.             bool compress_impl(bool printable = false);
69.
70.             int min_level_;
71.             logical_function<term_type> func_, stdspf_;
72.             vector<logical_function<term_type>> simplified_;
```

```
73.             vector<table_type> table_;
74.             set_type prime_imp;
75.     };
76.
77.
78.
79. }   // namespace quine_mccluskey
80.
81.
82. #endif  // QUINE_MCCLUSKEY_HPP
```

ДОДАТОК Д.17 Файл опису класу, який мінімізує булеві функції методом Квайна–Мак-Класкі–Петрика

quine_mccluskey.cpp

```
1. #include <iostream>
2. #include <set>
3. #include <algorithm>
4. #include <stdexcept>
5. #include <numeric>
6. #include <cmath>
7. #include "logical_expr.h"
8. #include "quine_mccluskey.h"
9. #include "windows.h"
10.
11. using namespace std;
12. using namespace logical_expr;
13.
14. namespace quine_mccluskey {
15.
16.
17.     typedef simplifier::property_type property_type;
18.     typedef simplifier::term_type term_type;
19.     typedef simplifier::set_type set_type;
20.     typedef simplifier::table_type table_type;
21.
22.     // Make standard sum of products form
23.     const logical_function<term_type>& simplifier::make_std_spf() {
24.             stdspf_.clear();
25.             arg_generator<> generator(0, std::pow(2, func_.term_size()),
func_.term_size());
26.             for (auto arg : generator)
27.                     if (func_(arg))
28.                             stdspf_ += logical_term<term_mark>(arg);
29.             return stdspf_;
30.     }
31.
32.     const table_type& simplifier::make_min_table() {
33.             table_[0].resize(func_.term_size() + 1, set_type());
34.             for (auto term : stdspf_)
35.                     table_[0][term.num_of_value(true)].push_back(term);
36.             return table_[0];
37.     }
38.
39.     void simplifier::compress_table(bool printable) {
40.             for (;; ) {
41.                     if (printable)
42.                             //cout << get_current_level() + 1 << "-level
compression:" << endl;
43.                     if (!compress_impl(printable)) break;
44.             }
45.             for (auto table : table_)
```

```
46.                     for (auto set : table)
47.                             for (const logical_term<term_mark> &term : set)
48.                                     if (!property_get(term))
49.                                             prime_imp.push_back(term);
50.             make_unique(prime_imp);
51.     }
52.
53.     const vector<logical_function<term_type>>& simplifier::simplify() {
54.             vector<int> next_index(prime_imp.size());
55.             std::iota(next_index.begin(), next_index.end(), 0);
56.             // bool: wether simplifying finished  int: loop number that
simplifying took
57.             std::pair<bool, int> end_flags = { false, 0 };
58.             for (int i = 1; i <= prime_imp.size(); ++i) {
59.                     if (end_flags.first && end_flags.second < i)
60.                             break;
61.                     do {
62.                             logical_function<term_type> func;
63.                             auto log_func_comp
64.                                     = [&](const logical_function<term_type>
&f) { return f.is_same(func); };
65.                             for (int j = 0; j < i; ++j)
66.                                     func += prime_imp[next_index[j]];
67.                             if (func == stdspf_ &&
68.                                     std::find_if(simplified_.begin(),
simplified_.end(), log_func_comp)
69.                                     == simplified_.end()) {
70.                                     simplified_.push_back(func);
71.                                     end_flags = { true, i };
72.                             }
73.                     } while (next_permutation(next_index.begin(),
next_index.end()));
74.             }
75.             return simplified_;
76.     }
77.
78.     void simplifier::add_table(const table_type& table) {
79.             table_.push_back(table);
80.     }
81.
82.     void simplifier::clear_table() {
83.             for (int i = 0; i < table_.size(); ++i)
84.                     table_[i].clear();
85.     }
86.
87.     template<typename T>
88.     void simplifier::make_unique(vector<T> &vec) {
89.             for (auto it = vec.begin(); it != vec.end(); ++it) {
90.                     auto rm_it = remove(it + 1, vec.end(), *it);
91.                     vec.erase(rm_it, vec.end());
92.             }
93.     }
94.
95.     // Try to find prime implicants
96.     // Return true while trying to find them
97.     // Return false if it finished
98.     bool simplifier::compress_impl(bool printable) {
99.             int number_cout;
100.            table_type next_table;
101.            next_table.resize(func_.term_size(), set_type());
102.            int count = 0;
103.            for (int i = 0; i + 1 < table_[min_level_].size(); ++i) {
104.                    for (int j = 0; j < table_[min_level_][i].size(); ++j) {
```

```
105.                            for (int k = 0; k < table_[min_level_][i +
1].size(); ++k) {
106.                                    try {
107.                                            // Throw an exception if could not
minimize
108.                                            auto term =
onebit_minimize(table_[min_level_][i][j], table_[min_level_][i + 1][k], false);
109.                                            if (printable)
110.                                                    //cout << "COMPRESS(" <<
table_[min_level_][i][j] << ", " << table_[min_level_][i + 1][k] << ") = " <<
term << endl;
111.                                            if
(std::find_if(next_table[term.num_of_value(true)].begin(),
next_table[term.num_of_value(true)].end(),
112.                                                    [&](const term_type &t) {
return t.is_same(term); }) == next_table[term.num_of_value(true)].end())
113.
        next_table[term.num_of_value(true)].push_back(term);
114.                                            ++count;
115.                                            // Mark the used term for
minimization
116.
        property_set(table_[min_level_][i][j], true);
117.                                            property_set(table_[min_level_][i
+ 1][k], true);
118.                                    }
119.                                    catch (std::exception &e) {}
120.                            }
121.                    }
122.            }
123.            if (count) {
124.                    ++min_level_;
125.                    add_table(next_table);
126.            }
127.          //gotoxy(0, 16);
128.            for (int x = 0; x < 12; x++)
129.            {
130.                    std::cout << char(221);
131.            }
132.          //gotoxy(0, 16);
133.            for (int x = 0; x < 12; x++)
134.            {
135.                    std::cout << " ";
136.            }
137.            return (count ? true : false);
138.    }
139.
140.
141. }   // namespace quine_mccluskey
```

ДОДАТОК Д.18 Заголовочний файл, у якому містяться функції для мінімізації булевих функцій методом Квайна–Мак-Класкі–Петрика

transformation_and_minimization.h

```
1. #ifndef TRANSFORMATION_AND_MINIMIZATION
2. #define TRANSFORMATION_AND_MINIMIZATION
3. #include <fstream>
4. #include <string>
5. #include <math.h>
6. #include <conio.h>
7. #include <iostream>
8. #include <string>
```

```
9. #include <utility>
10. #include <stdexcept>
11. #include <cmath>
12. #include <cstdlib>
13. #include <boost/program_options.hpp>
14. #include "logical_expr.h"
15. #include "quine_mccluskey.h"
16. #include<vector>
17.
18.
19. using namespace std;
20.
21. namespace trans
22. {
23.     void gotoxy(int column, int line)
24.     {
25.             COORD coord;
26.             coord.X = column;
27.             coord.Y = line;
28.             SetConsoleCursorPosition(
29.                     GetStdHandle(STD_OUTPUT_HANDLE),
30.                     coord
31.             );
32.     }
33.
34.     //using namespace std;
35.     template <typename Property>
36.     void print_term_expr(const logical_expr::logical_term<Property> &term,
ofstream & fout, char first_char = 'A',  char inverter = '~')
37.     {
38.             for (int i = 0; i < term.size(); ++i) {
39.                     if (term[i] == false)
40.                     {
41.                             //cout << inverter;
42.                             fout << inverter;
43.                     }
44.                     if (term[i] != logical_expr::dont_care)
45.                     {
46.                             fout << static_cast<char>(first_char + i);
47.                     }
48.             }
49.             fout << '+';
50.     }
51.
52.     template<typename TermType>
53.     void print_func_expr(const logical_expr::logical_function<TermType>
&func, char first_char = 'A', const string &funcname = "f", char inverter = '~')
54.     {
55.             cout << funcname << " = ";
56.             for (auto it = func.begin(); it != func.end(); ++it) {
57.                     print_term_expr(*it, first_char, inverter);
58.                     if (it + 1 != func.end())
59.                             std::cout << " + ";
60.             }
61.             gotoxy(0, 16);
62.             for (int x = 0; x < 12; x++)
63.             {
64.                     std::cout << char(221);
65.             }
66.             gotoxy(0, 16);
67.             for (int x = 0; x < 12; x++)
68.             {
69.                     std::cout << " ";
```

```
70.             }
71.     }
72.
73.     template<typename TermType>
74.     void print_truth_table(const logical_expr::logical_function<TermType>
&f, char first_char = 'A', const string &funcname = "f")
75.     {
76.             std::cout << "Truth Table: ";
77.             print_func_expr(f, first_char, funcname);
78.             for (char c = first_char; c != first_char + f.term_size(); ++c)
79.                     cout << c;
80.             std::cout << " | " << funcname << "()" << endl;
81.             for (int i = 0; i < f.term_size() + 6; ++i)
82.                     std::cout << ((i == f.term_size() + 1) ? '|' : '-');
83.             std::cout << endl;
84.             logical_expr::arg_generator<> generator(0, std::pow(2,
f.term_size()), f.term_size());
85.             for (auto arg : generator)
86.                     std::cout << arg << " |  " << f(arg) << endl;
87.             gotoxy(0, 16);
88.             for (int x = 0; x < 12; x++)
89.             {
90.                     std::cout << char(221);
91.             }
92.             gotoxy(0, 16);
93.             for (int x = 0; x < 12; x++)
94.             {
95.                     std::cout << " ";
96.             }
97.     }
98.
99.
100.    string *transFuncToMinimizeMGC(int base)
101.    {
102.            const int Count_zminna = ceil((log(base)) / (log(2)));
103.
104.            std::ifstream *fout_Q = new std::ifstream[Count_zminna];
105.            for (int i = 0; i < Count_zminna; i++)
106.            {
107.                    std::string str4 = "D://Resourse_Q" + std::to_string(i) +
".txt";
108.                    std::string str5 = str4;
109.                    fout_Q[i].open(str5, std::ios::in | std::ios::binary |
std::ios::ate);
110.            }
111.            int *filesize = new int[Count_zminna];
112.            char **A = new char *[Count_zminna];
113.            for (int i = 0; i < Count_zminna; i++)
114.            {
115.                    filesize[i] = fout_Q[i].tellg();
116.                    A[i] = new char[filesize[i]];
117.                    fout_Q[i].clear();
118.                    fout_Q[i].seekg(0, ios::beg);
119.            }
120.            for (int i = 0; i < Count_zminna; i++)
121.            {
122.              fout_Q[i].getline(A[i], filesize[i] + 1);
123.            }
124.            for (int j = 0; j < Count_zminna; j++)
125.            {
126.                    for (int i = 0; i < filesize[j] + 1; i++)
127.                    {
128.                            if ((A[j][i] == ')') || (A[j][i] == '('))
```

```
129.                            {
130.                                    A[j][i] = ' ';
131.                            }
132.                            if (A[j][i] == 'n')
133.                            {
134.                                    A[j][i] = '~';
135.                                    A[j][i + 1] = ' ';
136.                                    A[j][i + 2] = ' ';
137.                            }
138.                            if (A[j][i] == 'a')
139.                            {
140.                                    A[j][i] = ' ';
141.                                    A[j][i + 1] = ' ';
142.                                    A[j][i + 2] = ' ';
143.                            }
144.                            if (A[j][i] == 'o')
145.                            {
146.                                    A[j][i] = '+';
147.                                    A[j][i + 1] = ' ';
148.                            }
149.                    }
150.            }
151.            //string str6 = string(A[0]);
152.            int *filesizeB = new int[Count_zminna];
153.            for (int j = 0; j < Count_zminna; j++)
154.            {
155.                    filesizeB[j] = 0;
156.                    for (int i = 0; i < filesize[j] + 1; i++)
157.                    {
158.                            if (A[j][i] != ' ')
159.                            {
160.                                    filesizeB[j]++;
161.                            }
162.                    }
163.            }
164.
165.            char **B = new char *[Count_zminna];
166.            for (int i = 0; i < Count_zminna; i++)
167.            {
168.                    B[i] = new char[filesizeB[i] + (Count_zminna * 3) * 2 +
3];
169.            }
170.
171.            for (int j = 0; j < Count_zminna; j++)
172.            {
173.                    int p = (Count_zminna * 3) * 2 + 3;
174.                    for (int i = 0; i < filesize[j] + 1; i++)
175.                    {
176.                            if (A[j][i] != ' ')
177.                            {
178.                                    B[j][p] = A[j][i];
179.                                    p++;
180.                            }
181.                            //p++;
182.                    }
183.            }
184.
185.            for (int i = 0; i < Count_zminna; i++)
186.            {
187.                    //cout << A[i] << endl;
188.                    //cout << B[i] << endl;
189.            }
190.
```

```
191.            for (int j = 0; j < Count_zminna; j++)
192.            {
193.                    for (int i = (Count_zminna * 3) * 2 + 3; i < filesizeB[j]
+ (Count_zminna * 3) * 2 + 3; i++)
194.                    {
195.                            if (B[j][i] == 'A')
196.                            {
197.                                    B[j][i] = 'A' + Count_zminna - 1 - (B[j][i
+ 1] - 48);
198.                                    B[j][i + 1] = ' ';
199.                                    continue;
200.                            }
201.                            if (B[j][i] == 'B')
202.                            {
203.                                    B[j][i] = 'A' + 2 * Count_zminna - 1 -
(B[j][i + 1] - 48);
204.                                    B[j][i + 1] = ' ';
205.                                    continue;
206.                            }
207.                            if (B[j][i] == 'C')
208.                            {
209.                                    B[j][i] = 'A' + 3 * Count_zminna - 1 -
(B[j][i + 1] - 48);
210.                                    B[j][i + 1] = ' ';
211.                                    continue;
212.                            }
213.                    }
214.            }
215.            for (int j = 0; j < Count_zminna; j++)
216.            {B[j][0] = 'f';
217.
218.                    B[j][1] = '(';
219.                    B[j][(Count_zminna * 3) * 2 + 3 - 1] = '=';
220.                    B[j][(Count_zminna * 3) * 2 + 3 - 2] = ')';
221.            }
222.            for (int j = 0; j < Count_zminna; j++)
223.            {
224.                    int k = 0;
225.                    for (int i = 2; i <  (Count_zminna * 3) * 2 + 3 - 2; i++)
226.                    {
227.                            if (i % 2 == 0)
228.                            {
229.                                    B[j][i] = 'A' + k;
230.                                    k++;
231.                            }
232.                            else
233.                            {
234.                                    B[j][i] = ',';
235.                            }
236.                    }
237.            }
238.
239.            string *arr = new string[Count_zminna];
240.            for (int i = 0; i < Count_zminna; i++)
241.            {
242.                    arr[i] = B[i];
243.            }
244.
245.            for (int i = 0; i < Count_zminna; i++)
246.            {
247.                    //cout << A[i] << endl;
248.                    //cout << B[i] << endl;
249.              //fout_Q[i].close();
```

```
250.            }
251.            //string ** s =
252.
253.            return arr;
254.    }
255.
256.    void convertminimizedFunctionMGC( int basem)
257.    {
258.            const int Count_zminna = ceil((log(basem)) / (log(2)));
259.            std::ifstream *fout_Q = new std::ifstream[Count_zminna];
260.            for (int i = 0; i < Count_zminna; i++)
261.            {
262.                    std::string str4 = "D://Resourse_min_Q" +
std::to_string(i) + ".txt";
263.                    std::string str5 = str4;
264.                    fout_Q[i].open(str5, std::ios::in | std::ios::binary |
std::ios::ate);
265.            }
266.            int *filesize = new int[Count_zminna];
267.            char **A = new char *[Count_zminna];
268.            for (int i = 0; i < Count_zminna; i++)
269.            {
270.                    filesize[i] = fout_Q[i].tellg();
271.                    A[i] = new char[filesize[i]];
272.                    fout_Q[i].clear();
273.                    fout_Q[i].seekg(0, ios::beg);
274.            }
275.            for (int i = 0; i < Count_zminna; i++)
276.            {
277.              //if (!fout_Q[i]) return 1;
278.                    //char *mas = A[i];
279.              fout_Q[i].getline(A[i], filesize[i] + 1);
280.                    A[i][filesize[i] - 1] = ' ';
281.                    //cout << A[i] << endl;
282.            }
283.
284.            char **B = new char *[Count_zminna];
285.            for (int i = 0; i < Count_zminna; i++)
286.            {
287.                    B[i] = new char[filesize[i] * 10];
288.            }
289.
290.            //std::vector<std::vector<char>> B(Count_zminna);
291.            //for (int i = 0; i < Count_zminna; i++)
292.            //      B[i].resize(filesize[i] * 6);
293.
294.
295.            for (int j = 0; j < Count_zminna; j++)
296.            {
297.                    for (int i = 0; i < (filesize[j]) * 10; i++)
298.                    {
299.                            B[j][i] = NULL;
300.                    }
301.                    B[j][filesize[j] * 10 - 1] = NULL;
302.            }
303.
304.            for (int j = 0; j < Count_zminna; j++)
305.            {
306.                    if (Count_zminna == 1)
307.                    {
308.                            int p = 0;
309.                            B[j][p] = '(';
310.                            p++;
```

```
311.                            for (int i = 0; i < filesize[j] - 1; i++)
312.                            {
313.                                    if ((A[j][i] != '~') && (A[j][i] != '+'))
314.                                    {
315.                                            char k = A[j][i] - 'A';
316.                                            char ABC = k / Count_zminna + 1;
317.                                            char chislo = Count_zminna - 1 -
(k % Count_zminna);
318.                                            if (ABC == 1)
319.                                            {
320.                                                    B[j][p] = 'A';
321.                                                    p++;
322.                                            }
323.                                            else if (ABC == 2)
324.                                            {
325.                                                    B[j][p] = 'B';
326.                                                    p++;
327.                                            }
328.                                            else if (ABC == 3)
329.                                            {
330.                                                    B[j][p] = 'C';
331.                                                    p++;
332.                                            }
333.                                            if (((ABC == 3) || (ABC == 2)) &&
((A[j][i + 1] == '+') || ((i + 2) == filesize[j])))
334.                                            {
335.                                                    B[j][p] = ')';
336.                                                    p++;
337.                                                    B[j][p] = ' ';
338.                                                    p++;
339.                                            }
340.                                            else
341.                                            {
342.                                                    B[j][p] = ' ';
343.                                                    p++;
344.                                                    B[j][p] = 'a';
345.                                                    p++;
346.                                                    B[j][p] = 'n';
347.                                                    p++;
348.                                                    B[j][p] = 'd';
349.                                                    p++;
350.                                                    B[j][p] = ' ';
351.                                                    p++;
352.                                            }
353.                                    }
354.                                    else if (A[j][i] == '~')
355.                                    {
356.                                            B[j][p] = 'n';
357.                                            p++;
358.                                            B[j][p] = 'o';
359.                                            p++;
360.                                            B[j][p] = 't';
361.                                            p++;
362.                                            B[j][p] = ' ';
363.                                            p++;
364.                                    }
365.                                    else
366.                                    {
367.                                            B[j][p] = 'o';
368.                                            p++;
369.                                            B[j][p] = 'r';
370.                                            p++;
371.                                            B[j][p] = ' ';
```

```
372.                                            p++;
373.                                            B[j][p] = '(';
374.                                            p++;
375.                                    }
376.                            }
377.                    }
378.                    else
379.                    {
380.                    int p = 0;
381.                    B[j][p] = '(';
382.                    p++;
383.                    for (int i = 0; i < filesize[j] - 1; i++)
384.                    {
385.                            if ((A[j][i] != '~') && (A[j][i] != '+'))
386.                            {
387.                                    char k = A[j][i] - 'A';
388.                                    char ABC = k / Count_zminna + 1;
389.                                    char chislo = Count_zminna - 1 - (k %
Count_zminna);
390.                                    if (ABC == 1)
391.                                    {
392.                                            B[j][p] = 'A';
393.                                            p++;
394.                                    }
395.                                    else if (ABC == 2)
396.                                    {
397.                                            B[j][p] = 'B';
398.                                            p++;
399.                                    }
400.                                    else if (ABC == 3)
401.                                    {
402.                                            B[j][p] = 'C';
403.                                            p++;
404.                                    }
405.                                    if ((ABC == 3) && ((chislo == 0) ||
(A[j][i + 1] == '+') || ((i + 2) == filesize[j])))
406.                                    {
407.                                            B[j][p] = '(';
408.                                            p++;
409.                                            B[j][p] = 48 + chislo;
410.                                            p++;
411.                                            B[j][p] = ')';
412.                                            p++;
413.                                    }
414.                                    else
415.                                    {
416.                                            B[j][p] = '(';
417.                                            p++;
418.                                            B[j][p] = 48 + chislo;
419.                                            p++;
420.                                            B[j][p] = ')';
421.                                            p++;
422.                                            B[j][p] = ' ';
423.                                            p++;
424.                                            B[j][p] = 'a';
425.                                            p++;
426.                                            B[j][p] = 'n';
427.                                            p++;
428.                                            B[j][p] = 'd';
429.                                            p++;
430.                                            B[j][p] = ' ';
431.                                            p++;
432.                                    }
```

```
433.                                    if ((ABC == 3) && ((chislo == 0) ||
(A[j][i + 1] == '+') || ((i + 2) == filesize[j])))
434.                                    {
435.                                            B[j][p] = ')';
436.                                            p++;
437.                                            B[j][p] = ' ';
438.                                            p++;
439.                                    }
440.                            }
441.                            else if (A[j][i] == '~')
442.                            {
443.                                    B[j][p] = 'n';
444.                                    p++;
445.                                    B[j][p] = 'o';
446.                                    p++;
447.                                    B[j][p] = 't';
448.                                    p++;
449.                                    B[j][p] = ' ';
450.                                    p++;
451.                            }
452.                            else
453.                            {
454.                                    B[j][p] = 'o';
455.                                    p++;
456.                                    B[j][p] = 'r';
457.                                    p++;
458.                                    B[j][p] = ' ';
459.                                    p++;
460.                                    B[j][p] = '(';
461.                                    p++;
462.                            }
463.                    }
464.              }
465.            }
466.            std::ofstream *fout = new std::ofstream[Count_zminna];
467.            for (int i = 0; i < Count_zminna; i++)
468.            {
469.                    std::string str4 = "D://Resourse_min_Q" +
std::to_string(i) + ".txt";
470.                    std::string str5 = str4;
471.                    fout[i].open(str5, std::ios::out | std::ios::binary |
std::ios::ate);
472.            }
473.            for (int i = 0; i < Count_zminna; i++)
474.            {
475.                    fout[i] << B[i];
476.            }
477.            for (int i = 0; i < Count_zminna; i++)
478.            {
479.                    fout[i].close();
480.            }
481.            //for (int i = 0; i < Count_zminna; i++)
482.            //{
483.            //      cout << B[i] << endl;
484.            //}
485.    }
486.
487.
488.
489.    int QM_MGC(int basem)
490.    {
491.            int base = basem;
492.            const int Count_zminna = ceil((log(base)) / (log(2)));
```

```
493.            int arg1 = NULL;
494.            char **arg2 = NULL;
495.            int exit_code = EXIT_SUCCESS;
496.
497.            std::ofstream *fout_Q = new std::ofstream[Count_zminna];
498.            for (int i = 0; i < Count_zminna; i++)
499.            {
500.                    std::string str4 = "D://Resourse_min_Q" +
std::to_string(i) + ".txt";
501.                    std::string str5 = str4;
502.                    fout_Q[i].open(str5, std::ios::out | std::ios::binary );
503.            }
504.
505.            for (int i = 0; i < Count_zminna; i++)
506.            {
507.                    try {
508.                            bool print_process = true;
509.                            char first_char = 'A';
510.                            constexpr char inverter = '~';
511.
512.                            if (print_process)
513.                                    //std::cout << "Logical Function
Simplifier (Quine-McCluskey)" << std::endl
514.                                    //<< "[*] Enter a logical function to be
simplified" << std::endl
515.                                    //<< "    (ex. \"f(A, B, C) = A + BC +
~A~B + ABC\" )" << std::endl
516.                                    //<< "[*] Input: "
517.                            cout<< flush;
518.                            std::string *line = transFuncToMinimizeMGC(base);
519.                            logical_expr::function_parser<inverter, true>
parser(line[i], first_char);
520.                            auto token = parser.parse();
521.                            // Create a logical function with
logical_term<term_mark>
522.                            typedef
quine_mccluskey::simplifier::property_type PropertyType;
523.                            typedef quine_mccluskey::simplifier::term_type
TermType;
524.                            logical_expr::logical_function<TermType>
function;
525.                            for (std::string term : token.second)
526.                                    function +=
logical_expr::parse_logical_term<PropertyType, inverter>(term,
token.first.size(), first_char);
527.
528.                            // Create a simplifier using Quine-McCluskey
algorithm
529.                            quine_mccluskey::simplifier qm(function);
530.                            if (print_process) {
531.                                    //std::cout << std::endl << "Sum of
products form:" << std::endl;
532.                                    //print_truth_table(qm.get_std_spf(),
first_char);    // Print the function in sum of products form
533.                                    //std::cout << std::endl << "Compressing
..." << std::endl;
534.                                    qm.compress_table(true);
// Compress the compression table
535.                                    //std::cout << std::endl << "Prime
implicants: " << std::endl;
536.                                    for (const auto &term :
qm.get_prime_implicants()) {      // Print the prime implicants
```

```
537.                                            print_term_expr(term, fout_Q[i],
first_char);
538.                                            std::cout << "  ";
539.                                    }
540.                                    //std::cout << std::endl << std::endl <<
"Result of simplifying:" << std::endl;
541.                            }
542.                            else
543.                                    qm.compress_table(false);
544.
545.                            //for (const auto &func : qm.simplify())
// Simplify and print its results
546.                                    //print_func_expr(func, first_char,
parser.function_name() + "\'");
547.                    }
548.                    catch (std::exception &e) {
549.                            std::cerr << std::endl << "[-] Exception: " <<
e.what() << std::endl;
550.                            exit_code = EXIT_FAILURE;
551.                    }
552.            }
553.            for (int i = 0; i < Count_zminna; i++)
554.            {
555.                    fout_Q[i].close();
556.            }
557.            //cout << flush;
558.            convertminimizedFunctionMGC(base);
559.            gotoxy(0,15);
560.            cout << "Minimization of MGC is finished" << endl;
561.            //_getch();
562.            return exit_code;
563.    }
564.
565.    string *transFuncToMinimizeF(int base)
566.    {
567.            const int Count_zminna = ceil((log(base)) / (log(2)));
568.
569.            std::ifstream *fout_Q = new std::ifstream[Count_zminna];
570.            for (int i = 0; i < Count_zminna; i++)
571.            {
572.                    std::string str4 = "D://Resourse_F" + std::to_string(i) +
".txt";
573.                    std::string str5 = str4;
574.                    fout_Q[i].open(str5, std::ios::in | std::ios::binary |
std::ios::ate);
575.            }
576.            int *filesize = new int[Count_zminna];
577.            char **A = new char *[Count_zminna];
578.            for (int i = 0; i < Count_zminna; i++)
579.            {
580.                    filesize[i] = fout_Q[i].tellg();
581.                    A[i] = new char[filesize[i]];
582.                    fout_Q[i].clear();
583.                    fout_Q[i].seekg(0, ios::beg);
584.            }
585.            for (int i = 0; i < Count_zminna; i++)
586.            {
587.                    //if (!fout_Q[i]) return 1;
588.                    //char *mas = A[i];
589.                    fout_Q[i].getline(A[i], filesize[i] + 1);
590.            }
591.            for (int j = 0; j < Count_zminna; j++)
592.            {
```

```
593.                    for (int i = 0; i < filesize[j] + 1; i++)
594.                    {
595.                            if ((A[j][i] == ')') || (A[j][i] == '('))
596.                            {
597.                                    A[j][i] = ' ';
598.                            }
599.                            if (A[j][i] == 'n')
600.                            {
601.                                    A[j][i] = '~';
602.                                    A[j][i + 1] = ' ';
603.                                    A[j][i + 2] = ' ';
604.                            }
605.                            if (A[j][i] == 'a')
606.                            {
607.                                    A[j][i] = ' ';
608.                                    A[j][i + 1] = ' ';
609.                                    A[j][i + 2] = ' ';
610.                            }
611.                            if (A[j][i] == 'o')
612.                            {
613.                                    A[j][i] = '+';
614.                                    A[j][i + 1] = ' ';
615.                            }
616.                    }
617.            }
618.            //string str6 = string(A[0]);
619.            int *filesizeB = new int[Count_zminna];
620.            for (int j = 0; j < Count_zminna; j++)
621.            {
622.                    filesizeB[j] = 0;
623.                    for (int i = 0; i < filesize[j] + 1; i++)
624.                    {
625.                            if (A[j][i] != ' ')
626.                            {
627.                                    filesizeB[j]++;
628.                            }
629.                    }
630.            }
631.
632.            char **B = new char *[Count_zminna];
633.            for (int i = 0; i < Count_zminna; i++)
634.            {
635.                    B[i] = new char[filesizeB[i] + (Count_zminna) * 2 + 3];
636.            }
637.
638.            for (int j = 0; j < Count_zminna; j++)
639.            {
640.                    int p = (Count_zminna) * 2 + 3;
641.                    for (int i = 0; i < filesize[j] + 1; i++)
642.                    {
643.                            if (A[j][i] != ' ')
644.                            {
645.                                    B[j][p] = A[j][i];
646.                                    p++;
647.                            }
648.                            //p++;
649.                    }
650.            }
651.
652.            for (int i = 0; i < Count_zminna; i++)
653.            {
654.                    //cout << A[i] << endl;
655.                    //cout << B[i] << endl;
```

```
656.            }
657.
658.            for (int j = 0; j < Count_zminna; j++)
659.            {
660.                    for (int i = (Count_zminna) * 2 + 3; i < filesizeB[j] +
(Count_zminna) * 2 + 3; i++)
661.                    {
662.                            if (B[j][i] == 'A')
663.                            {
664.                                    B[j][i] = 'A' + Count_zminna - 1 - (B[j][i
+ 1] - 48);
665.                                    B[j][i + 1] = ' ';
666.                                    continue;
667.                            }
668.                            if (B[j][i] == 'B')
669.                            {
670.                                    B[j][i] = 'A' + 2 * Count_zminna - 1 -
(B[j][i + 1] - 48);
671.                                    B[j][i + 1] = ' ';
672.                                    continue;
673.                            }
674.                            if (B[j][i] == 'C')
675.                            {
676.                                    B[j][i] = 'A' + 3 * Count_zminna - 1 -
(B[j][i + 1] - 48);
677.                                    B[j][i + 1] = ' ';
678.                                    continue;
679.                            }
680.                    }
681.            }
682.            for (int j = 0; j < Count_zminna; j++)
683.            {
684.                    B[j][0] = 'f';
685.
686.                    B[j][1] = '(';
687.                    B[j][(Count_zminna) * 2 + 3 - 1] = '=';
688.                    B[j][(Count_zminna) * 2 + 3 - 2] = ')';
689.            }
690.            for (int j = 0; j < Count_zminna; j++)
691.            {
692.                    int k = 0;
693.                    for (int i = 2; i < (Count_zminna) * 2 + 3 - 2; i++)
694.                    {
695.                            if (i % 2 == 0)
696.                            {
697.                                    B[j][i] = 'A' + k;
698.                                    k++;
699.                            }
700.                            else
701.                            {
702.                                    B[j][i] = ',';
703.                            }
704.                    }
705.            }
706.
707.            string *arr = new string[Count_zminna];
708.            for (int i = 0; i < Count_zminna; i++)
709.            {
710.                    arr[i] = B[i];
711.            }
712.
713.            for (int i = 0; i < Count_zminna; i++)
714.            {
```

```
715.                    //cout << A[i] << endl;
716.                    //cout << B[i] << endl;
717.                    //fout_Q[i].close();
718.            }
719.            //string ** s =
720.
721.            return arr;
722.    }
723.
724.    void convertminimizedFunctionF(int basem)
725.    {
726.            const int Count_zminna = ceil((log(basem)) / (log(2)));
727.            std::ifstream *fout_Q = new std::ifstream[Count_zminna];
728.            for (int i = 0; i < Count_zminna; i++)
729.            {
730.                    std::string str4 = "D://Resourse_min_F" +
std::to_string(i) + ".txt";
731.                    std::string str5 = str4;
732.                    fout_Q[i].open(str5, std::ios::in | std::ios::binary |
std::ios::ate);
733.            }
734.            int *filesize = new int[Count_zminna];
735.            char **A = new char *[Count_zminna];
736.            for (int i = 0; i < Count_zminna; i++)
737.            {
738.                    filesize[i] = fout_Q[i].tellg();
739.                    A[i] = new char[filesize[i]];
740.                    fout_Q[i].clear();
741.                    fout_Q[i].seekg(0, ios::beg);
742.            }
743.            for (int i = 0; i < Count_zminna; i++)
744.            {
745.                    //if (!fout_Q[i]) return 1;
746.                    //char *mas = A[i];
747.                    fout_Q[i].getline(A[i], filesize[i] + 1);
748.                    A[i][filesize[i] - 1] = ' ';
749.                    //cout << A[i] << endl;
750.            }
751.
752.
753.
754.
755.            char **B = new char *[Count_zminna];
756.            for (int i = 0; i < Count_zminna; i++)
757.            {
758.                    B[i] = new char[filesize[i] * 10];
759.            }
760.
761.            //std::vector<std::vector<char>> B(Count_zminna);
762.            //for (int i = 0; i < Count_zminna; i++)
763.            //      B[i].resize(filesize[i] * 6);
764.
765.
766.            for (int j = 0; j < Count_zminna; j++)
767.            {
768.                    for (int i = 0; i < (filesize[j]) * 10; i++)
769.                    {
770.                            B[j][i] = NULL;
771.                    }
772.                    B[j][filesize[j] * 10 - 1] = NULL;
773.            }
774.
775.            for (int j = 0; j < Count_zminna; j++)
```

```
776.            {
777.                    int p = 0;
778.                    B[j][p] = '(';
779.                    p++;
780.                    for (int i = 0; i < filesize[j] - 1; i++)
781.                    {
782.                            if ((A[j][i] != '~') && (A[j][i] != '+'))
783.                            {
784.                                    char k = A[j][i] - 'A';
785.                                    char ABC = k / Count_zminna + 1;
786.                                    char chislo = Count_zminna - 1 - (k %
Count_zminna);
787.                                    if (ABC == 1)
788.                                    {
789.                                            B[j][p] = 'A';
790.                                            p++;
791.                                    }
792.                                    if ((ABC == 1) || ((i + 2) == filesize[j])
|| (A[j][i + 1] == '+'))
793.                                    {
794.                                            B[j][p] = '(';
795.                                            p++;
796.                                            B[j][p] = 48 + chislo;
797.                                            p++;
798.                                            B[j][p] = ')';
799.                                            p++;
800.                                    }
801.                                    else
802.                                    {
803.                                            B[j][p] = '(';
804.                                            p++;
805.                                            B[j][p] = 48 + chislo;
806.                                            p++;
807.                                            B[j][p] = ')';
808.                                            p++;
809.                                            B[j][p] = ' ';
810.                                            p++;
811.                                            B[j][p] = 'a';
812.                                            p++;
813.                                            B[j][p] = 'n';
814.                                            p++;
815.                                            B[j][p] = 'd';
816.                                            p++;
817.                                            B[j][p] = ' ';
818.                                            p++;
819.                                    }
820.                                    if ((ABC == 1) || (A[j][i + 1] == '+') ||
((i + 2) == filesize[j]))
821.                                    {
822.                                            B[j][p] = ')';
823.                                            p++;
824.                                            B[j][p] = ' ';
825.                                            p++;
826.                                    }
827.                            }
828.                            else if (A[j][i] == '~')
829.                            {
830.                                    B[j][p] = 'n';
831.                                    p++;
832.                                    B[j][p] = 'o';
833.                                    p++;
834.                                    B[j][p] = 't';
835.                                    p++;
```

```
836.                                    B[j][p] = ' ';
837.                                    p++;
838.                            }
839.                            else
840.                            {
841.                                    B[j][p] = 'o';
842.                                    p++;
843.                                    B[j][p] = 'r';
844.                                    p++;
845.                                    B[j][p] = ' ';
846.                                    p++;
847.                                    B[j][p] = '(';
848.                                    p++;
849.                            }
850.                    }
851.            }
852.            std::ofstream *fout = new std::ofstream[Count_zminna];
853.            for (int i = 0; i < Count_zminna; i++)
854.            {
855.                    std::string str4 = "D://Resourse_min_F" +
std::to_string(i) + ".txt";
856.                    std::string str5 = str4;
857.                    fout[i].open(str5, std::ios::out | std::ios::binary |
std::ios::ate);
858.            }
859.            for (int i = 0; i < Count_zminna; i++)
860.            {
861.                    fout[i] << B[i];
862.            }
863.            for (int i = 0; i < Count_zminna; i++)
864.            {
865.                    fout[i].close();
866.            }
867.            //for (int i = 0; i < Count_zminna; i++)
868.            //{
869.            //      cout << B[i] << endl;
870.            //}
871.    }
872.
873.
874.
875.    int QM_F(int basem)
876.    {
877.            int base = basem;
878.            const int Count_zminna = ceil((log(base)) / (log(2)));
879.            int arg1 = NULL;
880.            char **arg2 = NULL;
881.            int exit_code = EXIT_SUCCESS;
882.
883.            std::ofstream *fout_Q = new std::ofstream[Count_zminna];
884.            for (int i = 0; i < Count_zminna; i++)
885.            {
886.                    std::string str4 = "D://Resourse_min_F" +
std::to_string(i) + ".txt";
887.                    std::string str5 = str4;
888.                    fout_Q[i].open(str5, std::ios::out | std::ios::binary);
889.            }
890.
891.            for (int i = 0; i < Count_zminna; i++)
892.            {
893.                    try {
894.                            bool print_process = true;
895.                            char first_char = 'A';
```

```
896.                            constexpr char inverter = '~';
897.                            if (print_process)
898.                                    //std::cout << "Logical Function
Simplifier (Quine-McCluskey)" << std::endl
899.                                    //<< "[*] Enter a logical function to be
simplified" << std::endl
900.                                    //<< "    (ex. \"f(A, B, C) = A + BC +
~A~B + ABC\" )" << std::endl
901.                                    //<< "[*] Input: "
902.                                    cout << flush;
903.                            std::string *line = transFuncToMinimizeF(base);
904.                            logical_expr::function_parser<inverter, true>
parser(line[i], first_char);
905.                            auto token = parser.parse();
906.                            // Create a logical function with
logical_term<term_mark>
907.                            typedef
quine_mccluskey::simplifier::property_type PropertyType;
908.                            typedef quine_mccluskey::simplifier::term_type
TermType;
909.                            logical_expr::logical_function<TermType>
function;
910.                            for (std::string term : token.second)
911.                                    function +=
logical_expr::parse_logical_term<PropertyType, inverter>(term,
token.first.size(), first_char);
912.
913.                            // Create a simplifier using Quine-McCluskey
algorithm
914.                            quine_mccluskey::simplifier qm(function);
915.                            if (print_process) {
916.                                    //std::cout << std::endl << "Sum of
products form:" << std::endl;
917.                                    //print_truth_table(qm.get_std_spf(),
first_char);    // Print the function in sum of products form
918.                                    //std::cout << std::endl << "Compressing
..." << std::endl;
919.                                    qm.compress_table(true);
// Compress the compression table
920.
921.                                    for (const auto &term :
qm.get_prime_implicants()) {      // Print the prime implicants
922.                                            print_term_expr(term, fout_Q[i],
first_char);
923.                                            std::cout << "  ";
924.                                    }
925.                                    //std::cout << std::endl << std::endl <<
"Result of simplifying:" << std::endl;
926.                            }
927.                            else
928.                                    qm.compress_table(false);
929.
930.                            //for (const auto &func : qm.simplify())
// Simplify and print its results
931.                            //print_func_expr(func, first_char,
parser.function_name() + "\'");
932.                    }
933.                    catch (std::exception &e) {
934.                            std::cerr << std::endl << "[-] Exception: " <<
e.what() << std::endl;
935.                            exit_code = EXIT_FAILURE;
936.                    }
937.            }
```

```
938.            for (int i = 0; i < Count_zminna; i++)
939.            {
940.                    fout_Q[i].close();
941.            }
942.            //cout << flush;
943.            convertminimizedFunctionF(base);
944.            gotoxy(0, 16);
945.            cout << "Minimization of F is finished" << endl;
946.            return exit_code;
947.    }
948.
949. };
950.
951. #endif
```

ДОДАТОК Е. ПРОГРАМНИЙ КОД ГЕНЕРАТОРА ПОМНОЖУВАЧІВ ЕЛЕМЕНТІВ РОЗШИРЕНИХ ПОЛІВ ГАЛУА *GF(p^n)* З СТРУКТУРОЮ ФВ

ДОДАТОК Е.1 Заголовочний файл класу, який створює *VHDL*-опис вузла *MGC*

create_mgc.h

```
1. #ifndef CREATE_MGC_H
2. #define CREATE_MGC_H
3.
4. #include <iostream>
5.
6. class createMGC
7. {
8. public:
9.      createMGC(int basis, std::string fileMGC, std::string fileMUL,
std::string fileSUM);
10.     int creaMGC();
11.     void gotoxy(int column, int line);
12. private:
13.     int Basis;
14.     int Count_zminna;
15.     std::string fileNameMGC;
16.     std::string fileNameMUL;
17.     std::string fileNameSUM;
18. };
19.
20. #endif // CREATE_MGC_H
```

ДОДАТОК Е.2 Файл опису класу, який створює *VHDL*-опис вузла *MGC*

create_mgc.cpp

```
1. #include <iostream>
2. #include <fstream>
3. #include <io.h>
4. #include <math.h>
5. #include "create_MGC.h"
6. #include "windows.h"
7. #include "string"
8.
9. createMGC::createMGC(int basis, std::string fileMGC, std::string fileMUL,
std::string fileSUM )
10. {
11.     Basis = basis;
12.     fileNameMGC = fileMGC;
13.     fileNameMUL = fileMUL;
14.     fileNameSUM = fileSUM;
15. }
16.
17. void createMGC::gotoxy(int column, int line)
18. {
19.     COORD coord;
20.     coord.X = column;
21.     coord.Y = line;
22.     SetConsoleCursorPosition(
23.             GetStdHandle(STD_OUTPUT_HANDLE),
```

```
24.             coord
25.     );
26. }
27.
28. int createMGC::creaMGC()
29. {
30.
31.     if (Basis > 998)
32.     {
33.             std::cout << "The basis is too large" << std::endl;
34.             return 1;
35.     }
36.     const int Count_zminna = ceil((log(Basis)) / (log(2)));
37.
38.     std::ofstream fout_MGC;
39.     //fout_MGC.clear();
40.     fout_MGC.open("D://" + fileNameMGC + ".txt", std::ios::out |
std::ios::trunc);
41.     fout_MGC << "library IEEE;" << std::endl;
42.     //fout_MGC << std::endl;
43.     //std::cout << "library IEEE;" << std::endl;
44.     //std::cout << std::endl;
45.     fout_MGC << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
46.     //fout_MGC;
47.     //fout_MGC << "/n";
48.     //std::cout << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
49.     //fout_MGC << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
50.     //std::cout << std::endl;
51.     fout_MGC << std::endl;
52.     //std::cout << "entity " << fileName << " is" << std::endl;
53.     fout_MGC << "entity " << fileNameMGC << " is" << std::endl;
54.     //std::cout << "       port(" << std::endl;
55.     fout_MGC << "       port(" << std::endl;
56.
57.     if (Count_zminna > 1)
58.     {
59.             //std::cout << "          A : in STD_LOGIC_VECTOR(" <<
Count_zminna - 1 << " downto 0);" << std::endl;
60.             fout_MGC << "          A : in STD_LOGIC_VECTOR(" << Count_zminna
- 1 << " downto 0);" << std::endl;
61.             //std::cout << "          B : in STD_LOGIC_VECTOR(" <<
Count_zminna - 1 << " downto 0);" << std::endl;
62.             fout_MGC << "          B : in STD_LOGIC_VECTOR(" << Count_zminna
- 1 << " downto 0);" << std::endl;
63.             //std::cout << "          C : in STD_LOGIC_VECTOR(" <<
Count_zminna - 1 << " downto 0);" << std::endl;
64.             fout_MGC << "          C : in STD_LOGIC_VECTOR(" << Count_zminna
- 1 << " downto 0);" << std::endl;
65.             //std::cout << "          S : out STD_LOGIC_VECTOR(" <<
Count_zminna - 1 << " downto 0);" << std::endl;
66.             fout_MGC << "          S : out STD_LOGIC_VECTOR(" << Count_zminna
- 1 << " downto 0)" << std::endl;
67.     }
68.     else
69.     {
70.             fout_MGC << "          A : in STD_LOGIC;" << std::endl;
71.             fout_MGC << "          B : in STD_LOGIC;" << std::endl;
72.             fout_MGC << "          C : in STD_LOGIC;" << std::endl;
73.             fout_MGC << "          S : out STD_LOGIC;" << std::endl;
74.     }
75.
76.     //std::cout << "          );" << std::endl;
77.     fout_MGC << "          );" << std::endl;
```

```
78.     //std::cout << "end " << fileName << std::endl;
79.     fout_MGC << "end " << fileNameMGC << ";" << std::endl;
80.     //std::cout << std::endl;
81.     //std::cout << std::endl;
82.     fout_MGC << std::endl;
83.     fout_MGC << std::endl;
84.     //std::cout << "architecture "<< fileName <<" of "<< fileName <<" is" <<
std::endl;
85.     fout_MGC << "architecture " << fileNameMGC << " of " << fileNameMGC << "
is" << std::endl;
86.     //std::cout << "begin" << std::endl;
87.     if (Count_zminna > 1)
88.     {
89.             fout_MGC << "component  " << fileNameMUL << std::endl;
90.             fout_MGC << "   port(" << std::endl;
91.             fout_MGC << "       A: in STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0);" << std::endl;
92.             fout_MGC << "       B: in STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0);" << std::endl;
93.             fout_MGC << "       S: out STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0)" << std::endl;
94.             fout_MGC << ");" << std::endl;
95.             fout_MGC << "end component;" << std::endl;
96.             fout_MGC << std::endl;
97.             fout_MGC << "component  " << fileNameSUM << std::endl;
98.             fout_MGC << "   port(" << std::endl;
99.             fout_MGC << "       A: in STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0);" << std::endl;
100.            fout_MGC << "       B: in STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0);" << std::endl;
101.            fout_MGC << "       S: out STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0)" << std::endl;
102.            fout_MGC << ");" << std::endl;
103.            fout_MGC << "end component;" << std::endl;
104.    }
105.    else
106.    {
107.            fout_MGC << "component  " << fileNameMUL << std::endl;
108.            fout_MGC << "   port(" << std::endl;
109.            fout_MGC << "       A: in STD_LOGIC;" << std::endl;
110.            fout_MGC << "       B: in STD_LOGIC;" << std::endl;
111.            fout_MGC << "       S: out STD_LOGIC" << std::endl;
112.            fout_MGC << ");" << std::endl;
113.            fout_MGC << "end component;" << std::endl;
114.            fout_MGC << std::endl;
115.            fout_MGC << "component  " << fileNameSUM << std::endl;
116.            fout_MGC << "   port(" << std::endl;
117.            fout_MGC << "       A: in STD_LOGIC;" << std::endl;
118.            fout_MGC << "       B: in STD_LOGIC;" << std::endl;
119.            fout_MGC << "       S: out STD_LOGIC" << std::endl;
120.            fout_MGC << ");" << std::endl;
121.            fout_MGC << "end component;" << std::endl;
122.    }
123.    fout_MGC << std::endl;
124.    fout_MGC << std::endl;
125.    fout_MGC << std::endl;
126.    fout_MGC << std::endl;
127.    if (Count_zminna > 1)
128.    {
129.            fout_MGC << "signal BUS50 : STD_LOGIC_VECTOR(" << Count_zminna -
1 << " downto 0);" << std::endl;
130.    }
131.    else
```

```
132.    {
133.            fout_MGC << "signal BUS50 : STD_LOGIC;" << std::endl;
134.    }
135.    fout_MGC << std::endl;
136.    fout_MGC << std::endl;
137.    fout_MGC << "begin" << std::endl;
138.    fout_MGC << std::endl;
139.    fout_MGC << std::endl;
140.    fout_MGC << "U1 : " << fileNameMUL << std::endl;
141.    fout_MGC << "  port map(" << std::endl;
142.    fout_MGC << "       A => B," << std::endl;
143.    fout_MGC << "       B => C," << std::endl;
144.    fout_MGC << "       S => BUS50" << std::endl;
145.    fout_MGC << "  );" << std::endl;
146.    fout_MGC << std::endl;
147.    fout_MGC << "U2: " << fileNameSUM << std::endl;
148.    fout_MGC << "  port map(" << std::endl;
149.    fout_MGC << "       A => A," << std::endl;
150.    fout_MGC << "       B => BUS50," << std::endl;
151.    fout_MGC << "       S => S" << std::endl;
152.    fout_MGC << "  );" << std::endl;
153.    fout_MGC << std::endl;
154.    fout_MGC << std::endl;
155.    fout_MGC << "end " << fileNameMGC << ";" << std::endl;
156.
157.    gotoxy(0, 19);
158.    for (int x = 0; x < 120; x++)
159.    {
160.            std::cout << " ";
161.    }
162.    gotoxy(0, 19);
163.
164.    std::cout << "Creating MGC is finished" << std::endl;
165.    fout_MGC.close();
166.    //delete[] A;
167.
168.    return 0;
169. }
```

ДОДАТОК Е.3 Заголовочний файл класу, який створює *VHDL*-опис вузла *MUL*

create_MUL.h

```
1. #ifndef CREATE_MUL_H
2. #define CREATE_MUL_H
3.
4. #include <iostream>
5.
6. class createMUL
7. {
8. public:
9.      createMUL(int basis, std::string fileN);
10.     int creaMUL();
11.     void gotoxy(int column, int line);
12. private:
13.     int Basis;
14.     int Count_zminna;
15.     std::string fileName;
16. };
17.
18. #endif // CREATE_MGC_H
```

ДОДАТОК Е.4 Файл опису класу, який створює *VHDL*-опис вузла *MUL*

create_MUL.cpp

```
#include <iostream>
#include <fstream>
#include <io.h>
#include <math.h>
#include "create_MUL.h"
#include "windows.h"
#include "string"

createMUL::createMUL(int basis, std::string fileN)
{
    Basis = basis;
    fileName = fileN;
}

void createMUL::gotoxy(int column, int line)
{
    COORD coord;
    coord.X = column;
    coord.Y = line;
    SetConsoleCursorPosition(
        GetStdHandle(STD_OUTPUT_HANDLE),
        coord
    );
}

int createMUL::creaMUL()
{
    if (Basis > 998)
    {
        std::cout << "The basis is too large" << std::endl;
        return 1;
    }
    const int Count_zminna = ceil((log(Basis)) / (log(2)));

    std::ifstream *fout_Q = new std::ifstream[Count_zminna];
    std::ifstream *fout_Qi = new std::ifstream[Count_zminna];
    for (int i = 0; i < Count_zminna; i++)
    {
        std::string str4 = "D://Resourse_min_MUL" + std::to_string(i) +
".txt";
        std::string str5 = str4;
        fout_Q[i].open(str5, std::ios::in | std::ios::binary |
std::ios::ate);
    }
    for (int j = 0; j < Count_zminna; j++)
    {
        std::string str4 = "D://Resourse_min_MUL" + std::to_string(j) +
".txt";
        std::string str5 = str4;
        fout_Qi[j].open(str5, std::ios::in | std::ios::binary);
    }
    int *filesize = new int[Count_zminna];
    char **A = new char *[Count_zminna];
    for (int i = 0; i < Count_zminna; i++)
    {
        filesize[i] = fout_Q[i].tellg();
        A[i] = new char[filesize[i]];
    }
    for (int i = 0; i < Count_zminna; i++)
```

```
57.     {
58.         fout_Q[i].close();
59.     }
60.     for (int i = 0; i < Count_zminna; i++)
61.     {
62.         if (!fout_Qi[i]) return 1;
63.         fout_Qi[i].getline(A[i], filesize[i] + 1);
64.     }
65.
66.     for (int i = 0; i < Count_zminna; i++)
67.     {
68.         fout_Qi[i].close();
69.     }
70.
71.     std::ofstream fout_MGC;
72.     fout_MGC.open("D://" + fileName + ".txt", std::ios::out |
std::ios::trunc);
73.     fout_MGC << "library IEEE;" << std::endl;
74.     fout_MGC << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
75.     fout_MGC << std::endl;
76.     fout_MGC << "entity " << fileName << " is" << std::endl;
77.     fout_MGC << "       port(" << std::endl;
78.
79.     if (Count_zminna > 1)
80.     {
81.         fout_MGC << "          A : in STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0);" << std::endl;
82.         fout_MGC << "          B : in STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0);" << std::endl;
83.         fout_MGC << "          S : out STD_LOGIC_VECTOR(" << Count_zminna -
1 << " downto 0)" << std::endl;
84.     }
85.     else
86.     {
87.         fout_MGC << "          A : in STD_LOGIC;" << std::endl;
88.         fout_MGC << "          B : in STD_LOGIC;" << std::endl;
89.         fout_MGC << "          S : out STD_LOGIC;" << std::endl;
90.     }
91.
92.     fout_MGC << "          );" << std::endl;
93.     fout_MGC << "end " << fileName << ";" << std::endl;
94.     fout_MGC << std::endl;
95.     fout_MGC << std::endl;
96.     fout_MGC << "architecture " << fileName << " of " << fileName << " is"
<< std::endl;
97.     fout_MGC << "begin" << std::endl;
98.     fout_MGC << "         process(A, B)" << std::endl;
99.     fout_MGC << "       begin" << std::endl;
100.     if (Count_zminna > 1)
101.     {
102.         for (int i = 0; i < Count_zminna; i++)
103.         {
104.             fout_MGC << "          S(" << i << ") <= " << A[i] << ';' <<
std::endl;
105.         }
106.     }
107.     else
108.     {
109.         for (int i = 0; i < Count_zminna; i++)
110.         {
111.             fout_MGC << "          S <= " << A[i] << ";" << std::endl;
112.         }
113.     }
```

```
114.     fout_MGC << "       end process;" << std::endl;
115.     fout_MGC << std::endl;
116.     fout_MGC << std::endl;
117.     fout_MGC << "end " << fileName << ";" << std::endl;
118.
119.     gotoxy(0, 11);
120.     for (int x = 0; x < 120; x++)
121.     {
122.         std::cout << " ";
123.     }
124.     gotoxy(0, 11);
125.
126.     std::cout << "Creating MUL is finished" << std::endl;
127.     fout_MGC.close();
128.     delete[] A;
129.     return 0;
130. }
```

ДОДАТОК Е.5 Заголовочний файл класу, який створює *VHDL*-опис вузла *SUM*

create_SUM.h

```
1. #ifndef CREATE_SUM_H
2. #define CREATE_SUM_H
3.
4. #include <iostream>
5.
6. class createSUM
7. {
8. public:
9.     createSUM(int basis, std::string fileN);
10.    int creaSUM();
11.    void gotoxy(int column, int line);
12. private:
13.    int Basis;
14.    int Count_zminna;
15.    std::string fileName;
16. };
17.
18. #endif // CREATE_MGC_H
```

ДОДАТОК Е.6 Файл опису класу, який створює *VHDL*-опис вузла *SUM*

create_SUM.cpp

```
1. #include <iostream>
2. #include <fstream>
3. #include <io.h>
4. #include <math.h>
5. #include "create_SUM.h"
6. #include "windows.h"
7. #include "string"
8.
9. createSUM::createSUM(int basis, std::string fileN)
10. {
11.    Basis = basis;
12.    fileName = fileN;
13. }
14.
15. void createSUM::gotoxy(int column, int line)
16. {
17.    COORD coord;
```

```
18.     coord.X = column;
19.     coord.Y = line;
20.     SetConsoleCursorPosition(
21.         GetStdHandle(STD_OUTPUT_HANDLE),
22.         coord
23.     );
24. }
25.
26. int createSUM::creaSUM()
27. {
28.     if (Basis > 998)
29.     {
30.         std::cout << "The basis is too large" << std::endl;
31.         return 1;
32.     }
33.     const int Count_zminna = ceil((log(Basis)) / (log(2)));
34.
35.     std::ifstream *fout_Q = new std::ifstream[Count_zminna];
36.     std::ifstream *fout_Qi = new std::ifstream[Count_zminna];
37.     for (int i = 0; i < Count_zminna; i++)
38.     {
39.         std::string str4 = "D://Resourse_min_SUM" + std::to_string(i) +
".txt";
40.         std::string str5 = str4;
41.         fout_Q[i].open(str5, std::ios::in | std::ios::binary |
std::ios::ate);
42.     }
43.     for (int j = 0; j < Count_zminna; j++)
44.     {
45.         std::string str4 = "D://Resourse_min_SUM" + std::to_string(j) +
".txt";
46.         std::string str5 = str4;
47.         fout_Qi[j].open(str5, std::ios::in | std::ios::binary);
48.     }
49.     int *filesize = new int[Count_zminna];
50.     char **A = new char *[Count_zminna];
51.     for (int i = 0; i < Count_zminna; i++)
52.     {
53.         filesize[i] = fout_Q[i].tellg();
54.         A[i] = new char[filesize[i]];
55.     }
56.     for (int i = 0; i < Count_zminna; i++)
57.     {
58.         fout_Q[i].close();
59.     }
60.     for (int i = 0; i < Count_zminna; i++)
61.     {
62.         if (!fout_Qi[i]) return 1;
63.         fout_Qi[i].getline(A[i], filesize[i] + 1);
64.     }
65.
66.     for (int i = 0; i < Count_zminna; i++)
67.     {
68.         fout_Qi[i].close();
69.     }
70.
71.     std::ofstream fout_MGC;
72.     fout_MGC.open("D://" + fileName + ".txt", std::ios::out |
std::ios::trunc);
73.     fout_MGC << "library IEEE;" << std::endl;
74.     fout_MGC << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
75.     fout_MGC << std::endl;
76.     fout_MGC << "entity " << fileName << " is" << std::endl;
```

```
77.     fout_MGC << "       port(" << std::endl;
78.
79.     if (Count_zminna > 1)
80.     {
81.         fout_MGC << "          A : in STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0);" << std::endl;
82.         fout_MGC << "          B : in STD_LOGIC_VECTOR(" << Count_zminna - 1
<< " downto 0);" << std::endl;
83.         fout_MGC << "          S : out STD_LOGIC_VECTOR(" << Count_zminna -
1 << " downto 0)" << std::endl;
84.     }
85.     else
86.     {
87.         fout_MGC << "          A : in STD_LOGIC;" << std::endl;
88.         fout_MGC << "          B : in STD_LOGIC;" << std::endl;
89.         fout_MGC << "          S : out STD_LOGIC;" << std::endl;
90.     }
91.     fout_MGC << "          );" << std::endl;
92.     fout_MGC << "end " << fileName << ";" << std::endl;
93.     fout_MGC << std::endl;
94.     fout_MGC << std::endl;
95.     fout_MGC << "architecture " << fileName << " of " << fileName << " is"
<< std::endl;
96.     fout_MGC << "begin" << std::endl;
97.     fout_MGC << "         process(A, B)" << std::endl;
98.     fout_MGC << "       begin" << std::endl;
99.     if (Count_zminna > 1)
100.     {
101.         for (int i = 0; i < Count_zminna; i++)
102.         {
103.             fout_MGC << "          S(" << i << ") <= " << A[i] << ';' <<
std::endl;
104.         }
105.     }
106.     else
107.     {
108.         for (int i = 0; i < Count_zminna; i++)
109.         {
110.             fout_MGC << "          S <= " << A[i] << ";" << std::endl;
111.         }
112.     }
113.     fout_MGC << "       end process;" << std::endl;
114.     fout_MGC << std::endl;
115.     fout_MGC << std::endl;
116.     fout_MGC << "end " << fileName << ";" << std::endl;
117.
118.     gotoxy(0, 14);
119.     for (int x = 0; x < 120; x++)
120.     {
121.         std::cout << " ";
122.     }
123.     gotoxy(0, 14);
124.
125.     std::cout << "Creating SUM is finished" << std::endl;
126.     fout_MGC.close();
127.     delete[] A;
128.     return 0;
129. }
130.
```

ДОДАТОК Е.7 Заголовочний файл класу, який генерує булеві функції для вузла *MUL*

generate_functions_for_MUL.h

```
1. #ifndef GENERATE_FUNCTIONS_FOR_MUL_H
2. #define GENERATE_FUNCTIONS_FOR_MUL_H
3.
4. class generateFunctionsForMUL
5. {
6. public:
7.      generateFunctionsForMUL(int basis);
8.      int generateFunction();
9.      void gotoxy(int column, int line);
10. private:
11.     int Basis;
12.     //int Count_zminna;
13. };
14.
15. #endif // GENERATE_FUNCTIONS_FOR_MGC_H
```

ДОДАТОК Е.8 Файл опису класу, який генерує булеві функції для вузла *MUL*

generate_functions_for_MUL.cpp

```
1. #include "generate_functions_for_MUL.h"
2. #include <fstream>
3. #include <iostream>
4. #include <math.h>
5. #include <bitset>
6. #include <windows.h>
7. #include <string.h>
8.
9. generateFunctionsForMUL::generateFunctionsForMUL(int basis)
10. {
11.     Basis = basis;
12. }
13.
14. void generateFunctionsForMUL::gotoxy(int column, int line)
15. {
16.     COORD coord;
17.     coord.X = column;
18.     coord.Y = line;
19.     SetConsoleCursorPosition(
20.         GetStdHandle(STD_OUTPUT_HANDLE),
21.         coord
22.     );
23. }
24.
25. int generateFunctionsForMUL::generateFunction()
26. {
27.     gotoxy(0, 9);
28.     if (Basis > 998)
29.     {
30.         std::cout << "The basis is too large" << std::endl;
31.         return 1;
32.     }
33.     const int Count_zminna = ceil((log(Basis)) / (log(2)));
34.     std::ofstream *fout_Q = new std::ofstream[Count_zminna];
35.     for (int i = 0; i < Count_zminna; i++)
36.     {
37.         std::string str4 = "D://Resourse_MUL" + std::to_string(i) + ".txt";
38.         std::string str5 = str4;
39.         fout_Q[i].open(str5, std::ofstream::out | std::ofstream::trunc);
```

```
40.     }
41.     std::bitset<10> A;
42.     std::bitset<10> B;
43.     //std::bitset<10> C;
44.     int res;
45.     bool *oor = new bool[Count_zminna];
46.     for (int m = 0; m < Count_zminna; m++)
47.     {
48.         oor[m] = false;
49.     }
50.     int countProcessing = 0;
51.     for (int i = 0; i < Basis; i++)
52.     {
53.         A = i;
54.         for (int j = 0; j < Basis; j++)
55.         {
56.             B = j;
57.             res = ((j * i) % Basis) ;
58.             //std::cout << std::hex << int(res) << std::endl;
59.             if (res != 0)
60.             {
61.                 for (int r = 0; r < Count_zminna; r++)
62.                 {
63.                     int YangerBit = (res >> r) & 0x01;
64.                     if (YangerBit == 1)
65.                     {
66.                         if (oor[r] == true)
67.                         {
68.                             fout_Q[r] << " or ";
69.                         }
70.                         for (int s = Count_zminna - 1; s >= 0; s--)
71.                         {
72.                             oor[r] = true;
73.                             if (s != Count_zminna - 1)
74.                             {
75.                                 fout_Q[r] << " and ";
76.                             }
77.                             if (s == (Count_zminna - 1))
78.                             {
79.                                 fout_Q[r] << "( ";
80.                             }
81.                             if (A[s] == 0)
82.                             {
83.                                 fout_Q[r] << "not ";
84.                                 std::string str6 = "A(" + std::to_string(s)
+ ")";
85.                                 std::string str7 = str6;
86.                                 fout_Q[r] << str7;
87.                             }
88.                             if (A[s] == 1)
89.                             {
90.                                 std::string str6 = "A(" + std::to_string(s)
+ ")";
91.                                 std::string str7 = str6;
92.                                 fout_Q[r] << str7;
93.                             }
94.                         }
95.                         for (int s = Count_zminna - 1; s >= 0; s--)
96.                         {
97.                             fout_Q[r] << " and ";
98.                             if (B[s] == 0)
99.                             {
100.                                fout_Q[r] << "not ";
```

```
                                    std::string str6 = "B(" + std::to_string(s)
+ ")";
                                    std::string str7 = str6;
                                    fout_Q[r] << str7;
                                }
                                if (B[s] == 1)
                                {
                                    std::string str6 = "B(" + std::to_string(s)
+ ")";
                                    std::string str7 = str6;
                                    fout_Q[r] << str7;
                                }
                                if (s == 0)
                                {
                                    fout_Q[r] << " )";
                                }
                            }
                        }
                        if (countProcessing < 120)
                        {
                            std::cout << char(221);
                            countProcessing++;
                        }
                        else
                        {
                            gotoxy(0, 9);
                            countProcessing = 0;
                            for (int x = 0; x < 120; x++)
                            {
                                std::cout << " ";
                            }
                            gotoxy(0, 9);
                        }
                    }
                }
            }
        }

        for (int l = 0; l < Count_zminna; l++)
        {
            fout_Q[l].close();
        }

        gotoxy(0, 9);
        countProcessing = 0;
        for (int x = 0; x < 120; x++)
        {
            std::cout << " ";
        }
        gotoxy(0, 9);
        gotoxy(0, 9);

        return 0;
    }
```

ДОДАТОК Е.9 Заголовочний файл класу, який генерує булеві функції для вузла *SUM*

generate_functions_for_SUM.h

```
#ifndef GENERATE_FUNCTIONS_FOR_SUM_H
#define GENERATE_FUNCTIONS_FOR_SUM_H
```

```
3.
4. class generateFunctionsForSUM
5. {
6. public:
7.     generateFunctionsForSUM(int basis);
8.      int generateFunction();
9.      void gotoxy(int column, int line);
10. private:
11.     int Basis;
12. };
13.
14. #endif // GENERATE_FUNCTIONS_FOR_MGC_H
```

ДОДАТОК Е.10 Файл опису класу, який генерує булеві функції для вузла *SUM*

generate_functions_for_SUM.cpp

```
1. #include "generate_functions_for_SUM.h"
2. #include <fstream>
3. #include <iostream>
4. #include <math.h>
5. #include <bitset>
6. #include <windows.h>
7. #include <string.h>
8.
9. generateFunctionsForSUM::generateFunctionsForSUM(int basis)
10. {
11.     Basis = basis;
12. }
13.
14. void generateFunctionsForSUM::gotoxy(int column, int line)
15. {
16.     COORD coord;
17.     coord.X = column;
18.     coord.Y = line;
19.     SetConsoleCursorPosition(
20.             GetStdHandle(STD_OUTPUT_HANDLE),
21.             coord
22.     );
23. }
24.
25. int generateFunctionsForSUM::generateFunction()
26. {
27.     gotoxy(0, 12);
28.     if (Basis > 998)
29.     {
30.         std::cout << "The basis is too large" << std::endl;
31.         return 1;
32.     }
33.     const int Count_zminna = ceil((log(Basis)) / (log(2)));
34.     std::ofstream *fout_Q = new std::ofstream[Count_zminna];
35.     for (int i = 0; i < Count_zminna; i++)
36.     {
37.         std::string str4 = "D://Resourse_SUM" + std::to_string(i) + ".txt";
38.         std::string str5 = str4;
39.         fout_Q[i].open(str5, std::ofstream::out | std::ofstream::trunc);
40.     }
41.     std::bitset<10> A;
42.     std::bitset<10> B;
43.     int res;
44.     bool *oor = new bool[Count_zminna];
45.     for (int m = 0; m < Count_zminna; m++)
```

```
46.     {
47.         oor[m] = false;
48.     }
49.     int countProcessing = 0;
50.     for (int i = 0; i < Basis; i++)
51.     {
52.         A = i;
53.         for (int j = 0; j < Basis; j++)
54.         {
55.             B = j;
56.             res = ((j + i) % Basis);
57.             //std::cout << std::hex << int(res) << std::endl;
58.             if (res != 0)
59.             {
60.                 for (int r = 0; r < Count_zminna; r++)
61.                 {
62.                     int YangerBit = (res >> r) & 0x01;
63.                     if (YangerBit == 1)
64.                     {
65.                         if (oor[r] == true)
66.                         {
67.                             fout_Q[r] << " or ";
68.                         }
69.                         for (int s = Count_zminna - 1; s >= 0; s--)
70.                         {
71.                             oor[r] = true;
72.                             if (s != Count_zminna - 1)
73.                             {
74.                                 fout_Q[r] << " and ";
75.                             }
76.                             if (s == (Count_zminna - 1))
77.                             {
78.                                 fout_Q[r] << "( ";
79.                             }
80.
81.
82.                             if (A[s] == 0)
83.                             {
84.                                 fout_Q[r] << "not ";
85.                                 std::string str6 = "A(" + std::to_string(s) +
")";
86.                                 std::string str7 = str6;
87.                                 fout_Q[r] << str7;
88.                             }
89.                             if (A[s] == 1)
90.                             {
91.                                 std::string str6 = "A(" + std::to_string(s) +
")";
92.                                  std::string str7 = str6;
93.                                 fout_Q[r] << str7;
94.                             }
95.                         }
96.                         for (int s = Count_zminna - 1; s >= 0; s--)
97.                         {
98.                             fout_Q[r] << " and ";
99.                             if (B[s] == 0)
100.                            {
101.                                fout_Q[r] << "not ";
102.                                std::string str6 = "B(" + std::to_string(s) +
")";
103.                                std::string str7 = str6;
104.                                fout_Q[r] << str7;
105.                            }
```

```
106.                            if (B[s] == 1)
107.                            {
108.                                std::string str6 = "B(" + std::to_string(s) +
")";
109.                                std::string str7 = str6;
110.                                fout_Q[r] << str7;
111.                            }
112.                            if (s == 0)
113.                            {
114.                                fout_Q[r] << " )";
115.                            }
116.                        }
117.                    }
118.                if (countProcessing < 120)
119.                {
120.                    std::cout << char(221);
121.                    countProcessing++;
122.                }
123.                else
124.                {
125.                    gotoxy(0, 12);
126.                    countProcessing = 0;
127.                    for (int x = 0; x < 120; x++)
128.                    {
129.                        std::cout << " ";
130.                    }
131.                    gotoxy(0, 12);
132.                }
133.            }
134.        }
135.      }
136.    }
137.
138.
139.    gotoxy(0, 12);
140.    countProcessing = 0;
141.    for (int x = 0; x < 120; x++)
142.    {
143.            std::cout << " ";
144.    }
145.    gotoxy(0, 12);
146.    gotoxy(0, 12);
147.
148.
149.    for (int l = 0; l < Count_zminna; l++)
150.    {
151.            fout_Q[l].close();
152.    }
153.    return 0;
154. }
```

ДОДАТОК Є. ПРОГРАМНИЙ КОД ГЕНЕРАТОРА ПОМНОЖУВАЧІВ ЕЛЕМЕНТІВ РОЗШИРЕНИХ ПОЛІВ ГАЛУА *GF(p^n)* З СТРУКТУРОЮ ЛЕ

ДОДАТОК Є.1 Заголовочний файл класу, який генерує *VHDL*-опис вузла MGC

create_mgc.cpp

```
1. #ifndef CREATING_MATRIX_H
2. #define CREATING_MATRIX_H
3. #include <iostream>
4. #include "SMn_element.h"
5. #include "Rn_element.h"
6. #include "SUM_element.h"
7. #include "Sn_element.h"
8. #include "Rn_element.h"
9. #include "BaseClass.h"
10.
11. class CreateMGC : public BaseClass
12. {
13. public:
14.     CreateMGC(int mbase, int mlarge, std::string mSMn, std::string mSMch,
std::string mMGC);
15.     void fillMGC();
16.     void printFile();
17.     int wherex();
18.     int wherey();
19.     ~CreateMGC();
20. private:
21.     std::string strSMn;
22.     std::string strSMch;
23.     std::string strMGC;
24.     int busCount;
25.     int base;
26.     int large;
27.     SMn** SMn_matr;
28.     SMch** SMch_matr;
29.     SnElement* Sn;
30.     RnElement* Rn;
31.     SUMElement* SUM;
32. };
33.
34. #endif // CREATING_MATRIX_H
35.
```

ДОДАТОК Є.2 Файл опису класу, який генерує *VHDL*-опис вузла *MGC*

create_mgc.cpp

```
1. #ifndef CREATE_MGC_H
2. #define CREATE_MGC_H
1. #include "creating_MGC_LLE.h"
2. #include "mgc_element.h"
3. #include "f_element.h"
4. #include <iostream>
5. #include <fstream>
6. #include <math.h>
7. #include "windows.h"
```

```
8. #include <stdio.h>
9. #include <string>
10.
11.
12. CreateMGC::CreateMGC(int mbase, int mlarge, std::string mSMn, std::string
mSMch, std::string mMGC)
13. {
14.     strSMn = mSMn;
15.     strSMch = mSMch;
16.     strMGC = mMGC;
17.     base = mbase;
18.     busCount = 0;
19.     large = mlarge;
20.     int matrix_size = (large * 2) - 1;
21.     // Create matrix of SMn elements
22.     SMn_matr = new SMn *[large - 1];
23.     for (int i = 0; i < matrix_size; i++)
24.         SMn[i] = new SMn[matrix_size];
25.     // Create matrix of SMch elements
26.     SMch_matr = new SMch *[large - 1];
27.     for (int i = 0; i < matrix_size; i++)
28.         SMch[i] = new SMch[matrix_size];
29.     // Create vector of elements Sn
30.     Sn = new SnElement[large - 1];
31.     for (int i = 0; i < large - 1; i++)
32.     {
33.         Sn[i].initialSn(base, large);
34.     }
35.     // Create vector of elements Rn
36.     Rn = new RnElement[large - 1];
37.     for (int i = 0; i < large - 1; i++)
38.     {
39.         Rn[i].initialRn(base, large);
40.     }
41.     // Create vector of elements SUM
42.     SUM = new SUMElement[large - 1];
43.     for (int i = 0; i < large - 1; i++)
44.     {
45.         SUM[i].initialSUM(base, large);
46.     }
47.     std::cout << "Dates are processing. Wait for finishing" << std::endl;
48. }
49.
50. CreateMGC::~CreateMGC()
51. {
52.     for (int count = 0; count < large -1; count++) {
53.         delete[] SMn[count];
54.         delete[] SMch[count];
55.     }
56.     delete[]Sn;
57.     delete[]Rn;
58. }
59.
60. void CreateMGC::fillMGC()
61. {
62.     int countProcessing;
63.     int matrix_size = (large * 2) - 1;
64.     int m = large, k = m - 1;
65.     int p = 0;
66.     //Cicle for setting elements SMn full or empty
67.     for (int j = 0; j <= m - 1; j++)
68.     {
69.         for (int i = k; i <= 2 * m - 2 - p; i++)
```

```
70.         {
71.             SMn[j][i].MGCSetFull();
72.         }
73.         if (j < m - 1)
74.         {
75.             k--;
76.             p++;
77.         }
78.         else {
79.             k++;
80.             p--;
81.         }
82.     }
83.     k = m - 1;
84.     p = 0;
85.     //Cicle for setting elements SMch full or empty
86.     for (int j = 0; j <= m - 1; j++)
87.     {
88.         for (int i = k; i <= 2 * m - 2 - p; i++)
89.         {
90.             SMch[j][i].MGCSetFull();
91.         }
92.         if (j < m - 1)
93.         {
94.             k--;
95.             p++;
96.         }
97.         else {
98.             k++;
99.             p--;
100.         }
101.     }
102.     k = m - 1;
103.     p = 0;
104.     //Cicle for setting in input C elements Sn
105.     for (int j = 0; j < m - 1; j++)
106.     {
107.         for (int i = 0; i < matrix_size; i++)
108.         {
109.             if ((i + j == large - 1) || (j == 0 && i >= large - 1))
110.             {
111.                 SMch[j][i].AddC(-5);
112.             }
113.         }
114.     }
115.     //Cicle for setting A, B and C on elements of array SMn
116.     int elem;
117.     for (int j = 0; j <= m - 1; j++)
118.     {
119.         elem = large - 1;
120.         for (int i = k; i <= 2 * m - 2 - p; i++)
121.         {
122.             if (j <= large - 1)
123.             {
124.                 SMn[j][i].AddA(j);
125.                 SMn[j][i].AddB(elem);
126.                 SMn[j][i].AddC(i);
127.             }
128.             else
129.             {
130.                 SMn[j][i].AddA(elem);
131.             }
132.             elem--;
```

```
133.         }
134.         if (j < m - 1)
135.         {
136.             k--;
137.             p++;
138.         }
139.         else {
140.             k++;
141.             p--;
142.         }
143.     }
144.     k = m - 1;
145.     p = 0;
146.     k = m - 1;
147.     p = 0;
148.     //Cicle for setting A, B and C on elements of array SMch
149.     for (int j = 0; j <= m - 1; j++)
150.     {
151.         elem = large - 1;
152.         for (int i = k; i <= 2 * m - 2 - p; i++)
153.         {
154.             if (j <= large - 1)
155.             {
156.                 SMch[j][i].AddA(j);
157.                 SMch[j][i].AddB(elem);
158.                 SMch[j][i].AddC(i);
159.             }
160.             else
161.             {
162.                 SMch[j][i].AddA(elem);
163.             }
164.             elem--;
165.         }
166.         if (j < m - 1)
167.         {
168.             k--;
169.             p++;
170.         }
171.         else {
172.             k++;
173.             p--;
174.         }
175.     }
176.     k = m - 1;
177.     p = 0;
178.     int vel;
179.     k = m - 1;
180.     p = 0;
181.     //Cicle for add adreses amonth elements SMn that create koeficient on
that we multiply polynom
182.     int aa = large - 2;
183.     int b = -1;
184.     for (int j = 0; j <= m - 1; j++)
185.     {
186.         int r = 0;
187.         vel = -1;
188.         for (int i = k; i <= 2 * m - 2 - p; i++)
189.         {
190.             if (j > large - 1)
191.             {
192.                 if (r == 0)
193.                 {
194.                     aa++;
```

```
195.                     b++;
196.                 }
197.                 vel++;
198.                 if (vel == 0)
199.                 {
200.                     busCount++;
201.                 }
202.                 SMn[j][i].Addb(&SMn[aa][b]);
203.                 SMn[aa][b].Adds(&SMn[j][i]);
204.
205.                 SMn[j][i].AddBbus(busCount - 1);
206.                 SMn[aa][b].AddSbus(busCount - 1);
207.                 r++;
208.             }
209.         }
210.         if (j < m - 1)
211.         {
212.             k--;
213.             p++;
214.         }
215.         else {
216.             k++;
217.             p--;
218.         }
219.     }
220.     k = m - 1;
221.     p = 0;
222.     //Cicle for add adreses amonth elements SMch that create koeficient on
that we multiply polynom
223.     for (int j = 0; j <= m - 1; j++)
224.     {
225.         int r = 0;
226.         vel = -1;
227.         for (int i = k; i <= 2 * m - 2 - p; i++)
228.         {
229.             if (j > large - 1)
230.             {
231.                 if (r == 0)
232.                 {
233.                     aa++;
234.                     b++;
235.                 }
236.                 vel++;
237.                 if (vel == 0)
238.                 {
239.                     busCount++;
240.                 }
241.                 SMch[j][i].Addb(&SMch[aa][b]);
242.                 SMch[aa][b].Adds(&SMch[j][i]);
243.
244.                 SMch[j][i].AddBbus(busCount - 1);
245.                 SMch[aa][b].AddSbus(busCount - 1);
246.                 r++;
247.             }
248.         }
249.         if (j < m - 1)
250.         {
251.             k--;
252.             p++;
253.         }
254.         else {
255.             k++;
256.             p--;
```

```
257.         }
258.     }
259.     k = m - 1;
260.     p = 0;
261.     //Cicle for setting elements Rn
262.     for (int j = 0; j < m - 1; j++)
263.     {
264.         for (int i = 0; i < matrix_size; i++)
265.         {
266.             if ((i + j == large - 1) || (j == 0 && i >= large - 1))
267.             {
268.                 Rn[j][i].AddC(-5);
269.             }
270.         }
271.     }
272.     //Cicle for adding elements SUM
273.     for (int j = 0; j < m - 1; j++)
274.     {
275.         for (int i = 0; i < matrix_size; i++)
276.         {
277.             if ((i + j == large - 1) || (j == 0 && i >= large - 1))
278.             {
279.                 SUM[j][i].AddR(-5);
280.             }
281.         }
282.     }
283.     //Circuit for adding result elements
284.     int val = large;
285.     for (int i = 0; i <= (2 * large) - 2; i++)
286.     {
287.         if (SUM[(2 * large) - 2][i].returnSUMempty() == true)
288.         {
289.             SUM[(2 * large) - 2][i].AddS(val - 1);
290.             val--;
291.         }
292.     }
293.     //Circle for adding connections between elements in array SMn
294.     k = m - 1;
295.     p = 0;
296.     for (int j = 0; j <= m - 1; j++)
297.     {
298.         for (int i = k; i <= 2 * m - 2 - p; i++)
299.         {
300.             if (SMn[j][i].ReturnS() == -1)
301.             {
302.                 for (int s = j + 1; s <= (2 * large) - 2; s++)
303.                 {
304.                     if (SMn[s][i].returnMGCempty() == true)
305.                     {
306.                         busCount++;
307.                         SMn[j][i].Adds(&SMn[s][i]);
308.                         SMn[j][i].AddSbus(busCount - 1);
309.                         SMn[s][i].Addc(&SMn[j][i]);
310.                         SMn[s][i].AddCbus(busCount - 1);
311.                         break;
312.                     }
313.                 }
314.             }
315.         }
316.         if (j < m - 1)
317.         {
318.             k--;
319.             p++;
```

```
320.         }
321.         else {
322.             k++;
323.             p--;
324.         }
325.     }
326.     //Circle for adding connections between elements in array SMch
327.     k = m - 1;
328.     p = 0;
329.     for (int j = 0; j <= m - 1; j++)
330.     {
331.         for (int i = k; i <= 2 * m - 2 - p; i++)
332.         {
333.             if (SMch[j][i].ReturnS() == -1)
334.             {
335.                 for (int s = j + 1; s <= (2 * large) - 2; s++)
336.                 {
337.                     if (SMch[s][i].returnMGCempty() == true)
338.                     {
339.                         busCount++;
340.                         SMch[j][i].Adds(&SMch[s][i]);
341.                         SMch[j][i].AddSbus(busCount - 1);
342.                         SMch[s][i].Addc(&SMch[j][i]);
343.                         SMch[s][i].AddCbus(busCount - 1);
344.                         break;
345.                     }
346.                 }
347.             }
348.         }
349.         if (j < m - 1)
350.         {
351.             k--;
352.             p++;
353.         }
354.         else {
355.             k++;
356.             p--;
357.         }
358.     }
359.     //Circle for adding elements Sn to SMch
360.     int countInputBits = ceil((log(base)) / (log(2)));
361.     if (countInputBits > 1)
362.     {
363.         int i = 0;
364.         for (int j = large - 1; j < (2 * large) - 2; j++)
365.         {
366.             SMch[j][i].AddSbus(busCount);
367.             Sn[i].AddAbus(busCount);
368.             int l = SMch[j + 1][i + 1].ReturnBbus();
369.             Sn[i].AddBbus(SMch[j + 1][i + 1].ReturnBbus());
370.             busCount++;
371.             i++;
372.         }
373.     }
374. }
375.
376. void CreateMGC::printFile()
377. {
378.     int countProcessing = 0;
379.     int U = 0;
380.     int countInputBits = ceil((log(base)) / (log(2)));
381.     std::ofstream fout;
382.     //std::string str = "D://" + strMGC + ".txt";
```

```
383.     fout.open("D://" + strMultiplier + ".txt");
384.     if (countInputBits >1)
385.     {
386.         fout << "library IEEE;" << std::endl;
387.         fout << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
388.         fout << std::endl;
389.         fout << "entity " << strMultiplier << " is" << std::endl;
390.         fout << "       port(" << std::endl;
391.         fout << "          A : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
392.         fout << "          B : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
393.         fout << "          C : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
394.         fout << "          R : out STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0)" << std::endl;
395.         fout << "          );" << std::endl;
396.         fout << "end " << strMultiplier << ";" << std::endl;
397.         fout << std::endl;
398.         fout << std::endl;
399.         fout << "architecture " + strMGC + " of " + strMGC + " is" <<
std::endl;
400.         fout << std::endl;
401.         fout << std::endl;
402.         fout << "component " + strMGC << std::endl;
403.         fout << "  port (" << std::endl;
404.         fout << "       A : in STD_LOGIC_VECTOR(" << countInputBits - 1 <<
" downto 0);" << std::endl;
405.         fout << "       B : in STD_LOGIC_VECTOR(" << countInputBits - 1 <<
" downto 0);" << std::endl;
406.         fout << "       C : in STD_LOGIC_VECTOR(" << countInputBits - 1 <<
" downto 0);" << std::endl;
407.         fout << "       S : out STD_LOGIC_VECTOR(" << countInputBits - 1 <<
" downto 0)" << std::endl;
408.         fout << "       );" << std::endl;
409.         fout << "end component;" << std::endl;
410.         fout << std::endl;
411.         fout << std::endl;
412.         fout << "component " + strF << std::endl;
413.         fout << "  port (" << std::endl;
414.         fout << "       A : in STD_LOGIC_VECTOR(" << countInputBits - 1 <<
" downto 0);" << std::endl;
415.         fout << "       B : out STD_LOGIC_VECTOR(" << countInputBits - 1 <<
" downto 0)" << std::endl;
416.         fout << "       );" << std::endl;
417.         fout << "end component;" << std::endl;
418.         fout << std::endl;
419.         fout << std::endl;
420.         for (int i = 0; i < busCount; i++)
421.         {
422.             fout << "signal BUS" << i << " : STD_LOGIC_VECTOR (" <<
countInputBits - 1 << " downto 0);" << std::endl;
423.         }
424.         fout << std::endl;
425.         fout << std::endl;
426.         fout << "begin" << std::endl;
427.         fout << std::endl;
428.         fout << std::endl;
429.         for (int j = 0; j <= large - 1; j++)
430.         {
431.             for (int i = 0; i <= 2 * large - 2; i++)
432.             {
433.                 if (j < large)
```

```
434.                 {
435.                     if (SMn[j][i].returnMGCempty())
436.                     {
437.                         fout << "U" << U << " : " + strMGC << std::endl;
438.                         fout << "  port map(" << std::endl;
439.                         for (int k = 0; k < countInputBits; k++)
440.                         {
441.                             fout << "       A(" << k << ") =>" << "A(" <<
(SMn[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
442.                         }
443.                         for (int k = 0; k < countInputBits; k++)
444.                         {
445.                             fout << "       B(" << k << ") =>" << "B(" <<
(SMn[j][i].ReturnB() * countInputBits) + k << ")," << std::endl;
446.                         }
447.                         if (SMn[j][i].ReturnC() == -5)
448.                         {
449.                             fout << "       C => L," << std::endl;
450.                         }
451.                         else
452.                         {
453.                             fout << "       C" << " => " << "BUS" <<
SMn[j][i].ReturnCbus() << "," << std::endl;
454.                         }
455.                         fout << "       S" << " => " << "BUS" <<
SMn[j][i].ReturnSbus() << std::endl;
456.                         fout << "  );" << std::endl;
457.                         fout << std::endl;
458.                         U++;
459.                     }
460.                 }
461.                 else
462.                 {
463.                     if (SMn[j][i].returnSMchEmpty())
464.                     {
465.                         fout << "U" << U << " : " + strMGC << std::endl;
466.                         fout << "  port map(" << std::endl;
467.                         for (int k = 0; k < countInputBits; k++)
468.                         {
469.                             fout << "       A(" << k << ") =>" << "p(" <<
(SMn[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
470.                         }
471.                         fout << "       B" << " => " << "BUS" <<
SMn[j][i].ReturnBbus() << "," << std::endl;
472.                         if (SMn[j][i].ReturnC() != -5)
473.                         {
474.                             fout << "       C" << " => " << "BUS" <<
SMn[j][i].ReturnCbus() << "," << std::endl;
475.                         }
476.                         if (SMn[j][i].ReturnS() == -1)
477.                         {
478.                             fout << "       S" << " => " << "BUS" <<
SMn[j][i].ReturnSbus() << std::endl;
479.                             fout << "  );" << std::endl;
480.                         }
481.                         else
482.                         {
483.                             for (int k = 0; k < countInputBits; k++)
484.                             {
485.                                 if (k == countInputBits - 1)
486.                                 {
487.                                     fout << "       S(" << k << ") =>" <<
"R(" << (SMn[j][i].ReturnS() * countInputBits) + k << ")" << std::endl;
```

```
488.                                 }
489.                                 else
490.                                 {
491.                                     fout << "       S(" << k << ") =>" <<
"R(" << (SMn[j][i].ReturnS() * countInputBits) + k << ")" << "," << std::endl;
492.                                 }
493.                             }
494.                             fout << "  );" << std::endl;
495.                         }
496.                         fout << std::endl;
497.                         U++;
498.                     }
499.                 }
500.             }
501.             if (countProcessing < 120)
502.             {
503.                 std::cout << char(221);
504.                 countProcessing++;
505.             }
506.             else
507.             {
508.                 gotoxy(0, 10);
509.                 countProcessing = 0;
510.                 for (int x = 0; x < 120; x++)
511.                 {
512.                     std::cout << " ";
513.                 }
514.                 gotoxy(0, 10);
515.             }
516.         }
517.         for (int j = 0; j <= large - 1; j++)
518.         {
519.             for (int i = 0; i <= 2 * large - 2; i++)
520.             {
521.                 if (j < large)
522.                 {
523.                     if (SMch[j][i].returnMGCempty())
524.                     {
525.                         fout << "U" << U << " : " + strMGC << std::endl;
526.                         fout << "  port map(" << std::endl;
527.                         for (int k = 0; k < countInputBits; k++)
528.                         {
529.                             fout << "       A(" << k << ") =>" << "A(" <<
(SMnch[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
530.                         }
531.                         for (int k = 0; k < countInputBits; k++)
532.                         {
533.                             fout << "       B(" << k << ") =>" << "B(" <<
(SMch[j][i].ReturnB() * countInputBits) + k << ")," << std::endl;
534.                         }
535.                         if (SMch[j][i].ReturnC() == -5)
536.                         {
537.                             fout << "       C => L," << std::endl;
538.                         }
539.                         else
540.                         {
541.                             fout << "       C" << " => " << "BUS" <<
SMch[j][i].ReturnCbus() << "," << std::endl;
542.                         }
543.                         fout << "       S" << " => " << "BUS" <<
SMch[j][i].ReturnSbus() << std::endl;
544.                         fout << "  );" << std::endl;
545.                         fout << std::endl;
```

```
546.                         U++;
547.                     }
548.                 }
549.                 else
550.                 {
551.                     if (SMch[j][i].returnSMchEmpty())
552.                     {
553.                         fout << "U" << U << " : " + strMGC << std::endl;
554.                         fout << "  port map(" << std::endl;
555.                         for (int k = 0; k < countInputBits; k++)
556.                         {
557.                             fout << "       A(" << k << ") =>" << "p(" <<
(SMch[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
558.                         }
559.                         fout << "       B" << " => " << "BUS" <<
SMch[j][i].ReturnBbus() << "," << std::endl;
560.                         if (SMch[j][i].ReturnC() != -5)
561.                         {
562.                             fout << "       C" << " => " << "BUS" <<
SMch[j][i].ReturnCbus() << "," << std::endl;
563.                         }
564.                         if (SMch[j][i].ReturnS() == -1)
565.                         {
566.                             fout << "       S" << " => " << "BUS" <<
SMch[j][i].ReturnSbus() << std::endl;
567.                             fout << "  );" << std::endl;
568.                         }
569.                         else
570.                         {
571.                             for (int k = 0; k < countInputBits; k++)
572.                             {
573.                                 if (k == countInputBits - 1)
574.                                 {
575.                                     fout << "       S(" << k << ") =>" <<
"R(" << (SMch[j][i].ReturnS() * countInputBits) + k << ")" << std::endl;
576.                                 }
577.                                 else
578.                                 {
579.                                     fout << "       S(" << k << ") =>" <<
"R(" << (SMch[j][i].ReturnS() * countInputBits) + k << ")" << "," << std::endl;
580.                                 }
581.                             }
582.                             fout << "  );" << std::endl;
583.                         }
584.                         fout << std::endl;
585.                         U++;
586.                     }
587.                 }
588.             }
589.             if (countProcessing < 120)
590.             {
591.                 std::cout << char(221);
592.                 countProcessing++;
593.             }
594.             else
595.             {
596.                 gotoxy(0, 10);
597.                 countProcessing = 0;
598.                 for (int x = 0; x < 120; x++)
599.                 {
600.                     std::cout << " ";
601.                 }
602.                 gotoxy(0, 10);
```

```
603.             }
604.         }
605.         for (int i = 0; i < large - 1; i++)
606.         {
607.             fout << "U" << U << " : " + strF << std::endl;
608.             fout << "  port map(" << std::endl;
609.             fout << "       A" << " => " << "BUS" << Sn[i].ReturnAbus() <<
"," << std::endl;
610.             fout << "       B" << " => " << "BUS" << Sn[i].ReturnBbus() <<
std::endl;
611.             fout << "  );" << std::endl;
612.             fout << std::endl;
613.             U++;
614.         }
615.         for (int i = 0; i < large - 1; i++)
616.         {
617.             fout << "U" << U << " : " + strF << std::endl;
618.             fout << "  port map(" << std::endl;
619.             fout << "       A" << " => " << "BUS" << Rn[i].ReturnAbus() <<
"," << std::endl;
620.             fout << "       B" << " => " << "BUS" << Rn[i].ReturnBbus() <<
std::endl;
621.             fout << "  );" << std::endl;
622.             fout << std::endl;
623.             U++;
624.         }
625.         for (int i = 0; i < large - 1; i++)
626.         {
627.             fout << "U" << U << " : " + strF << std::endl;
628.             fout << "  port map(" << std::endl;
629.             fout << "       A" << " => " << "BUS" << SUM[i].ReturnAbus() <<
"," << std::endl;
630.             fout << "       B" << " => " << "BUS" << SUM[i].ReturnBbus() <<
std::endl;
631.             fout << "       C" << " => " << "BUS" << SUM[i].ReturnCbus() <<
std::endl;
632.             fout << "       Co" << " => " << "BUS" << SUM[i].ReturnCobus()
<< std::endl;
633.             fout << "  );" << std::endl;
634.             fout << std::endl;
635.             U++;
636.         }
637.         fout << std::endl;
638.         fout << std::endl;
639.         fout << "end " + strMultiplier + ";";
640.     }
641.     else if (countInputBits = 1)
642.     {
643.         fout << "library IEEE;" << std::endl;
644.         fout << "use IEEE.STD_LOGIC_1164.all;" << std::endl;
645.         fout << std::endl;
646.         fout << "entity " << strMultiplier << " is" << std::endl;
647.         fout << "       port(" << std::endl;
648.         fout << "          A : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
649.         fout << "          B : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
650.         fout << "          C : in STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0);" << std::endl;
651.         fout << "          R : out STD_LOGIC_VECTOR(" << (countInputBits *
large) - 1 << " downto 0)" << std::endl;
652.         fout << "          );" << std::endl;
653.         fout << "end " << strMultiplier << ";" << std::endl;
```

```
654.          fout << std::endl;
655.          fout << std::endl;
656.          fout << "architecture" << strMultiplier << "of " << strMultiplier
<< " is" << std::endl;
657.          fout << std::endl;
658.          fout << std::endl;
659.          fout << "component " + strMGC << std::endl;
660.          fout << "  port (" << std::endl;
661.          fout << "       A : in STD_LOGIC;" << std::endl;
662.          fout << "       B : in STD_LOGIC;" << std::endl;
663.          fout << "       C : in STD_LOGIC;" << std::endl;
664.          fout << "       S : out STD_LOGIC" << std::endl;
665.          fout << "       );" << std::endl;
666.          fout << "end component;" << std::endl;
667.          fout << std::endl;
668.          fout << std::endl;
669.          for (int i = 0; i < busCount; i++)
670.          {
671.              fout << "signal BUS" << i << " : STD_LOGIC;" << std::endl;
672.          }
673.          fout << std::endl;
674.          fout << std::endl;
675.          fout << "begin" << std::endl;
676.          fout << std::endl;
677.          fout << std::endl;
678.          for (int j = 0; j <= large - 1; j++)
679.          {
680.              for (int i = 0; i <= 2 * large - 2; i++)
681.              {
682.                  if (j < large)
683.                  {
684.                      if (SMn[j][i].returnMGCempty())
685.                      {
686.                          fout << "U" << U << " : " << strMGC << std::endl;
687.                          fout << "  port map(" << std::endl;
688.                          for (int k = 0; k < countInputBits; k++)
689.                          {
690.                              fout << "       A =>" << "A(" <<
(SMn[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
691.                          }
692.                          for (int k = 0; k < countInputBits; k++)
693.                          {
694.                              fout << "       B =>" << "B(" <<
(SMn[j][i].ReturnB() * countInputBits) + k << ")," << std::endl;
695.                          }
696.                          if (SMn[j][i].ReturnC() == -5)
697.                          {
698.                              fout << "       C => L," << std::endl;
699.                          }
700.                          else
701.                          {
702.                              fout << "       C" << " => " << "BUS" <<
SMn[j][i].ReturnCbus() << "," << std::endl;
703.                          }
704.                          fout << "       S" << " => " << "BUS" <<
SMn[j][i].ReturnSbus() << std::endl;
705.                          fout << "  );" << std::endl;
706.                          fout << std::endl;
707.                          U++;
708.                      }
709.                  }
710.                  else
711.                  {
```

```
712.                     if (SMn[j][i].returnSMnEmpty())
713.                     {
714.                         fout << "U" << U << " : " << strMGC << std::endl;
715.                         fout << "  port map(" << std::endl;
716.                         for (int k = 0; k < countInputBits; k++)
717.                         {
718.                             fout << "       A =>" << "p(" <<
(SMn[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
719.                         }
720.                         fout << "       B" << " => " << "BUS" <<
SMn[j][i].ReturnBbus() << "," << std::endl;
721.                         if (SMn[j][i].ReturnC() != -5)
722.                         {
723.                             fout << "       C" << " => " << "BUS" <<
SMn[j][i].ReturnCbus() << "," << std::endl;
724.                         }
725.                         if (SMn[j][i].ReturnS() == -1)
726.                         {
727.                             fout << "       S" << " => " << "BUS" <<
SMn[j][i].ReturnSbus() << std::endl;
728.                             fout << "  );" << std::endl;
729.                         }
730.                         else
731.                         {
732.                             for (int k = 0; k < countInputBits; k++)
733.                             {
734.                                 if (k == countInputBits - 1)
735.                                 {
736.                                     fout << "       S =>" << "R(" <<
(SMn[j][i].ReturnS() * countInputBits) + k << ")" << std::endl;
737.                                 }
738.                                 else
739.                                 {
740.                                     fout << "       S =>" << "R(" <<
(SMn[j][i].ReturnS() * countInputBits) + k << ")," << std::endl;
741.                                 }
742.                             }
743.                             fout << "  );" << std::endl;
744.                         }
745.                         fout << std::endl;
746.                         U++;
747.                     }
748.                 }
749.             }
750.             for (int j = 0; j <= large - 1; j++)
751.             {
752.                 for (int i = 0; i <= 2 * large - 2; i++)
753.                 {
754.                     if (j < large)
755.                     {
756.                         if (SMch[j][i].returnMGCempty())
757.                         {
758.                             fout << "U" << U << " : " << strMGC <<
std::endl;
759.                             fout << "  port map(" << std::endl;
760.                             for (int k = 0; k < countInputBits; k++)
761.                             {
762.                                 fout << "       A =>" << "A(" <<
(SMch[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
763.                             }
764.                             for (int k = 0; k < countInputBits; k++)
765.                             {
```

```
766.                                fout << "       B =>" << "B(" <<
(SMch[j][i].ReturnB() * countInputBits) + k << ")," << std::endl;
767.                            }
768.                            if (SMch[j][i].ReturnC() == -5)
769.                            {
770.                                fout << "       C => L," << std::endl;
771.                            }
772.                            else
773.                            {
774.                                fout << "       C" << " => " << "BUS" <<
SMch[j][i].ReturnCbus() << "," << std::endl;
775.                            }
776.                            fout << "       S" << " => " << "BUS" <<
SMch[j][i].ReturnSbus() << std::endl;
777.                            fout << "  );" << std::endl;
778.                            fout << std::endl;
779.                            U++;
780.                        }
781.                    }
782.                    else
783.                    {
784.                        if (SMch[j][i].returnSMnEmpty())
785.                        {
786.                            fout << "U" << U << " : " << strMGC <<
std::endl;
787.                            fout << "  port map(" << std::endl;
788.                            for (int k = 0; k < countInputBits; k++)
789.                            {
790.                                fout << "       A =>" << "p(" <<
(SMch[j][i].ReturnA() * countInputBits) + k << ")," << std::endl;
791.                            }
792.                            fout << "       B" << " => " << "BUS" <<
SMch[j][i].ReturnBbus() << "," << std::endl;
793.                            if (SMch[j][i].ReturnC() != -5)
794.                            {
795.                                fout << "       C" << " => " << "BUS" <<
SMch[j][i].ReturnCbus() << "," << std::endl;
796.                            }
797.                            if (SMch[j][i].ReturnS() == -1)
798.                            {
799.                                fout << "       S" << " => " << "BUS" <<
Sch[j][i].ReturnSbus() << std::endl;
800.                                fout << "  );" << std::endl;
801.                            }
802.                            else
803.                            {
804.                                for (int k = 0; k < countInputBits; k++)
805.                                {
806.                                    if (k == countInputBits - 1)
807.                                    {
808.                                        fout << "       S =>" << "R(" <<
(SMch[j][i].ReturnS() * countInputBits) + k << ")" << std::endl;
809.                                    }
810.                                    else
811.                                    {
812.                                        fout << "       S =>" << "R(" <<
(SMch[j][i].ReturnS() * countInputBits) + k << ")," << std::endl;
813.                                    }
814.                                }
815.                                fout << "  );" << std::endl;
816.                            }
817.                            fout << std::endl;
818.                            U++;
```

```
819.                         }
820.                     }
821.                 }
822.             }
823.             if (countProcessing < 120)
824.             {
825.                 std::cout << char(221);
826.                 countProcessing++;
827.             }
828.             else
829.             {
830.                 gotoxy(0, 10);
831.                 countProcessing = 0;
832.                 for (int x = 0; x < 120; x++)
833.                 {
834.                     std::cout << " ";
835.                 }
836.                 gotoxy(0, 10);
837.             }
838.         }
839.         fout << std::endl;
840.         fout << std::endl;
841.         fout << "end " + strMultiplier + ";";
842.     }
843.     gotoxy(0, 10);
844.     for (int x = 0; x < 120; x++)
845.     {
846.         std::cout << " ";
847.     }
848.     gotoxy(0, 10);
849.     std::cout << "Processing is finished" << std::endl;
850. }
```