НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кваліфікаційна наукова
праця на правах рукопису

Іванова Олена Миколаївна

УДК 004.056.5:004.415.24

ДИСЕРТАЦІЯ

МОДЕЛІ ТА МЕТОДИ ГІБРИДНОГО МАСКУВАННЯ ДАНИХ У ПРОЦЕСІ МОНІТОРИНГУ ПРОГРАМНОГО КОДУ FPGA-КОМПОНЕНТІВ ІНФОРМАЦІЙНИХ СИСТЕМ

Спеціальність 122 – Комп’ютерні науки
Галузь знань: 12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії.

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело
_____ О. М. Іванова

Науковий керівник – Антощук Світлана Григорівна, доктор технічних наук, професор

Одеса – 2026

АНОТАЦІЯ

Іванова О.М. Моделі та методи гібридного маскування даних у процесі моніторингу програмного коду FPGA-компонентів інформаційних систем. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 122 – «Комп'ютерні науки». – Національний університет «Одеська політехніка» Міністерства освіти і науки України, Одеса, 2026.

У **вступі** обґрунтовано актуальність та доцільність дисертаційного дослідження; показано зв'язок роботи з науковими програмами, планами, темами; сформульовано мету та задачі дослідження, наукову новизну та практичне значення одержаних результатів; визначено методи дослідження; зазначено особистий внесок здобувача, наведено дані про апробацію результатів дисертаційних досліджень та публікації.

В першому розділі проведено аналіз існуючих підходів, методів та засобів забезпечення моніторингу програмного коду FPGA-компонентів інформаційних систем. Визначено місце FPGA серед сучасних програмно-керованих компонентів комп'ютерних інформаційних систем. Показано, що FPGA використовуються переважно у складі високопродуктивних систем, що обумовлено придатністю структури цих компонентів для природньої організації паралельних обчислень. Це також значною мірою є мотивацією до використання FPGA в складі комп'ютерних інформаційних систем критичного застосування.

Проведено аналіз інцидентів останніх років, пов'язаних зі дестабілізацією роботи технічних об'єктів підвищеного ризику. Встановлено, що найзначнішим чинником збоїв в роботі критично важливих комп'ютерних систем є зловмисне втручання в їх функціонування. Виконано аналіз методів та засобів протидії втручанню в функціонування FPGA-компонентів інформаційних систем. Показано, що одним з найефективніших підходів до протидії втручанню є оперативний

моніторинг стану програмного коду цих компонентів. Виконання моніторингу потребує формування контрольних даних в момент підготовки програмного коду моніторингу, та їх зберігання з метою використання під час виконання актів моніторингу. Показано, що основним способом обходу моніторингу програмного коду для здійснення втручання є фальсифікація контрольних даних, яка найчастіше реалізується в інсайдерський спосіб зсередини організації. Найбільш суттєвими факторами, які обумовлюють можливість втручання визначено спосіб ті місце зберігання контрольних даних моніторингу.

Розглянуто існуючі підходи до зберігання контрольних даних систем моніторингу програмного коду. Встановлено, що підходи, які основані на збереженні контрольних даних разом з інформаційним об'єктом програмного коду в файлової системі або пам'яті, або які базуються на приєднанні контрольних даних до інформаційного об'єкта програмного коду і зберіганні їх в якості його частини, мають наступні недоліки: очевидність факту виконання моніторингу; доступність еталонних контрольних даних для зчитування, аналізу та можливої фальсифікації. Підходи, в межах яких контрольні дані зберігаються в централізованій базі даних та отримуються з неї в момент виконання процедури моніторингу пов'язані з проблемою організації доступу до цієї бази даних; складністю захисту бази даних від витоків та не зменшують можливість інсайдерського втручання.

Наявні підходи до зберігання контрольних даних, що базуються на стеганографічних методах, мають переваги, які полягають у приховуванні контрольних даних та факту виконання моніторингу, а також забезпечують утворення єдиного цілого між об'єктом програмного коду та контрольними даними. Такі можливості зазначених підходів забезпечені еквівалентними перетвореннями програмного коду. Однак ці підходи мають і недоліки, які полягають у відносно малому доступному обсязі даних, які можна зберегти в стеганографічний спосіб.

Спроби збільшити цей обсяг в межах наявних стеганографічних підходів стикається з *протиріччям* між тим фактом, що програмний код FPGA являє собою точні дані, спотворення яких не є можливим та неможливістю забезпечення достатнього для задач прихованого моніторингу обсягу зберігання даних за рахунок еквівалентних перетворень програмного коду FPGA.

З урахуванням зазначених факторів виконано обґрунтування напрямку досліджень, направленого на збільшення доступного обсягу для замаскованого зберігання контрольних даних в процесі прихованого моніторингу програмного коду FPGA-компонентів комп'ютерних інформаційних систем шляхом розробки моделей та методів гібридного стеганографічного зберігання цих даних що дає змогу *розірвати вказане протиріччя*. В результаті обґрунтування напрямку досліджень сформульовано мету та визначено задачі дисертації.

У **другому розділі** сформульовано твердження, про те, що при реалізації наближеної обробки даних на FPGA-компонентах, ця наближеність переноситься на точно поданий програмний код FPGA-компонентів. Наслідком цього є можливість замаскованого стеганографічного вбудовування в нееквівалентний спосіб (подібне до того, що використовується стосовно інформаційних контейнерів з наближено поданими елементарними одиницями) додаткових даних в точно поданий інформаційний контейнер, яким є програмний код FPGA-компонентів. Це твердження набуло формалізації у вигляді моделей, які виділяють два види надлишкових ресурсів у складі програмного коду FPGA, що можуть бути використані зі метою замаскованого зберігання даних: а) несуттєві блоки LUT в структурі тих осередків FPGA-компонентів, які виконують наближену обробку даних; б) несуттєві розряди програмного коду блоків LUT при виконанні наближеної обробки даних.

Для визначення множини надлишкових інформаційних ресурсів програмного коду FPGA, які виникають в процесі виконання наближених обчислень, та можуть бути використані для стеганографічного зберігання

контрольних даних, розроблено модель замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, *яка основана* на використанні нееквівалентних перетворень програмного коду FPGA *та враховує* особливості виконання наближених арифметичних операцій в середовищі FPGA.

Експериментальне дослідження, проведене в середовищі розробленого програмного забезпечення, яке реалізує запропоновану модель, підтвердило наявність надлишкових інформаційних ресурсів програмного коду FPGA, а також дозволило їх локалізувати та оцінити їх частку в загальному обсязі програмного коду блоків LUT FPGA-проєктів. За результатами експериментів отримано верхню оцінку кількості несуттєвих блоків LUT для проєктів, що були використані в експерименті, в діапазоні 39.35 % … 42.52 % та нижню оцінку кількості зазначених блоків в діапазоні 16.67 % … 29.91 % від загальної кількості блоків LUT в проєкті. Експериментально отримано оцінку частки несуттєвих розрядів програмних кодів блоків LUT для обчислювачів, що виконують наближену обробку даних, яка склала для експериментальних проєктів 16.8 % … 26.3 % від сукупної кількості розрядів програмних кодів блоків LUT у відповідних FPGA-проєктах.

Сформульовано перший пункт наукової новизни*: вперше* розроблено модель замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, *яка основана* на використанні нееквівалентних перетворень програмного коду FPGA *та враховує* особливості виконання наближених арифметичних операцій в середовищі FPGA, *що дозволяє* визначити множину надлишкових інформаційних ресурсів, які виникають в процесі виконання таких обчислень, та можуть бути використані для прихованого зберігання контрольних даних.

Для збільшення обсягу, доступного для прихованого зберігання додаткових даних в середовищі програмного коду FPGA-компонентів та необхідного в задачах моніторингу характеристик безпеки програмного

коду FPGA, розроблено метод нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, *який характеризується* використанням надлишковості програмного коду FPGA, яка виникає при виконанні наближеної обробки даних.

Виконано експериментальне дослідження функціонування реалізації запропонованого методу. Додатковий обсяг для замаскованого зберігання даних, забезпечений запропонованим методом склав для виділеного надлишкового стеганографічного ресурсу програмних кодів несуттєвих блоків LUT в експериментальних FPGA-проєктах за верхньою оцінкою: від 12 до 145 біт при використанні з метою замаскованого збереження даних тільки одного розряду програмного коду несуттєвих блоків LUT та від 192 до 2320 біт при використанні всіх розрядів програмного коду зазначених блоків; та за нижньою оцінкою від 5 до 102 біт при використанні одного розряду програмного коду блоків LUT та від 80 до 1632 біт при використанні всіх розрядів програмного коду зазначених блоків. Для експериментальних FPGA-проєктів кількість несуттєвих розрядів програмного коду блоків LUT склала від 89 до 690.

Запропоновані та експериментально досліджені в даному розділі дисертації надлишкові інформаційні ресурси додають до обсягів прихованого зберігання даних в програмному коді FPGA, забезпечених відомими методами, додаткові обсяги для зберігання контрольних даних, що дає змогу збереження контрольних даних більшої кількості видів оперативного моніторингу та додає варіативності у виборі розміру виходу хеш-функцій, використовуваних для отримання контрольних даних.

Сформульовано другий пункт наукової новизни: *вперше* розроблено метод нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, *що характеризується* використанням надлишковості програмного коду FPGA, яка виникає при виконанні наближених арифметичних операцій, *що дозволяє* збільшити

обсяг, доступний для прихованого зберігання контрольних даних в задачах моніторингу характеристик безпеки програмного коду FPGA.

В **третьому розділі** дисертації запропоновано підходи до гібридного замаскованого зберігання даних в програмному коді FPGA, які поєднують еквівалентні та нееквівалентні перетворення програмного коду для вбудовування даних, а також дають можливість застосування відновної обфускації в процесі вбудовування додаткових даних в програмний код.

Для збільшення обсягу контрольних даних, які можуть бути приховано вбудованими в програмний код FPGA, не збільшуючи здатність традиційних методів стегоаналізу до виявлення цих даних в програмному коді, розроблено метод гібридного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA, *який відрізняється* поєднанням еквівалентного та нееквівалентного підходів до перетворення програмного коду FPGA, а також поєднанням процесів стеганографічного вбудовування та відновної обфускації даних.

Гібридність методу, полягає в двох видах комбінування властивостей відомого еквівалентного підходу та, запропонованого в даній роботі, нееквівалентного підходу до замаскованого зберігання контрольних даних в програмному коді FPGA:

1) комбінування стеганографічних ресурсів, забезпечених еквівалентними та нееквівалентними перетвореннями програмного коду FPGA, для досягнення потрібного обсягу даних в середовищі програмного коду та послідовному виділенні цих ресурсів для мінімізації обчислювальної складності їх застосування;

2) комбінування стеганографічного вбудовування (з використанням обох підходів) даних в програмний код FPGA та зворотної (відновної) обфускації програмного коду FPGA, базованої на моделі еквівалентних перетворень, та призначеної для ускладнення стегоаналізу і виявлення прихованих в програмному коді даних.

Виконано експериментальне дослідження ефективності розробленого гібридного методу для задач прихованого зберігання контрольних даних в процесі моніторингу програмного коду FPGA-компонентів. Для експериментальних FPGA проєктів збільшення обсягу потенційно доступних додаткових даних за рахунок використання надлишкових інформаційних ресурсів несуттєвих блоків LUT при застосуванні швидкого способу локалізації цих блоків склало від 7,1% до 40,8% та в середньому 22,6%, а при застосуванні базового способу локалізації зазначене збільшення склало від 9,2% до 73,4% та в середньому 34,0%.

Збільшення обсягу потенційно доступних додаткових даних за рахунок використання крім цього також надлишкових інформаційних ресурсів несуттєвих розрядів суттєвих блоків LUT при застосуванні швидкого способу локалізації цих блоків склало від 31,7% до 138,9% та в середньому 93,7%. При застосуванні базового способу локалізації зазначене збільшення склало від 33,8% до 157,9% та в середньому 105,1%.

Виконано експериментальну оцінку ймовірності виявлення відомими методами та засобами стегоаналізу наявності даних, приховано вбудованих в програмний код FPGA, запропонованим гібридним методом. Оцінка ймовірності наявності додаткових даних при їх вбудовуванні гібридним методом без виконання етапу відновної обфускації збільшується в середньому на 2,31%, порівняно з застосуванням еквівалентного підходу, та зменшується на 6,43% в разі виконання відновної обфускації.

Сформульовано третій пункт наукової новизни: *вперше* розроблено метод гібридного замаскованого зберігання даних в середовищі програмного коду FPGA, *який відрізняється* поєднанням еквівалентного та нееквівалентного підходів до перетворення програмного коду FPGA, а також поєднанням процесів стеганографічного вбудовування та відновної обфускації даних, *що дозволяє* збільшити обсяг контрольних даних, які можуть бути приховано вбудованими в програмний код FPGA, не збільшуючи при цьому здатність традиційних методів стегоаналізу до виявлення цих даних в програмному коді.

Для введення можливості балансування між обсягом частини програмного коду, яка модифікується в результаті замаскованого вбудовування даних та ефективним обсягом стеганографічного контейнера, потрібним для збереження контрольних даних, розроблено метод формування стеганографічного ключа для прихованого збереження контрольних даних в програмному коді FPGA, *який відрізняється* можливістю адаптації до структури зв'язків між елементарними одиницями FPGA та до необхідного обсягу контрольних даних. Метод може бути застосований в якості доповнення як до традиційних методів стеганографічного вбудовування даних в програмний код FPGA, так і до гібридного методу.

Виконано експериментальне дослідження ефективності розробленого методу формування стеганографічного ключа, які показали середнє збільшення на 22,8% ефективного обсягу цифрового водяного знака, який містить контрольні данні та може бути приховано вбудований в програмний код FPGA.

Сформульовано четвертий пункт наукової новизни: *дістав подальшого розвитку* метод формування стеганографічного ключа для замаскованого збереження даних в програмному коді FPGA, *який відрізняється* можливістю адаптації до структури зв'язків між елементарними одиницями FPGA та до необхідного обсягу контрольних даних, *що дозволяє* виконувати балансування між обсягом частини програмного коду, яка модифікується в результаті вбудовування даних та ефективним обсягом стеганографічного контейнера, потрібним для збереження контрольних даних.

В **четвертому розділі** дисертаційної роботи розроблено підсистему гібридного замаскованого зберігання контрольних даних в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів. Сформульовано мету та задачі функціонування зазначеної підсистеми у складі головної інформаційної системи.

Специфіковано сукупність інформаційних потоків підсистеми, яка визначає вхідні та вихідні дані, інформаційну взаємодію з іншими компонентами систем, зовнішнім програмним середовищем та користувачами. Визначено перелік функціональних та нефункціональних вимог до підсистеми. Для розроблюваного програмного забезпечення створено діаграми логічного уявлення, розгортання, визначено структуру програмного проєкту та виконано аналіз програмного коду, що в сукупності формалізувало архітектуру підсистеми. Проведено комплексне тестування розробленої підсистеми, що довело її готовність до використання в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів.

Запропоновані в дисертаційній роботі методи та розроблені на їх основі програмні засоби були впроваджені в науково-дослідницьку діяльність та навчальний процес Національного університету «Одеська політехніка».

Ключові слова: зберігання інформації в комп'ютерних системах; доступу до інформації в комп'ютерних системах; замасковане зберігання даних, прихований моніторинг програмного коду FPGA-компонентів інформаційних систем, обробка наближених даних, обфускація програмного коду.

Список публікацій здобувача за темою дисертації

Праці, які відображають основні наукові результати дисертації

1. Антощук С.Г., Іванова О.М. Швидка локалізація осередків програмного коду FPGA, придатних для стеганографічного зберігання додаткових даних. *Вісник Херсонського національного технічного університету*. Херсон, 2025. № 4 (95) Ч. 3. С. 9 – 14. DOI: https://doi.org/10.35546/kntu2078-4481.2025.4.3.1. *Видання включено до переліку наукових фахових видань України (категорія Б).*

https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/1265/1215

2. Антощук С.Г., Іванова О.М. Метод формування стего-ключа для збільшення обсягу прихованого зберігання даних в середовищі програмного коду FPGA. *Вісник Херсонського національного технічного університету*. Херсон, 2025. № 1 (92). С. 9 – 16. DOI: https://doi.org/10.35546/kntu2078-4481.2025.1.2.1. *Видання включено до переліку наукових фахових видань України (категорія Б).*

https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/836/803

3. Антощук С.Г., Іванова О.М. Підхід до збільшення обсягу прихованого зберігання даних, призначених для моніторингу FPGA-компонентів комп'ютерних систем. *Вчені записки Таврійського національного університету. Серія: «Технічні науки»*. Київ, 2023. Том 34 (73), № 6. DOI: https://doi.org/10.32782/2663-5941/2023.6/08. *Видання включено до переліку наукових фахових видань України (категорія Б)*.

https://www.tech.vernadskyjournals.in.ua/journals/2023/6_2023/8.pdf

4. Іванова О.М., Дрозд О.В., Защолкін К.В., Кузнєцов М.О. Підхід до нееквівалентного стеганографічного вбудовування додаткових даних в програмний код блоків LUT FPGA. *Вісник Кременчуцького національного університету імені М. Остроградського*. Кременчук, 2021. № 6 (131). С. 60–65. DOI: https://doi.org/10.30929/1995-0519.2021.6.60-65. *Видання включено до переліку наукових фахових видань України (категорія Б).*

https://visnikkrnu.kdu.edu.ua/statti/2021_6_2021-6-60-65.pdf

5. Zashcholkin K., Drozd O., Antoshchuk S., Ivanova O., Sachenko O. Steganographic Resources of FPGA-based Systems for Approximate Data Processing. *CEUR-WS*. 2021. Vol. 2864. P. 324-333. *Наукове періодичне іноземне видання, Німеччина, ISSN 1613-0073. Видання включено до наукометричної бази* ***SCOPUS***

https://ceur-ws.org/Vol-2864/paper28.pdf

6. Zashcholkin K., Drozd O., Ivanova O., Shaporin R., Kuznietsov M. An Approach to Stego-Container Organization in FPGA Systems for Approximate Data Processing. *CEUR-WS*. 2021. Vol. 2853. P. 527–536. *Наукове періодичне іноземне видання, Німеччина, ISSN 1613-0073. Видання включено до наукометричної бази* ***SCOPUS***

https://ceur-ws.org/Vol-2853/paper55.pdf

7. Ivanova O., Drozd O., Zashcholkin K., Sulima Y. Combined Use of Equivalent and Non-Equivalent Transformations of FPGA Program Code to Embedding Additional Security Data. *IEEE East-West Design and Test Symposium (EWDTS)*. 2021. P. 191 – 195. DOI: https://doi.org/10.1109/EWDTS52692.2021.9580984. *Наукове періодичне іноземне видання, США, ISSN: 2373-826X. Видання включено до наукометричної бази* ***SCOPUS***

https://ieeexplore.ieee.org/abstract/document/9580984

8. Zashcholkin K., Drozd O., Ivanova O., Bykovyy P. Formation of the Interval Stego Key for the Digital Watermark Used in Integrity Monitoring of FPGA-based Systems. *CEUR-WS*. 2020. Vol. 2623. P. 267 – 276. *Наукове періодичне іноземне видання, Німеччина, ISSN 1613-0073. Видання включено до наукометричних баз* ***SCOPUS*** *та* ***Web of Science Core Collection***

https://ceur-ws.org/Vol-2623/paper23.pdf

9. Zashcholkin K., Drozd O., Shaporin R., Ivanova O., Drozd M. Co-Embedding Additional Security Data and Obfuscating Low-Level FPGA Program Code. *IEEE East-West Design and Test Symposium (EWDTS)*. 2020. P. 115 – 119. DOI: https://doi.org/10.1109/EWDTS50664.2020.9225111. *Наукове періодичне іноземне видання, США, ISSN: 2373-826X. Видання включено до наукометричної бази* ***SCOPUS***

https://ieeexplore.ieee.org/document/9225111

10. Патент на винахід № 122276 Україна, МПК G06F 11/263 (2006.01), G06F 7/544 (2006.01). Програмований пристрій для обчислення

логічної функції N змінних / К.В. Защолкін, О.В. Дрозд, Р.О. Шапорін, О.М. Іванова, Ю.В. Дрозд; заявник Одеський національний політехнічний університет. – № a201811671; заявлено 27.11.2018; опубліковано 12.10.2020; Бюл. № 19/2020.

https://sis.nipo.gov.ua/uk/search/detail/1458192

Наукові праці апробаційного характеру

11. Антощук С.Г., Іванова О.М., Защолкін К.В. Стеганографічне вбудовування додаткових даних в програмний код мікросхем FPGA. *Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем, MEICS-2023*: матеріали VIII Всеукраїнської науково-практичної конференції. Дніпро, 2023.

https://www.dnu.dp.ua/docs/ndc/2023/materiali%20konf/25_MEICS-2023.pdf

12. Іванова О.М., Дрозд О.В., Защолкін К.В. Особливості стеганографічного нееквівалентного вбудовування цифрових водяних знаків в програмний код FPGA. *Інформатика, управління та штучний інтелект, ІУШІ-2021*: тези VIII Міжнародної науково-технічної конференції. Харків, 2021.

https://web.kpi.kharkov.ua/ai/wp-content/uploads/sites/249/2024/10/TEZY_YUYY_2021.pdf

13. Іванова О.М., Михайлов Д.О., Защолкін К.В. Підхід до 3D стеганографічного вбудовування даних та його програмна реалізація. *Сучасні інформаційні технології, МІТ-2021*: матеріали XI Міжнародної наукової конференції. Одеса, 2021. С. 36 – 37.

Публікації, які додатково висвітлюють результати дисертації

14. Drozd O., Maevsky D., Maevskaya O., Martynyuk O., Parkhomenko A., Gladkova O., Drozd M., Ivanova O., Surkov S., Zashcholkin K. Internet of Things for Smart Building and City. Ministry of Education and Science of Ukraine, 2019. 156 p.

https://aliot.eu.org/wp-content/uploads/2020/01/ALIOT_ITM2_IoT-for-Smart-Build-and-City_web.pdf

ABSTRACT

Ivanova O.M. Models and methods for hybrid data masking in the process of monitoring the program code of FPGA-components of information systems. – Qualification scientific work in the form of manuscript.

Thesis for the PhD degree in specialty 122 Computer science. – Odesa Polytechnic National University, Ministry of Education and Science of Ukraine, Odesa, 2026.

The **introduction** substantiates the relevance and practicability of the dissertation research; demonstrates the connection between the work and scientific programs, plans, and topics; formulates the purpose and objectives of the research, the scientific novelty and practical significance of the results obtained; defines the research methods; the applicant's personal scientific contribution is indicated, data on the testing of the dissertation research results are provided, as well as information on publications.

The **first chapter** analyzes existing approaches, methods, and means of monitoring the program code of FPGA-components in information systems. The role of FPGAs among modern software-controlled components of computer information systems has been identified. It is shown that FPGAs are mainly used in high-performance computer systems, which is due to the suitability of the structure of these chips for the natural organization of parallel computing. This is also a significant motivation for the use of FPGA in critical computer systems.

An analysis of incidents in recent years related to the destabilization of high-risk technical facilities has been conducted. It has been established that the most significant factor in the failure of critical computer systems is malicious interference in their operation. An analysis of methods and means of counteracting interference in the operation of FPGA-components of computer systems has been carried out. It has been shown that one of the most effective approaches to countering interference is the operational monitoring of the state of the software code of these components. Monitoring requires the formation of

control data at the time the program code is prepared for monitoring, and the storage of that data for use during the execution of monitoring acts. It is shown that the main way to bypass program code monitoring for the purpose of interference is to falsify control data, which is most often done by insiders from within the organization. The most significant factor determining the possibility of interference is the method of storing control data.

Existing approaches to storing control data for program code monitoring systems are considered. It has been established that approaches based on storing control data together with the program code information object in a file system or memory, or based on attaching control data to the program code information object and storing it as part of it, have the following disadvantages: the obviousness of the fact that monitoring is being performed; the availability of reference control data for reading, analysis, and possible falsification. Approaches in which control data is stored in a centralized database and retrieved from it at the time of monitoring are associated with the problem of organizing access to this database; the complexity of protecting the database from leaks; and do not reduce the possibility of insider interference.

Existing approaches to storing control data based on the steganographic approach have advantages that consist in hiding control data and the fact of monitoring, as well as ensuring the creation of a single whole between the object of the program code and the control data. These capabilities of the above approaches are provided by equivalent transformations of the program code. However, these approaches also have disadvantages, which consist in the relatively small volume of data that can be stored in a steganographic way.

Attempts to increase this volume within the limits of existing steganographic approaches are faced with a contradiction between the fact that FPGA program code represents exact data, the distortion of which is impossible, and the impossibility of ensuring sufficient data storage volume for hidden monitoring tasks by means of equivalent transformations of FPGA program code.

Taking into account the above factors, the research direction aimed at increasing the available masked storage capacity for control data in the process of hidden monitoring of the program code of FPGA components of computer systems was justified by developing models and methods of hybrid steganographic storage of these data, which makes it possible to resolve the above contradiction. As a result of justifying the research direction, the objective of the thesis was formulated and its tasks were defined.

The **second chapter** formulates the statement that when implementing approximate data processing on an FPGA, this approximation is transferred to the exactly specified FPGA program code. The result is the possibility of masked embedding in a non-equivalent way (similar to that used for information containers with approximately presented elementary units) of additional data into a exactly specified information container, which is the FPGA program code. This statement has been formalized in the form of models that identify two types of redundant resources in the FPGA program code that can be used for steganographic purposes: a) insignificant LUT units in the structure of compute units that perform approximate data processing; b) insignificant bits of the LUT unit program code when performing approximate data processing.

To identify a set of redundant information resources in FPGA program code that arise during the execution of approximate calculations and can be used for steganographic storage of control data, a model of masked data storage in the FPGA component program code environment has been developed, which is based on the use of non-equivalent transformations of the FPGA program code and takes into account the peculiarities of performing approximate arithmetic operations in the FPGA environment.

An experimental research carried out in the environment of the developed software, which implements the proposed model, confirmed the presence of redundant information resources in the FPGA program code, and also allowed to localize them and estimate their share in the total volume of the program code of FPGA-projects. The results of the experiments obtained an upper estimate of the number of insignificant LUT units for experimental projects in the range of

39,35% to 42,52% and a lower estimate of the number of these units in the range of 16,67% to 29,91% of the total number of LUT units in the project. An estimate of the proportion of insignificant bits of LUT block program codes for computing units performing approximate data processing was also obtained experimentally, which amounted to 16,8% to 26,3% of the total number of bits of LUT unit program codes for experimental projects.

The first point of scientific novelty has been formulated: a model of masked data storage within the FPGA program code environment has been developed, which is based on the use of non-equivalent transformations of the FPGA program code and takes into account the peculiarities of performing approximate arithmetic operations in the FPGA environment, This allows us to identify a set of redundant information resources that arise in the process of performing such calculations and can be used for hidden storage of control data.

To increase the volume available for hidden storage of additional data in the FPGA program code environment and required for monitoring the security characteristics of FPGA program code, a method of non-equivalent masked data storage within the FPGA program code environment has been developed, which is characterized by the use of FPGA program code redundancy that arises during approximate data processing.

An experimental study of the results of the proposed method's implementation was conducted. The additional volume for masked data storage provided by the proposed method amounted to the following upper estimates for the allocated excessive steganographic resource of software codes of insignificant LUT units of experimental FPGA projects: from 12 to 145 bits when used for steganographic data storage of only one bit of the program code of insignificant LUT units, and from 192 to 2320 bits when using all bits of the program code of these units; and, according to the lower estimate, from 5 to 102 bits when using one bit of the LUT unit program code and from 80 to 1632 bits when using all bits of the program code of the specified blocks. For experimental FPGA projects, the number of non-essential bits of the LUT unit program code ranged from 89 to 690.

The redundant information resources proposed and experimentally researched in this chapter of the thesis add to the volume of hidden data storage in the FPGA program code, provided by known methods, additional resources for storing control data, which allows for the storage of control data for a greater number of types of operational monitoring and adds variability in the selection of the output size of hash functions used to obtain control data.

The second point of scientific novelty is formulated: a method of non-equivalent masked data storage in the FPGA program code environment has been developed, which is characterized by the use of redundancy in the FPGA program code, which arises when performing approximate arithmetic operations, allowing to increase the volume available for hidden storage of additional data in tasks of monitoring the security characteristics of FPGA program code.

The **third chapter** of the thesis proposes approaches to hybrid masked data storage in FPGA program code, which combine equivalent and non-equivalent transformations of program code during embedding, and also enable the use of recoverable obfuscation in the process of embedding additional data into program code.

To increase the volume of control data that can be hidden in the FPGA program code without increasing the ability of traditional steganalysis methods to detect this data in the program code, a method of hybrid masked storage of control data in the FPGA program code environment has been developed. This method is distinguished by a combination of equivalent and non-equivalent approaches to the transformation of FPGA program code, as well as a combination of steganographic embedding and data recoverable obfuscation processes.

The hybrid nature of the method consists of two types of combining the properties of the known equivalent approach and the non-equivalent approach proposed in this thesis for masked storage of control data in FPGA program code:

1) combining steganographic resources provided by equivalent and non-equivalent transformations of FPGA program code to achieve the required

volume of data in the program code environment and sequentially allocating these resources to minimize the computational complexity of their application;

2) combining steganographic embedding (using both approaches) of data into FPGA program code and reversible (recoverable) obfuscation of FPGA program code based on a model of equivalent transformations and designed to complicate steganalysis and detection of data hidden in program code.

An experimental research was carried out to evaluate the effectiveness of the developed hybrid method for hidden steganographic storage of control data in the process of monitoring FPGA program code. For experimental FPGA projects, the increase in the volume of potentially available additional data due to the use of the stego-resource of insignificant LUT units when applying a fast method of localizing these units ranged from 7,1% to 40,8% and averaged 22,6%, and when using the basic localization method, the increase ranged from 9,2% to 73,4% and averaged 34,0%.

The increase in the volume of potentially available additional data due to the use of redundant information resources of insignificant bits of significant LUT units when applying the fast method of localizing these units ranged from 31,7% to 138,9% and averaged 93,7%. By using the steganographic resource of insignificant bits of significant LUT units when applying the basic localization method, the increase ranged from 33,8% to 157,9% and averaged 105,1%.

An experimental assessment was performed of the probability of detecting, using known methods and means of steganalysis, the presence of data embedded in the FPGA program code using the proposed hybrid method. The probability of additional data being present when embedding using the hybrid method without performing the recoverable obfuscation stage increases by an average of 2,31% compared to using an equivalent approach, and decreases by 6,43% when performing the recoverable obfuscation stage.

The third point of scientific novelty has been formulated: a method of hybrid masked storage of control data in the FPGA program code environment has been developed, which is distinguished by a combination of equivalent and

non-equivalent approaches to the transformation of FPGA program code, as well as a combination of steganographic embedding and recoverable obfuscation of control data, which allows increasing the volume of control data that can be hidden embedded in FPGA program code without increasing the ability of traditional steganalysis methods to detect this data in the program code.

To provide a balance between the volume of program code modified as a result of steganographic data embedding and the effective volume of the steganographic container a method for forming a steganographic key for hidden storage of control data in FPGA program code has been developed, which is distinguished by its ability to adapt to the structure of connections between elementary FPGA units and to the required volume of control data. The method can be applied as a supplement to both traditional methods of steganographic data embedding in FPGA program code and to the hybrid method.

An experimental research was carried out to evaluate the effectiveness of the developed method of steganographic key formation, which showed an average increase of 22,8% in the effective volume of the digital watermark containing control data and can be hidden embedded in the FPGA program code.

The fourth point of scientific novelty has been formulated: the method of forming a steganographic key for hidden storage of control data in the FPGA program code has been further developed, which is distinguished by its ability to adapt to the structure of connections between elementary FPGA units and to the required volume of control data, allowing for a balance between the volume of the part of the program code that is modified as a result of data embedding and the effective volume of the steganographic container required to store control data.

The **fourth chapter** of the thesis develops a subsystem for hybrid masked storage of control data as part of an information system for hidden monitoring of FPGA program code. The purpose and objectives of the subsystem's functioning as part of the main information system are formulated.

A set of information flows of the subsystem is specified, which defines the input and output data of the subsystem, its information interaction with other

system components, the external software environment, and users. A list of functional and non-functional requirements for the subsystem has been defined. Diagrams of the logical representation of the subsystem under development and its deployment have been developed, the structure of the software project has been determined, and an analysis of the software code has been performed, which together formalized the architecture of the subsystem. Comprehensive testing of the developed subsystem has been carried out, proving its readiness for use as part of an information system for hidden monitoring of FPGA software code.

The methods proposed in the dissertation and the software tools developed based on them have been applied in research activities and academic courses at Odessa Polytechnic National University.

Keywords: information storage in computer systems; access to information in computer systems; masked data storage, hidden monitoring of FPGA-components program code, approximate data processing, program code obfuscation.

List of publications

Publications where the main scientific results of the dissertation are published

1. Antoschuk S., Ivanova O. Fast localization of FPGA program code elements suitable for steganographic storage of additional data. *Visnyk of Kherson National Technical University*. Kherson, 2025. N 4 (95). P. 3. P. 9 – 14. DOI: https://doi.org/10.35546/kntu2078-4481.2025.4.3.1. *The publication is included in the list of scientific professional publications of Ukraine (category B).*

https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/1265/1215

2. Antoschuk S., Ivanova O. The interval stego-key method for increasing the volume of hidden data storage in the environment of FPGA chips program code. *Visnyk of Kherson National Technical University*. Kherson, 2025. № 1 (92). С. 9 – 16. DOI: https://doi.org/10.35546/kntu2078-4481.2025.1.2.1. *The*

publication is included in the list of scientific professional publications of Ukraine (category B).

https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/836/803

3. Antoschuk S.H., Ivanova O.M. Approach to increasing the volume of hidden storage of control data for monitoring FPGA components of computer systems. *Scientific Notes of the Tavria National University. Series: "Technical Sciences".* Kyiv, 2023. Vol. 34 (73), No 6. DOI: https://doi.org/10.32782/2663-5941/2023.6/08. *The publication is included in the list of scientific professional publications of Ukraine (category B).*

https://www.tech.vernadskyjournals.in.ua/journals/2023/6_2023/8.pdf

4. Ivanova O.M., Drozd O.V., Zashcholkin K.V., Kuznietsov M.O. An approach to non-equivalent steganographic embedding of additional data into the program code of FPGA LUT units. *Transactions of Kremenchuk Mykhailo Ostrohradskyi National University.* Kremenchuk, 2021. No 6 (131). P. 60–65. DOI: https://doi.org/10.30929/1995-0519.2021.6.60-65. *The publication is included in the list of scientific professional publications of Ukraine (category B).*

https://visnikkrnu.kdu.edu.ua/statti/2021_6_2021-6-60-65.pdf

5. Zashcholkin K., Drozd O., Antoshchuk S., Ivanova O., Sachenko O. Steganographic Resources of FPGA-based Systems for Approximate Data Processing. *CEUR-WS*. 2021. Vol. 2864. P. 324-333. *Scientific periodical foreign issue, Germany, ISSN 1613-0073. The publication is included in the scientometric base* ***SCOPUS***

https://ceur-ws.org/Vol-2864/paper28.pdf

6. Zashcholkin K., Drozd O., Ivanova O., Shaporin R., Kuznietsov M. An Approach to Stego-Container Organization in FPGA Systems for Approximate Data Processing. *CEUR-WS*. 2021. Vol. 2853. P. 527–536. *Scientific periodical foreign issue, Germany, ISSN 1613-0073. The publication is included in the scientometric base* ***SCOPUS***

https://ceur-ws.org/Vol-2853/paper55.pdf

7. Ivanova O., Drozd O., Zashcholkin K., Sulima Y. Combined Use of Equivalent and Non-Equivalent Transformations of FPGA Program Code to Embedding Additional Security Data. *IEEE East-West Design and Test Symposium (EWDTS)*. 2021. P. 191 – 195. DOI: https://doi.org/10.1109/EWDTS52692.2021.9580984. 9580984. *Scientific periodical foreign issue, USA, ISSN: 2373-826X. The publication is included in the scientometric base* ***SCOPUS***

https://ieeexplore.ieee.org/abstract/document/9580984

8. Zashcholkin K., Drozd O., Ivanova O., Bykovyy P. Formation of the Interval Stego Key for the Digital Watermark Used in Integrity Monitoring of FPGA-based Systems. *CEUR-WS*. 2020. Vol. 2623. P. 267 – 276. *Scientific periodical foreign issue, Germany, ISSN 1613-0073. The publication is included in the scientometric bases* ***SCOPUS*** *and* ***Web of Science Core Collection***

https://ceur-ws.org/Vol-2623/paper23.pdf

9. Zashcholkin K., Drozd O., Shaporin R., Ivanova O., Drozd M. Co-Embedding Additional Security Data and Obfuscating Low-Level FPGA Program Code. *IEEE East-West Design and Test Symposium (EWDTS)*. 2020. P. 115 – 119. DOI: https://doi.org/10.1109/EWDTS50664.2020.9225111. *Scientific periodical foreign issue, USA, ISSN: 2373-826X. The publication is included in the scientometric base* ***SCOPUS***

https://ieeexplore.ieee.org/document/9225111

10. Patent for invention No. 122276 Ukraine, IPC G06F 11/263 (2006.01), G06F 7/544 (2006.01). Programmable device for calculating a logical function of N variables / K.V. Zashcholkin, O.V. Drozd, R.O. Shaporin, O.M. Ivanova, Y. V. Drozd; applicant Odessa National Polytechnic University. – No. a201811671; declared on 27.11.2018; published on 12.10.2020; Bull. No. 19/2020.

https://sis.nipo.gov.ua/uk/search/detail/1458192

Publications of approbation type

11. Antoschuk S.H., Ivanova O.M., Zashcholkin K.V. Steganographic embedding of additional data into the program code of FPGA. *Prospective directions of modern electronics, information and computer systems, MEICS-2023*: materials of the VIII All-Ukrainian scientific and practical conference. Dnipro, 2023.

https://www.dnu.dp.ua/docs/ndc/2023/materiali%20konf/25_MEICS-2023.pdf

12. Ivanova O.M., Drozd O.V., Zashcholkin K.V. Features of steganographic non-equivalent embedding of digital watermarks into FPGA program code. Features of steganographic non-equivalent embedding of digital watermarks into FPGA program code. Informatics, Control and Artificial Intelligence, ICAI-2021: theses of the VIII International Scientific and Technical Conference. Kharkiv, 2021.

https://web.kpi.kharkov.ua/ai/wp-content/uploads/sites/249/2024/10/TEZY_YUYY_2021.pdf

13. Ivanova O.M., Mikhailov D.O., Zashcholkin K.V. An approach to 3D steganographic data embedding and its software implementation. *Modern Information Technologies, MIT-2021*: Proceedings of the XI International Scientific Conference. Odesa, 2021. P. 36 – 37.

Publications that additionally demonstrate the results of the thesis

14. Drozd O., Maevsky D., Maevskaya O., Martynyuk O., Parkhomenko A., Gladkova O., Drozd M., Ivanova O., Surkov S., Zashcholkin K. Internet of Things for Smart Building and City. Ministry of Education and Science of Ukraine, Odessa National Polytechnic University, Zaporizhzhia National Technical University, 2019. 156 p.

https://aliot.eu.org/wp-content/uploads/2020/01/ALIOT_ITM2_IoT-for-Smart-Build-and-City_web.pdf

ЗМІСТ

ВСТУП

Актуальність теми. Комп’ютерні системи та інформаційні технології займають дедалі вагоміше місце в розвитку людства. Сьогодні більшість сфер людської діяльності пов’язано з їх використанням. За умов коректного функціонування такі системи є ефективним інструментом розвитку технологій. Водночас некоректне функціонування цих систем може привести до фінансових збитків, зменшення продуктивності технологічних процесів, а у випадку комп’ютерних систем критичного застосування – до техногенних катастроф.

Інциденти останніх років показали, що найзначнішим чинником збоїв в роботі критично важливих інформаційних систем є зловмисне втручання в їх функціонування. Для комп’ютерних компонентів з жорсткою логікою таке втручання зазвичай здійснюється шляхом фізичних впливів на апаратуру. Для програмованих компонентів до цього шляху додається втручання засобами зловмисних маніпуляцій з програмним кодом. Клас програмно-керованих компонентів є домінуючим в структурі сучасних інформаційних систем. Через це проблема захисту компонентів цього класу від зловмисних дій з програмним кодом, виходить на передній план.

В галузях, задачі яких потребують високопродуктивних програмно-керованих обчислень, вельми часто використовуються програмовані логічні інтегральні схеми, найбільш досконалими представниками яких є FPGA (Field Programmable Gate Arrays). Ці компоненти мають відмінності, як архітектурного, так і безпекового плану, від мікропроцесорів, мікроконтролерів та графічних процесорів. Можливість забезпечення високої продуктивності обчислень робить FPGA досить часто використовуваними для реалізації інформаційних систем критичного застосування. Зазначені фактори виводять FPGA в окремий *домен інтересів* щодо протидії втручанням, здійсненим через експлуатацію програмного коду.

Серед засобів захисту FPGA-компонентів інформаційних систем від зловмисного втручання особливу роль відіграє оперативний моніторинг стану їх програмного коду. Моніторинг проводиться за окремими характеристиками (цілісністю, автентичністю, шляхами розповсюдження, легітимністю використання) або комплексно за групою характеристик. Найбільш дієві підходи до організації моніторингу базуються на використанні контрольних даних, які зберігаються разом з програмним кодом або у віддаленій базі даних.

Основним способом зловмисного обходу моніторингу є фальсифікація контрольних даних, яка найчастіше реалізується в інсайдерський спосіб зсередини організації. Найбільш суттєвим фактором, що обумовлює можливість фальсифікація контрольних даних, є уразливі способи зберігання цих даних. Підходи, які основані на збереженні контрольних даних разом з інформаційним об’єктом програмного коду мають такі недоліки, як очевидність факту виконання моніторингу; доступність контрольних даних для зчитування, аналізу та можливої фальсифікації. Підходи, в межах яких контрольні дані зберігаються в централізованій базі даних пов’язані з проблемою організації доступу; складністю захисту від витоків; існуванням можливість інсайдерського втручання.

Існують також підходи до зберігання контрольних даних, що базуються на стеганографічних методах. В межах цих підходів контрольні дані маскуються шляхом прихованого вбудовування в інформаційний об’єкт програмного коду, щодо якого здійснюється моніторинг. Такі підходи мають переваги, які полягають у інформаційному маскуванні контрольних даних та приховуванні факту виконання моніторингу, а також забезпечують утворення єдиного цілого між об’єктом програмного коду та контрольними даними. Ці можливості забезпечені еквівалентними перетвореннями програмного коду. Однак такі підходи мають і недоліки, які полягають у відносно малому доступному обсязі даних, які можливо зберегти в стеганографічний спосіб.

Дефіцит доступного обсягу замаскованого зберігання контрольних даних при застосуванні еквівалентних перетворень програмного коду приводить до спрощення процесу потенційного зловмисного обходу моніторингу. Причиною такого спрощення є використання тільки контрольних даних малої розмірності та зменшення кількість видів прихованого моніторингу, які одночасно можуть бути застосовані до програмного коду FPGA-компонентів.

Спроби збільшити цей обсяг в межах наявних стеганографічних підходів до маскування контрольних даних стикаються з ***протиріччям*** між тим фактом, що програмний код FPGA являє собою точні дані, спотворення яких не є можливим, та неможливістю забезпечення достатнього для задач прихованого моніторингу обсягу зберігання даних за рахунок еквівалентних перетворень програмного коду FPGA.

Виходячи з цього ***актуальною*** науково-прикладною задачею є збільшення доступного обсягу для замаскованого зберігання контрольних даних в процесі прихованого моніторингу програмного коду FPGA-компонентів інформаційних систем шляхом розробки моделей та методів гібридного стеганографічного зберігання цих даних що дає змогу *розв'язати вказане протиріччя*.

Зв'язок роботи з науковими програмами, планами, темами. Дослідження та розробка моделей та методів гібридного інформаційно маскованого зберігання контрольних даних для прихованого моніторингу програмного коду FPGA-компонентів інформаційних систем виконувалися у відповідності до планів дослідницьких робіт Національного університету «Одеська політехніка» в межах держбюджетної НДР № 99-145 (ДР №0115U000833) "Розробка методів кратного ефекту та його фокусування для проєктування та діагностики комп'ютерних інформаційних систем і їх цифрових компонентів", а також у європейському освітньому проєкті ERASMUS+ "ALIOT" Internet of Things: Emerging Curriculum for Industry and Human Applications, 573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP.

Мета і задачі дослідження. ***Метою роботи*** є збільшення доступного обсягу для інформаційно замаскованого зберігання контрольних даних у процесі прихованого моніторингу програмного коду FPGA-компонентів інформаційних систем шляхом розробки моделей та методів гібридного стеганографічного вбудовування цих даних в програмний код.

Для досягнення мети в роботі вирішені такі ***задачі***:

– досліджено процеси забезпечення характеристик безпеки, а також характеристики правомірності використання та поширення щодо програмного коду FPGA-компонентів інформаційних систем; визначено фактори, які впливають на якість виконання моніторингу зазначених характеристик в умовах атак на систему моніторингу; визначено важливість фактору місця та способу зберігання контрольних даних прихованого моніторингу програмного коду FPGA;

– розроблено модель нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів та виконано оцінку обсягу надлишкових інформаційних ресурсів, забезпечених запропонованою моделлю, придатних для замаскованого зберігання контрольних даних;

– розроблено метод нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, оснований на запропонованій моделі та виконано експериментальне дослідження ефективності цього методу для задач прихованого моніторингу програмного коду FPGA;

– розроблено метод гібридного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA, який поєднує еквівалентний та нееквівалентний підхід до перетворення програмного коду FPGA-компонентів та процеси стеганографічного вбудовування та відновної обфускації даних; виконано експериментальне дослідження

ефективності цього методу для задач зберігання контрольних даних в процесі прихованого моніторингу програмного коду FPGA;

– розроблено метод формування стеганографічного ключа для прихованого збереження контрольних даних в програмному коді FPGA, який дозволяє здійснити балансування між обсягом модифікованої частини програмного коду та ефективним обсягом стеганографічного контейнера, потрібним для збереження контрольних даних, а також виконано експериментальне дослідження ефективності цього методу;

– розроблено підсистему гібридного інформаційно замаскованого зберігання контрольних даних в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів.

Об'єкт дослідження – процес моніторингу характеристик безпеки програмного коду FPGA-компонентів інформаційних систем.

Предмет дослідження – моделі, методи та засоби гібридного інформаційно замаскованого зберігання контрольних даних та доступу до них при виконанні прихованого моніторингу програмного коду FPGA-компонентів інформаційних систем.

Методи досліджень. При розробці моделі нееквівалентного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA-компонентів використовувалися арифметичні та алгоритмічні основи комп'ютерних обчислень, а також теорії наближених обчислень. Модель еквівалентних перетворень програмного коду блоків LUT FPGA для спільних процесів вбудовування контрольних даних в програмний код та його обфускації ґрунтується на застосуванні елементів булевої алгебри та теорії множин. Для розробки методу гібридного замаскованого зберігання контрольних даних використано теоретичні основи цифрової стеганографії та цифрових водяних знаків. Для аналізу та перевірки запропонованих теоретичних положень було застосовано комп'ютерне моделювання, методи та алгоритми розв'язання прикладних задач комп'ютерних наук та сучасні технології програмування.

Наукова новизна отриманих результатів полягає у розробці моделей та методів гібридного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, що забезпечує вирішення актуальної науково-прикладної задачі збільшення доступного обсягу для приховування контрольних даних в процесі таємного моніторингу програмного коду FPGA-компонентів інформаційних систем в умовах атак на систему моніторингу.

Основні наукові результати, отримані в роботі, полягають в наступному:

– *Вперше* розроблено модель замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, *яка основана* на використанні нееквівалентних перетворень програмного коду FPGA *та враховує* особливості виконання наближених арифметичних операцій в середовищі FPGA, *що дозволяє* визначити множину надлишкових інформаційних ресурсів, які виникають в процесі виконання таких обчислень, та можуть бути використані для прихованого зберігання контрольних даних.

– *Вперше* розроблено метод нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, *що характеризується* використанням надлишковості програмного коду FPGA, яка виникає при виконанні наближених арифметичних операцій, *що дозволяє* збільшити обсяг, доступний для прихованого зберігання контрольних даних в задачах моніторингу характеристик безпеки програмного коду FPGA.

– *Вперше* розроблено метод гібридного замаскованого зберігання даних в середовищі програмного коду FPGA, *який відрізняється* поєднанням еквівалентного та нееквівалентного підходів до перетворення програмного коду FPGA, а також поєднанням процесів стеганографічного вбудовування та відновної обфускації даних, *що дозволяє* збільшити обсяг контрольних даних, які можуть бути приховано вбудованими в програмний код FPGA, не збільшуючи при цьому здатність традиційних методів стегоаналізу до виявлення цих даних в програмному коді.

– *Дістав подальшого розвитку* метод формування стеганографічного ключа для замаскованого збереження даних в програмному коді FPGA, *який відрізняється* можливістю адаптації до структури зв'язків між елементарними одиницями FPGA та до необхідного обсягу контрольних даних, *що дозволяє* виконувати балансування між обсягом частини програмного коду, яка модифікується в результаті вбудовування даних та ефективним обсягом стеганографічного контейнера, потрібним для збереження контрольних даних.

Практичне значення отриманих результатів.

На базі запропонованих моделей та розроблених методів створено програмну підсистему гібридного замаскованого зберігання контрольних даних в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів.

Розроблена підсистема містить наступні основні програмні модулі:

– програмний модуль, який забезпечує доступ до контрольних даних, вбудованих в програмний код FPGA, відповідно до методу гібридного замаскованого зберігання;

– програмний модуль, який забезпечує вбудовування контрольних даних в програмний код FPGA, відповідно до методу гібридного замаскованого зберігання контрольних даних;

– програмні засоби, які реалізують моделі, призначені для виявлення множини надлишкових інформаційних ресурсів, що виникають в процесі виконання наближених арифметичних операцій на FPGA;

– програмний додаток, призначений для добуванні детальної інформації про структуру FPGA-проєкту з внутрішньої бази даних САПР FPGA-систем для задач зберігання контрольних даних в середовищі програмного коду FPGA-компонентів.

Використання розроблених програмних засобів, а також запропонованих моделей та методів дозволило збільшити порівняно з традиційним підходом обсяг потенційно доступних додаткових даних, які

можуть бути приховано вбудовані в програмний код FPGA в стеганографічний спосіб в середньому від 22,6% до 34,0% за рахунок використання надлишкового ресурсу несуттєвих блоків LUT FPGA та збільшити цей обсяг в середньому від 93,7% до 105,1% за рахунок використання двох видів виявлених надлишкових ресурсів: несуттєвих блоків LUT FPGA та несуттєвих розрядів суттєвих блоків LUT FPGA. Це дає змогу підвищити розрядність контрольних даних, які можуть бути замасковано збережені в програмному коді FPGA та збільшити кількості різновидів прихованого моніторингу програмного коду, контрольні дані яких зберігаються в такий спосіб.

На основі теоретичних результатів дисертації було отримано патент України на винахід.

Розроблені в роботі методи та інструментальні засоби впроваджені у діяльність товариства з обмеженою відповідальністю ТЕХНО.КОМ (довідка про впровадження від 10 грудня 2025 року).

Отримані в роботі результати, а також програмні засоби впроваджено в навчальний процес європейських університетів за міжнародним ERASMUS+ проєктом “ALIOT – Internet of Things: Emerging Curriculum for Industry and Human Applications” 573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP, а також в навчальний процес Національного університету «Одеська політехніка» при викладанні дисципліни “Технології захисту інформації”.

Особистий внесок здобувача. Всі наукові положення, висновки та рекомендації дисертації були отримані здобувачем особисто. В роботах, що написано в співавторстві, автором запропоновано: модель інформаційно замаскованого зберігання контрольних даних в середовищі програмного коду FPGA-компонентів [1, 5, 6]; метод нееквівалентного інформаційно замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів [1, 4-7]; метод гібридного інформаційно замаскованого зберігання контрольних даних в середовищі програмного коду FPGA [3, 9]; метод формування стеганографічного ключа для

прихованого збереження контрольних даних в програмному коді FPGA [2, 8]; архітектура програмних засобів гібридного інформаційно замаскованого зберігання контрольних даних [4, 11, 13]; принцип виконання еквівалентних перетворень при виконанні обфускації програмного коду FPGA-компонентів [10]; принципи організації обчислювальних процесів обробки множини блоків LUT FPGA для гібридного замаскованого зберігання контрольних даних [12, 14].

Апробація роботи. Результати роботи доповідалися та обговорювалися на наступних наукових конференціях:

– міжнародній науково-технічній конференції “Computer Modeling and Intelligent Systems, CMIS-2021” (Запоріжжя, 2021);

– міжнародній науково-технічній конференції “Інформатика, управління та штучний інтелект, ІУШІ-2021” (Харків, 2021);

– міжнародній науково-технічній конференції “Сучасні інформаційні технології, МІТ” (Одеса, 2021);

– всеукраїнській науково-практичної конференції “Перспективні напрямки сучасної електроніки, інформаційних і комп’ютерних систем, MEICS-2023” (Дніпро, 2023).

Публікації. За матеріалами дисертації опубліковано 14 праць, зокрема: 9 статей у наукових виданнях (5 з яких у закордонних наукових періодичних виданнях за напрямом дисертації, проіндексованих у наукометричних базах Scopus та Web of Science Core Collection; 4 статті – у виданнях, що входять до переліку наукових фахових видань України); 1 патент України на винахід; 3 публікації в матеріалах науково-технічних конференцій; 1 навчальний посібник, який додатково висвітлює результати дисертації.

Структура і обсяг дисертації. Дисертація має загальний обсяг 214 сторінок і складається з вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Основний текст дисертації викладено на 156 сторінках. Робота містить 30 рисунків і 6 таблиць. Список використаних джерел складається з 149 найменувань.

1 АНАЛІЗ ПРОБЛЕМИ ЗБЕРІГАННЯ КОНТРОЛЬНИХ ДАНИХ В ЗАДАЧАХ ПРИХОВАНОГО МОНІТОРИНГУ ПРОГРАМОВАНИХ КОМПОНЕНТІВ КОМП'ЮТЕРНИХ СИСТЕМ

1.1 Місце програмованих обчислювальних компонентів в складі сучасних комп'ютерних та інформаційних систем

Основними класами компонентів комп'ютерних систем є компоненти з жорсткою логікою функціонування та програмовані компоненти [1]. Компоненти з жорсткою логікою мають фіксовану поведінку, яка визначається їх структурою на етапі виробництва. Програмовані компоненти налаштовуються на вирішення конкретного завдання шляхом завантаження в них програмного коду. Поведінка програмованих компонентів може бути в будь-який момент часу частково або повністю змінена шляхом заміни програмного коду, який керує логікою їх функціонування. Застосування в складі комп'ютерних систем компонентів обох видів дозволяє забезпечити як гнучкість, так і високу ефективність обробки інформації.

Компоненти з жорсткою логікою функціонування мають фіксовану структуру і виконують строго визначені функції. Їх логіка задається на етапі проєктування і не підлягає зміні в процесі експлуатації комп'ютерної системи. У складі комп'ютерних систем компоненти такого роду зазвичай вирішують вузькоспеціалізовані, заздалегідь визначені завдання, для виконання яких потрібна висока продуктивність і енергоефективність. До цього типу компонентів відносяться спеціалізовані інтегральні схеми (ASIC) [2], компоненти стандартної логіки, контролери інтерфейсів, кодеки, а також різні схеми управління і обробки сигналів.

Традиційно виділяють наступні переваги компонентів з жорсткою логікою функціонування [3]: 1) максимальна продуктивність, обумовлена оптимізацією схеми під конкретну операцію; 2) енергоефективність,

обумовлена відсутністю накладних витрат на вибірку і декодування інструкцій, як це відбувається при функціонуванні, наприклад, мікропроцесорів; 3) мінімізація площі компонента і обсягу його апаратних ресурсів, причиною чого також є вузька спеціалізація компонента.

До недоліків таких компонентів слід віднести: 1) відсутність модифікованості, що виражається в необхідності перепроєктування компонента при виявленні помилок функціонування або при зміні алгоритмів функціонування чи протоколів взаємодії з іншими компонентами; 2) високу вартість проєктування, великі часові та фінансові вкладення на етапі проєктування та підготовки до виробництва.

Програмовані компоненти характеризуються тим, що їх функціональність визначається програмним кодом, який завантажується у внутрішню або окрему пам’ять програм таких компонентів [4]. До даного типу компонентів відносяться: центральні процесори (CPU) [5], графічні процесори (GPU) [6], мікроконтролери (MCU) [7], а також програмовані логічні інтегральні схеми (CPLD/FPGA) [8]. Основне призначення компонентів такого роду полягає у виконанні алгоритмів, обробці даних, управлінні пристроями та реалізації складної логіки, яка може змінюватися без фізичної модифікації апаратури. Узагальнено такі компоненти призначені для інтерпретації та виконання послідовності команд (програмного коду), що дозволяє одній і тій саме апаратній платформі вирішувати необмежений спектр завдань.

До основних переваг програмованих компонентів [9-10] належать висока гнучкість і адаптивність. Зміна функціональності цих компонентів здійснюється шляхом зміни програмного коду. Такі мікросхеми дозволяють реалізовувати складні обчислювальні та керуючі алгоритми в багатофункціональних і динамічно мінливих системах. Недоліками програмованих компонентів є нижча продуктивність і енергоефективність при виконанні спеціалізованих завдань порівняно з компонентами, які мають жорстку логіку функціонування.

У складі сучасних комп’ютерних систем програмовані компоненти займають домінуюче положення [11], оскільки саме такі компоненти забезпечують виконання обчислень, управління процесами та взаємодію з користувачем. Також слід відзначити ієрархічне домінування програмованих компонентів. Компоненти з жорсткою логікою зазвичай складають значну, але допоміжну частину апаратного забезпечення комп’ютерних систем, виконуючи функції прискорення, підтримки периферії та обробки спеціалізованих операцій [12-13].

Таким чином можна констатувати, що більшість сучасних комп’ютерних систем містять як компоненти з жорсткою логікою, так і програмовані компоненти. При цьому програмовані компоненти забезпечують універсальність обчислювальних, керуючих і інформаційних процесів, а компоненти з жорсткою логікою забезпечують утворюючі обчислювальну систему і допоміжні функції.

1.2 Місце FPGA серед програмованих компонентів комп’ютерних систем

Мікросхеми FPGA (Field-Programmable Gate Array) [14-15] поряд з мікропроцесорами (CPU), мікроконтролерами та графічними процесорами (GPU) є програмованими компонентами комп’ютерних систем. Структура FPGA являє собою двовимірну матрицю програмованих обчислювальних блоків, з’єднаних програмованою системою міжз’єднань [16]. Кожен обчислювальний блок, як правило, містить простий обчислювач (блок LUT) [17], прості елементи пам’яті та допоміжні елементи, що дозволяють реалізовувати довільні логічні функції. Додатково до складу FPGA входять спеціалізовані апаратні ресурси, такі як блоки вбудованої пам’яті, апаратні помножувачі, блоки цифрової обробки сигналів та інтерфейсні модулі вводу-виводу. За рахунок дуже великої кількості елементарних обчислювальних блоків у складі програмованої матриці FPGA (від десятків

тисяч до десятків мільйонів) ці компоненти здатні вирішувати складні обчислювальні та керуючі задачі.

Функціонування FPGA задається конфігураційними даними (програмним кодом), які завантажуються в мікросхему перед початком функціонування компонента [18]. Ці дані налаштовують поведінку логічних блоків і структуру з'єднань між ними.

Особливостями FPGA, що відрізняють їх від інших програмованих компонентів, є:

1) суто паралельний принцип функціонування;

2) дуже велика кількість елементарних обчислювальних блоків (від десятків тисяч до десятків мільйонів);

3) програмний код, що завантажується в FPGA, змінює внутрішню структуру схеми і функції обчислювальних блоків, що призводить до зміни поведінки компонента.

На відміну від мікроконтролерів і мікропроцесорів, що виконують програми послідовно в середовищі апаратної схеми з фіксованою структурою, FPGA дозволяють створювати спеціалізовану апаратну структуру під конкретну задачу. Це забезпечує високу продуктивність і низькі затримки при обробці даних.

В ієрархії програмованих компонентів FPGA розглядаються як найбільш універсальний і потужний клас реконфігурованих пристроїв. Їх застосування виправдане в завданнях, де потрібне поєднання високої продуктивності, паралельної обробки і можливості модифікації апаратної логіки без зміни фізичної структури системи.

По відношенню до мікропроцесорів (CPU) і графічних процесорів (GPU) мікросхеми FPGA позиціонуються наступним чином [19-20]. У середовищі CPU обчислювальний процес організований шляхом послідовного виконання інструкцій з обмеженим рівнем паралелізму, що реалізується за рахунок конвеєрів і багатоядерності. У FPGA обчислювальний процес реалізується у вигляді спеціалізованої апаратної

структури, в якій множина операцій може виконуватися одночасно і незалежно. Це дозволяє істотно знизити затримки обробки даних і забезпечити мінімальний час відгуку. Крім того, FPGA характеризуються високою гнучкістю на рівні архітектури, оскільки апаратна конфігурація може бути адаптована під конкретну вирішувану задачу.

Найчастіше FPGA застосовуються в завданнях, де потрібний високий ступінь паралелізму, мінімальні затримки і низьке енергоспоживання.

Задачі апаратної обробки растрової графіки та видео. Багато задач обробки растрової графіки та відео базується на використанні матричного помноження [21]. При програмній реалізації на CPU або GPU такі операції складається з декількох вкладених циклів які послідовно ітерують по елементам матриць операндів. При реалізації на FPGA виникає можливість розгортання зазначених циклів і таким чином операція помноженні матриць може бути подана у вигляді розгорнутих декількох або всіх зазначених циклів, та реалізована паралельно [22].

Цифрова обробка сигналів. Використання FPGA у задачах цифрової обробки сигналів забезпечує суттєві переваги під час реалізації алгоритмів із жорсткими вимогами до затримки та пропускної здатності [23]. Алгоритми цифрової обробки сигналів можуть реалізуються на FPGA у вигляді повністю конвеєрних структур, що дозволяють обробляти один відлік сигналу за такт.

Задачі високошвидкісної передачі даних. Застосування FPGA у системах високошвидкісної передачі даних зумовлене їх здатністю реалізовувати протоколи та фізичні рівні зв’язку безпосередньо на апаратному рівні [24]. Сучасні FPGA оснащуються високошвидкісними трансиверами (GTY, GTH, SERDES) [25], що підтримують швидкості передачі від 10 до 112 Гбіт/с на лінію, з апаратною корекцією помилок (FEC), еквалізацією та відновленням тактової частоти (CDR). Це дозволяє реалізовувати інтерфейси PCI Express Gen4/Gen5, Ethernet 100G/400G та Serial RapidIO з мінімальною затримкою [26]. На відміну від процесорних

рішень, обробка пакетів і керування потоками даних у FPGA виконуються паралельно та детерміновано, без участі операційної системи. Затримка передачі може бути знижена до десятків наносекунд завдяки прямій конвеєрній маршрутизації даних. Крім того, можливість адаптації схеми під конкретну задачу дозволяє оптимізувати ширину шин, формат кадрів і алгоритми буферизації під конкретне завдання, знижуючи енергоспоживання та підвищуючи ефективну пропускну здатність [27]. Ці властивості роблять FPGA критично важливими в телекомунікаційних, датацентрових і наукових вимірювальних системах.

Реалізація криптографічних алгоритмів. Завдяки просторовому паралелізму FPGA дозволяють реалізовувати криптографічні примітиви (AES, SHA-2, SHA-3, RSA, ECC) у вигляді спеціалізованих апаратних конвеєрів із фіксованою затримкою [28]. Наприклад, апаратна реалізація AES-128 на FPGA може забезпечувати пропускну здатність понад 100 Гбіт/с при латентності менше 100 нс, що я в 10-15 разів кращим результатом, порівняно з програмними реалізаціями на CPU [29]. Енергоефективність таких рішень досягає значень менше 1 нДж на один зашифрований блок, що також суттєво нижче за аналогічні показники GPU [30]. Реконфігурованість FPGA забезпечує можливість адаптації до змін криптографічних стандартів без заміни апаратної платформи.

Обчислення для задач блокчейн. Використання FPGA для реалізації блокчейн-обчислень має низку переваг порівняно з CPU та GPU. Архітектура FPGA дозволяє реалізовувати глибоко конвеєрні та повністю паралельні апаратні реалізації криптографічних алгоритмів, SHA-256 [31], Keccak [32] або операції над еліптичними кривими (ECDSA, EdDSA) [33]. Це забезпечує значно меншу латентність ніж при застосуванні чисто програмної реалізації на CPU або GPU. Наприклад, апаратна реалізація алгоритму SHA-256 на FPGA може досягати затримок порядку десятків тактів і пропускної здатності в сотні гігахешів за секунду при енергоспоживанні на рівні одиниць ват, що дає виграш за

енергоефективністю (Дж/хеш) у 5-20 разів порівняно з GPU [34]. На відміну від ASIC, FPGA зберігають реконфігурованість, що є критичним для блокчейн-протоколів, схильних до змін алгоритмів консенсусу або хеш-функцій [35]. Крім того, FPGA ефективно підтримують потокову обробку транзакцій і валідацію підписів у вузлах, знижуючи навантаження на центральний процесор і підвищуючи загальну пропускну здатність мережі.

Обробка задач високочастотного трейдінгу. Використання FPGA у високочастотному трейдінгу [36] зумовлене їхньою здатністю забезпечувати наднизькі та детерміновані затримки обробки даних. На відміну від CPU, де обробка ринкових даних виконується послідовно та залежить від планувальника операційної системи, FPGA реалізують алгоритми у вигляді апаратних конвеєрів із фіксованим часом проходження сигналу. Типові затримки обробки пакета на FPGA становлять 50–200 нс, тоді як програмні рішення на CPU – одиниці мікросекунд [37]. FPGA дозволяють виконувати парсинг протоколів (FIX, OUCH, ITCH) [38], побудову книги заявок і генерацію ордерів повністю в апаратурі, минаючи мережевий стек операційної системи [39]. Паралельна архітектура забезпечує одночасну обробку тисяч цінових рівнів без деградації латентності. Крім того, FPGA демонструють нижче енергоспоживання на операцію [40], а детермінізм затримок підвищує відтворюваність торгових стратегій і точність арбітражних моделей трейдінгу.

Задачі прискорення нейронних мереж. Використання FPGA для реалізації нейронних мереж зумовлене можливістю тонкої апаратної оптимізації обчислень і потоків даних. На етапі навчання FPGA застосовуються для прискорення операцій лінійної алгебри, насамперед множення матриць і згорток, за рахунок масивів DSP-блоків та конвеєризації обчислень [41]. Мікросхеми FPGA можуть реалізовувати настроюванні функції активації та оптимізатори безпосередньо в апаратурі, зменшуючи затримки обміну з пам’яттю [42].

На етапі інференсу FPGA є особливо ефективними завдяки детермінованій латентності та високому співвідношенню продуктивності до споживаної потужності [43]. Апаратна реалізація нейронної мережі у вигляді потокового графа дозволяє виконувати інференс із затримками в десятки мікросекунд, що є критичним для систем реального часу. Використання квантизованих моделей та on-chip пам'яті [44] FPGA зменшує кількість звернень до зовнішньої пам'яті, знижуючи енергоспоживання. На відміну від GPU, FPGA забезпечують передбачуваний час відгуку за високого навантаження, що робить їх застосування доцільним в складі вбудованих AI-систем [45].

Таким чином, можна констатувати, що FPGA займають нішу реконфігурованих апаратних автономних обчислювачів [46] та прискорювачів [47], які функціонують спільно з CPU і GPU. Це робить FPGA важливим елементом сучасних комп'ютерних систем, орієнтованих на високопродуктивні застосування [48-50].

1.3 Структура FPGA, принципи функціонування та програмування

Мікросхеми FPGA являють собою клас програмованих компонентів, структура і функціональність яких можуть бути визначені після виготовлення за допомогою конфігурування (програмування) [51]. Структурно FPGA складається з регулярної двовимірної матриці програмованих логічних блоків (рис. 1.1) [52], системи програмованих міжз'єднань [53] і блоків вводу-виводу (I/O) [54]. Логічні блоки є основними елементами обчислень і включають в себе блоки LUT (Look-Up Table) [55-56], тригери і допоміжні логічні елементи. LUT реалізують довільні логічні функції від певного числа вхідних змінних, що дозволяє будувати комбінаційні схеми різного рівня складності. Тригери, які входять до складу

логічних блоків використовуються для зберігання стану і реалізації послідовної логіки, а також для реалізації розподіленої пам’яті [57].

Система програмованих міжз’єднань являє собою ієрархічну мережу комутаторів і провідників, що забезпечують з’єднання логічних блоків між собою і з блоками вводу-виводу. Дані міжз’єднання можуть змінюватися в процесі конфігурації, що дозволяє формувати довільну конфігурацію цифрової схеми [58]. Блоки вводу-виводу забезпечують зв’язки FPGA із зовнішніми пристроями та підтримують різні електричні стандарти і режими передачі даних [58].

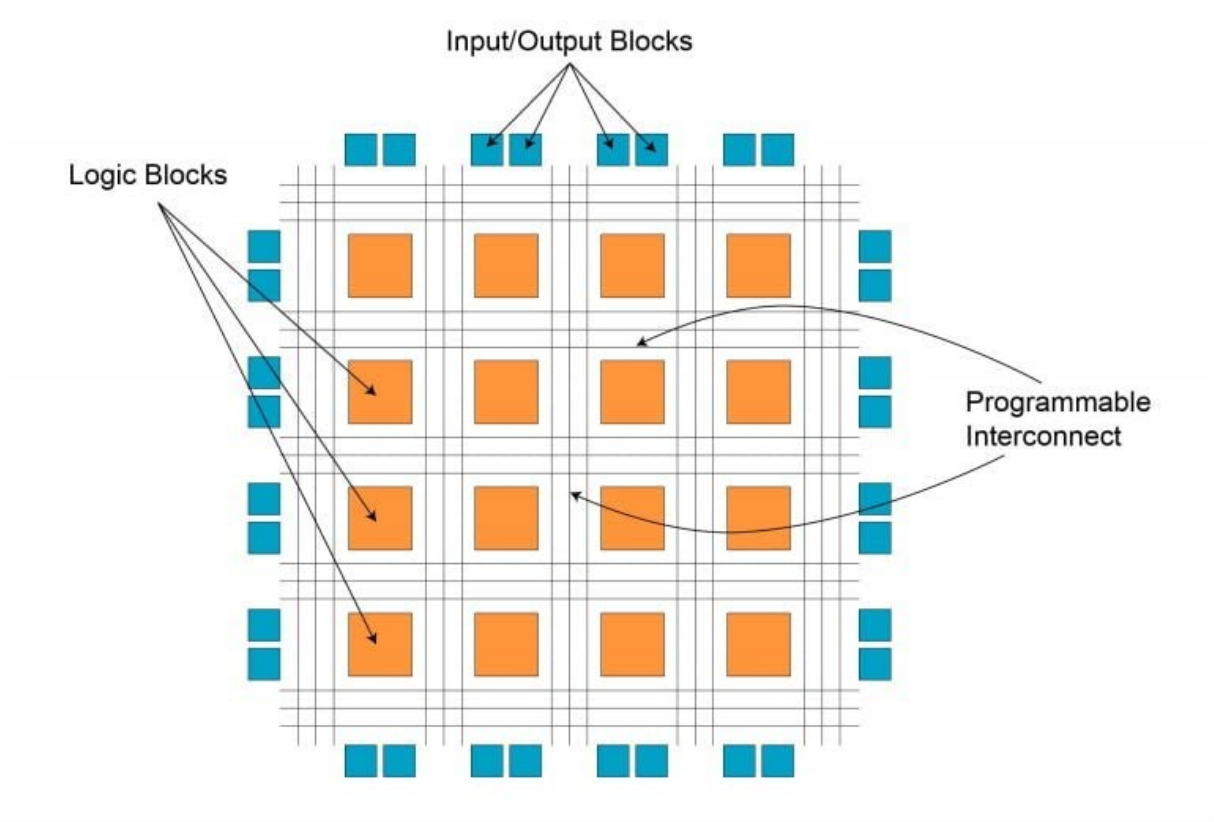


Рисунок 1.1 – Структура матриці FPGA [59]

Принцип функціонування FPGA базується на паралельній організації обчислень, в рамках якої всі логічні блоки функціонують одночасно відповідно до заданої конфігурації, що забезпечує високу продуктивність і мінімізує час відгуку [60]. Поведінка FPGA визначається програмним кодом не в явному вигляді, а апаратною структурою, сформованою з логічних

елементів і міжз’єднань під дією програмного коду, що завантажується в мікросхему [61]. Для опису цієї структури використовуються мови опису апаратури, такі як VHDL [62] або Verilog [63], які дозволяють задати цифрову систему на рівні регістрових передач [64] і логічних зв’язків.

Принцип конфігурації FPGA полягає в завантаженні в мікросхему конфігураційних даних (низькорівневого програмного коду) [65], що визначають стан всіх програмованих елементів. Ці дані зберігаються в конфігураційній пам’яті і задають вміст блоків LUT, режими роботи тригерів, маршрутизацію сигналів і параметри блоків вводу-виводу. У більшості сучасних FPGA для зберігання конфігурації використовується статична пам’ять типу SRAM [66], внаслідок чого конфігурація є енергозалежною і повинна завантажуватися при кожному включенні живлення. Джерелом конфігурації може виступати зовнішній модуль енергонезалежної пам’яті, мікроконтролер або інтерфейс програмування .

Таким чином, FPGA являють собою універсальну апаратну платформу, що поєднує гнучкість програмного підходу з перевагами апаратної реалізації. Архітектура, принцип функціонування і механізм конфігурації цих компонентів забезпечують можливість створення високопродуктивних спеціалізованих цифрових систем без необхідності розробки спеціалізованих інтегральних схем [67].

Основними обчислювальними елементами FPGA є блоки LUT (Look-Up Table), які призначені для реалізації довільних логічних функцій. Блоки LUT являють собою програмовані комбінаційні пристрої, конфігурація яких задається на етапі конфігурування мікросхеми шляхом завантаження в них двійкового програмного коду [68] (рис. 1.2).

Структурно LUT реалізується у вигляді масиву пам’яті, як правило, на основі статичних комірок SRAM. Кількість адресних входів блоку LUT визначає його розрядність і кількість задіяних комірок пам’яті. Так блок LUT з n входами містить 2^n комірок пам’яті, кожна з яких зберігає одне логічне значення, що відповідає конкретній комбінації вхідних сигналів [69].

Принцип функціонування блоків LUT базується на адресації масиву пам’яті. Вхідні сигнали LUT використовуються в якості адреси, за якою здійснюється вибір відповідної комірки пам’яті. Значення, що зберігається в обраній комірці, подається на вихід LUT. Таким чином, вихідний сигнал є функцією вхідних сигналів, що визначається вмістом пам’яті LUT.

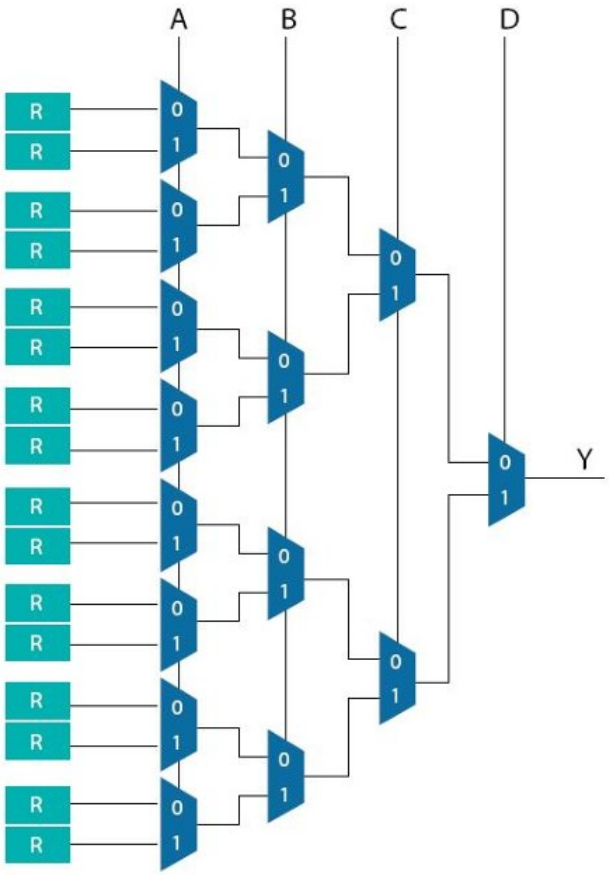


Рисунок 1.2 – Логічна структура блоків LUT FPGA [70]

Конфігурація вмісту блоків LUT здійснюється при завантаженні конфігураційних даних в FPGA. Ці дані формуються засобами автоматизованого проєктування на основі опису логіки мовами опису апаратури. Після конфігурації блок LUT функціонує як фіксований комбінаційний пристрій до наступного перезавантаження конфігурації.

У сучасних FPGA блоки LUT, як правило, входять до складу більших логічних комірок (Logic Elements [71] або Configurable Logic Blocks [72]),

які також включають тригери, мультиплексори та додаткові елементи комутації. Це дозволяє реалізовувати як комбінаційні, так і послідовнісні логічні схеми, а також підвищує гнучкість і щільність логічної реалізації.

1.4 Вплив зовнішнього втручання на функціонування програмованих та непрограмованих компонентів комп'ютерних систем

Зовнішні зловмисні втручання в функціонування комп'ютерних систем є одним з ключових факторів ризику для функціонування зазначених систем [73]. Під таким втручанням розуміють цілеспрямовані дії третіх осіб, спрямовані на порушення конфіденційності [74], цілісності [75] або доступності обчислювальних ресурсів [76]. Залежно від способу реалізації їх прийнято поділяти на фізичні втручання в апаратну частину [77] і логічні втручання, що здійснюються через маніпуляції з програмним кодом [78].

Фізичні втручання передбачають безпосередній вплив на елементи апаратури комп'ютерної системи. До таких дій відносяться несанкціонований доступ до обчислювальних вузлів, заміна або модифікація компонентів, підключення сторонніх пристроїв, а також пошкодження обладнання. Наприклад, установка апаратних закладок може дозволити зловмиснику перехоплювати передані дані або отримувати прихований віддалений доступ до системи. Пошкодження кабельних з'єднань, блоків живлення або систем охолодження здатне викликати збої, деградацію продуктивності і повну відмову обладнання. Особливу небезпеку становлять втручання в центрах обробки даних, де фізичний вплив на один елемент може спричинити каскадні відмови взаємопов'язаних систем [79].

Втручання через програмний код реалізуються без прямого контакту з апаратурою і базуються на використанні вразливостей програмного забезпечення [80]. До них відносяться вбудовування шкідливого коду,

експлуатація помилок в операційних системах і прикладних програмах, а також несанкціоноване змінення конфігурацій [81]. Шкідливий програмний код може виконувати широкий спектр деструктивних функцій: від прихованого збору інформації та підміни даних до блокування роботи системи. Експлуатація вразливостей дозволяє зловмисникам підвищувати свої привілеї, обходити механізми захисту та отримувати контроль над критично важливими процесами [82].

Негативний вплив втручань через експлуатацію програмного коду посилюється їх масштабованістю і складністю виявлення. На відміну від фізичного впливу, логічні атаки можуть здійснюватися віддалено і одночасно зачіпати велику кількість систем. Крім того, шкідливий код здатний маскуватися під легітимні процеси, що ускладнює його своєчасне виявлення і нейтралізацію.

Існують приклади втручання у функціонування комп'ютерних систем підвищеного ризику, що містять у своєму складі мікросхеми FPGA. Дані втручання були здійснені за допомогою маніпуляції програмним кодом комп'ютерних систем віддалено або із залученням інсайдерів.

Одним з найбільш відомих прикладів є інцидент застосування шкідливого низькорівневого апаратного вірусу Stuxnet [83-84], виявленого в 2010 році і спрямованого проти ядерного об'єкта в Натанзі (Іран). В результаті втручання в роботу програмованих компонентів системи управління відбулося прискорене зношування і вихід з ладу газових центрифуг. Даний інцидент став першим підтвердженим випадком, коли кібератака призвела до фізичного руйнування промислового обладнання, що продемонструвало принципово новий рівень загроз для критичної інфраструктури.

У 2016 році в результаті дій Industroyer [85] відбулася низка атак на системи управління в енергетиці (електричні мережі та енергетичну інфраструктуру) через віддалені маніпуляцій з програмним кодом.

У 2017 році на нафтохімічному підприємстві в Саудівській Аравії було виявлено шкідливе програмне забезпечення Triton (Trisis) [86], націлене на системи аварійного захисту. Втручання в їх функціонування могло призвести до неконтрольованого розвитку аварійної ситуації, включаючи вибух і загибель персоналу. Хоча катастрофу було попереджено за рахунок аварійної зупинки, сам факт атаки на захисні механізми істотно підвищив оцінку ризиків кібервпливу.

Інцидент в системі управління каналізацією округу Маручі (Австралія) [87] в 2020 році проявився в тому, що в результаті несанкціонованого доступу до програмованих компонентів системи управління відбулося багаторазове скидання неочищених стічних вод, що спричинило серйозну екологічну шкоду.

У 2021 році кібератака на трубопровід Colonial Pipeline в США призвела до зупинки поставок палива в ряді штатів [88-89]. Незважаючи на відсутність фізичного руйнування інфраструктури, наслідки виразилися в масштабних економічних втратах і порушенні логістики. Цей інцидент підтвердив системну взаємопов’язаність критичної інфраструктури та її залежність від стійкості цифрових компонентів.

1.5 Оперативний моніторинг характеристик безпеки програмного коду FPGA

Мікросхеми FPGA мають відмінності як архітектурного, так і безпекового плану, від мікропроцесорів та мікроконтролерів. Крім того, можливість забезпечення високої продуктивності робить FPGA вельми часто використовуваною елементною базою для реалізації комп’ютерних систем критичного застосування [90-92]. Зазначені фактори виводять FPGA в окремий від мікропроцесорів та мікроконтролерів домен інтересів щодо протидії втручанню через експлуатацію програмного коду.

Традиційно виділяють наступні засоби захисту програмованих компонентів від зловмисного втручання:

1) організаційні заходи [93] – правила роботи з програмним кодом та визначення кола осіб, які мають доступ до програмного коду на різних етапах життєвого циклу комп'ютерної системи;

2) засоби фізичного обмеження повторного запису програмного коду в конфігураційну пам'ять [94];

3) криптографічний захист програмного коду [95-96];

4) оперативний моніторинг [97-98] характеристик безпеки програмного коду FPGA: цілісності, автентичності, шляхів розповсюдження, легітимності використання.

Серед засобів захисту FPGA компонентів від зловмисного втручання саме оперативний моніторинг стану програмного коду цих компонентів відіграє особливу роль. Моніторинг проводиться за окремими характеристиками (цілісністю, автентичністю, шляхами розповсюдження, легітимністю використання) або комплексно за групою характеристик. При цьому найбільш дієві підходи до організації моніторингу базуються на використанні контрольних даних, які зберігаються разом з програмним кодом або в якості його частини.

Для виконання моніторингу програмного коду використовуються контрольні дані, які обчислюються або призначаються визначеним чином. Обчислювальні контрольні дані повинні відповідати наступним вимогам:

1) контрольні данні повинні мати значно менший обсяг, ніж програмний код який контролюються за їх допомогою;

2) незначні зміни програмного коду, який контролюється, повинні приводити до суттєвих змін контрольних даних.

Прикладами обчислювальних контрольних даних є хеш-суми [99] для моніторингу цілісності програмного коду або коди аутентифікації [100] для моніторингу його автентичності.

Атрибутивні контрольні дані, на відміну від обчислюваних, призначаються об’єкту програмного коду. Прикладом атрибутивних контрольних даних є контрольні дані, які застосовуються в контролі розповсюдження програмного коду або в контролі легітимності його використання [101].

На рис. 1.3 наведено схему виконання моніторингу програмного коду програмованих компонентів на етапі підготовки контрольних даних. На цьому етапі для об’єкта програмного коду формуються контрольні дані. Обчислювані контрольні дані створюються за допомогою відповідних алгоритмів, а атрибутивні контрольні дані призначаються об’єкту програмного коду. Отриманий набір контрольних даних оголошується еталонним і зберігається.

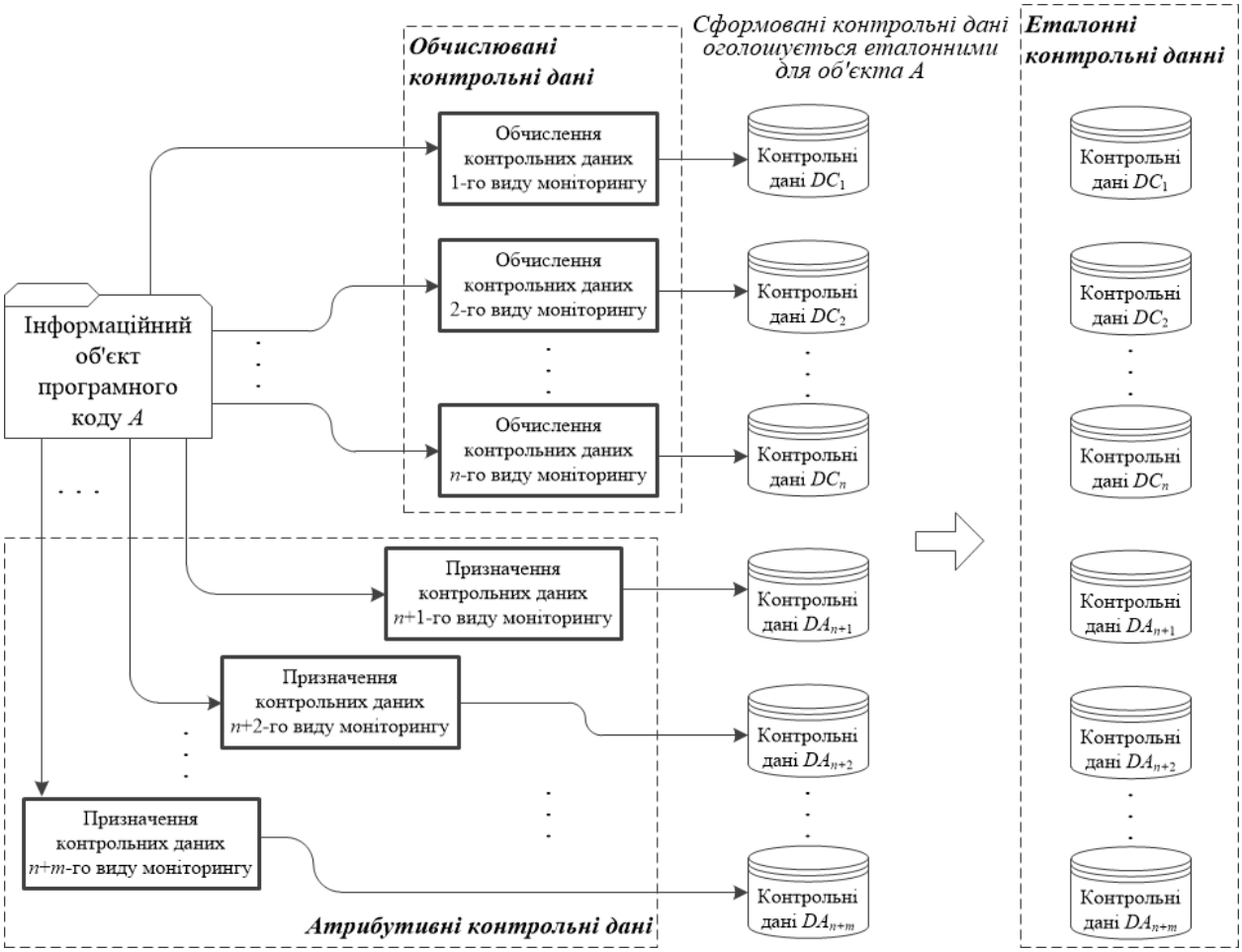


Рисунок 1.3 – Підготовка та збереження контрольних даних при виконанні моніторингу програмного коду

В момент виконання актів контролю в межах моніторингу програмного коду (рис. 1.4) обчислювані контрольні дані повторно обчислюються для програмного коду, щодо якого здійснюється моніторинг, а атрибутивні контрольні дані зчитуються з об’єкта програмного коду. Далі цей набір контрольних даних порівнюється з еталонними контрольними даними, які були збережені на етапі підготовки контролю в межах мониторінгу. За результатами такого порівняння робиться висновок про статус характеристик безпеки програмного коду (цілісність, автентичність, шляхи розповсюдження, легітимність застосування) під час актів контролю в межах моніторингу певного виду.

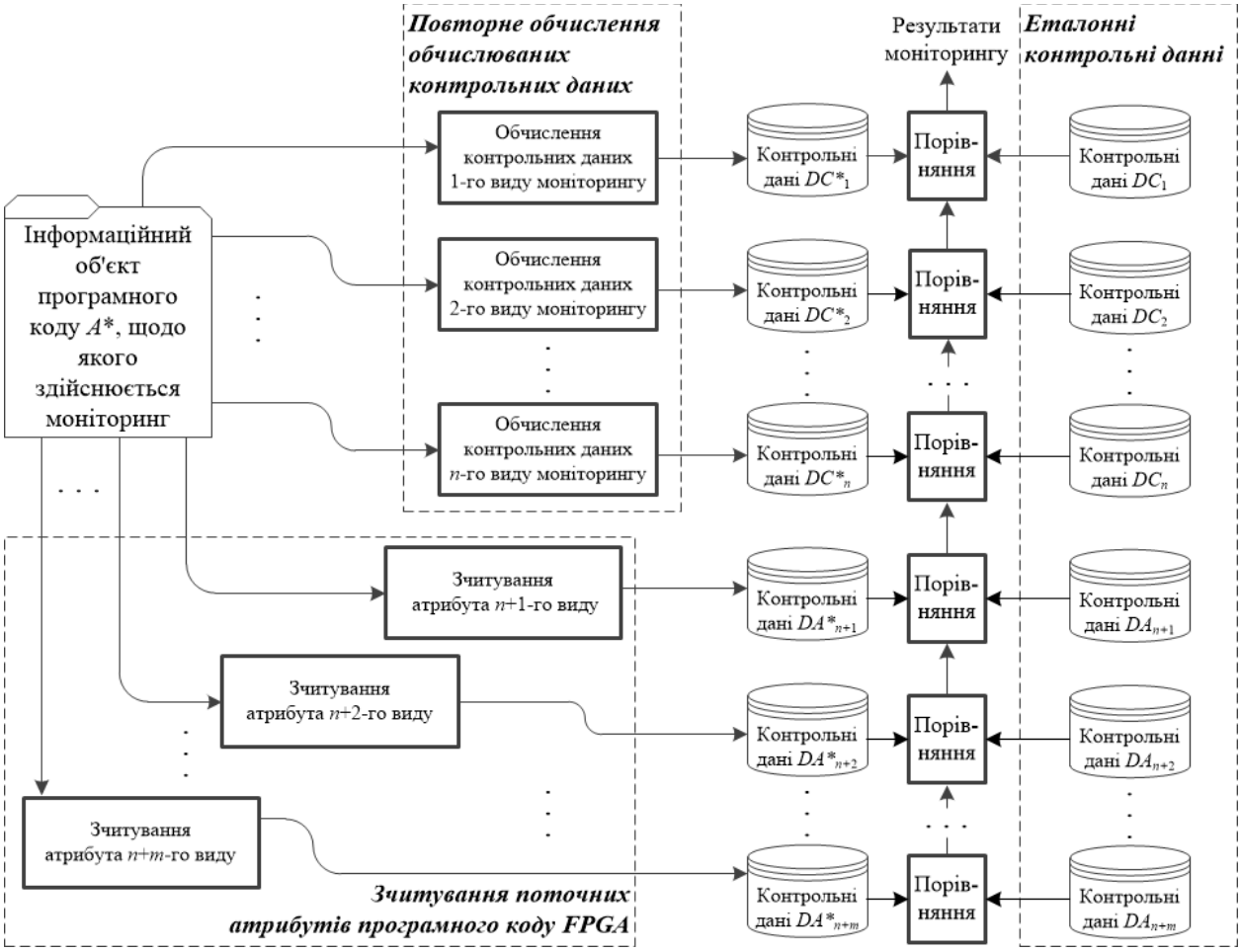


Рисунок 1.4 – Використання збережених контрольних даних на етапі здійснення мониторінгу програмного коду

Аналіз, наведених в попередньому підрозділі дисертації, інцидентів останніх років, пов’язаних зі дестабілізацією роботи технічних об’єктів підвищеного ризику показує, що найзначнішим чинником збоїв в роботі критично важливих комп’ютерних систем є зловмисне втручання в їх функціонування [102]. Одним з найефективніших підходів до протидії втручанню є оперативний моніторинг стану програмного коду цих компонентів. Виконання моніторингу потребує формування контрольних даних в момент підготовки програмного коду моніторингу, та їх зберігання з метою використання під час виконання актів моніторингу. При цьому основним способом обходу моніторингу програмного коду для здійснення втручання є фальсифікація контрольних даних [103], а найбільш суттєвим фактором, який обумовлює можливість втручання є спосіб зберігання контрольних даних.

1.6 Підходи до зберігання контрольних даних моніторингу програмного коду програмованих компонентів

В практиці побудови систем контролю програмного коду використовується декілька підходів до зберігання контрольних даних. Один з часто застосовуваних підходів базується на збереженні контрольних даних разом з інформаційним об’єктом програмного коду в файлової системі або пам’яті [104] у вигляді окремих файлів, логічно пов'язаних з контрольованими інформаційними об'єктами програмного коду. У цьому випадку для кожного контрольованого об'єкта програмного коду створюється окремий файл з еталонним значенням, що спрощує адміністрування і дозволяє оновлювати контрольні дані незалежно від основного вмісту. Проте, при спільному зберіганні контрольованого файлу і файлу з контрольною сумою зберігається ризик їх одночасної компрометації, особливо в умовах відсутності суворого розмежування прав доступу.

Інший існуючий підхід зберігання контрольних даних мониторінгу базуються на приєднанні контрольних даних до інформаційного об’єкта програмного коду і зберіганні в якості його частини [105]. В межах цього підходу контрольні дані розміщуються в складі контрольованого інформаційного об’єкта. Даний підхід використовується в файлових форматах і контейнерах даних, оскільки забезпечує автономність контрольованого об'єкта і спрощує його передачу та обробку. Однак з точки зору безпеки даний спосіб має обмежену ефективність, оскільки при наявності у порушника прав на зміну файлу він може скоригувати як вміст, так і відповідне контрольне значення.

Таким чином, зазначені вище два підходи до зберігання контрольних даних узагальнено мають наступні недоліки:

1) очевидність факту виконання моніторингу;

2) доступність еталонних контрольних даних для зчитування, аналізу та можливої фальсифікації.

Також часто використовуються підходи, в межах яких контрольні дані зберігаються в централізованій базі даних та отримуються з неї в момент виконання процедури моніторингу [106]. Ці підходи пов’язані з проблемою організації доступу до бази даних; складністю захисту бази даних від витоків та не зменшують можливість інсайдерського втручання [107].

Окрему групу рішень складають криптографічно пов’язані структури та захищені контейнери даних [108]. У даному випадку контрольні дані використовується у складі криптографічного механізму. Однак такий підхід успадковує недоліки базового підходу:

1) контрольні дані (хоч і в зашифрованому вигляді) доступні широкому колу осіб;

2) факт виконання моніторингу є відкритим;

3) не мінімізується можливість інсайдерської маніпуляції контрольними даними.

Один з перспективних підходів до зберігання контрольних даних базується на використанні стеганографічного [109] приховування даних безпосередньо в середовищі об’єкта програмного коду FPGA. Цей підхід має переваги, які полягають у приховуванні контрольних даних та факту виконання моніторингу, а також у забезпеченні утворення єдиного цілого між об’єктом програмного коду та контрольними даними. Однак вони мають і недоліки, що полягають у відносно малому доступному обсязі даних, які можна зберегти в стеганографічний спосіб. Це обмежує обсяг контрольних даних та кількість видів моніторингу, які можна застосувати до програмного коду.

1.7 Стеганографічний підхід до зберігання контрольних даних моніторингу

Як було зазначено в попередньому підрозділі, існує підхід до зберігання контрольних даних моніторингу програмного коду, який відрізняється від розглянутих традиційних підходів. Цей підхід ґрунтується на використанні теорії цифрової стеганографії [109]. Стеганографія спрямована на захист даних шляхом таємного вбудовування (приховування) даних одного виду в дані іншого виду. Захищувані дані вбудовуються у стего-контейнер [110], яким слугує спеціально підготовлений інформаційний об’єкт. При використанні стеганографічного підходу прихованим стає сам факт наявності захищуваних даних. Основною вимогою до такого підходу є відсутність будь-яких змін (функціональних, візуальних, структурних) стего-контейнера після вбудовування в нього додаткових даних.

Використання методів цифрової стеганографії для зберігання контрольних даних полягає в наступному. Контрольні дані обчислюються в традиційний спосіб. Після цього вони вбудовуються у програмний код в стеганографічний спосіб. При цьому вбудовані контрольні дані не повинні

впливати на функціонування пристрою, який програмується цим програмним кодом. Також вбудовування не повинно призводити до структурних або візуальних змін програмного коду. Таким чином, програмний код виступає у ролі стего-контейнера, а контрольні дані – у ролі додаткових даних, що вбудовуються у цей контейнер. При використанні такого підходу сам факт наявності контрольних даних є прихованим від зовнішнього спостерігача. Факт того, що до програмного коду здійснюється моніторинг, також є прихованим. У момент виконання активної стадії моніторингу вбудовані у програмний код контрольні дані витягуються з інформаційного стего-контейнера програмного коду. Витягування можливе за наявності стего-ключа, який описує правила локалізації додаткових даних у програмному коді.

Традиційно стеганографічний підхід до вбудовування даних моніторингу у програмний код FPGA ґрунтується на еквівалентних перетвореннях цього програмного коду [111–113]. У ході еквівалентних перетворень програмних кодів елементарних блоків LUT FPGA зберігаються функції, які задаються програмним кодом. Контрольні дані вбудовуються у програмний код у вигляді цифрового водяного знака, не змінюючи розмір програмного коду та функціонування FPGA. Після такого вбудовування цифровий водяний знак утворює з програмним кодом єдине ціле. При цьому виділити місце розташування вбудованого цифрового водяного знака у програмному коді не видається можливим.

Після вбудовування цільові біти вбудованого цифрового водяного знака виконують основну функцію програмного коду та одночасно є прихованим сховищем вбудованих даних. Витягання контрольної інформації, збереженої таким способом, можливе лише за наявності спеціального ключа (стеганографічного ключа [114]), який виявляє локалізацію цифрового водяного знака. За відсутності стего-ключа зовнішній спостерігач не може прийняти рішення щодо наявності або відсутності контрольної інформації в інформаційному об’єкті програмного

коду. Це призводить до того, що факт моніторингу цілісності також є прихованим від зовнішнього спостерігача.

Для забезпечення зазначених властивостей цифрового водяного знака використовується його вбудовування за допомогою еквівалентних перетворень програмного коду FPGA. Еквівалентні перетворення виконуються у просторі програмного коду елементарних обчислювальних блоків FPGA – блоків LUT (Look Up Table). Блок LUT – це програмований обчислювач, який зазвичай має від 4 до 8 входів для різних сімейств FPGA. Цей блок виконує обчислення однієї логічної функції від вхідних змінних. Налаштування блока LUT на обчислення конкретної функції здійснюється за допомогою двійкового програмного коду розміром 2^n розрядів, де n – кількість входів блока LUT.

Еквівалентні перетворення складаються з елементарних дій, кожна з яких виконується для пари послідовно з'єднаних блоків LUT. У межах кожної елементарної дії програмний код першого блока LUT пари порозрядно інвертується. Ця інверсія компенсується операцією певної перестановки розрядів програмного коду другого блока LUT пари. Конкретний вид компенсуючої перестановки залежить від ваги входу другого блока LUT пари, який підключений до виходу першого блока пари. За ваги входу k перестановка полягає в обміні місцями груп послідовно розташованих 2^k-1 розрядів програмного коду.

На рис. 1.5 показано дві пари блоків LUT, що мають еквівалентні програмні коди. Ліва пара блоків має програмні коди $code_1$ та $code_2$ відповідно. У правій парі блоків: блок LUT_1 має код $I(code_1)$ – порозрядну інверсію коду $code_1$; блок LUT_2 має код $P(v, code_2)$ – компенсуючу перестановку розрядів двійкового програмного коду $code_2$, конкретний вид якої залежить від ваги v того входу блока LUT_2, який підключений до виходу блока LUT_1.

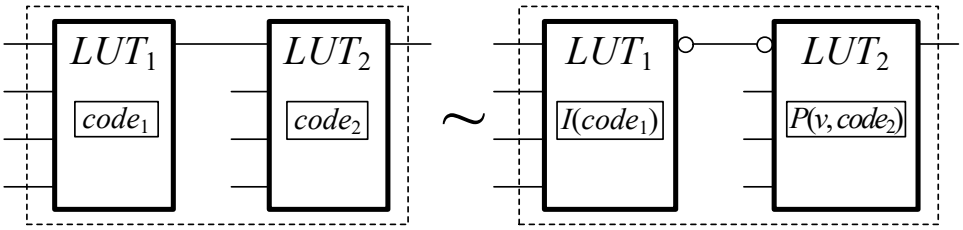


Рисунок 1.5 – Інверсія та компенсуюча перестановка при еквівалентному перетворенні програмного коду пари блоків LUT

У результаті елементарної дії еквівалентного перетворення у програмний код першого блока LUT пари вбудовується один розряд контрольного цифрового водяного знака.

Однак ефективний обсяг даних, які можна вбудувати у програмний код в такий спосіб, є обмеженим [115]. Ця обмеженість призводить до того, що у програмний код FPGA на практиці вдається вбудувати контрольні дані лише одного виду моніторингу. У цьому випадку контрольні дані інших використовуваних видів моніторингу доводиться зберігати за допомогою відкритих традиційних способів зберігання контрольних даних.

1.8 Оцінка методів FPGA IP Watermarking відносно стеганографічного підходу до зберігання додаткових даних в програмному коді FPGA

В межах технологій створення комп'ютерних систем на основі FPGA-компонентів поширеним є застосування FPGA IP Core (Intellectual Property Core) – готових функціональних блоків для FPGA, які реалізують певну функцію та можуть бути використані богаторазово в різних проєктах.

Зазвичай FPGA IP Core поставляється у вигляді високорівневого програмного HDL-коду на мовах Verilog або VHDL, gate-level netlist або у вигляді готового bitstream модуля.

Головна мета використання FPGA IP Core полягає в прискоренні розробки FPGA-систем за рахунок повторного використання вже створених та перевірених рішень.

Для захисту авторства на FPGA IP Core та виявлення їх незаконного копіювання використовують методи FPGA IP Watermarking, які полягають у вбудовуванні певних цифрових водяних знаків у ці модулі. Методи FPGA IP Watermarking мають деякі подібні риси до методів стеганографічного підходу. У даному підрозділі виконується оцінка методів IP Watermarking та визначається їх придатність для застосування в задачах прихованого моніторингу програмного коду FPGA.

Можна виділити наступні групи методів FPGA IP Watermarking.

1) Методи, що використовують структурний підхід до вбудовування цифрового водяного знака. Ці методи додають до структури FPGA-компонента фіктивні модулі та сигнали, які не впливають на основну логіку, але здатні за певних умов видавати унікальну послідовність бітів. Недолік цих методів полягає в необхідності використання додаткових ресурсів, зосередженості джерела цифрового водяного знака та можливості виявлення фіктивних модулів в структурі компонента.

2) Методи, які використовують ті частини логіки FPGA-компонента, що функціонують за принципом кінцевого автомата. В межах цих методів до графів кінцевих автоматів додаються приховані підграфи, перехід в які здійснюється при певній недокументованій вхідній послідовності або певному внутрішньому стані. Потрапивши в прихований підграф, компонент починає генерувати послідовність бітів цифрового водяного знака. Недолік цих методів полягає в ймовірності потрапляння в прихований підграф в процесі нормального функціонування компонента, що призведе до збою; необхідності залучення додаткових ресурсів та можливості виявлення прихованих підграфів при аналізі структури компонента.

3) Методи використання незадіяних частин таблиці істинності блоків LUT. При використанні блоком LUT не всіх його входів частина таблиці істинності є незадіяною. Дана група методів застосовує цю частину таблиці для збереження цифрового водяного знака. Недоліком цієї групи методів є

тривіальна можливість виявлення частин водяного знака та наявність блокування застосування невикористаної частини таблиці з боку систем проєктування FPGA-компонентів.

4) Методи, які вводять до комбінаційної логіки модулів FPGA додаткові входи, розширюючи їх таблицю істинності. Зчитування водяного знака здійснюється за допомогою спеціальної комбінації значень додаткових входів, яка відкриває доступ до прихованої частини таблиці істинності, яка містить цифровий водяний знак. Основними недоліками такого підходу є потреба у додаткових апаратних ресурсах, а також ризик потрапляння в приховані області таблиці істинності, що провокує збій у роботі компонента.

5) Методи створення унікального розміщення компонентів та трасування сигналів між ними в просторі FPGA-компонента. Недоліком цих методів є витрати ресурсів та нестійкість до повторного синтезу або оптимізації.

6) Використання вільних ресурсів у самому файлі bitstream, які ігноруються при завантаженні, але можуть бути зчитані спеціальним програмним забезпеченням. Недоліками цих методів є низька стійкість до повторної генерації bitstream, а також легке виявлення та видалення вбудованого цифрового водяного знака.

Виходячи із зазначеного аналізу методів FPGA IP Watermarking можна констатувати, що ці методи мають значні обмеження, а саме: потребують застосування додаткових ресурсів (програмного коду, простору FPGA-компонентів, енергоспоживання); концентрують цифровий водяний знак в певних ділянках простору FPGA-компонентів; деякі з них крім цього підвищують ймовірність збоїв при переході в режим генерації водяного знака. У цілому методи FPGA IP Watermarking хоча і можуть забезпечувати захист авторства та виявлення незаконного копіювання FPGA IP Core, не мають властивостей, потрібних для зберігання контрольних даних в задачах прихованого моніторингу програмного коду FPGA.

1.9 Висновки до першого розділу

В першому розділі дисертації проведено аналіз наявних підходів, методів та засобів забезпечення моніторингу програмного коду FPGA компонентів комп'ютерних систем. Визначено місце мікросхем FPGA в номенклатурі елементної бази сучасних комп'ютерних систем. Показано, що FPGA використовуються переважно у складі високопродуктивних комп'ютерних систем, що обумовлено придатністю структури цих мікросхем для природньої організації паралельних обчислень. Це також значною мірою є мотивацією до використання мікросхем FPGA в складі комп'ютерних систем критичного застосування.

Проведено аналіз інцидентів останніх років, пов'язаних зі дестабілізацією роботи технічних об'єктів підвищеного ризику. Встановлено, що найзначнішим чинником збоїв в роботі критично важливих комп'ютерних систем є зловмисне втручання в їх функціонування. Виконано аналіз методів та засобів протидії втручанню в функціонування FPGA компонентів комп'ютерних систем. Показано, що одним з найефективніших підходів до протидії втручанню є оперативний моніторинг стану програмного коду цих компонентів. Виконання моніторингу потребує формування контрольних даних в момент підготовки програмного коду моніторингу, та їх зберігання з метою використання під час виконання актів моніторингу. Найбільш суттєвим фактором, який обумовлює можливість втручання визначено спосіб зберігання контрольних даних.

Розглянуто наявні традиційні підходи до зберігання контрольних даних систем моніторингу програмного коду. Також розглянуто перспективні підходи, які базуються на використанні цифрової стеганографії для збереження контрольних даних та мають переваги, які полягають у приховування контрольних даних та факту виконання моніторингу, а також забезпечують утворення єдиного цілого між об'єктом програмного коду та контрольними даними. Однак ці підходи мають і

недоліки, які полягають у відносно малому доступному обсязі даних, які можна зберегти в стеганографічний спосіб.

З урахуванням зазначених факторів виконано обґрунтування напрямку досліджень, направленого на збільшення обсягу стего-контейнера, утвореного програмним кодом FPGA при зберіганні контрольних даних прихованого моніторингу програмного коду.

В результаті обґрунтування напрямку досліджень сформульовано мету та визначено задачі дисертації.

Метою роботи є збільшення доступного обсягу для інформаційно замаскованого зберігання контрольних даних у процесі прихованого моніторингу програмного коду FPGA-компонентів інформаційних систем шляхом розробки моделей та методів гібридного стеганографічного вбудовування цих даних в програмний код.

Для досягнення мети в роботі необхідно вирішити такі ***задачі***:

– дослідити процеси забезпечення характеристик безпеки, а також характеристики правомірності використання та поширення щодо програмного коду FPGA-компонентів інформаційних систем; визначити фактори, які впливають на якість виконання моніторингу зазначених характеристик в умовах атак на систему моніторингу;

– розробити модель нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів та виконати оцінку обсягу надлишкових інформаційних ресурсів, забезпечених запропонованою моделлю, придатних для замаскованого зберігання контрольних даних;

– розробити метод нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, оснований на запропонованій моделі та виконати експериментальне дослідження ефективності цього методу для задач прихованого моніторингу програмного коду FPGA;

– розробити метод гібридного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA, який поєднує еквівалентний та нееквівалентний підхід до перетворення програмного коду FPGA-компонентів та процеси стеганографічного вбудовування та відновної

обфускації даних; виконати експериментальне дослідження ефективності цього методу для задач зберігання контрольних даних в процесі прихованого моніторингу програмного коду FPGA;

– розробити метод формування стеганографічного ключа для прихованого збереження контрольних даних в програмному коді FPGA, який дозволяє здійснити балансування між обсягом модифікованої частини програмного коду та ефективним обсягом стеганографічного контейнера, потрібним для збереження контрольних даних, а також виконати експериментальне дослідження ефективності цього методу;

– розробити підсистему гібридного інформаційно замаскованого зберігання контрольних даних в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів.

2 МОДЕЛІ ТА МЕТОД НЕЕКВІВАЛЕНТНОГО ЗАМАСКОВАНОГО ЗБЕРІГАННЯ КОНТРОЛЬНИХ ДАНИХ В СЕРЕДОВИЩІ ПРОГРАМНОГО КОДУ FPGA-КОМПОНЕНТІВ

Даний розділ дисертаційної роботи присвячено вирішенню задач розробки і дослідження моделей та методів замаскованого зберігання контрольних даних в середовищі програмного коду FPGA-компонентів комп'ютерних систем, шляхом виконання визначених нееквівалентних перетворень програмного коду.

В розділі запропоновано модель, яка формалізує умови та правила застосування нееквівалентних перетворень програмного коду FPGA для виділення надлишкових інформаційних ресурсів програмного коду та використання цих ресурсів при стеганографічному зберігання контрольних даних. На основі розробленої моделі у розділі виділено два види надлишкових інформаційних ресурсів програмного коду FPGA та на їх основі запропоновано правила формування стеганографічного простору для прихованого зберігання додаткових даних.

В розділі також вирішується задача оцінки обсягу стеганографичних ресурсів, забезпечених запропонованою моделлю. Зроблено висновки про достатність зазначеного обсягу для зберігання контрольних даних оперативного моніторингу програмного коду FPGA.

2.1 Модель нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів

2.1.1 Стеганогарфічне зберігання даних в інформаційних контейнерах з наближено поданими елементарними одиницями

В першому розділі дисертаційної роботи було показано, що елементами моніторингу програмного коду, які потребують розвитку та вдосконалення, є спосіб зберігання контрольних даних та спосіб доступу до

них. Одним із ефективних підходів до зберігання контрольних даних є стеганографічний підхід. Найбільшого свого розвитку цей підхід отримав стосовно мультимедійних стеганографічних контейнерів, таких як растрові зображення, цифрове відео та цифровий звук [116]. Це пояснюється тим, що зазначені контейнери мають аналогову природу походження. Цифрове представлення зображень, відео та звуку отримано шляхом реєстрації аналогових явищ. Результати цієї реєстрації оцифровані та подані у відповідних мультимедійних форматах. В результаті таких перетворень дані елементарних одиниць (пікселів, семплів) мультимедійних контейнерів мають наближене подання. Це означає, що незначні спотворення цих даних не тільки не призводять до деградації контейнера, але й зазвичай не реєструється.

Молодші біти елементарних одиниць інформаційного мультимедійного контейнера в просторовому домені або домені перетворень несуть настільки малий внесок в основну функцію контейнера, що їх спотворення не може бути сприйнято. З огляду на це, до мультимедійних контейнерів стеганографічне зберігання додаткових даних ефективно реалізується шляхом нееквівалентних перетворень. А саме, біти елементарних одиниць просторового або частотного доменів контейнера використовуються як сховища для стеганографічних даних [117].

Сучасні мультимедійні інформаційні контейнери мають значну кількість елементарних одиниць. Так, наприклад, типове растрове зображення, яке використовується як фотографія, зазвичай містить десятки або сотні мільйонів пікселів. Типовий звуковий файл – десятки мільйонів семплів. В силу цього обсяг додаткових даних, які можуть бути вбудовані стеганографічним способом у такі мультимедійні контейнери шляхом спотворюючих нееквівалентних перетворень, є значним (рис 2.1). Це призводить до відсутності дефіциту обсягу для стеганографічного зберігання додаткових контрольних даних у складі мультимедійних інформаційних контейнерів.

2.1.2 Можливості стеганографічного зберігання даних в інформаційних контейнерах з точно поданими елементарними одиницями

На відміну від мультимедійних інформаційних контейнерів, програмний код (і зокрема програмний код мікросхем FPGA) є інформаційним контейнером з точно поданими даними. Спотворення бітів програмного коду призводить до спотворення функціонування пристрою, що програмується цим кодом. Для стеганографічного вбудовування додаткових даних у програмний код FPGA зазвичай використовують методи еквівалентного перетворення [111-113]. Ці методи не змінюють функціонування мікросхеми FPGA та обсяг програмного коду, проте дозволяють приховано вбудувати в цей код додаткові дані.

Методи стеганографічного вбудовування додаткових даних у програмний код FPGA, які використовують еквівалентні перетворення є практичними, проте забезпечують малий потенційно доступний для зберігання обсяг додаткових даних. Це часто призводить до дефіциту обсягу зберігання контрольних даних у задачах оперативного моніторингу програмного коду FPGA, а також зменшує кількість різновидів моніторингу, які можуть бути застосовані до програмного коду (рис 2.1).

Рисунок 2.1 – Типи інформаційних контейнерів та механізми стеганографічного зберігання в них додаткових даних

В даній дисертаційній роботі показується, що для точно поданого інформаційного контейнера, яким є програмний код FPGA, також можливе стеганографічне вбудовування додаткових даних, подібне до того, що використовується в інформаційних контейнерах з наближено поданими елементарними одиницями (рис. 2.2). Виявлення, оцінка обсягу та використання інформаційних ресурсів для прихованого зберігання даних у програмному коді FPGA, що забезпечуються цією можливістю, є основними задачами дисертаційної роботи.

Зазначена можливість полягає в тому, що при реалізації наближеної обробки даних на FPGA, ця наближеність опосередковано переноситься на точно поданий програмний код. При виконанні наближених обчислень на FPGA частина елементарних блоків FPGA беруть участь у обчисленні лише несуттєвих розрядів результатів обчислень [118]. Програмний код таких елементарних блоків може бути змінений в нееквівалентний спосіб для вбудовування додаткових даних. Причому таке нееквівалентне вбудовування не відбивається на коректності виконання функції пристрою.

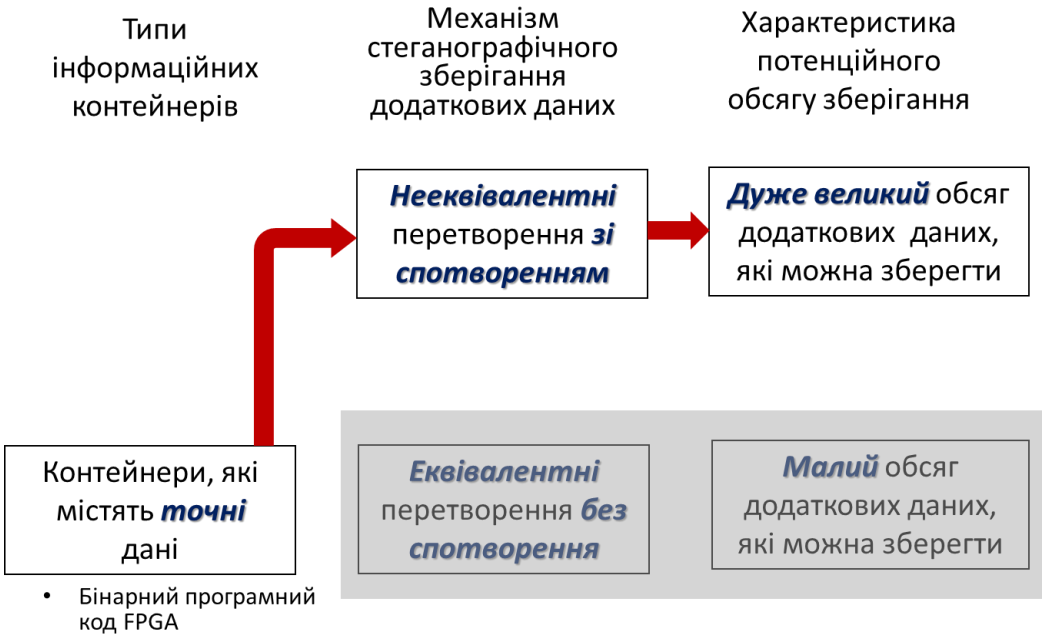


Рисунок 2.2 – Шлях застосування нееквівалентних перетворень для вбудовування додаткових даних в контейнери, що містять точні дані

У разі виконання наближеної обробки даних на FPGA програмний код FPGA має потенціал для застосування до нього нееквівалентних перетворень. Ці перетворення подібні до тих, які використовуються для мультимедійних контейнерів з наближено поданими елементарними інформаційними одиницями. Однак при цьому такі перетворення, що змінюють програмний код, в силу специфіки виконання операцій над наближеними даними, не впливають на основну функцію коду.

На основі цього спільно з відомими (основаними на еквівалентних перетвореннях) підходами до стеганографічного зберігання додаткових даних в програмному коді FPGA, пропонується застосувати підхід, що базується на нееквівалентних перетвореннях програмного коду тих частин FPGA, які виконують наближену обробку даних (рис. 2.3).

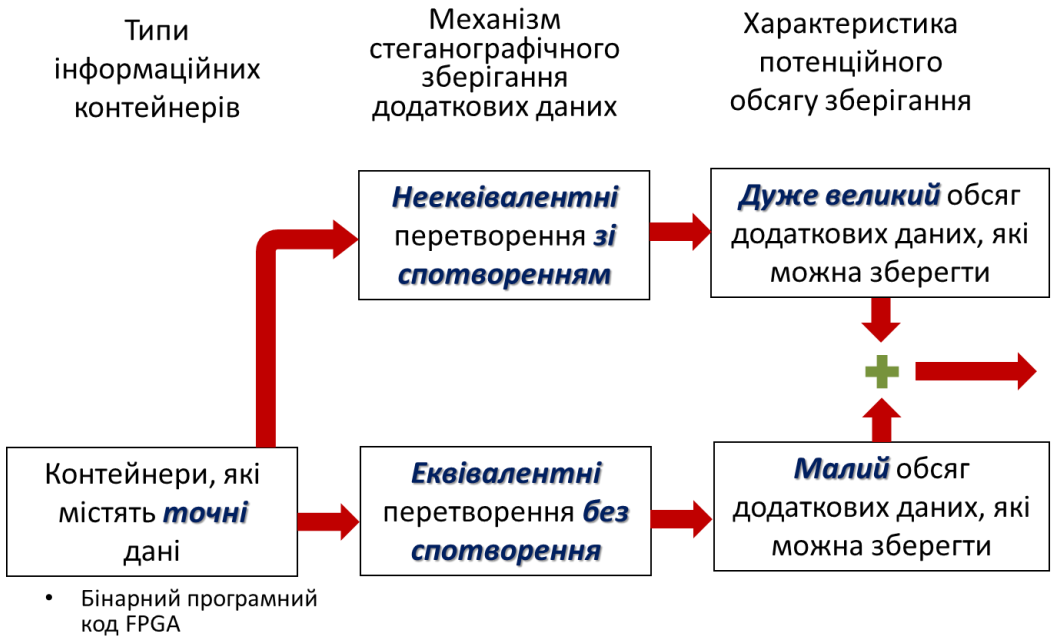


Рисунок 2.3 – Шляхи спільного застосування еквівалентних та нееквівалентних механізмів стеганографічного зберігання даних до програмного коду FPGA

Виходячи з цього, одними з основних задач даної дисертаційної роботи є задачі виявлення та оцінки обсягу стеганографічних ресурсів

програмного коду FPGA, які дозволяють виконувати нееквівалентне приховане вбудовування додаткових даних в нееквівалентний спосіб. Виявлення зазначених ресурсів у даній роботі виконується для FPGA-проєктів, модулі яких здійснюють наближену обробку даних. Оцінка ресурсів робиться для завдання стеганографічного зберігання контрольних даних, необхідних для прихованого оперативного моніторингу програмного коду FPGA.

2.1.3 Передумови виділення ресурсів програмного коду FPGA, які можуть бути використані для стеганографічного зберігання даних

Цілочисленні арифметичні операції в комп'ютерних системах зазвичай виконуються у форматах з фіксованою комою. Наближені обчислення виконуються у форматах із рухомою комою. При цьому над порядками чисел з рухомою комою виконуються цілі операції з фіксованою комою, а мантиси обробляються наближено. Найчастіше мантиси обробляються із збереженням єдиної розрядності для операндів і результатів, а обчислення завершуються округленням з втратою молодших обчислених розрядів та корекцією результату [118].

Виходячи з цього специфікації багатьох операцій наближеною обробки даних вимагають однаковості формату операндів та результату [119]. Однак вельми часто виникає ситуація, при якій результат операції природно має більшу розрядність в порівнянні з розрядністю операндів. У цьому випадку спочатку виконується обчислення повного результату, який потім доводиться до формату операндів шляхом відкидання деякої кількості молодших розрядів з наступним округленням.

На теперішній час одним з головних міжнародних стандартів, що специфікує подання чисел з рухомою комою та методи виконання операцій з такими числами є стандарт IEEE 754 [120]. Цей стандарт використовується в програмних та апаратних реалізаціях арифметичних

операцій. Доволі велика частка операцій в межах цього стандарту специфікована таким чином, що розмір результату повинен співпадати з розміром та форматом операндів.

Одним з прикладів таких операцій є арифметичні операції над числами з рухомою комою. Числа з рухомою комою містять в якості основних компонентів мантису та порядок. Порядки при виконанні арифметичних операцій обробляються точно, але мантиси обробляються наближено. При цьому найчастіше специфікація операції вимагає, щоб розрядність мантис результату дорівнювала розрядності мантис операндів (рис. 2.4).

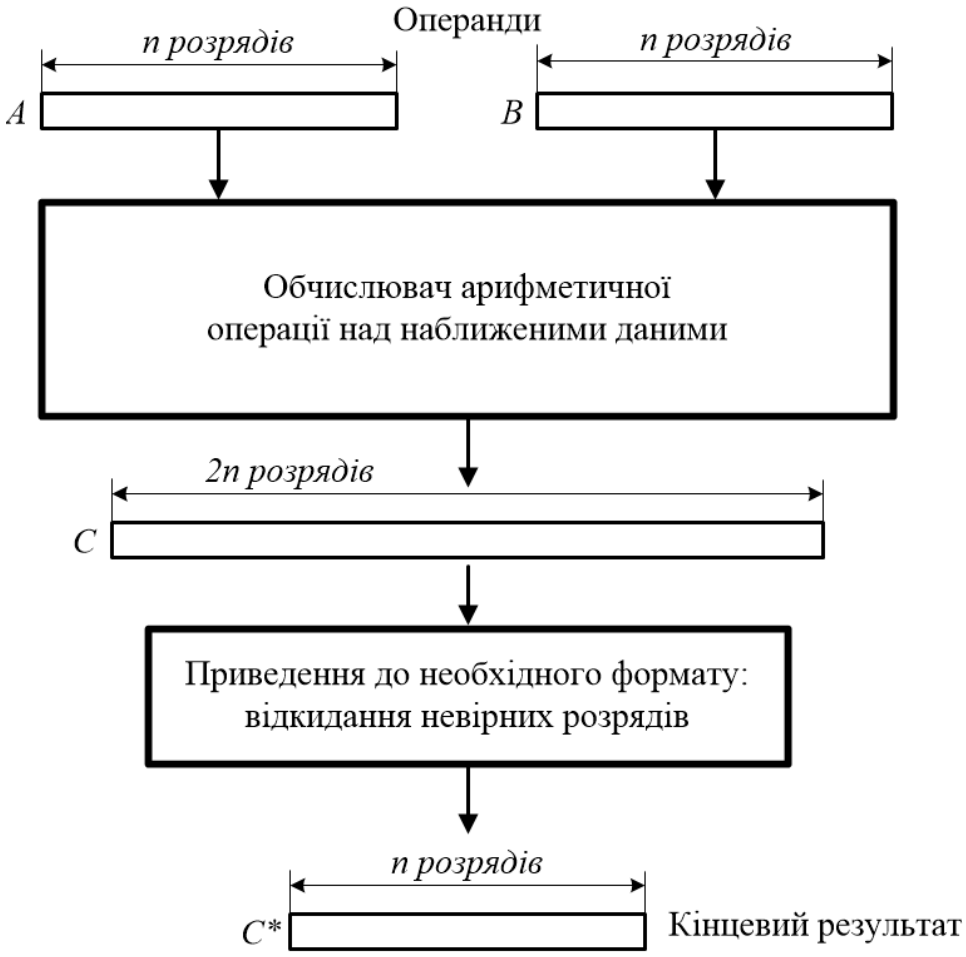


Рисунок 2.4 – Приведення розрядності мантис результату до розрядності мантис операндів

Наприклад, для отримання результату помноженні чисел з рухомою комою порядки операндів підсумовуються, а мантиси помножуються. Результатом помноження двох цілих двійкових n-розрядних чисел (якими є мантиси) є $2n$-розрядне двійкове число. Результат операції помноження чисел з рухомою комою найчастіше повинен мати розрядність мантиси, що дорівнює розрядності мантиси операндів. Для приведення $2n$-розрядного результату помноження мантис до n-розрядного формату молодші з $2n$ розрядів відкидаються, а результуючий порядок коректується, щоб компенсувати це відкидання.

Таким чином, при виконанні арифметичних операцій над числами з рухомою комою, у випадку необхідності приведення розрядності мантис результату до розрядності мантис операндів, виконується два крок:

– обчислюється повна версія мантиси;

– розрядність повної версії мантис приводиться до розрядності мантиси операндів, шляхом відкидання молодших розрядів.

При виконанні зазначених операцій, множина розрядів результату складається з двох непересічних підмножин: несуттєвих розрядів (що відкидаються) та суттєвих розрядів (що залишаються) в кінцевому результаті операції. Спотворення несуттєвих молодших розрядах результату не впливають на підсумковий результат операції.

Основними елементарними блоками структури мікросхем FPGA є блоки LUT (Look Up Table). У сучасних серіях FPGA кількість таких блоків становить мільйони та десятки мільйонів. Блок LUT має m входів (m для різних сімейств FPGA лежить зазвичай у діапазоні від 4 до 8) та виконує обчислення однієї логічної функції від m змінних. Блоки LUT є програмованими блоками. Налаштування блока реалізації конкретної логічної функції виконується 2^m бітним двійковим програмним кодом. Програмування мікросхеми FPGA призводить до налаштування кожного з блоків LUT на реалізацію необхідної логічної функції та створення необхідних зв’язків таких блоків один з одним.

При реалізації зазначений вище наближених арифметичних операцій над числами з рухомою комою на FPGA в її структурі можна виділити елементарні обчислювальні блоки LUT, які беруть участь у формуванні тільки несуттєвих розрядів результату. В силу цього програмний код таких блоків може бути модифікований і така модифікація не буде мати властивості впливу на результат роботи пристрою та його параметри.

2.1.4 Оцінка обмежень в застосуванні ресурсів програмного коду FPGA, які можуть бути використані для замаскованого зберігання даних

Ресурси програмного коду FPGA (які було показано в п. 2.1.3), що можуть бути використані для стеганографічного зберігання даних, виникають при виконанні наближених обчислень над числами з рухомою комою. Необхідно визначити обмеження застосування цих ресурсів, встановивши яке місце посідають наближенні обчислення в сучасній комп'ютерній сфері і яке місце займають ці обчислення в середовищі FPGA-компонентів.

Точні числа створюють основу обчислень, в той час як наближені числа роблять обчислення практичним. Тому в більшість прикладних задач обчислення виконуються саме над наближеними числами.

До сфер, де переважно використовуються наближені обчислення відносяться:

– *наукові обчислення*: моделювання клімату, моделювання погоди, обчислення в галузі фізики, хімії, біології тощо;

– *машинне навчання*: ваги нейронних мереж та проміжні значення, які виникають в процесі машинного навчання зазвичай є числами з рухомою комою;

– більша частина обчислень в *комп'ютерній 3D графіці*: розрахунки освітлення, тіней, текстур, фізики руку об'єктів;

– *цифрова обробка сигналів*: зв'язок, обробка цифрового звуку, відео;

– *фізичні симуляції* процесів та об'єктів;

– обчислення в задачах *комп'ютерного* зору.

Більшість зазначених областей потребує високопродуктивних обчислень часто в реальному часі. Природний паралелізмом та швидкісні характеристики FPGA-компонентів, обумовлюють суттєве поширення FPGA в цих галузях. Через це, а також з урахуванням статистики ринку FPGA можна констатувати, що наближені обчислення є найпоширенішою нішею використання FPGA-компонентів.

2.1.5 Формальне визначення моделі нееквівалентного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA-компонентів

Наявність множини несуттєвих розрядів, які відкидаються в процесі обчислень, та наявність множини елементарних блоків LUT FPGA, які ці розряди обчислюють, робить можливим використання програмного коду таких блоків LUT в якості інформаційного ресурсу для стеганогрфічного збереження контрольних даних моніторингу програмного коду. Узагальнення виділення та використання зазначеного інформаційного ресурсу представлено у вигляді моделі замаскованого зберігання контрольних даних в середовищі програмного коду блоків LUT FPGA. Модель пропонується для двох рівнів: локального та загального. Модель M_A^i локального рівня виділяє інформаційний ресурс для стеганографічного зберігання контрольних даних в середовищі програмного коду блоків LUT, які використовуються для реалізації однієї арифметичної операції над наближеними даними, поданими у вигляді чисел з рухомою комою. Модель загального рівня виділяє зазначений інформаційний ресурс для всіх операцій над наближеними даними, які виконуються в середовищі мікросхеми FPGA.

Модель локального рівня, що пропонується, являє собою кортеж виду:

$$M_A^i =< Op^i, A^i, B^i, C^i, C_E^i, C_{IE}^i, LUT_s^i, L_E^i, L_{IE}^i, L_M^i, L_{OIE}^i > \quad (2.1)$$

де Op^i – арифметична операція, яка виконується над двома наближеними числами, поданими у вигляді чисел з рухомою комою, та формує кінцевий результат у форматі, який співпадає з форматом операндів;

$A^i =< a_n^i,\ a_{n-1}^i,\ \dots, a_1^i >$ – n-розрядна мантиса першого операнда, який приймає участь у виконанні наближеної арифметичної операції Op^i над числами з рухомою комою;

$B^i =< b_n^i,\ b_{n-1}^i,\ \dots, b_1^i >$ – n-розрядна мантиса другого операнда, який приймає участь у виконанні наближеної арифметичної операції Op^i над числами з рухомою комою;

$C^i =< c_{2n}^i,\ c_{2n-1}^i, \dots, c_{n+1}^i,\ c_n^i,\ c_{n-1}^i, \dots,\ c_1^i >$ – $2n$-розрядний повний результат операції Op^i над мантисами A^i та B^i;

C_E^i та C_{IE}^i – відповідно суттєві та несуттєві (ті, що відкидаються при приведенні до кінцевого формату результату) розряди повного результату ($C_E^i \in C^i,\ C_{IE}^i \in C^i$);

LUT_s^i – множина блоків LUT FPGA, які використовуються для реалізації арифметичної операції Op^i;

L_E^i та L_{IE}^i – підмножини блоків LUT, які беруть участь в обчисленні відповідно суттєвих та несуттєвих розрядів повного результату. Причому $LUT_s^i = L_E^i \cup L_{IE}^i$;

L_M^i – підмножина блоків LUT, які одночасно беруть участь в обчисленні, як суттєвих, так і несуттєвих розрядів результату. Причому $L_M^i = L_E^i \cap L_{IE}^i$;

L_{OIE}^i – підмножина блоків LUT які беруть участь в обчисленні тільки несуттєвих розрядів результату, $L_{OIE}^i = L_{IE}^i \setminus L_E^i$.

Співвідношення між розрядами та підмножинами блоків LUT FPGA в межах моделі M_A^i графічно показані на рис. 2.5. Блоки LUT $L_{OIE}^i = L_{IE}^i \setminus L_E^i$, які беруть участь в обчисленні тільки несуттєвих розрядів результату, в

подальшому будуть називатися несуттєвими блоками LUT. Програмні коди цих блоків складають надлишковий інформаційний ресурс мікросхем FPGA, який виявляється моделюю M_A^i та може бути використаний для стеганографічного зберігання контрольних даних.

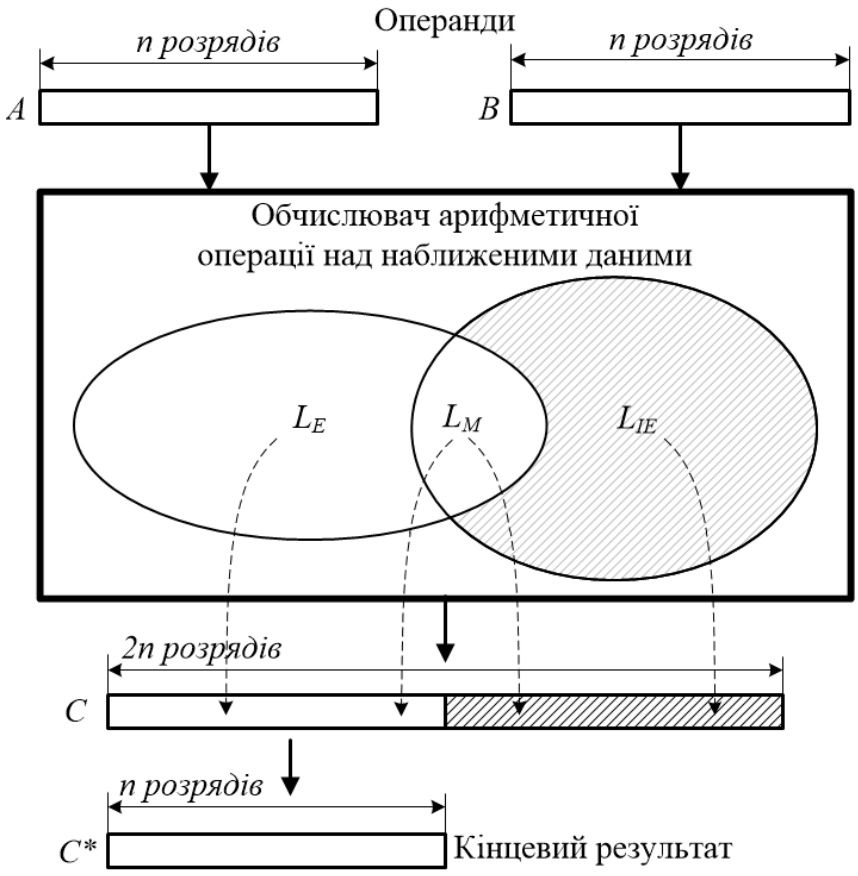


Рисунок 2.5 – Множини блоків LUT FPGA, задіяних при обчисленні розрядів, що залишаються та відкидаються в межах моделі M_A^i

Компонентами впорядкованої множини LUT_s^i блоків LUT FPGA, які використовуються для реалізації арифметичної операції Op^i в межах моделі (2.1) є m-входові блоки LUT:

$$LUT_s^i =< LUT^{i,1}, LUT^{i,2}, \dots, LUT^{i,w_i} > \qquad (2.2)$$

де w_i – кількість блоків LUT, що використовуються для виконання арифметичної операції Op^i.

Кожен з компонентів множини LUT_s^i є двокомпонентним кортежем виду:

$$LUT^{i,h} =< id^{i,h}, Code^{i,h} >; \quad h = 1 \dots w_i \tag{2.3}$$

де $id^{i,h}$ – унікальний ідентифікатор блока $LUT^{i,h}$ в межах структури мікросхеми FPGA;

$Code^{i,h}$ – програмний код блока $LUT^{i,h}$.

За умови, що кількості входів блоків $LUT^{i,h} \in LUT_s^i$ дорівнює m, кількість двійкових розрядів програмного коду кожного з цих блоків становить 2^m. Програмний код блока $LUT^{i,h}$ являє собою двійковий 2^m-компонентний вектор:

$$Code^{i,h} =< code_0^{i,h}, code_1^{i,h}, \dots, code_{2^m-1}^{i,h} > \tag{2.4}$$

Програмні коди несуттєвих блоків LUT $L_{OIE}^i = L_{IE}^i \setminus L_E^i$ (2.1), які беруть участь в обчисленні тільки несуттєвих розрядів результату, складають надлишковий інформаційний ресурс мікросхем FPGA, який виявляється моделюю M_A^i та може бути використаний для стеганографічного зберігання контрольних даних. Сукупність програмних кодів блоків L_{OIE}^i утворює простір для стеганографічного зберігання додаткових даних в середовищі програмного коду FPGA:

$$IRes_{Op^i} = \bigcup Code^{i,g} | \, LUT^{i,g} \in L_{OIE}^i \tag{2.5}$$

де $IRes_{Op^i}$ – надлишковий інформаційний ресурс FPGA, який виникає при реалізації наближеної арифметичної операції Op^i над числами з рухомою комою.

Запропонована модель M_A^i виділяє інформаційний ресурс для стеганографічного зберігання додаткових даних при здійсненні двоопераднних арифметичних операцій над числами з рухомою комою. Зазначена модель може бути пощирена на випадок арифметичних операцій з більшою кількість операндів.

Модель замаскованого зберігання контрольних даних в середовищі програмного коду LUT FPGA загального рівня, що пропонується, являє собою кортеж виду:

$$M_A =< M_A^1, M_A^2, \dots, M_A^p > \tag{2.6}$$

де компоненти $M_A^1, M_A^2, \dots, M_A^p$ кортежу є локальними моделями M_A^i для кожної з наближених арифметичних операцій, що виконується в середовищі даної FPGA над числами, поданими у вигляді чисел з рухомою комою, та формують кінцевий результат у форматі, який співпадає з форматом операндів. При цьому на компоненти кортежу (2.6) накладено обмеження, яке полягає в тому, що множини $LUT_S^i \in M_A^i$ кожного з компонентів кортежу (2.6) не пересікаються:

$$LUT_S^w \cap LUT_S^q = \emptyset; \tag{2.7}$$

де $LUT_S^w \in M_A^w;\ \ LUT_S^q \in M_A^q;\ w = 1 \dots p;\ \ q = 1 \dots p;\ w \neq q.$

Зазначене обмеження полягає в тому, що арифметичні операції $Op^i \in M_A^i$ з компонентів кортежу (2.6) реалізуються з використанням індивідуальних блоків LUT FPGA. Оскільки в практиці реалізації арифметичних операцій

на FPGA найчастіше операції реалізуються непересічними наборами блоків LUT, це обмеження не є звужуючим відносно виявлення ресурсів для зберігання додаткових даних в межах моделі M_A.

Сукупність несуттєвих блоків LUT $L_{OIE}^{i} = L_{IE}^{i} \setminus L_{E}^{i}$, які беруть участь в обчисленні тільки несуттєвих розрядів результату, всіх компонентів кортежу (2.6) становлять множину несуттєвих блоків LUT в межах всієї FPGA:

$$L_{OIE} = \bigcup_{k=1}^{p} (L_{OIE}^{k} \in M_A^k) \tag{2.8}$$

Програмні коди цих блоків L_{OIE}:

$$IRes_1 = \bigcup Code^{i,v} | \, LUT^{i,v} \in L_{OIE} \tag{2.9}$$

складають надлишковий інформаційний ресурс програмного коду мікросхем FPGA на загальному рівні, який виявляється моделюю M_A та може бути використаний для стеганографічного зберігання контрольних даних.

2.1.6 Види інформаційних ресурсів для стеганографічного зберігання додаткових даних в програмному коді FPGA, які проявляються запропонованою моделлю

Структуру реалізацій операцій наближеної обробки даних на FPGA спрямовано на обчислення повного природного результату з наступним штучним приведенням його до необхідного формату. Так, при виконанні операцій наближеної обробки даних для приведення результату до формату

операндів спочатку обчислюють повний результат, а потім його несуттєві розряди відкидають із застосуванням подальшого процесу округлення.

Виходячи з цього запропонована в цьому розділі дисертації модель дозволяє виділити в структурі схеми FPGA-системи підмножина блоків LUT, які виконують обчислення тільки несуттєвих розрядів результату і не беруть участь у обчисленні суттєвих розрядів. Якщо подібні блоки LUT мають місце у схемі FPGA-системи, то програмний код таких блоків, що становить надлишковий інформаційний ресурс, може бути змінений при вбудовуванні додаткових даних, і ця зміна не призведе до отримання неправильного результату функціонування системи. Такі блоки LUT відповідно до моделі M_A, запропонованої в даній дисертації, відносяться до множини несуттєвих блоків LUT. Всі інші блоки LUT (що не відносяться до підмножини несуттєвих) відносяться до множини суттєвих блоків LUT.

В цих умовах програмний код несуттєвих блоків LUT (цілком або окремі його розряди) може бути використаний як сховище прихованих додаткових даних, що вбудовуються в програмний код FPGA в стеганографічний спосіб. Таким чином, програмний код несуттєвих блоків LUT (2.9) можна виділити як основний надлишковий інформаційний ресурс $IRes_1$ у структурі мікросхем FPGA, який можна використовувати для вбудовування додаткових даних у програмний код FPGA шляхом виконання нееквівалентних перетворень.

При виконанні наближених арифметичних операцій над числами з рухомою комою несуттєвими можуть бути як блоки LUT, так й окремі розряди програмного коду блоків LUT. Несуттєвими розрядами програмного коду блоків LUT вважатимемо розряди, до яких під час обчислення суттєвих розрядів результату потенційно не проводиться звернення. Такі розряди доцільно виділити лише у програмному коді суттєвих L_E^i блоків LUT для кожної з арифметичних операцій Op^i відповідно до моделі M_A^i (2.1) на локальному рівні. Це пояснюється тим, що програмний код несуттєвих блоків LUT було виділено в даному розділі дисертаційної

роботи в якості самостійного виду стеганографічного ресурсу (2.9) відповідно до моделі M_A (2.6).

Сукупність двійкових розрядів програмного коду суттєвих блоків LUT L_E^i відповідно до моделі M_A (2.6) визначається як:

$$Code_E = \bigcup Code^{i,d} | \, LUT^{i,d} \in L_E \qquad (2.10)$$

Визначимо функцію *access* для вибору несуттєвих розрядів програмного коду суттєвих блоків LUT наступним чином:

якщо LUT^i – блок LUT, який бере участь у реалізації наближеної арифметичної операції Op^i над даними з рухомою комою і $code_E^i$ – розряд програмного коду блока LUT^i, тоді:

– $acces(code_E^r) = 1$, якщо в ході виконання арифметичної операції Op^i потенційно може мати місце звернення до розряду програмного коду $code_E^r$;

– $acces(code_E^r) = 0$ в іншому випадку.

Виходячи з цього, сукупність *несуттєвих розрядів* програмного коду блоків LUT, що реалізують наближені арифметичні операції над даними з рухомою комою, можна визначити як:

$$IRes_2 = \bigcup code_E^r | \, code_E^r \in Code_E \,\&\, acces(code_E^r) = 0 \qquad (2.11)$$

Ця сукупність розрядів становить надлишковий інформаційний ресурс програмного коду FPGA, який може бути використаний для стеганографічного зберігання контрольних даних.

Таким чином, у даному розділі дисертації на основі моделі M_A (2.6) виділено два види надлишкових інформаційних ресурсів програмного коду FPGA, які можуть бути використані для стеганографічного зберігання додаткових даних. На відміну від традиційних стеганографічних ресурсів, для ресурсів, виділених у даному розділі, вбудовування додаткових даних здійснюється в результаті застосування нееквівалентних перетворень їх значень. Ці ресурси віднесені до двох видів:

– $IRes_1$ (2.9) – сукупність двійкових розрядів програмних кодів несуттєвих блоків LUT L_{OIE} FPGA, що задіяні у виконанні наближених арифметичних операцій над числами з рухомою комою і обчислюють тільки несуттєві розряди результатів цих операцій;

– $IRes_2$ (2.11) – сукупність несуттєвих розрядів програмного коду блоків LUT FPGA, які:

а) є розрядами програмного коду блоків LUT, задіяних у виконанні наближених арифметичних операцій над числами з рухомою комою і обчислюють значення суттєвих розрядів результатів;

б) характеризуються тим, що до них при виконанні арифметичної операції потенційно не здійснюється звернення.

Формалізація методу використання двох виділених ресурсів та оцінка їх обсягу складають наступні завдання, що вирішуються в даному розділі дисертації.

2.2 Метод нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів

В даному підрозділі дисертації вирішується задача розробки методу, призначеного для замаскованого зберігання додаткових даних в середовищі програмного коду FPGA-компонентів, основаного на використанні нееквівалентних перетворень програмного коду відповідно до запропонованої моделі M_A (2.6).

Вхідні дані методу:

а) інформаційний об’єкт програмного коду FPGA-базованого компонента на рівні блоків LUT, побудований відповідно до моделі LUT-контейнера запропонованої в роботі [121]. Зазначений інформаційний об’єкт є сукупність: множини блоків LUT FPGA; множини зв’язків цих блоків між собою; множин входів та виходів LUT-контейнера, а також множини зв’язків блоків LUT з входами та виходами контейнера. Кожен з блоків в межах LUT-контейнера є сукупністю: ідентифікатора блока LUT, програмного коду цього блока, інформації про локалізацію блока LUT в програмованій матриці FPGA; множини вхідних портів та вихідного порту блока LUT;

б) послідовність двійкових розрядів $D = < d_1,\ d_2, \dots, d_s >$ додаткових даних, що вбудовуються до LUT-контейнера;

в) стего-ключ *Key*, який містить правило отримання впорядкованої множини блоків, що складають стего-шлях вбудовування.

Вихідні дані методу: заповнений LUT-контейнер, в програмний код якого в стеганографічний спосіб вбудовано двійкову послідовність *D*. Результуючий LUT-контейнер є функціонально та параметрично еквівалентним первісному LUT-контейнеру.

Пропонований метод базується на наступних положеннях.

Перше положення методу: схема FPGA-системи, що виконує наближену обробку даних, містить несуттєві блоки LUT. Ці блоки беруть участь у обчисленні несуттєвих (тих, які відкидаються при приведенні до результуючого формату) розрядів результату і не беруть участь в обчисленні суттєвих (тих, що залишаються) розрядів.

Друге положення методу: якщо програмний код несуттєвих блоків LUT піддається нееквівалентній модифікації, це не призводить до спотворення результату арифметичної операції, що випливає з визначення несуттєвих блоків LUT.

Третє положення методу: програмний код несуттєвих блоків LUT може бути використаний як сховище для стеганографічно вбудовуваних в нього даних;

Четверте положення методу: програмний код суттєвих блоків LUT має несуттєві розряди. Це розряди, до яких потенційно відсутні звернення під час обчислення суттєвих розрядів результату операції.

П'яте положення методу: нееквівалентна модифікація несуттєвих розрядів програмного коду блоків LUT не впливає на правильність результату арифметичної операції, що випливає з визначення несуттєвих розрядів програмного коду блоків LUT.

Шосте положення методу: програмний код несуттєвих бітів LUT може бути використаний як сховище для стеганографічно вбудовуваних в нього даних.

Вся сукупність програмного коду несуттєвих блоків LUT L_{OIE} (2.4) FPGA відповідно до моделі (2.6) є простором для потенційного зберігання прихованих додаткових даних у складі програмного коду FPGA. Цей простір за аналогією з простором молодших розрядів (в просторовій області або області перетворень) мультимедійних інформаційних контейнерів може бути використаний повністю або частково для стеганографічного вбудовування додаткових прихованих даних. Локалізація використовуваних елементів зазначеного простору виконується на етапі вбудовування додаткових даних. Правило локалізації задається стегоаногрфічним ключем. При вилученні вбудованих у даний простір даних за допомогою стеганографічного ключа виконується повторна локалізація використовуваних елементів простору.

Метод, що пропонується, відповідно до зазначених положень, складається з наступних кроків.

Крок 1. В структурі FPGA-проєкту виділяються підсхеми, які виконують наближену обробку даних для операндів, представлених у форматі з рухомою комою.

Крок 2. З підсхем, виділених на кроці 1, вибираються ті підсхеми, специфікація яких вимагає, щоб формат і розмір результатів виконання арифметичних операцій над числами з рухомою комою збігалися з форматор і розміром операндів.

Крок 3. Формуються підмножини блоків LUT, які відносяться до кожної з вибраних на попередньому кроці підсхем.

Крок 4. Визначається які стего-ресурси, відповідно до моделі M_A (2.6), застосовуються в процесі вбудовування додаткових даних в нееквівалентний спосіб:

– тільки розряди програмних кодів несуттєвих блоків LUT – стего-ресурс $IRes_1$ (2.9);
– тільки несуттєві розряди програмних кодів суттєвих блоків LUT – стего-ресурс $IRes_2$ (2.11);
– розряди обох стего-ресурсів – об'єднання $IRes_1 \cup IRes_2$ множин розрядів

Крок 5. За результатами визначення, які було отримано на попередньому кроці, в наступний спосіб формується двозарядна двійкова змінна ER:

– ER=10 – якщо на кроці 4 для вбудовування були обрані тільки розряди стего-ресурсу $IRes_1$;
– ER=01 – якщо на кроці 4 для вбудовування були обрані тільки розряди стего-ресурсу $IRes_2$;
– ER=11 – якщо на кроці 4 для вбудовування були обрані розряди об'єднання множин стего-розрядів $IRes_1 \cup IRes_2$.

Параметр ER включається в стего-ключ вбудовування та діставання додаткових даних в якості компонента. При цьому значення параметра ER=00 інтерпретується як незастосування нееквівалентних стего-ресурсів для вбудовування додаткових даних в програмний код FPGA. В цьому випадку для вбудовування використовуються лише еквівалентні перетворення програмного коду відповідно до відомих методів.

Крок 6. Якщо ER(0) = 1, то на кожній з обраних на кроці 3 підмножин блоків LUT визначаються несуттєві блоки LUT. Для цього виконується моделювання кожної з підсхем, що виконують наближену обробку, з метою з'ясування, які з блоків LUT обчислюють тільки несуттєві розряди результату і не беруть участі в обчисленні суттєвих розрядів.

Якщо ER(1) = 1, то на кожній з обраних на кроці 3 підмножин блоків LUT визначається множина несуттєвих розрядів програмного коду суттєвих блоків LUT для підсхем, виділених на кроці 2.

Крок 7. З множин розрядів $IRes_1$ програмних кодів несуттєвих блоків LUT та несуттєвих розрядів $IRes_2$ програмних кодів суттєвих блоків LUT виконується формування ефективного нееквівалентного стего-ресурсу для вбудовування додаткових даних в програмний код FPGA: $IRes = IRes_1 \cup IRes_2$ при $ER = 11$; $IRes = IRes_1$ при $ER = 10$; $IRes = IRes_2$ ER $= 01$.

Крок 8. Розряди отриманого стего-ресурсу $IRes$ впорядковуються відповідно до первинного впорядкування блоків LUT і розрядів їх програмних кодів, яке задається архітектурою FPGA.

Крок 9. Формується компонент стего-ключа, який призначений для локалізації вбудовуваних даних у середовищі нееквівалентного стего-ресурсу, отриманого на попередньому кроці.

Крок 10. У середовище стего-ресурсу $IRes$, виділеного на кроці 7 відповідно до стего-ключа, сформованого на кроці 9, виконується вбудовування двійкової послідовності $D = < d_1, d_2, \dots, d_s >$ додаткових даних. Додаткові дані вбудовуються в цільові розряди стего-ресурсу $IRes$ шляхом заміни розрядів $IRes$ на відповідні розряди додаткових даних або в похідний спосіб визначений стего-ключем.

Наступні підрозділи даного розділу дисертації присвячені експериментальній оцінці обсягу стего-ресурсів $IRes_1$ (2.9) та $IRes_2$ (2.11), що виділяються моделлю M_A (2.6), а також експериментальному дослідженню ефективності запропонованого методу.

2.3 Експериментальна оцінка обсягу даних, який може бути вбудовано в програмний код FPGA методом нееквівалентного замаскованого зберігання контрольних даних

В даному підрозділі дисертації вирішується задача оцінки обсягу стего-ресурсів, виділених запропонованою моделлю M_A (2.6) та доступних для використання в межах розробленого методу зберігання даних в середовищі програмного коду FPGA.

2.3.1 Оцінка обсягу стего-ресурсів, утворених розрядами програмних кодів несуттєвих блоків LUT

Вирішується завдання оцінки обсягу даних які потенційно можуть бути вбудовані в програмний код FPGA завдяки використанню стего-ресурсу $IRes_1$ (2.9) – програмних кодів несуттєвих блоків LUT при виконанні на FPGA наближеної обробки даних, поданих у вигляді чисел з рухомою комою. Визначення несуттєвих блоків LUT, задіяних в зазначених обчисленнях, можливе шляхом моделювання підсхем обчислювачів (отриманих на кроці 2 запропонованого методу), які виконують обробку мантис чисел з рухомою комою за умови отримання результату в форматі, який співпадає з форматом операндів.

2.3.1.1 Базовий спосіб локалізації несуттєвих блоків LUT

Базовий спосіб моделювання (який не може бути застосований в загальному випадку, але є основою для похідних ефективних способів) для локалізації несуттєвих блоків LUT полягає в послідовному застосуванні до схеми обчислювача, що обробляє мантиси, всіх можливих наборів вхідних даних з обчисленням вірного результату. На кожному наборі вхідних даних крім цього виконуються всі можливі спотворення та комбінації таких

спотворень на виходах всіх блоків LUT підсхеми з аналізом помилок результату обчислення. Блок LUT, який при спотворенні його виходу на всіх вхідних наборах даний та при всіх комбінаціях спотворень виходів інших блоків LUT додає помилки тільки в несуттєві розряди результату за підсумками такого моделювання вважається несуттєвим блоком LUT.

Недоліком такого базового способу локалізації несуттєвих блоків LUT є вкрай велика кількості ітерацій моделювання. Це обмежує можливість застосування цього способу, через що, він може бути застосований до схемам обчислювачів дуже малої розмірності (розмірність кожного з операндів обмежується 4-5 розрядами, а кількість блоків LUT схеми обчислювача – 20-30 блоками).

Виходячи з неможливості застосування базового способу для обчислювачів реальної розмірності, пропонується наступний підхід для локалізації несуттєвих блоків LUT, який використовує певні елементи описаного вище базового способу, але виконує локалізацію за значно меншу кількість ітерацій моделювання, що дає змогу застосувати його до обчислювачів, які використовуються на практиці. Підхід оснований на врахуванні структурних та арифметичних особливостей побудови обчислювачів, які обробляють мантиси чисел з рухомою комою. В межах цього підходу пропонується визначити верхню та нижню оцінку кількості несуттєвих блоків LUT.

2.3.1.2 Отримання верхньої оцінки кількості несуттєвих блоків LUT

Верхня оцінка кількості несуттєвих блоків LUT блоків отримується шляхом обмеженого (порівняно з базовим способом моделювання) моделювання схеми обчислювача та визначення блоків LUT, які є суттєвими для виконання операції над наближеними даними.

Відповідно до моделі M_A^i (2.1) в множині блоків LUT_S^i які використовуються для реалізації арифметичної операції над наближеними даними виділяються дві підмножини: L_E^i та L_{IE}^i – підмножини блоків LUT,

які беруть участь в обчисленні відповідно суттєвих та несуттєвих розрядів повного результату. Причому $LUT_s^i = L_E^i \cup L_{IE}^i$ та $L_E^i \cap L_{IE}^i \neq \emptyset$. Отримання верхньої оцінки кількості несуттєвих блоків LUT L_{IE}^i можливо звести до оцінки кількості суттєвих блоків LUT L_E^i.

Нехай $|LUT_s^i|$ – потужність множини блоків LUT, що реалізують операцію Op^i наближеної обробки даних відповідно до моделі M_A^i (2.1), $|L_{IE}^i|$ – потужність підмножини несуттєвих блоків LUT цієї множини, а $|L_E^i|$ – потужність підмножини суттєвих блоків LUT в множині LUT_s^i. Тоді з співвідношень $LUT_s^i = L_E^i \cup L_{IE}^i$ та $L_E^i \cap L_{IE}^i \neq \emptyset$ випливає:

$$LUT_s^i = L_E^i \cup L_{IE}^i;$$
$$L_{IE}^i = LUT_s^i \setminus L_E^i;$$

тоді

$$\begin{aligned} \left|L_{IE}^i\right| = \left|LUT_s^i \setminus L_E^i\right| = \left|LUT_s^i\right| - \left|LUT_s^i \cap L_E^i\right| = \\ = \left|LUT_s^i\right| - \left|\left(L_E^i \cup L_{IE}^i\right) \cap L_E^i\right| = \left|LUT_s^i\right| - \left|L_E^i\right|. \end{aligned} \quad (2.12)$$

Таким чином при відомій загальній кількості блоків LUT LUT_s^i підсхеми обробки наближених даних оцінка кількості $|L_E^i|$ суттєвих блоків LUT дає можливість оцінити кількість несуттєвих блоків $|L_{IE}^i|$.

Кількість суттєвих блоків LUT може бути оцінена при моделюванні на всіх можливих вхідних даних як кількість блоків LUT, спотворення вихідного значення яких впливає на зміну значень суттєвих розрядів арифметичної операції, що виконується. Відмінність процедури моделювання для визначення суттєвих блоків LUT від базового способу моделювання полягає в відсутності необхідності оцінки впливу спотворень блока LUT при всіх можливих комбінаціях сотворений та неспотворених значень на виходах інших блоків підсхеми. Ця відмінність суттєво звужую простір пошуку рішення при моделюванні та кількість ітерацій моделювання відносно базового способу.

2.3.1.3 Отримання нижньої оцінки кількості несуттєвих блоків LUT

Отримання нижньої оцінки кількості несуттєвих блоків LUT для обчислювача, який виконує наближену обробку операндів, поданих у вигляді чисел з рухомою комою, базується на врахуванні структурних та арифметичних особливостей побудови таких обчислювачів. За умови вимоги про відповідність формату результату форматам операндів, обчислювач в частині обробки мантис за n-розрядними операндами отримує повний $2n$-розрядний результат з послідуючим округленням та відкиданням молодших n-розрядів.

Відомим є метод скорочення структури матриці помноження мантис для описаного випадку [118]. Відповідно до цього методу матриця помноження розбивається на старшу та молодшу частини. На рис. 2.6 структура матричного помножувача мантис умовно показана у вигляді матриці кон'юнкцій добутку двох n-розрядних двійкових чисел при виконанні їх помноження. Молодша частина матриці складається з k-стовбців. Метод скорочення структури матриці помноження [118] обумовлює виключення молодшої частини матриці зі схеми. При цьому старша частина матриці обчислює усічений $2n–k$ -розрядний результат $V(2n…k+1)$, старші n-розрядів якого складають кінцевий результат $V(2n…n+1)$, формат якого відповідає формату операндів.

В зазначеному методі величина k визначається за умови відсутності впливу молодшої частини схеми матриці на округлений кінцевий результат $V(2n…n+1)$. Значення k розраховуються виходячи з максимального значення складової повного результату, яку створює молодша частина матриці. Таке максимальне значення утворюється коли вся молодша частина матриці кон'юнкцій добутку заповнена одиницями. Відповідно до положень методу [118] значення k приближено оцінюється за формулою:

$$k = n – log_2 n;$$

де n – розмірність операндів та кінцевого результату.

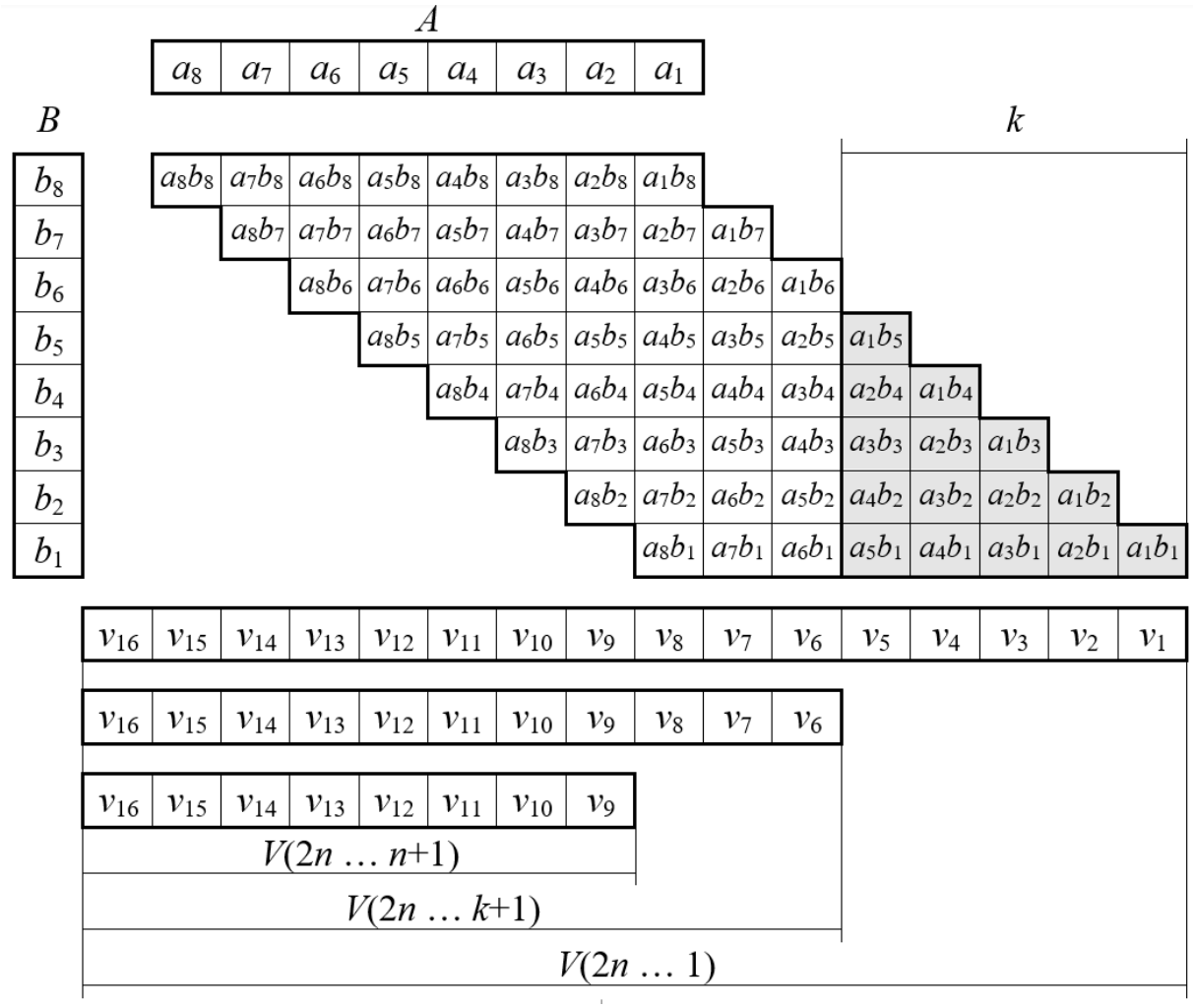


Рисунок 2.6 – Умовне позначення структури матричного помножувача мантис у вигляді матриці кон’юнкцій добутку *n*-розрядних двійкових чисел

Зазначені положення методу скорочення матриці помноження мантис [118] пропонується використати для встановлення нижньої оцінки кількості несуттєвих блоків LUT. Блоки LUT, які відповідають молодшим *k*-стовбцям матриці в структурі обчислювача є несуттєвими через відсутність впливу значень на виходах цих блоків на округлений кінцевий результат. Будь-які спотворення значень на виходах зазначених блоків, що відносяться до молодшої частини матриці в структурі обчислювача, не приводять до змін в суттєвих розрядах результату та розрядах кінцевого результату обчислення. Аналіз структури матриці обчислювача у вигляді списків блоків LUT, а також списків зв’язків цих блоків між собою та з входами та виходами

обчислювача дає можливість виявити сукупність блоків, які відносяться до молодшої частини матриці обчислювача.

2.3.2 Експериментальне дослідження доступного обсягу замаскованого зберігання даних, забезпеченого несуттєвими блоками LUT

В попередніх підрозділах дисертаційної роботи було обґрунтовано наявність в інформаційному контейнері, утвореному програмним кодом мікросхем FPGA, надлишкового інформаційного ресурсу $IRes_1$, який забезпечений несуттєвими блоками LUT та придатний для замаскованого зберігання додаткових даних.

Мета експериментального дослідження полягає в визначенні кількості несуттєвих блоків LUT у складі підсхем FPGA, які реалізують наближену обробку даних, поданих у вигляді чисел з рухомою комою.

Середовище проведення експерименту склали наступні програмні засоби.

1) Синтез та розміщення і трасування FPGA-проєктів виконувалося в САПР Intel Quartus Prime 20.1 Lite Edition [122] для цільових мікросхем FPGA Intel Cyclone IV EP4CE15F23A7 [123].

2) Для виконання експерименту було розроблено програмний додаток LUTListExtractor, призначення якого полягає в добуванні детальної інформації про структуру FPGA-проєкту з внутрішньої бази даних САПР Intel/Altera Quartus Prime. Ця детальна інформація містить список блоків LUT FPGA-проєкту, значення програмних кодів цих блоків і список зв’язків блоків LUT між собою. Додаток розроблено мовою TCL [124], застосування якої є стандартним механізмом взаємодії між модулями САПР в галузі проєктування систем на базі FPGA. Додаток LUTListExtractor отримує на вхід FPGA-проєкт, виконується в середовищі САПР Intel/Altera Quartus Prime, дістає з внутрішньої бази даних САПР детальну інформацію про

блоки LUT та структуру зв’язків між ними, та видає на вихід цю інформацію у вигляді сукупності структурованих текстових файлів.

3) Для виконання експерименту також було розроблено програмний додаток EsLUTGetter, який приймає від додатка LUTListExtractor на вхід інформацію про сукупність блоків LUT та їх зв’язки в FPGA-проєкті та виконує моделювання LUT-схеми за принципами наведеними в п. 2.3.1.2 дисертаційної роботи. Мета функціонування додатка полягає в визначенні кількості суттєвих блоків LUT, що дає змогу оцінити кількість несуттєвих блоків LUT.

4) Також для виконання експерименту було розроблено програмний додаток LMatrixLUTGetter, який приймає від додатка LUTListExtractor на вхід інформацію про сукупність блоків LUT та їх зв’язки в FPGA-проєкті, аналізує структуру матриці обчислювача, визначає розмір молодшої частини матриці в структурі обчислювача та відповідно до положень, наведених в п. 2.3.1.3 дисертації, виконує пошук блоків LUT, які відносяться до молодшої частини матриці.

В якості цільових мікросхем синтезу при проведенні експерименту були використані мікросхеми FPGA Intel Cyclone IV EP4CE15F23A7. Обрання мікросхем FPGA сімейства Intel Cyclone IV в якості цільових мікросхем синтезу обумовлено поширеністю їх використання, а також подібністю їх структури до структур класичних FPGA та сімейств Intel/Altera Cyclone II, III та III LS.

Моделювання проводилося в *середовищі обчислювальної системи* на базі 8-ядерного процесора AMD Ryzen 7 при обсязі оперативної пам’яті 16 ГБ.

Вихідні дані експерименту: п’ять FPGA-проєктів, які складаються з модулів помноження мантис та мають розрядність кожного з операндів відповідно 4, 6, 8, 12 та 16 розрядів та розрядність до округлення відповідно

8, 12, 16, 24 та 32 розряди. Після округлення розрядність результату збігається з розрядністю операндів.

Методика проведення експерименту полягає в наступному.

1) В середовищі САПР Intel Quartus Prime 20.1 Lite Edition формується п’ять FPGA-проєктів, кожен з яких містить модулі помноження мантис з розрядність операндів 4, 6, 8, 12, 16 відповідно.

2) В середовищі цієї ж САПР виконується синтез, розміщення та трасування проєктів для цільових мікросхем FPGA.

3) Кожний з експериментальних FPGA-проєктів подається на вхід розробленого додатка LUTListExtractor, код якого виконується в середовищі САПР Intel Quartus Prime 20.1 Lite Edition. Додаток отримує детальну інформацію про блоки LUT, їх програмні коди та структуру зв’язків блоків між собою з бази даних проєкту Intel Quartus Prime. В результаті роботи додатка LUTListExtractor для кожного з експериментальних FPGA-проєктів формуються впорядковані набори даних про схему обчислювача на рівні блоків LUT.

4) Дані, отримані на попередньому кроці експерименту подаються на вхід розробленого додатка EsLUTGetter, який виконує моделювання LUT-схеми за принципами наведеними в п. 2.3.1.2 дисертаційної роботи.

5) За результатом моделювання в розробленому програмному додатку EsLUTGetter отримується список суттєвих блоків LUT для кожного експериментального FPGA-проєкта. На основі цього списку підраховується кількість суттєвих блоків LUT та робиться верхня оцінка кількості несуттєвих блоків LUT.

6) Дані, отримані на кроці 3 експерименту подаються на вхід розробленого додатка LMatrixLUTGetter, який обчислює розмір молодшої частини матриці в структурі обчислювача та відповідно до положень, наведених в п. 2.3.1.3 дисертації, виконує пошук блоків LUT, які відносяться до молодшої частини матриці. За результатом функціонування додатка LMatrixLUTGetter отримується список несуттєвих блоків LUT, які

розміщені в молодшій частині матриці обчислювача для кожного експериментального FPGA-проєкта. На основі аналізу цього списку робиться нижня оцінка кількості несуттєвих блоків LUT.

Зв’язки між програмними додатками та програмними модулями, які становлять середовище виконання експерименту показано на рис. 2.7.

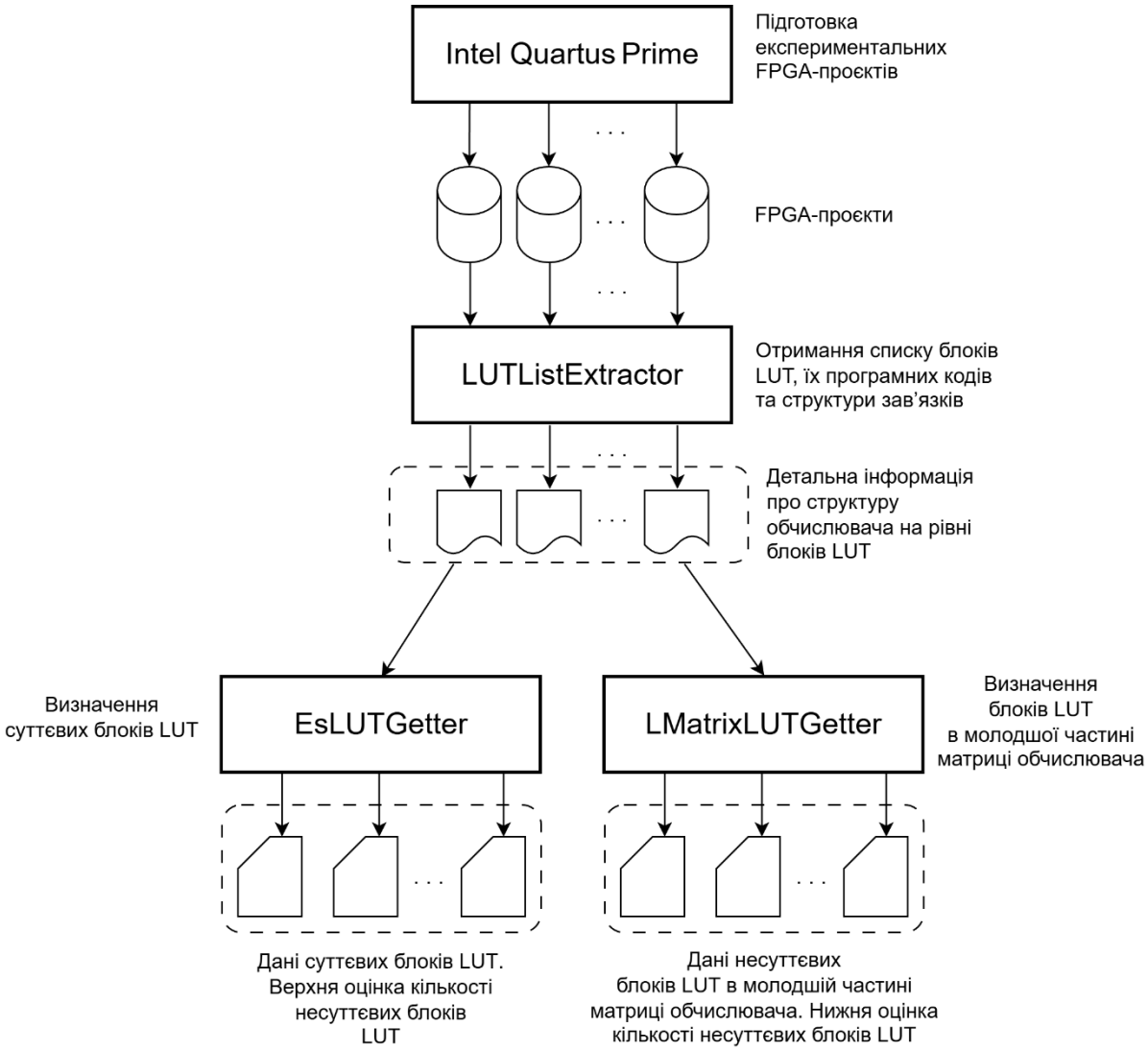


Рисунок 2.7 – Зв’язки між програмними додатками та програмними модулями, які становлять середовище виконання експерименту з дослідження стеганографічного ресурсу, забезпеченого несуттєвими блоками LUT

Експериментальне визначення кількість суттєвих блоків LUT та отримання верхньої оцінки кількості несуттєвих блоків LUT полягає в

наступному. Для кожного з можливих спільних значень операндів помножувача, який міститься в експериментальному FPGA-проєкті виконуються наступні кроки.

1. Обчислюється $2n$-розрядний результат помноження операндів.

2. Фіксуються вірні значення на виходах кожного із блоків LUT FPGA-проєкту помножувача.

3. Значення на виході кожного із блоків LUT FPGA-проєкту послідовно інвертуються (змінюються з вірного значення на невірне). При кожному з таких інвертувань фіксується величина різниці меж вірним результатом, та результатом, отриманим в при інвертуванні. Величина різниці обчислюється за модулем. При цьому кожному із блоків LUT проєкту ставиться у відповідність величина різниці, що виникла в результаті інвертування значення виходу цього блока. Збережена величина різниці оновляється при отриманні поточної різниці, більшої, за збережену.

Величини різниць, отримані в результаті аналізу значень операндів, дозволяють прийняти рішення щодо того, чи відноситься блок LUT до підмножини суттєвих блоків. Якщо різниця, яка поставлена у відповідність блоку LUT n-розрядного помножувача, є більшою за 2^n, то ця різниця проявляється у вигляді помилки в суттєвих розрядах результату C_E^i, а блок LUT відноситься до суттєвих блоків. Таким чином, продукційне правило для віднесення блока LUT n-розрядного помножувача, який реалізує операцію Op^i наближеної обробки даних, до підмножини суттєвих блоків має наступний вигляд:

$$\text{if } difference(LUT^{i,h}) \geq 2^n \text{ then } LUT^{i,h} \in L_E^i$$

де $difference(LUT^{i,h})$ – значення різниці меж вірним результатом, та результатом, отриманим при інвертуванні виходу блока, поставлене у відповідність блокові $LUT^{i,h}$.

На рис. 2.8 показане вікно інтерфейсу додатка EsLUTGetter наприкінці процедури моделювання для 8-розрядного помножувача мантис.

У вікні інтерфейсу для кожного з блоків LUT 1…101 обчислювача наведені значенні мінімальної та максимальної різниці між вірним значенням добутку операндів та значенням, отриманим в результаті інвертування виходу відповідного блока LUT на повному циклі моделювання.

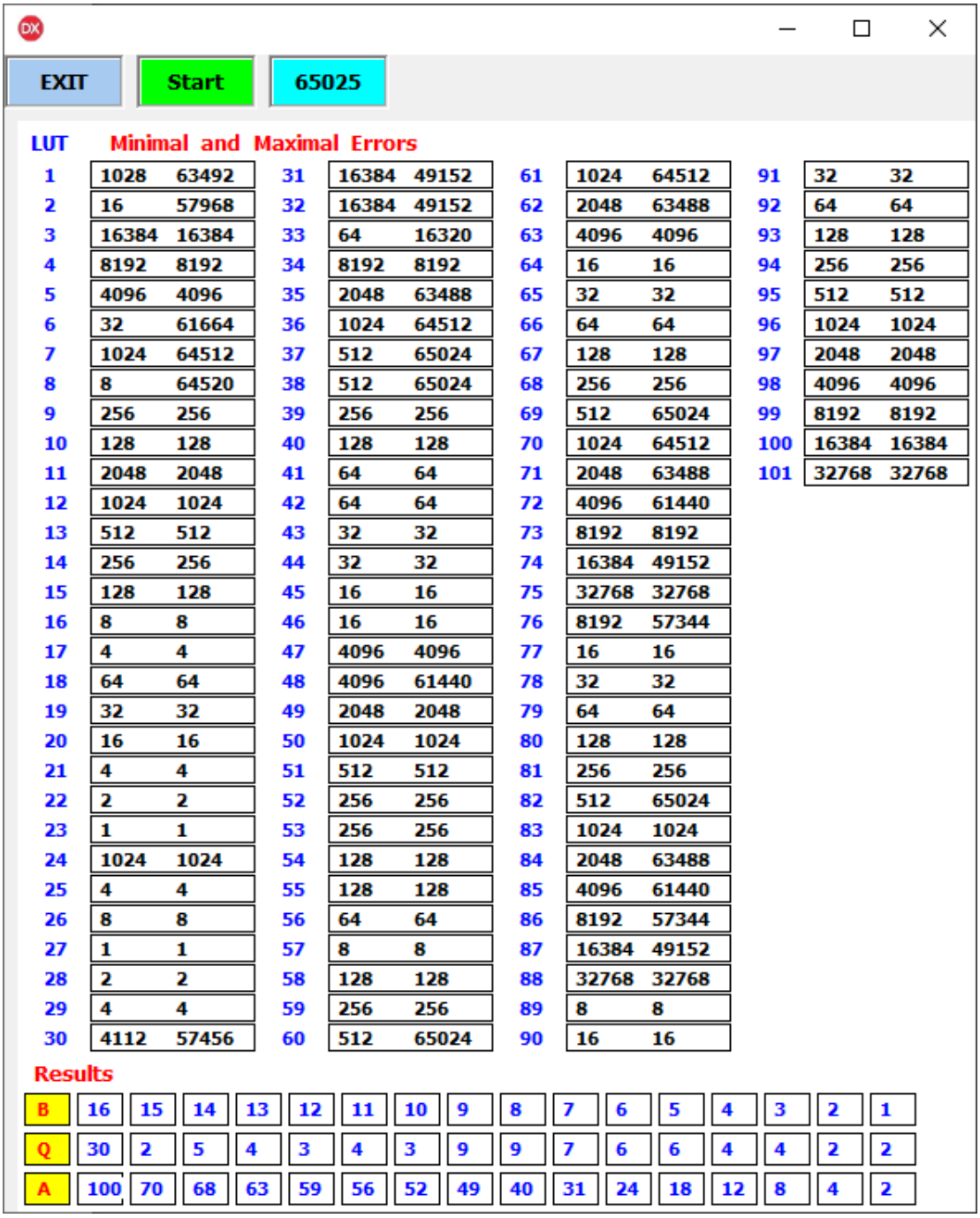


Рисунок 2.8 – Вікно розробленого додатка EsLUTGetter з результатами обробки 8-розрядного обчислювача

Обчислювач, результат обробки якого показаний у вікні додатка EsLUTGetter, має 101 блок LUT. У вікні додатка для кожного з блоків LUT

наведено значення мінімальної та максимальної різниці між вірним значенням добутку операндів та значенням, отриманим в результаті інвертування виходу відповідного блока LUT на повному циклі моделювання. При цьому для прийняття рішення стосовно віднесення блоку LUT до суттєвих використовується тільки значення максимальної різниці. Значення мінімальної різниці використовується тільки в якості допоміжного. Крім значень різниць додаток візуалізує глобальну статистику розподілення кількості блоків LUT за значеннями різниць. Значення B (Bits) показує номері розрядів результату виконання операції до округлення. Значення Q (Quantity) – кількість блоків LUT які впливають на значення у відповідному розряді. Значення A (Amount) – частка блоків LUT, вплив яких на результат не перевищує вагу даного розряду.

Результати експериментального дослідження з визначення кількості суттєвих блоків LUT представлені в табл. 2.1. У таблиці наведено експериментальні результати, отримані для помножувачів мантис чисел з рухомою комою з розрядністю операндів 4, 6, 8, 12, 16 розрядів. В експериментальних проєктах FPGA частка суттєвих блоків LUT лежить в діапазоні 57,48% ... 60,65% від загальної кількості блоків LUT проєктів. За значеннями кількості суттєвих блоків LUT отримано верхню оцінку (2.12) кількості несуттєвих блоків яка лежить в діапазоні від 39,35% … 42,52% від загальної кількості блоків LUT експериментальних FPGA-проєктів.

Показано абсолютні значення складності виконання моделювання для отримання кількості суттєвих блоків LUT. Показано, що складність моделювання практично лінійно залежить від кількості потенційно можливих вхідних даних обчислювачів на повному циклі моделювання.

Також в табл. 2.1. наведено наближену оцінку обсягу даних, які можуть бути в стеганографічний спосіб приховано вбудовані в програмний код обчислювачів, реалізованих в межах експериментальних FPGA-проєктів. Оцінку обсягу стего-контейнера за результатами даного

експерименту зроблено виходячи з верхньої оцінки кількості несуттєвих блоків LUT.

Таблиця 2.1 – Результати експериментального визначення кількості суттєвих блоків LUT та верхньої оцінки кількості несуттєвих блоків LUT

Розрядність операндів та кінцевого результату	4	6	8	12	16
Розрядність результату до округлення	8	12	16	24	32
Загальні кількість блоків LUT	30	61	101	208	341
Кількість суттєвих блоків LUT	18	37	61	122	196
Частка суттєвих блоків LUT серед усіх блоків	60%	60,65%	60,4%	58,65%	57,48%
Верхня оцінка кількості несуттєвих блоків LUT	12	24	40	86	145
Верхня оцінка частки несуттєвих блоків LUT серед усіх блоків	40%	39,35%	39,6%	41,35%	42,52%
Час моделювання (с)	0.00005	0.00095	0.01425	3.10364	821.75912
Мінімальний обсяг, доступний для зберігання даних (біт)	12	24	40	86	145
Максимальний обсяг, доступний для зберігання даних (біт) при кількості входів блоків LUT – 4	192	384	640	1376	2320

Функціонування блока LUT з m входами визначається 2^m-бітним програмним кодом (в залежності від конкретного сімейства мікросхем FPGA m становить від 4 до 8). Таким чином, кожний несуттєвий блок LUT може використовуватися для вбудовування в стеганографічний спосіб мінімум 1 і максимум 2^m розрядів додаткових даних. Для блока LUT з 4 входами обсяг пам'яті програмного коду складає 16 розрядів. В цьому випадку обсяг, доступний для зберігання даних в стеганографічний спосіб, забезпечений несуттєвими блоками LUT, змінюється від 12 до 145 біт при зміні розрядності операндів з 4 до 16 розрядів при використанні тільки одного біту стего-даних на блок LUT. При використанні для вбудовування стего-даних всіх розрядів програмного коду несуттєвих блоків LUT обсяг стего-контейнера в середовищі програмного коду експериментальних проєктів змінюється з 192 до 2320 біт при збільшенні розміру операндів з 4 до 16 розрядів.

На рис. 2.9. результати експерименту показані у вигляді діаграми.

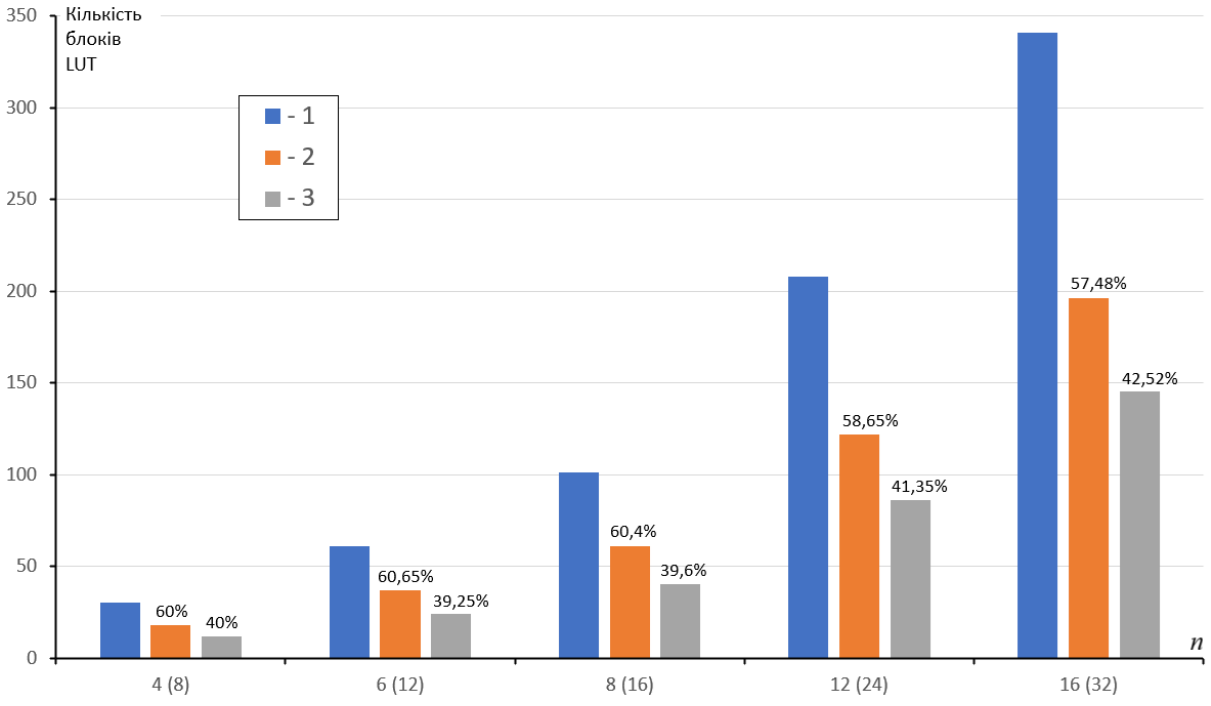


Рисунок 2.9 – Результати експериментального визначення кількості суттєвих блоків LUT та верхньої оцінки кількості несуттєвих блоків LUT
1 – загальна кількість блоків LUT в реалізації FPGA-проєкту;
2 – з них кількість експериментально визначених суттєвих блоків LUT;
3 – верхня оцінка кількості несуттєвих блоків LUT

На діаграмі за горизонтальною віссю для експериментальних FPGA-проєктів відкладено розрядність операндів та результату, а також розрядність результату до округлення. За вертикальною віссю відкладено кількість блоків LUT. На діаграмі показано порівняння абсолютних, а також відносних значень загальної кількості блоків LUT, кількості експериментально визначених суттєвих блоків LUT та верхня оцінка кількості несуттєвих блоків LUT для кожного FPGA-проєкта.

Визначення нижньої оцінки кількості несуттєвих блоків LUT.

Процес аналізу блоків LUT розробленим програмним додатком LMatrixLUTGetter для отримання нижньої оцінки кількості несуттєвих блоків LUT полягає в наступному. На основі даних про блоки LUT, їх зв'язки та програмні коди додаток LMatrixLUTGetter обчислює розмір молодшої частини матриці в структурі обчислювача та відповідно до положень, наведених в п. 2.3.1.3 дисертації, виконує пошук блоків LUT, які відносяться до молодшої частини матриці. За результатом функціонування додатка LMatrixLUTGetter отримується список несуттєвих блоків LUT, які розміщені в молодшій частині матриці обчислювача для кожного експериментального FPGA-проєкта. На основі аналізу цього списку виконується нижня оцінка кількості несуттєвих блоків LUT.

Результати експериментального дослідження з визначення нижньої оцінки кількості несуттєвих блоків LUT представлені в табл. 2.2. У таблиці наведено експериментальні результати, отримані для помножувачів мантис чисел з рухомою комою з розрядністю операндів 4, 6, 8, 12, 16 розрядів. В експериментальних FPGA-проєктах нижня оцінка частки несуттєвих блоків LUT лежить в діапазоні 16,67% ... 29,91% від загальної кількості блоків LUT проєктів.

Обсяг, доступний для зберігання даних в стеганографічний спосіб, забезпечений несуттєвими блоками LUT, за отриманою нижньою оцінкою змінюється від 5 до 102 бітів та від 80 до 1632 бітів при використанні для вбудовування стего-даних відповідно тільки одного або всіх розрядів програмного коду несуттєвих блоків LUT.

Таблиця 2.2 – Результати експериментального визначення нижньої оцінки кількості несуттєвих блоків LUT

Розрядність операндів та кінцевого результату	4	6	8	12	16
Розрядність результату до округлення	8	12	16	24	32
Загальні кількість блоків LUT	30	61	101	208	341
Нижня оцінка кількості несуттєвих блоків LUT	5	10	22	51	102
Нижня оцінка частки несуттєвих блоків LUT серед усіх блоків	16,67%	16,39%	21,78%	24,52%	29,91%
Мінімальний обсяг, доступний для зберігання даних (біт)	5	10	22	51	102
Максимальний обсяг, доступний для зберігання даних (біт) при кількості входів блоків LUT – 4	80	160	352	816	1632

На комплексній діаграмі (рис. 2.10) з результатами експериментів з дослідження інформаційних стеганографічних ресурсів, забезпечених несуттєвими блоками LUT, показано порівняння абсолютних, а також відносних значень загальної кількості блоків LUT, кількість експериментально визначених суттєвих блоків LUT, верхньої та нижньої оцінки кількості несуттєвих блоків LUT для кожного з експериментальних FPGA-проєктів.

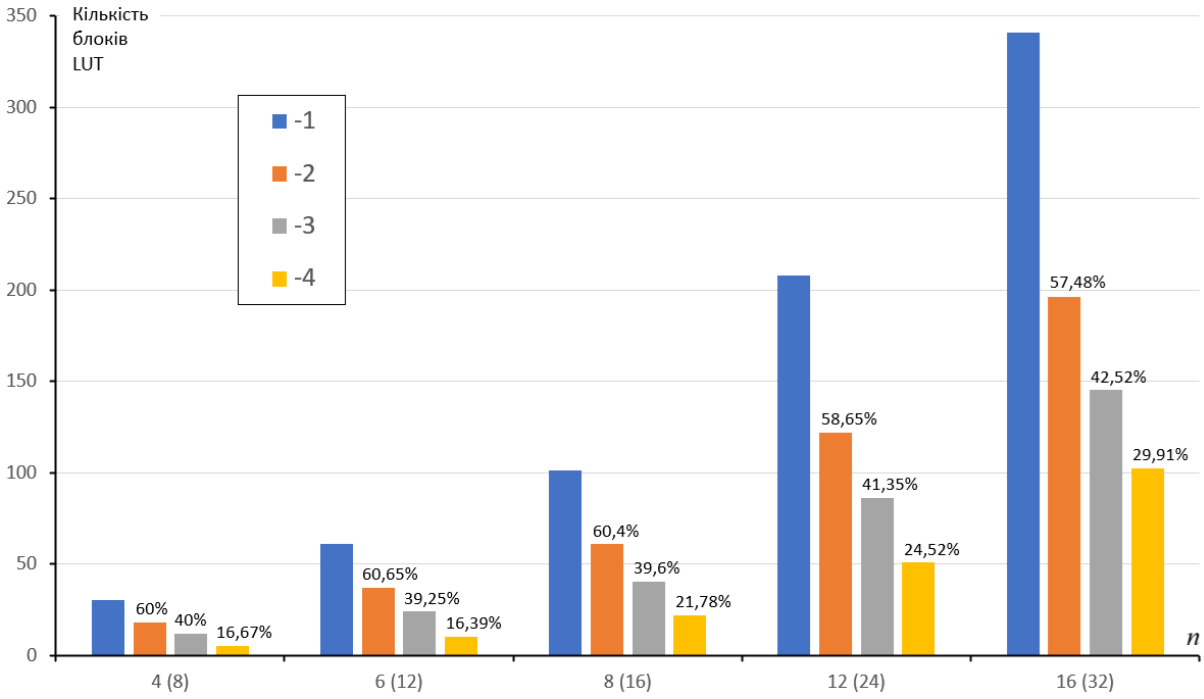


Рисунок 2.10 – Результати експериментального дослідження з урахуванням нижньої оцінки кількості несуттєвих блоків LUT

1 – загальна кількість блоків LUT в реалізації FPGA-проєкту;

2 – з них кількість експериментально визначених суттєвих блоків LUT;

3 – верхня оцінка (не більше ніж) кількості несуттєвих блоків LUT;

4 – нижня оцінка (не менше ніж) кількості несуттєвих блоків LUT

2.3.3 Експериментальне дослідження доступного обсягу замаскованого зберігання даних, забезпеченого несуттєвими розрядами програмних кодів блоків LUT

Вирішується завдання експериментальної оцінки обсягу даних які потенційно можуть бути вбудовані в програмний код FPGA завдяки використанню стего-ресурсу $IRes_2$ (2.11) – сукупності несуттєвих розрядів програмних кодів блоків LUT при виконанні на FPGA наближеної обробки даних, поданих у вигляді чисел з рухомою комою.

В ході експериментального дослідження у попередньому підрозділі дисертації елементи стего-ресурсу $IRes_1$ (2.9) були оцінені за обсягом та

локалізовані шляхом обмеженого перебору, а також шляхом бистрого пошуку з врахуванням арифметичних та структурних особливостей обчислювачів в складі яких виконувався пошук цих елементів (несуттєвих блоків LUT).

На відміну від процедур локалізації стего-ресурсу $IRes_1$ (2.9), локалізація несуттєвих розрядів програмних кодів блоків LUT, які складають стего-ресурс $IRes_2$ (2.11) можлива тільки переборним шляхом. Цей фактор робить стего-ресурс $IRes_2$ обмеженим для практичного застосування через велику обчислювальну складність локалізації елементів цього ресурсу. Стего-ресурс $IRes_2$ може біти виділений в обчислювачах обмеженої розрядності. Через що його застосування є доцільним тільки при вичерпанні обсягу всіх інших наявних стего-ресурсів: а) ресурсів, забезпечених еквівалентними перетвореннями програмних кодів FPGA; б) ресурсів, забезпечених використанням несуттєвих блоків LUT FPGA.

Однак суттєвим є фактор того, що для локалізації елементів стего-ресурсу $IRes_2$ необхідно виконати однократні обчислення. Це дає змогу локалізувати елементи, що відносяться до даного ресурсу один раз для кожного типу обчислювачів та скласти карту простору стего-ресурсу в просторі програмних кодів FPGA. Крім того для локалізація елементів $IRes_2$ можливе розбиття обчислювача на підсхеми, в межах яких обчислювальна складність пошуку елементів ресурсу є помірною.

В даному підрозділі дисертаційної роботи виконується експериментальне дослідження *метою* якого, є демонстрація наявності стего-ресурсу $IRes_2$ та оцінки його обсягу.

Середовище проведення експерименту склали програмні та апаратні засоби, які використовувалися при проведенні експериментального дослідження обсягу стего-ресурсів, забезпечених несуттєвими блоками LUT (підрозділ 2.3.2 дисертації).

До цих засобів додано, розроблений для цілей даного експерименту, програмний додаток IEBitsExtractor, який приймає від додатка LUTListExtractor на вхід інформацію про сукупність блоків LUT, їх програмні коди та зв’язки в FPGA-проєкті та виконує пошук розрядів

програмного коду блоків LUT до яких не відбувається звертання при будь-якому можливому наборі вхідних даних обчислювача.

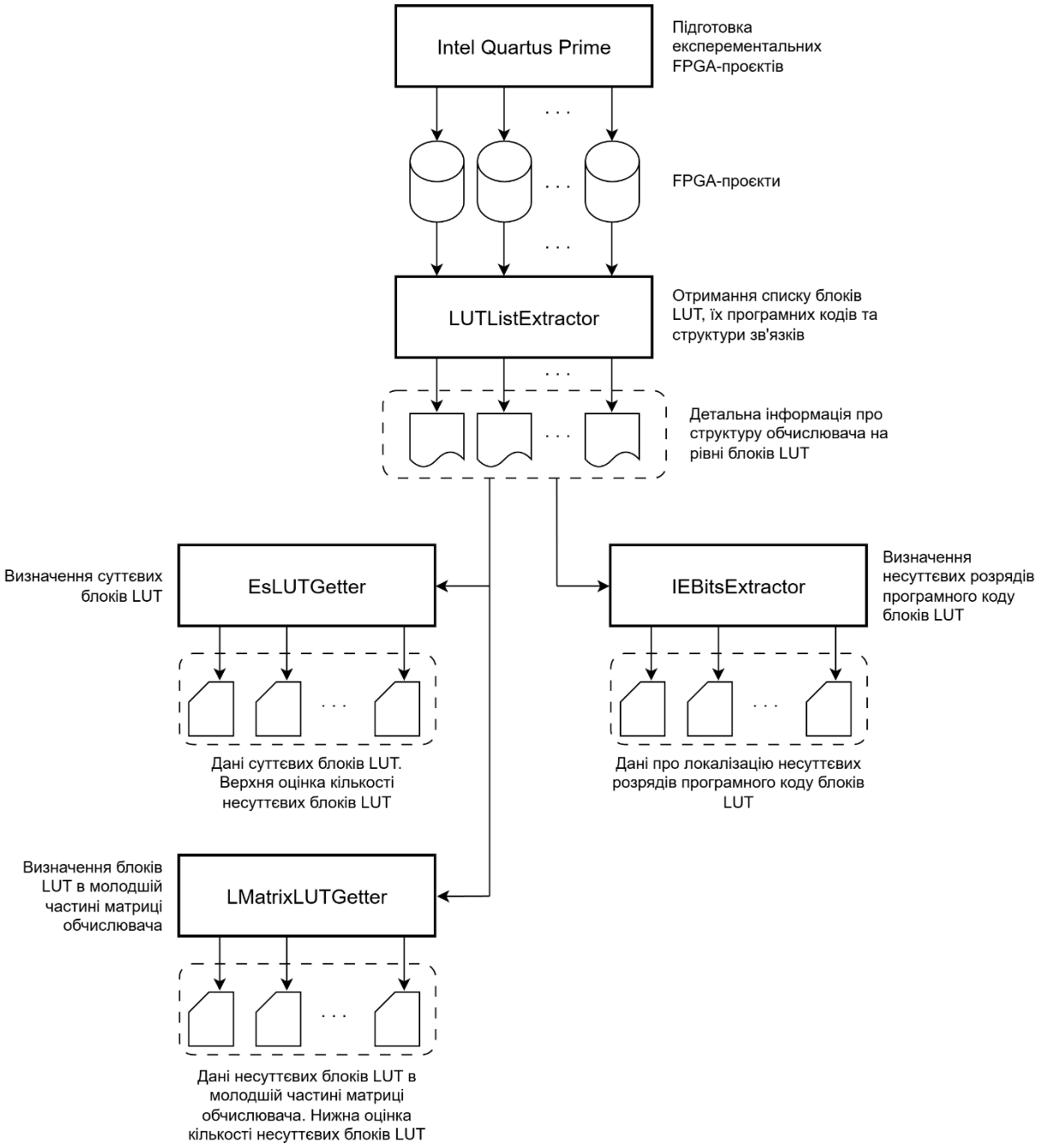


Рисунок 1.11 – Зв’язки між програмними додатками та програмними модулями, які становлять середовище виконання експерименту з дослідження стеганографічного ресурсу, забезпеченого несуттєвими розрядами програмних кодів блоків LUT

На рис. 2.12 показано вікно інтерфейсу додатка IEBitsExtractor наприкінці процедури моделювання для 8-розрядного помножувача мантис чисел з рухомою комою.

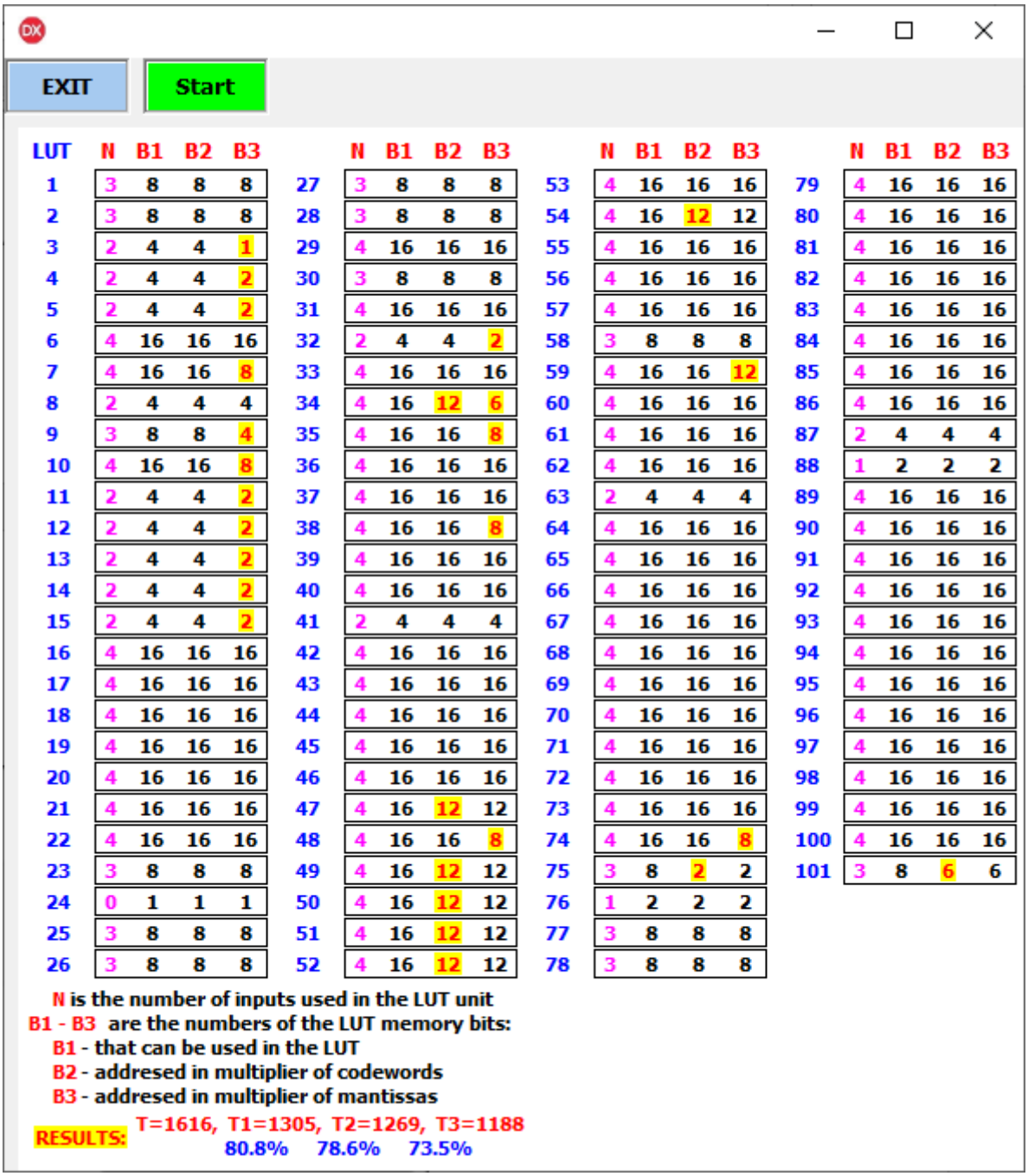


Рисунок 2.12 – Вікно розробленого додатка IEBitsExtractor з результатами обробки 8-розрядного обчислювача

Додаток IEBitsExtractor виконує перебір операндів помножувача. При цьому додаток аналізує, до яких розрядів кожного з блоків LUT FPGA-системи здійснюється звернення. У вікні додатка для кожного з блоків LUT в кінці процесу перебору значень операндів відображається:

– N – кількість використовуваних входів блоку LUT;

– B1 – кількість потенційно адресованих розрядів програмного коду блоку LUT (кількість розрядів програмного коду, потрібна для програмування блока LUT з кількістю входів N);

– B2 – кількість реально адресованих розрядів програмного коду при інтерпретації операндів, як кодових слів відповідної розрядності (операнди приймають всі можливі значення);

– B3 – кількість реально адресованих розрядів програмного коду при інтерпретації операндів, як нормалізованих мантис (операнди приймають половину можливих значень);

– T – сумарна кількість наявних розрядів програмних кодів всіх блоків LUT обчислювача;

– T1 – сумарна кількість потенційно адресованих розрядів програмних кодів всіх блоків LUT обчислювача;

– T2 – сумарна кількість реально адресованих розрядів програмних кодів всіх блоків LUT обчислювача при інтерпретації операндів, як кодових слів відповідної розрядності;

– T3 – сумарна кількість реально адресованих розрядів програмних кодів всіх блоків LUT обчислювача при інтерпретації операндів, як нормалізованих мантис;

– частки кількості розрядів T1, T2, T3 в сумарній кількості наявних розрядів програмних кодів всіх блоків LUT обчислювача (T1/T, T2/T, T3/T).

В табл. 2.3 наведено узагальнені результати підрахунку кількості несуттєвих розрядів в програмному коді блоків LUT для помножувачів мантис чисел з рухомою комою при розрядності операндів 4, 6, 8, 10 та 12 розрядів.

Таблиця 2.3 – Результати експериментального визначення кількості несуттєвих розрядів програмних кодів блоків LUT

Розрядність операндів та кінцевого результату		4	6	8	10	12
Розрядність результату до округлення		8	12	16	20	24
Загальна кількість блоків LUT		30	61	101	149	208
Час моделювання, с		$5.2 \cdot 10^{-5}$	$7.8 \cdot 10^{-4}$	0.01205	0.16714	2.59889
Загальний обсяг пам’яті (T) для програмних кодів блоків LUT, біт		480	976	1616	2384	3328
Кількість потенційно адресованих розрядів (T1)		396 (82.5%)	747 (76.6%)	1305 (80.8%)	1991 (83.5%)	2811 (84.5%)
Суттєві розряди	Помноження двійкових кодів (T2)	391 (81.5%)	719 (73.7%)	1269 (78.6%)	1983 (83.2%)	2752 (82.7%)
	Помноження нормалізова-них мантис (T3)	287 (59.8%)	658 (67.4%)	1188 (73.5%)	1882 (79.0%)	2638 (79.3%)
Несуттєві розряди	Помноження двійкових кодів	89 (18.5%)	257 (26.3%)	347 (21.4%)	401 (16.8%)	576 (17.3%)
	Помноження нормалізо-ваних мантис	193 (40.2%)	318 (32.6%)	428 (26.5%)	502 (21.0%)	690 (20.7%)

В таблиці для кожної розрядності експериментально досліджуваних помножувачів наведено загальну кількість розрядів програмного коду блоків LUT, яка для кожного з 4-входових блоків LUT складає 16 розрядів.

Деякі блоки LUT в експериментальних FPGA-проєктах мають меншу кількість входів, що обумовлено специфікою логічних функцій, які ці блоки реалізують. Через це для програмування таких блоків LUT використовуються менша кількість розрядів програмного коду: 8 розрядів для блоків LUT з 3-ма входами та 4 розряди для блоків LUT з 2-ма входами. Це приводить до того, що кількість потенційно адресованих розрядів програмного коду сукупності блоків LUT є меншою за загальну кількість розрядів пам’яті програмного коду цих блоків.

Кількість несуттєвих розрядів програмних кодів блоків LUT для експериментальних прєктів становить від 89 до 576 при інтерпретації операндів, як кодових слів розрядності від 4 до 12 розрядів, що складає від 16.8 % до 26.3 % обсягу пам’яті, задіяної для зберігання програмних кодів.

Кількість несуттєвих розрядів програмних кодів блоків LUT для експериментальних прєктів становить від 193 до 690 при інтерпретації операндів, як нормалізованих мантис розрядності від 4 до 12 розрядів, що складає від 20.7 % до 40. 2% обсягу пам’яті, задіяної для зберігання програмних кодів.

На діаграмі (рис. 2.13) наведено результати експериментів з дослідження інформаційних стеганографічних ресурсів, забезпечених несуттєвими розрядами блоків LUT. За горизонтальною віссю діаграми відкладено розрядності операндів помножувачів, які використовувалися в складі середовища проведення експериментів, а також показано сукупну розрядність результату обчислень до відкидання несуттєвих розрядів та округлення.

Наведено порівняння загальної кількості розрядів програмних кодів всіх блоків LUT FPGA-проєктів, задіяних в експерименті, кількості потенційно адресованих розрядів цих програмних кодів на повному циклі моделювання, кількості суттєвих розрядів при помноженні операндів,

інтерпретованих як двійкові коди та нормалізовані мантиси чисел з рухомою комою відповідно для кожного з експериментальних FPGA-проєктів.

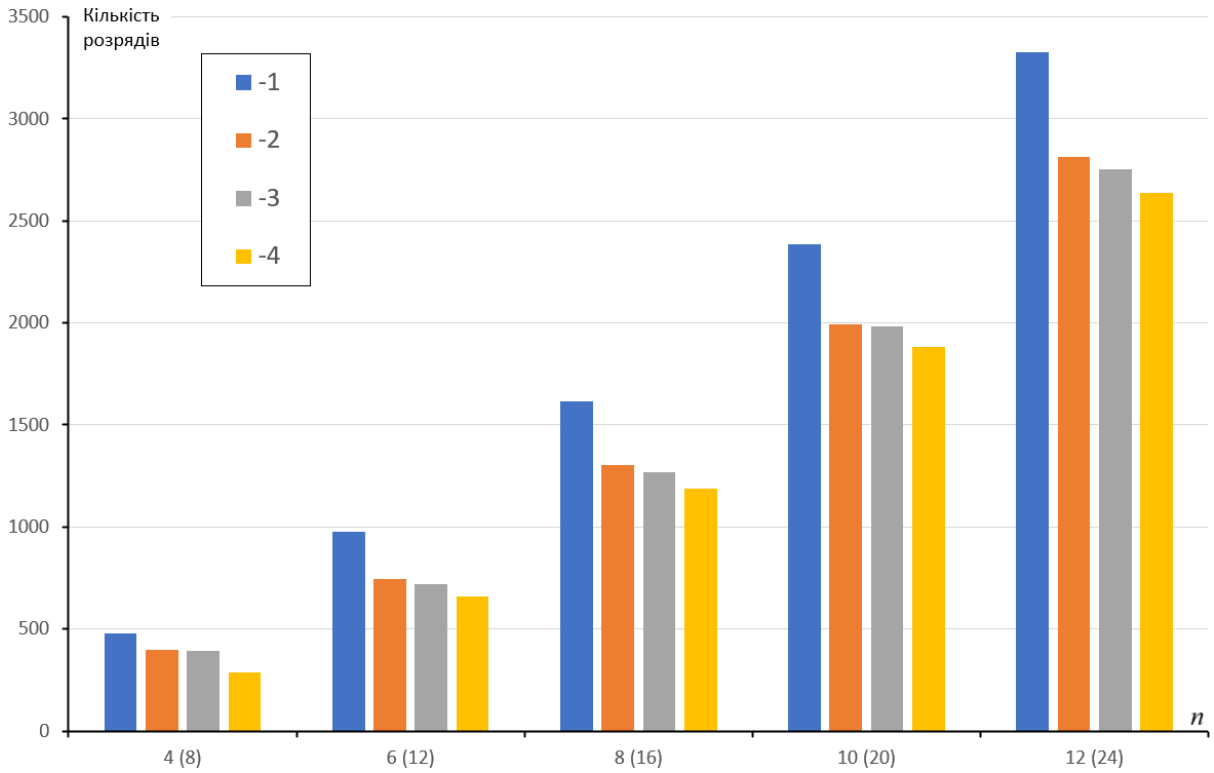


Рисунок 2.13 – Результати експериментального визначення кількості суттєвих розрядів програмного коду блоків LUT

1 – загальна кількість розрядів блоків LUT;

2 – кількість потенційно адресованих розрядів;

3 – кількість суттєвих розрядів при помноженні двійкових кодів;

4 – кількість суттєвих розрядів при помноженні нормалізованих мантис

На діаграмі (рис. 2.14) попередньо показані результати, деталізовано з представленням кількості несуттєвих розрядів програмного коду блоків LUT. Показана кількості несуттєвих розрядів програмного коду блоків LUT порівняно з кількістю потенційно адресованих розрядів при помноженні двійкових кодів та нормалізованих мантис.

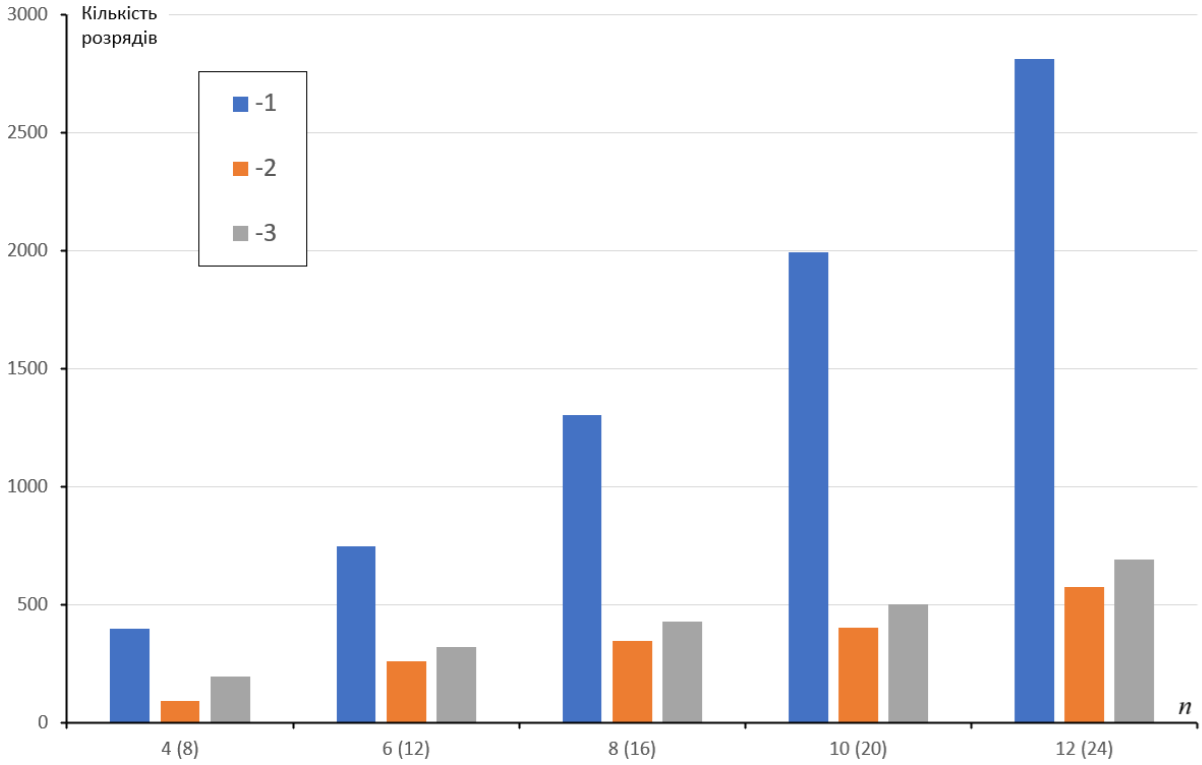


Рисунок 2.14 – Результати експериментального визначення кількості несуттєвих розрядів програмного коду блоків LUT

1 – кількість потенційно адресованих розрядів;

2 – кількість несуттєвих розрядів при помноженні двійкових кодів;

3 – кількість несуттєвих розрядів при помноженні нормалізованих мантис

2.3.4 Обговорення результатів експериментального дослідження та ефективності запропонованого методу

В даному розділі дисертаційної роботи сформульовано твердження, про те, що при реалізації наближеної обробки даних на FPGA, ця наближеність переноситься на точно поданий програмний код FPGA. Наслідком цього є можливість стеганографічного вбудовування в нееквівалентний спосіб (подібне до того, що використовується стосовно інформаційних контейнерів з наближено поданими елементарними одиницями) додаткових даних в точно поданий інформаційний контейнер, яким є програмний код FPGA. Це твердження набуло формалізації у вигляді

моделей, які виділяють два види надлишкових ресурсів у складі програмного коду FPGA, що можуть бути використані зі стеганографічною метою: а) несуттєві блоки LUT в структури обчислювачів, які виконують наближену обробку даних; б) несуттєві розряди програмного коду блоків LUT при виконанні наближеної обробки даних.

Проведені експериментальні дослідження підтвердили наявність зазначених надлишкових ресурсів, дозволили оцінити їх обсяг та виконати локалізацію в структурі експериментальних FPGA-проєктів.

За результатами експериментів отримано верхню оцінку кількості несуттєвих блоків LUT для експериментальних проєктів в діапазоні 39.35 % … 42.52 % та нижню оцінку кількості зазначених блоків в діапазоні 16.67 % … 29.91 % від загальної кількості блоків LUT в проєкті. Також експериментально отримано оцінку частки несуттєвих розрядів програмних кодів блоків LUT для обчислювачів, що виконують наближену обробку даних, яка склала для експериментальних проєктів 16.8 % … 26.3 % від сукупної кількості розрядів програмних кодів блоків LUT у відповідних FPGA-проєктах.

Блок LUT який має m входів та виконує обчислення однієї логічної функції від m змінних, програмується на реалізацію конкретної логічної функції 2^m бітним двійковим програмним кодом. Таким чином обсяг пам’ять програмного коду одного m-входового блока LUT складає 2^m розрядів. Для мікросхем FPGA Intel Cyclone IV EP4CE15F23A7, в середовищі яких проводилося експериментальне дослідження, кількість входів блоків LUT складає $m = 4$, а обсяг пам’яті, задіяної під програний код кожного з блоків LUT становить 16 двійкових розрядів. Подібні за кількістю входів та обсягом пам’яті програмного коду блоки LUT мають також FPGA класичних сімейств Intel/Altera Cyclone II, III та III LS.

Для FPGA-проєктів, які використовувалися в експериментах, кількість несуттєвих блоків LUT за верхньою оцінкою склала від 12 до 145. Таким чином, для цих випадків обсяг стего-контейнера, що забезпечується

ресурсом несуттєвих блоків LUT, становив від 12 до 145 біт при використанні з метою стеганографічного збереження даних тільки одного розряду програмного коду блоків LUT та від 192 до 2320 біт при використанні всіх розрядів програмного коду зазначених блоків. За нижньою оцінкою кількість несуттєвих блоків LUT склала від 5 до 102, що забезпечує обсяг для збереження додаткових даних від 5 до 102 біт при використанні одного розряду програмного коду блоків LUT та від 80 до 1632 біт при використанні всіх розрядів програмного коду зазначених блоків.

Отримані обсяги є достатніми для зберігання контрольних хеш-сум, одержуваних за допомогою хеш-функцій, що використовуються в практичних застосуваннях оперативного моніторингу характеристик безпеки програмного коду. Зі збільшенням розміру операндів помножувачів об’єм стего-контейнера стає достатнім також для зберігання додаткової службової інформацією контролю.

Для експериментальних FPGA-проєктів кількість несуттєвих розрядів програмного коду блоків LUT при множенні двійкових чисел склала від 89 до 576. При множенні нормалізованих мантис кількість несуттєвих розрядів програмного коду склала від 193 до 690. Отримані обсяги пам’яті для зберігання додаткових даних можуть бути використані разом з обсягами, забезпечених несуттєвими блоками LUT або самостійно. Спільне використання обсягів, забезпечених двома видами стего ресурсів дає можливість приховано зберігати в програмному коді FPGA в стеганографічний спосіб контрольні дані декількох видів оперативного моніторингу програмного коду, а також додаткової службової інформацією мніторингу.

Наявні підходи до стеганографічнго зберігання контрольних даних в програному коді FPGA, які базуються на еквівалентних перетвореннях, хоча і мають малу обчислювальну складність, але забезпечують можливість збереження тільки однієї або двох хеш-сум об’ємом 128-256 розрядів в залежності від розміру FPGA-проєкту. Запропоновані та експериментально

дослідженні в даному розділі дисертації стего-ресурси додають до цього обсягу додаткові ресурси для зберігання контрольних даних (діапазони кількості яких зазначені вище для кожного з запропонованих стего-ресурсів), що дає змогу збереження контрольних даних більшої кількості видів оперативного моніторингу та додає варіативності у виборі розміру виходу хеш-функцій, використовуваних для отримання контрольних даних. Так стего-ресурс несуттєвих блоків LUT в експериментальному проєкті, який містить помножувач для 16 розрядних операндів, за нижньою оцінкою при використанні всіх розрядів програмного коду цих блоків складає 1632 розряди, що становить обсяг для прихованого збереження 3 хеш-сум розміром 512 розрядів та 6 хеш-сум розміром 256 розрядів.

2.4 Висновки до другого розділу

В другому розділі дисертації вирішено задачу розробки моделі нееквівалентного стеганографічного зберігання контрольних даних в середовищі програмного коду FPGA-компонентів. В розділі запропоновано модель, яка формалізує умови та правила застосування нееквівалентних перетворень програмного коду FPGA для виділення надлишкових інформаційних ресурсів програмного коду та використання цих ресурсів при стеганографічному зберігання контрольних даних. На основі розробленої моделі у розділі виділено два види надлишкових інформаційних ресурсів програмного коду FPGA та на їх основі запропоновано правила формування стеганографічного простору для прихованого зберігання додаткових даних.

Вирішено задачу оцінки обсягу стеганографичних ресурсів для зберігання даних, забезпечених запропонованою моделлю. Експериментальне дослідження, проведене в середовищі розробленого програмного забезпечення, яке реалізує запропоновану модель, підтвердило

наявність надлишкових стеганографічних ресурсів програмного коду FPGA, а також дозволило їх локалізувати та оцінити їх частку в загальному обсязі програмного коду блоків LUT експериментальних FPGA-проєктів. За результатами експериментів отримано верхню оцінку кількості несуттєвих блоків LUT для експериментальних проєктів в діапазоні 39.35 % … 42.52 % та нижню оцінку кількості зазначених блоків в діапазоні 16.67 % … 29.91 % від загальної кількості блоків LUT в проєкті. Також експериментально отримано оцінку частки несуттєвих розрядів програмних кодів блоків LUT для обчислювачів, що виконують наближену обробку даних, яка склала для експериментальних проєктів 16.8 % … 26.3 % від сукупної кількості розрядів програмних кодів блоків LUT у відповідних FPGA-проєктах.

Вирішено задачу розробки методу нееквівалентного стеганографічного зберігання контрольних даних в середовищі програмного коду FPGA-компонентів, основаного на запропонованій моделі. Метод використовує надлишкові інформаційні ресурси програмного коду FPGA, які виникають при наближеній обробці даних та можуть бути використані зі стеганографічною метою.

Вирішено задачу експериментального дослідження ефективності розробленого методу та доведення достатності для задач прихованого моніторингу програмного коду FPGA того обсягу даних, які можуть бути вбудовані в програмний код запропонованим методом. Додатковий обсяг для стеганографічного зберігання даних, забезпечений запропонованим методом склав для виділеного надлишкового стеганографічного ресурсу програмних кодів несуттєвих блоків LUT експериментальних FPGA-проєктів за верхньою оцінкою: від 12 до 145 біт при використанні з метою стеганографічного збереження даних тільки одного розряду програмного коду несуттєвих блоків LUT та від 192 до 2320 біт при використанні всіх розрядів програмного коду зазначених блоків; та за нижньою оцінкою від 5

до 102 біт при використанні одного розряду програмного коду блоків LUT та від 80 до 1632 біт при використанні всіх розрядів програмного коду зазначених блоків. Для експериментальних FPGA-проєктів кількість несуттєвих розрядів програмного коду блоків LUT склала від 89 до 690.

Запропоновані та експериментально досліджені в даному розділі дисертації стего-ресурси додають до обсягів прихованого зберігання даних в програмному коді FPGA, забезпечених відомими методами, додаткові ресурси для зберігання контрольних даних, що дає змогу збереження контрольних даних більшої кількості видів оперативного моніторингу та додає варіативності у виборі розміру виходу хеш-функцій, використовуваних для отримання контрольних даних.

За результатом досліджень, описаних в даному розділі дисертаційної роботи, отримано наступні пункти наукової новизни:

– *вперше* розроблено модель замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, *яка основана* на використанні нееквівалентних перетворень програмного коду FPGA *та враховує* особливості виконання наближених арифметичних операцій в середовищі FPGA, *що дозволяє* визначити множину надлишкових інформаційних ресурсів, які виникають в процесі виконання таких обчислень, та можуть бути використані для прихованого зберігання контрольних даних.

– *вперше* розроблено метод нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, *що характеризується* використанням надлишковості програмного коду FPGA, яка виникає при виконанні наближених арифметичних операцій, *що дозволяє* збільшити обсяг, доступний для прихованого зберігання контрольних даних в задачах моніторингу характеристик безпеки програмного коду FPGA.

Основні результати розділу опубліковані у працях [125 – 131].

3 МЕТОДИ ГІБРИДНОГО ЗАМАСКОВАНОГО ЗБЕРІГАННЯ КОНТРОЛЬНИХ ДАНИХ СЕРЕДОВИЩІ ПРОГРАМНОГО КОДУ FPGA-КОМПОНЕНТІВ

Даний розділ дисертаційної роботи присвячено вирішенню задач розробки методів гібридного стеганографічного зберігання даних в програмному коді FPGA, які поєднують еквівалентні та нееквівалентні перетворення програмного коду в процесі вбудовування, а також дають можливість застосування відновної обфускації в процесі вбудовування додаткових даних в програмний код.

В розділі запропоновано метод який дає можливість виконати стеганографічне вбудовування додаткових даних в програмний код FPGA, використовуючи при цьому стеганографічні ресурси, забезпечені як еквівалентними, так і нееквівалентними перетвореннями програмного коду. Метод дозволяє виконати пріоритетний розподіл, даних що приховано зберігаються в програмному коді FPGA за видами стеганографічних ресурсів. Також метод дозволяє виконати відновну обфускацію програмного коду з визначеннями локалізації дії такої обфускації.

В розділі також вирішується задача оцінки ефективного стеганографічного обсягу зберігання даних, забезпеченого розробленим методом та впливу вбудовування даних на можливість виявлення факту наявності прихованих даних традиційними методами стегоаналізу.

3.1 Метод гібридного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA

3.1.1 Передумови розробки методу

Відомими є підходи до стеганографічного зберігання контрольних даних в середовищі програмного коду FPGA, базовані на використанні еквівалентних перетворень програмного коду [111-113]. Переваги цих

підходів полягають в малій обчислювальній складності та можливості застосування практично до всього наявного в FPGA-проєкті програмного коду блоків LUT. Такі підходи є практичними, але забезпечують малий потенційно доступний для зберігання обсяг додаткових даних. Це часто призводить до дефіциту обсягу зберігання контрольних даних у задачах оперативного моніторингу програмного коду FPGA, а також зменшує кількість різновидів моніторингу, які можуть бути застосовані до програмного коду.

Моделі та метод, запропоновані в другому розділі даної дисертаційної роботи базуються на використанні нееквівалентних перетворень елементів програмного коду блоків LUT FPGA-проєктів, в частині реалізації наближеної обробки даних, поданих у вигляді чисел з рухомою комою. Підходи, утворені зазначеними моделями та методом, мають як переваги так і певні обмеження, порівняно з відомими традиційними підходами, базованими на еквівалентних перетвореннях.

Переваги запропонованого підходу, базованого на нееквівалентних перетвореннях програмного коду FPGA, полягають в забезпеченні більшого обсягу, доступного для стеганографічного збереження даних, ніж обсяг, доступний завдяки застосуванню еквівалентних перетворень. До недоліків слід віднести більшу обчислювальну складність методів та локалізацію стеганографічного простору зберігання даних тільки в модулях, які виконують наближену обробку даних.

Виходячи з наявності переваг та певних обмежень в обох підходах до стеганографічного зберігання даних в середовищі програмного коду FPGA в даному розділі дисертації пропонується гібридний метод стеганографічного зберігання, який поєднує еквівалентний та нееквівалентний підходи до стеганографічного зберігання даних, дозволяє комбінувати стеганографічні ресурси, забезпечені цими підходами та дає можливість виконувати збереження даних відповідно до встановлених пріоритетів використання цих ресурсів.

Гібридність методу, що пропонується, полягає в двох видах комбінування властивостей відомого еквівалентного підходу та, запропонованого в даній роботі, нееквівалентного підходу до стеганографічного зберігання контрольних даних в програмному коді FPGA:

1) комбінування стеганографічних ресурсів, забезпечених еквівалентними та нееквівалентними перетвореннями програмного коду FPGA, для досягнення потрібного обсягу даних в середовищі програмного коду та послідовному виділенні цих ресурсів для мінімізації обчислювальної складності їх застосування;

2) комбінування стеганографічного вбудовування (з використанням обох підходів) даних в програмний код FPGA та зворотної (відновної) обфускації програмного коду FPGA, базованої на моделі еквівалентних перетворень, та призначеної для ускладнення стегоаналізу і виявлення прихованих в програмному коді даних.

3.1.2 Основні положення методу

3.1.2.1 Загальні положення

Локалізація стеганографічного ресурсу в середовищі програмного коду FPGA з використанням еквівалентного підходу має малу обчислювальну складність, але забезпечую малий обсяг ресурсу. Локалізація аналогічних стего-ресурсів з використанням нееквівалентного підходу має більшу обчислювальну складність, але забезпечує значно більший ніж еквівалентний підхід обсяг цих ресурсів. Виходячи з цього пропонується виконувати вбудовування даних в програмний код FPGA застосовуючи систему пріоритетів для визначення послідовності використання стеганографічних ресурсів.

Для визначення пріоритетів та послідовності використання стего-ресурсів пропонується застосовувати цілочислений вектор пріоритетів:

$$P_{res} = < p_{eq},\ p_{neq1},\ p_{neq2} > \quad (3.1)$$

де p_{eq} – пріоритет використання стего-ресурсу, забезпеченого еквівалентними перетвореннями програмного коду FPGA;

p_{neq1} – пріоритет використання стего-ресурсу, забезпеченого нееквівалентними перетвореннями програмного коду за рахунок використання несуттєвих блоків LUT;

p_{neq2} – пріоритет використання стего-ресурсу, забезпеченого нееквівалентними перетвореннями програмного коду за рахунок використання несуттєвих розрядів суттєвих блоків LUT.

Приклади типових значень вектора P_{res}:

– P_{res} = < 1, 0, 0> – задано тільки один пріоритет, який виділяє для використання стего-ресурс, забезпечений еквівалентним підходом. Для зберігання контрольних даних використовується тільки стего-ресурс, забезпечений еквівалентним підходом. У випадку, якщо обсягу цього ресурсу недостатньо, решта контрольних даних зберігається в нестеганографічний спосіб;

– P_{res} = < 1, 2, 0 > – задано два пріоритети використання стего-ресурсів. Найбільший пріоритет має стего-ресурс забезпечений еквівалентним підходом. Другий пріоритет має стего-ресурс забезпечений нееквівалентним підходом з використанням програмного коду несуттєвих блоків LUT. Для зберігання контрольних даних використовується стего-ресурс, забезпечений еквівалентним підходом. У випадку, якщо обсяг цього ресурсу вичерпано, використовується стего-ресурс, забезпечений несуттєвими блоками LUT. Тільки у випадку недостатності обсягів стеганографічного збереження даних, забезпеченого обома типами ресурсів, решта контрольних даних зберігається в нестеганографічний спосіб;

– P_{res} = < 1, 2, 3 > – задано три пріоритети використання стего-ресурсів. Найбільший пріоритет має стего-ресурс забезпечений еквівалентним підходом. Другий пріоритет має стего-ресурс забезпечений

нееквівалентним підходом з використанням програмного коду несуттєвих блоків LUT. Третій пріоритет має стего-ресурс забезпечений використанням несуттєвих розрядів суттєвих блоків LUT. Для зберігання контрольних даних використовується стего-ресурс, забезпечений еквівалентним підходом. У випадку, якщо обсяг цього ресурсу вичерпано, використовується стего-ресурс, забезпечений несуттєвими блоками LUT. При вичерпані зазначеного стего-ресурсу, використовується стего-ресурс, забезпечений несуттєвими розрядами суттєвих блоків LUT. Тільки у випадку недостатності обсягів стеганографічного збереження даних, забезпеченого всіма трьома типами ресурсів, решта контрольних даних зберігається в нестеганографічний спосіб.

Оскільки пріоритети задаються вектором P_{res} явно, можливі також інші комбінації пріоритетів, які задають іншу послідовність використання стего-ресурсів. Наприклад, більш пріоритетне використання нееквівалентних ресурсів та використання еквівалентних ресурсів тільки у разі вичерпання обсягу нееквівалентних.

Вхідні дані методу:

а) інформаційний об’єкт програмного коду FPGA-базованого компонента на рівні блоків LUT, побудований відповідно до моделі LUT-контейнера запропонованої в роботі [121]. Зазначений інформаційний об’єкт є сукупність: множини блоків LUT FPGA; множини зв’язків цих блоків між собою; множин входів та виходів LUT-контейнера, а також множини зв’язків блоків LUT з входами та виходами контейнера;

б) послідовність двійкових розрядів $D = < d_1,\ d_2, \dots, d_s >$ додаткових даних, що вбудовуються до LUT-контейнера;

в) стего-ключ *Key*, який містить правило отримання впорядкованої множини блоків, що складають стего-шлях вбудовування.

Вихідні дані методу: заповнений LUT-контейнер, в програмний код якого в стеганографічний спосіб вбудовано двійкову послідовність *D*.

Результуючий LUT-контейнер є функціонально та параметрично еквівалентним первісному LUT-контейнеру.

Пропонований гібридний метод базується на наступних положеннях.

3.1.2.2 Положення, які встановлюють принципи комбінування стеганографічних ресурсів, забезпечених еквівалентними та нееквівалентними перетвореннями програмного коду FPGA.

Перше положення методу полягає в тому, що для застосування методу FPGA-система в програмний код, якої в стеганографічний спосіб вбудовуються додаткові дані, повинна містити модулі, які виконують наближену обробку даних. У випадку відсутності таких модулів у складі FPGA-системи, вектор пріоритетів (3.1) дорівнює P_{res} = < 1, 0, 0 >, а метод, як окремий випадок, зводиться до відомого підходу до вбудовування стеганографічний даних в програмний код FPGA з використанням еквівалентних перетворень.

Друге положення методу: стеганографічні ресурси, забезпечені еквівалентними та нееквівалентними перетвореннями програмного коду FPGA не повинні перетинатися. Програмний код тих блоків LUT, які було обрано для вбудовування додаткових даних у еквівалентний спосіб не повинен далі модифікуватися в нееквівалентний спосіб. Програмний код тих блоків LUT, які було обрано для вбудовування додаткових даних у нееквівалентний спосіб не повинен далі модифікуватися в еквівалентний спосіб. Виключення складають методи еквівалентного стеганографічного вбудовування даних, які мають можливість відкату інформаційного контейнера до первісного стану при витяганні даних з нього [132]. При використанні подібних методів можливим є вбудовування даних в еквівалентний спосіб поверх даних, які було вбудовано в нееквівалентний спосіб.

Третє положення методу: вектор P_{res} задає пріоритетність та послідовність використання стего-ресурсів та використовується для регулювання сукупного доступних обсягу стего-ресурсів та складності

локалізації елементів цих ресурсів. Вектор P_{res} є компонентом стего-ключа та визначає в якій послідовності потрібно витягати вбудовані дані з інформаційного контейнера програмного коду FPGA.

Четверте положення методу: при ненульовому значенні компонента p_{neq1} вектора P_{res} (3.1) окремим параметром стего-ключа $neq1_{low}^{up}$ визначається спосіб локалізації несуттєвих блоків LUT:

– шляхом отримання нижньої оцінки кількості несуттєвих блоків LUT та їх локалізації (п. 2.3.1.3 дисертаційної роботи) $- neq1_{low}^{up} = 0$;

– шляхом отримання верхньої оцінки кількості несуттєвих блоків LUT та їх локалізації (п. 2.3.1.2 дисертаційної роботи) $- neq1_{low}^{up} = 1$;

3.1.2.3 Положення, які встановлюють принципи комбінування стеганографічного вбудовування (з використанням обох підходів) даних в програмний код FPGA та зворотної (відновної) обфускації програмного коду FPGA.

П'яте положення методу: пропонований метод в частині еквівалентного стеганографічного вбудовування даних використовує відомі еквівалентні перетворення [111-113]. Метод, так само, як і відомі методи вбудовування [140], вимагає виділення з множини блоків LUT множини пар $P_i = <LUT_{i1}, LUT_{i2}>$ послідовно з'єднаних блоків.

Шосте положення методу: еквівалентне вбудовування розряду даних в блок LUT_{i1} призводить до незворотного еквівалентного перетворення програмного коду блоків пари P_i. Незворотність обумовлена тим, що процедура виконання еквівалентного перетворення програмного коду блоку залежить від співвідношення розряду, який вбудовується та цільового розряду програмного коду.

Сьоме положення методу: обфускація програмного коду пари P_i блоків LUT здійснюється шляхом зворотного (відновного) еквівалентного перетворення:

1) перестановка розрядів блоку LUT другого рівня відповідно до принципів перестановки, які мають місце при еквівалентному вбудовування

даних в пару послідовно з'єднаних блоків LUT [140]. Повторне виконання цієї ж дії призводить до відкату стану програмного коду блоків LUT. В силу цього зазначене перетворення є зворотним.

2) інверсія програмного коду одного або декількох блоків LUT першого рівня, виходи яких підключені до входу блоку LUT другого рівня. Повторне виконання цієї ж дії також призводить до відкату до первісного стану програмного коду блоків LUT. В силу цього зазначене перетворення є зворотним.

Восьме положення методу: стего-ключ, що містить параметри, необхідні для стеганографічного вбудовування даних в стего-контейнер та діставання даних з стего-контейнера, включає параметри двох типів: параметри, що визначають елементи процесу вбудовування, і параметри, що визначають елементи процесу обфускації.

Дев'яте положення методу: параметрами еквівалентного перетворення програмного коду блоків LUT в частині обфускації є:

1) множина LUT_{i2} других блоків LUT пар P_i, для яких була проведена обфускація. Ця множина задається за допомогою формального правила, яке генерує номери блоків LUT_{i2};

2) ваги входів блоків LUT_{i2} з цієї множини, які визначають вид перестановки бітів блоків LUT_{i2}. Ці ваги можуть бути визначені фіксованим, псевдовипадковим, ітераційним або шаблонним способом.

Десяте положення методу: обфускація програмного коду блоків LUT виконується після вбудовування розрядів додаткових даних в цей програмний код. Для добування додаткових даних спочатку виконується деобфускація, а далі методами еквівалентних перетворень та з застосуваннями методу, запропонованого в даній роботі, виконується добування вбудованих даних з інформаційного контейнера програмного коду FPGA.

3.1.3 Послідовність кроків методу

Метод, що пропонується є послідовністю дій, виконання яких призводить до стеганографічного вбудовування додаткових даних у програмний код FPGA, а також до обфускації цього програмного коду. Кількість етапів методу залежить від кількості ненульових компонентів у векторі пріоритетів P_{res} (3.1), який є компонентом стего-ключа, а також від того, чи застосовується етап обфускації програмного коду блоків LUT FPGA.

Визначається шість потенційно можливих етапів виконання методу:

– підготовчий етап;

– три опціональні етапи вбудовування додаткових даних у програмний код FPGA;

– опціональний етап обфускації програмного коду блоків LUT FPGA.

Етап підготовки є обов'язковим і виконується першим у послідовності етапів методу. Етап обфускації є опціональним, у разі його застосування, виконується після етапів вбудовування даних. Кількість використовуваних етапів вбудовування та їх послідовність визначається складом компонентів вектора пріоритетів P_{res} (3.1).

Етап 1*: підготовчий етап*

Крок 1.1. Формуються компоненти вектора P_{res} (3.1), які визначають типи та підтипи стеганографічних ресурсів, які будуть використані для прихованого вбудовування додаткових даних, а також послідовність використання цих ресурсів. Вектор включається як частина до стего-ключа, необхідного для прихованого вбудовування додаткових даних в програмний код FPGA та діставання цих даних з програмного коду.

Крок 1.2. Визначається, чи буде застосовуватися зворотна (відновна) обфускація програмного коду FPGA після стеганографічного вбудовування додаткових даних. Відповідний компонент стего-ключа встановлюється в значення 1 у разі застосування обфускації та в значення 0 в іншому випадку.

Наступні три етапи методу виконуються в послідовності, яка задається компонентами вектора P_{res} (3.1).

***Етап 2**: вбудовування додаткових даних в програмний код FPGA з використанням еквівалентних перетворень програмного коду.*

Етап виконується, якщо компонент p_{eq} вектора P_{res} (3.1) має ненульове значення. Значення цього компонента задає послідовність виконання етапу 2 серед етапів 2 – 4.

Крок 2.1. Формуються компоненти стего-ключа, призначені для локалізації цільових розрядів вбудовування в середовищі стего-ресурсу програмного коду FPGA, забезпеченого еквівалентними перетвореннями програмного коду. Компонент формується з урахуванням вимоги не перетинання двох видів стего-ресурсів: еквівалентного і нееквівалентного.

Крок 2.2. Виконується вбудовування частини (яка має розмір, що відповідає обсягу стего-ресурсу, локалізованого на попередньому кроці) двійкової послідовності $D = < d_1,\ d_2, \dots, d_s >$ додаткових даних, призначених для прихованого вбудовування в програмний код FPGA. При цьому використовується один з відомих методів стеганографічного вбудовування даних, базованих на еквівалентних перетвореннях програмного коду FPGA [111-113]. Якщо в результатів вбудовування додаткові дані є вичерпаними (повністю вбудованими в програмний код), то наступні методи вбудовування з етапів 2 – 5 не застосовуються, а компоненти вектора P_{res}, що відповідають цим етапам в стего-ключі, обнуляються.

***Етап 3**: вбудовування додаткових даних в програмний код FPGA з використанням стего-ресурсу, забезпеченого нееквівалентними перетвореннями програмного коду за рахунок використання несуттєвих блоків LUT.*

Етап виконується, якщо компонент p_{neq1} вектора P_{res} (3.1) має ненульове значення. Значення цього компонента задає послідовність виконання етапу 3 серед етапів 2 – 4.

Виконується *кроки 1 – 10* методу нееквівалентного стеганографічного зберігання контрольних даних в середовищі програмного коду FPGA-компонентів, запропонованого в підрозділі 2.2 дисертаційної роботи. При цьому на кроці 5 методу, запропонованого в підрозділі 2.2, параметр ER встановлюється в значення «10», що визначає використання для вбудовування тільки розрядів програмних кодів несуттєвих блоків LUT – стего-ресурс IRes_1 (2.9).

На вхід методу подається невбудована на попередньому кроці частина двійкової послідовності $D = < d_1,\ d_2, \dots, d_s >$ додаткових даних, яка має розмір, що відповідає обсягу стего-ресурсу, локалізованого на *кроці 7* методу, запропонованого в підрозділі 2.2.

Компоненти стего ключа, отримані в результаті виконання кроків *1 – 10* методу, запропонованого в підрозділі 2.2, включаються до стего-ключа головного методу.

Якщо в результатів вбудовування додаткові дані є вичерпаними (повністю вбудованими в програмний код), то наступні методи вбудовування з етапів 2 – 4 не застосовуються, а компоненти вектора P_{res}, що відповідають цим етапам в стего-ключі, обнуляються.

Етап 4*: вбудовування додаткових даних в програмний код FPGA з використанням стего-ресурсу, забезпеченого нееквівалентними перетвореннями програмного коду за рахунок використання несуттєвих розрядів програмного коду суттєвих блоків LUT (п. 2.1.5 дисертаційної роботи).*

Етап виконується, якщо компонент p_{neq2} вектора P_{res} (3.1) має ненульове значення. Значення цього компонента задає послідовність виконання етапу 4 серед етапів 2 – 4.

Виконується *кроки 1 – 10* методу нееквівалентного стеганографічного зберігання контрольних даних в середовищі програмного коду FPGA-компонентів, запропонованого в підрозділі 2.2 дисертаційної роботи. При цьому на кроці 5 методу, запропонованого в підрозділі 2.2, параметр ER

встановлюється в значення «01», що визначає використання для вбудовування тільки розрядів програмних кодів несуттєвих блоків LUT – стего-ресурс $IRes_2$ (2.11).

На вхід методу подається невбудована на попередньому кроці частина двійкової послідовності $D = < d_1,\ d_2, \dots, d_s >$ додаткових даних, яка має розмір, що відповідає обсягу стего-ресурсу, локалізованого на *кроці 7* методу, запропонованого в підрозділі 2.2.

Компоненти стего ключа, отримані в результаті виконання кроків *1 – 10* методу, запропонованого в підрозділі 2.2, включаються до стего-ключа головного методу.

Якщо в результатів вбудовування додаткові дані є вичерпаними (повністю вбудованими в програмний код), то наступні методи вбудовування з етапів 2 – 4 не застосовуються, а компоненти вектора P_{res}, що відповідають цим етапам в стего-ключі, обнуляються.

Етап 5*: виконання зворотної (відновної) обфускації програмного коду блоків LUT FPGA.*

Крок 5.1. Формується набір K_{obf} компонентів стего-ключа, призначених для виконання зворотної (відновної) обфускації програмного коду блоків LUT FPGA. Зазначений набір є двокомпонентним кортежем виду:

$$K_{obf} =< LUTRule_{obf},\ InputsRule_{obf} > \qquad (3.2)$$

де $LUTRule_{obf}$ – формальне правило, що дозволяє сформувати список

$$LUTList_{obf} =< l_{1_{obf}},\ l_{2_{obf}},\ \dots,\ l_{n_{obf}} > \qquad (3.3)$$

блоків LUT, до програмного коду яких застосовується обфускація. Далі вважається, що компоненти списку $LUTList_{obf}$ є другими блоками LUT_{i2} у виділених парах P_i послідовно з'єднаних блоків LUT відповідно до моделі

еквівалентних перетворень [111] програмного коду FPGA для стеганографічного вбудовування даних (восьме положення методу);

$InputsRule_{obf}$ – формальне правило, що дозволяє сформувати список

$$WList_{obf} =< w_{l1_{obf}},\ w_{l2_{obf}},\ \dots,\ w_{ln_{obf}} > \qquad (3.4)$$

компонентами якого є кортежі $w_{li_{obf}} \in WList_{obf}$ виду:

$$w_{li_{obf}} =< wLUT_{i_1},\ wLUT_{i_2},\ \dots,\ wLUT_{i_k} > \qquad (3.5)$$

які задають множину ваг входів блоків $l_i \in LUTList_{obf}$, що підключені до виходів інвертованих блоків LUT_{i1} пар P_i послідовно з'єднаних блоків LUT (восьме положення методу).

Правило $LUTRule_{obf}$ задає порядок перерахування блоків LUT для отримання множини $LUTList_{obf}$. Це правило може бути задано фіксованим, псевдовипадковим ітераційним або шаблонним способом. Правило $InputsRule_{obf}$ задає ваги входів блоків LUT, які беруть участь у компенсації інверсії. Це правило також може бути описане аналогічним чином в фіксований, псевдовипадковий, ітераційний або шаблонний спосіб.

Крок 5.2. Для кожного блоку LUT $l_i \in LUTList_{obf}$ за відповідним компонентом $w_{li_{obf}} \in WList_{obf}$ формується список

$$InvList_{i_{obf}} =< InvLUT_{1i},\ \dots,\ InvLUT_{si} > \qquad (3.6)$$

блоків LUT, які є першими компонентами для блоку $l_i \in LUTList_{obf}$ в парі P_i послідовно з'єднаних блоків LUT (восьме положення методу). Програмний код блоків зі списку (3.6) підлягає інвертуванню в ході обфускації відповідно до моделі еквівалентних перетворень [111] програмного коду FPGA.

Крок 5.3. Виконується обфускація програмного коду блоків LUT. Для цього виконуються наступні дії для кожного з блоків $l_i \in LUTList_{obf}$:

а) програмний код кожного з блоків $InvLUT_{qi} \in InvList_{i_{obf}}$ порозрядно інвертується;

б) розряди програмного коду кожного блоку $l_i \in LUTList_{obf}$ переставляються відповідно до правил перестановки, визначених для сукупності ваг $WList_{obf}$ відповідно до моделі еквівалентних перетворень [111] програмного коду FPGA для стеганографічного вбудовування даних (восьме положення методу).

Стего-ключ, який містить інформацію, необхідну для вбудовування додаткових даних в програмний код блоків LUT FPGA та витягання вбудованих даних формується за результатами виконання кроків методу та являє собою чотирьох-компонентний кортеж наступного вигляду:

$$SKey =< P_{res},\ K_{eq},\ K_{neq},\ K_{obf} > \qquad (3.7)$$

де P_{res} – цілочислений вектор пріоритетів використання стего-ресурсів (3.1), забезпечених еквівалентними та нееквівалентними перетвореннями програмного коду блоків LUT FPGA;

K_{eq} та K_{neq} – компоненти стего-ключа, які визначають локалізацію та використання стего-ресурсів, забезпечених, відповідно еквівалентними та нееквівалентними перетвореннями програмного коду блоків LUT FPGA;

K_{obf} – компонент, що визначає виконання зворотної (відновної) обфускації програмного коду блоків LUT FPGA.

На етапі добування вбудованих додаткових даних з програмного коду FPGA стего-ключ вважається відомим стороні добування. Для добування додаткових даних спочатку виконується деобфускація, яка полягає в повторному виконанні обфускації відповідно до правил 6.1 – 6.3 етапу 6 даного методу. Після виконання деобфускації з застосуваннями методу, запропонованого в даній роботі виконується добування вбудованих даних з інформаційного контейнера програмного коду FPGA.

3.2 Експериментальне дослідження результатів застосування запропонованого гібридного методу

Мета експериментального дослідження полягає у визначенні практичного ефекту запропонованого методу. В ході експерименту реалізовано два напрямки оцінки практичного ефекту. Перший напрямок – оцінка збільшення обсягу, доступного для прихованого стеганографічного зберігання додаткових даних в середовищі програмного коду FPGA за рахунок використання гібридного методу вбудовування даних порівняно з вбудовуванням, забезпеченим традиційним підходом. Другий напрямок – оцінка ймовірності виявлення відомими методами та засобами стегоаналізу наявності вбудованих даних середовищі програмного коду FPGA.

Середовище проведення експерименту склали наступні програмні засоби.

1) Синтез та розміщення і трасування FPGA-проєктів виконувалося в САПР Intel Quartus Prime 20.1 Lite Edition [122].

2) Розроблений в даній дисертаційній роботі програмний додаток LUTListExtractor, призначення якого полягає в добуванні детальної інформації про структуру FPGA-проєкту з внутрішньої бази даних САПР Intel/Altera Quartus Prime.

3) Сукупність програмних засобів розроблених в межах даної дисертаційної роботи та об’єднаних в єдину інформаційну систему: програмний додаток EsLUTGetter (п. 2.3.2 дисертації), програмний додаток LMatrixLUTGetter (п. 2.3.2 дисертації) та програмний додаток IEBitsExtractor (п. 2.3.3 дисертації).

В якості цільових мікросхем синтезу при проведенні експерименту були використані мікросхеми FPGA Intel Cyclone IV EP4CE15F23A7 [123]. Обрання мікросхем FPGA сімейства Intel Cyclone IV в якості цільових мікросхем синтезу обумовлено поширеністю їх використання, а також

подібністю їх структури до структур класичних FPGA та сімейств Intel/Altera Cyclone II, III та III LS [134].

Вихідні дані експерименту: Для виконання експерименту було відібрано вісім FPGA проєктів різного розміру та призначення. Ці проєкти мають різні частки модулів, що виконують арифметичні обчислення над наближеними даними.

Виконання експериментального дослідження проводилося в *середовищі обчислювальної системи* на базі 8-ядерного процесора AMD Ryzen 7 при обсязі оперативної пам'яті 16 ГБ.

Процедура проведення експериментів полягала в наступному. Для кожному з експериментальних FPGA проєктів виконувалося три етапи локалізації та оцінка обсягу стего-ресурсів відповідно до запропонованого гібридного методу:

1) з використанням тільки еквівалентних перетворень програмного коду FPGA (вектор пріоритетів (3.1) P_{res} = < 1, 0, 0 >);

2) з послідовним використанням стего-ресурсів забезпечених еквівалентними перетвореннями програмного коду FPGA та нееквівалентними перетвореннями за рахунок використання програмного коду несуттєвих блоків LUT (вектор пріоритетів P_{res} = < 1, 2, 0 >);

3) з послідовним використанням стего-ресурсів забезпечених еквівалентними перетвореннями програмного коду FPGA, нееквівалентними перетвореннями за рахунок використання програмного коду несуттєвих блоків LUT та несуттєвих розрядів суттєвих блоків LUT (вектор пріоритетів P_{res} = < 1, 2, 3 >).

При цьому на етапах 2 та 3 експерименту визначалося обидва способи локалізації несуттєвих блоків LUT: шляхом отримання нижньої оцінки кількості несуттєвих блоків LUT та їх локалізації (параметр стего-ключа $neq1_{low}^{up} = 0$) та шляхом отримання верхньої оцінки кількості несуттєвих блоків LUT та їх локалізації (параметр стего-ключа $neq1_{low}^{up} = 1$).

Після цього виконувалася зворотна (відновна) обфускація програмного коду блоків LUT FPGA.

При виконанні етапів 2 та 3 експерименту використовувалися ті ж самі компоненти K_{eq} стего ключа (3.7), що використовувалися на етапі 1 експерименту. Для вбудовування даних в нееквівалентний стего ресурс використовувався тільки один біт програмного коду несуттєвих блоків LUT. Хоча потенційно має місце можливість вбудовування від 1 до 2^n розрядів даних в кожен з несуттєвих блоків LUT.

В табл. 3.1 представлені результати експериментальної оцінки потенційно доступного обсягу даних, які можуть бути вбудовані в програмний код використаних FPGA-проєктів. В таблиці показано загальну кількість блоків LUT у схемі проєкту. Також показано кількість блоків LUT, які задіяні в схемах для виконання наближених арифметичних операцій та їх частку в загальній кількості блоків LUT.

В стовбці, позначеному вектором пріоритетів P_{res} = <1, 0, 0> гібридного методу, показано кількість розрядів додаткових даних, які можна вбудувати в програмний код FPGA використовуючи тільки еквівалентні стегоресурси. В стовбцях, позначений вектором пріоритетів P_{res} = <1, 2, 0> гібридного методу, наведено кількість розрядів додаткових даних, які можна вбудувати в програмний код FPGA використовуючи послідовну комбінацію еквівалентних стего-ресурсів та стего-ресурсів, забезпеченими несуттєвими блоками LUT. При цьому стовбець, позначений параметром стего-ключа $neq1_{low}^{up} = 0$, містить кількість розрядів додаткових даних за умови отримання нижньої оцінки кількості несуттєвих блоків LUT та їх локалізації, а стовбець, позначений параметром стего-ключа $neq1_{low}^{up} = 1$ – за умови отримання верхньої оцінки кількості несуттєвих блоків LUT та їх локалізації. В кожній з комірок зазначених стовбців показано два значення обсягу стего-ресурсу, забезпеченого несуттєвими блоками LUT:

1) при використанні тільки одного розряду програмного коду в кожному з таких блоків;

2) при використанні всіх розрядів програмного коду таких блоків.

Стовбець, позначений вектором пріоритетів P_{res} = <1, 2, 3> гібридного методу містить кількість розрядів стего-ресурсу, забезпеченого несуттєвими розрядами суттєвих блоків LUT, які можуть бути додані до обсягу стего-ресурсів, наведених в попередніх стовбцях таблиці за умови використання несуттєвих розрядів.

Таблиця 3.1 – Результати експериментального дослідження гібридного методу замаскованого зберігання додаткових даних

FPGA проєкти	Кількість блоків LUT		Потенційно доступний обсяг додаткових даних			
	Загальна	В арифметичних схемах	P_{res} = <1, 0, 0>	P_{res} = <1, 2, 0>		P_{res} = <1, 2, 3>
				$neq1_{low}^{up}$ = 0	$neq1_{low}^{up}$ = 1	
1	421	120 (28,5%)	49	20 320	36 576	+27
2	597	152 (20,3%)	95	25 400	43 688	+107
3	780	162 (20,8%)	176	32 512	51 816	+181
4	975	202 (20,7%)	198	44 704	62 992	+208
5	2183	549 (25,1%)	415	153 2448	195 3120	+359
6	4212	757 (17,9%)	1011	204 3264	261 4176	+571
7	9856	953 (9,66%)	3371	241 3856	310 4960	+829
8	10074	1124 (11,15%)	3675	328 5248	405 6480	+955

В табл. 3.2 наведено узагальнені результати експериментального дослідження. Стовбці, позначені вектором пріоритетів P_{res} = <1, 2, 3> гібридного методу містять загальну кількість розрядів стего-ресурсу, забезпеченого несуттєвими розрядами суттєвих блоків LUT з урахуванням обсягів стего-ресурсів, зазначених в попередніх стовбцях. Також в таблиці наведено частки збільшення обсягу додаткових даних за рахунок використання гібридного методу порівняно з традиційним вбудовуванням, яке використовує еквівалентне перетворення програмного коду FPGA.

Таблиця 3.2 – Узагальнені результати експериментального дослідження гібридного методу

FPGA проєкти	Кількість блоків LUT		Потенційно доступний обсяг додаткових даних				
	Загальна	В арифметичних схемах	P_{res} = <1, 0, 0>	P_{res} = <1, 2, 0>		P_{res} = <1, 2, 3>	
				$neq1_{low}^{up} = 0$	$neq1_{low}^{up} = 1$	$neq1_{low}^{up} = 0$	$neq1_{low}^{up} = 1$
1	421	120 (28,5%)	49	69 (40,8%)	85 (73,4%)	96 (95,9%)	112 (128,5%)
2	597	152 (20,3%)	95	120 (26,3%)	138 (45,2%)	227 (138,9%)	245 (157,9%)
3	780	162 (20,8%)	176	208 (18,2%)	227 (28,9%)	389 (121%)	408 (131,8%)
4	975	202 (20,7%)	198	242 (22,2%)	260 (31,3%)	450 (127,2%)	468 (136,3%)
5	2183	549 (25,1%)	415	568 (36,8%)	610 (46,9%)	927 (123,3%)	969 (133,4%)
6	4212	757 (17,9%)	1011	1215 (20,2%)	1272 (25,8%)	1786 (76,6%)	1843 (82,3%)
7	9856	953 (9,66%)	3371	3612 (7,1%)	3681 (9,2%)	4441 (31,7%)	4510 (33,8%)
8	10074	1124 (11,15%)	3675	4003 (8,9%)	4080 (11,0%)	4958 (34,9%)	5035 (37,1%)

Для експериментальних FPGA проєктів збільшення обсягу потенційно доступних додаткових даних за рахунок використання стего-ресурсу несуттєвих блоків LUT при $neq1_{low}^{up} = 0$ склало від 7,1% до 40,8% та в середньому 22,6%. За рахунок використання стего-ресурсу несуттєвих блоків LUT при $neq1_{low}^{up} = 1$ зазначене збільшення склало від 9,2% до 73,4% та в середньому 34,0%.

Збільшення обсягу потенційно доступних додаткових даних за рахунок використання крім цього також стего-ресурсу несуттєвих розрядів суттєвих блоків LUT при $neq1_{low}^{up} = 0$ склало від 31,7% до 138,9% та в середньому 93,7%. За рахунок використання стего-ресурсу несуттєвих розрядів суттєвих блоків LUT при $neq1_{low}^{up} = 1$ зазначене збільшення склало від 33,8% до 157,9% та в середньому 105,1%.

Другим етапом експериментального дослідження була оцінка стійкості стеганографічного вбудовування до стегоаналізу. Для цього було задіяно декілька доступних програмних продуктів [135-137], які виконують стегоаналіз інформаційного контейнера. Ці програмні продукти формують результат свого функціонування як ймовірність наявності стеганографічно вбудованих додаткових даних в інформаційний контейнер. Оцінки ймовірності наявності додаткових даних та їх порівняння проводилася для еквівалентного вбудовування та для гібридного вбудовування при наявності та відсутності обфускації програмного коду FPGA. У підсумку для програмного коду кожного з експериментальних FPGA-проєктів була отримана усереднена оцінка ймовірності наявності вбудованих додаткових даних в програмному коді FPGA.

На рис. 3.1 показані результати експериментальної оцінки ймовірності наявності додаткових даних в програмному коді експериментальних FPGA-проєктів. Для кожного проєкту зроблено чотири оцінки цієї ймовірності:

1) при використанні тільки еквівалентного вбудовування даних;

2) при еквівалентному вбудовуванні з подальшою обфускацією програмного коду FPGA;

3) при гібридному вбудовуванні (використовуються обидва види стего ресурсів: еквівалентні та нееквівалентні);

4) при гібридному вбудовування з подальшою обфускацією програмного коду FPGA.

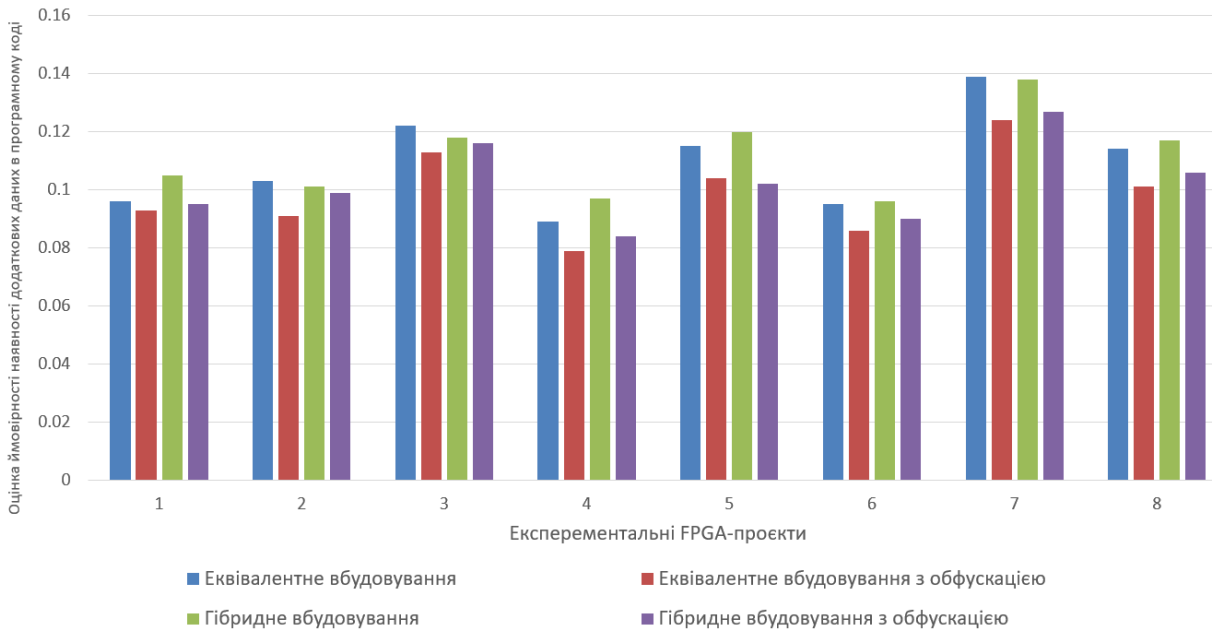


Рисунок 3.1 – Результати оцінки ймовірності наявності додаткових даних в програмному коді FPGA

Як видно з результатів стегоаналізу, оцінка ймовірності виявлення наявності додаткових даних, вбудованих в FPGA-проєкт мало залежить від розміру проєкту. Це можна пояснити тим, що існуючі методи стегоаналізу та програмні засоби, побудовані на їх основі, більше пристосовані до мультимедійних стего-контейнерів, ніж до інформаційний контейнерів програмного коду FPGA. Оцінка ймовірності наявності додаткових даних збільшується в середньому на 2,31% або на 0,002 в абсолютних значеннях при вбудовуванні гібридним методом порівняно з застосуванням еквівалентного підходу. Однак зазначена оцінка зменшується на 6,43%, після виконання запропонованої відновної обфускації. Це пояснюється впливом процедури обфускації на статистичну модель інформаційного контейнера програмного коду FPGA.

3.3 Обговорення результатів експериментального дослідження та ефективності запропонованого гібридного методу

Проведені експериментальні дослідження підтвердили можливість використання гібридного підходу до стеганографічного вбудовування додаткових даних в програмний код FPGA, який поєднує еквівалентний та нееквівалентний підходи до стеганографічного зберігання даних, дозволяє комбінувати стеганографічні ресурси, забезпечені цими підходами та дає можливість виконувати збереження даних відповідно до встановлених пріоритетів використання цих ресурсів.

Порівняно з існуючим еквівалентним підходом до стеганографічного зберігання додаткових даних, запропонований гібридний підхід дозволив збільшити потенційний обсяг прихованого зберігання додаткових даних в програмному коді FPGA. Так, наприклад, для малого за обсягом експериментального проєкта 1 потенційний обсяг стеганографічного зберігання збільшено з 49 розрядів, забезпечених традиційним підходом, до 69, 85, 96 та 112 розрядів в залежності від параметрів вбудовування. Це збільшення дає можливість перенести деякі атрибутивні дані моніторингу в область стего зберігання. Для більшого за обсягом експериментального проєкта 2 потенційний обсяг стеганографічного зберігання збільшено з 95 розрядів, забезпечених традиційним підходом, до 227 та 245 розрядів що дає можливість стеганографічного зберігання контрольних даних у вигляді хеш-суми SHA1 (160 біт) або SHA-224 та деяких атрибутивних даних моніторингу.

Для проєктів 3 та 4 за традиційного підходу мала місце можливість стеганографічного зберігання тільки 176 та 198 розрядів відповідно. Це не давало змогу зберегти в програмному коді контрольні дані, наприклад у вигляді хеш-суми SHA-256. При використанні гібридного методу обсяг зберігання збільшився для цих проєктів таким чином, що зберігання хеш-суми такого розміру стало можливим. Аналогічні порогові зміни в розмірі

обсягу зберігання мають місце також стосовно інших FPGA-проєктів, які брали участь в експерименті.

3.4 Метод формування стеганографічного ключа для прихованого збереження контрольних даних в програмному коді FPGA

3.4.1 Передумови розробки методу

Для більшості практично використовуваних методів оперативного моніторингу характеристик безпеки програмного коду FPGA, базованих на стеганографічному зберіганні контрольних даних (у вигляді цифрового водяного знака), існує необхідність відновлення початкового стану інформаційного об’єкта програмного коду в момент виконання контролю. Для відновлення початкового стану крім контрольних даних в цифровому водяному знаку повинні знаходитися дані, за якими можна виконати відновлення. Ефективний обсяг цифрового водяного знака залежить від обсягу і структури програмного коду FPGA, а також від обмежень, що визначаються стего-ключем вбудовування водяного знака. Значну частину обсягу цифрового водяного знака займають дані, необхідні для відновлення початкового стану. У цих умовах часто існує дефіцит ефективного обсягу цифрового водяного знака для зберігання контрольних даних.

Узагальнено проблема відновлення початкового стану програмного коду [138] полягає в необхідності стеганографічного збереження як самих контрольних даних, так і інформації для відновлення початкового стану програмного коду. Однак обсяг інформації, необхідної для відновлення, може займати значну частину обсягу цифрового водяного знака. Це значно зменшує частину обсягу цифрового водяного знака, яка містить безпосередньо контрольні дані моніторингу. В результаті часто виникає ситуація при якій ефективний обсяг цифрового водяного знака є

недостатнім для зберігання контрольних даних з необхідним для моніторингу розміром.

В попередньому підрозділі дисертації було запропоновано підхід до збільшення обсягу прихованого стеганографічного зберігання даних в середовищі програмного коду FPGA. Це зменшує дефіцит необхідного обсягу зберігання. Однак за умов необхідності відновлення початкового стану інформаційного об’єкта програмного коду за рахунок зберігання в програмному коді крім контрольних даних також даних, за якими можливо відновити початковий стан, бажаною є можливість додаткового збільшення обсягу стеганографічного зберігання даних в програмному коді FPGA.

Виходячи з цього в даному підрозділі дисертаційної роботи пропонується метод, направлений на збільшення обсягу контрольних даних, які можуть бути приховано збереженими в програмному коді FPGA у вигляді цифрового водяного знака, за рахунок застосування інтервального підходу до формування стего-ключа, що використовується при збереженні та зчитуванні контрольних даних.

3.4.2 Аналіз факторів, які впливають на ефективний обсяг цифрового водяного знака в середовищі програмного коду FPGA

Для формування основних положень методу, що пропонується, виконано аналіз двох основних факторів, які впливають на ефективний обсяг цифрового водяного знака:

1) фактора довжини шляху вбудовування;

2) фактора стего-ключа.

Цифровий водяний знак, що використовується для розглянутих методів моніторингу, складається з трьох компонентів (рис. 3.2):

CD – контрольні дані;

ISRec – інформація, необхідна для відновлення початкового стану програмного коду FPGA у момент виконання актів моніторингу;

S – дані, необхідні для визначення місця розташування компонентів цифрового водяного знака під час моніторингу.

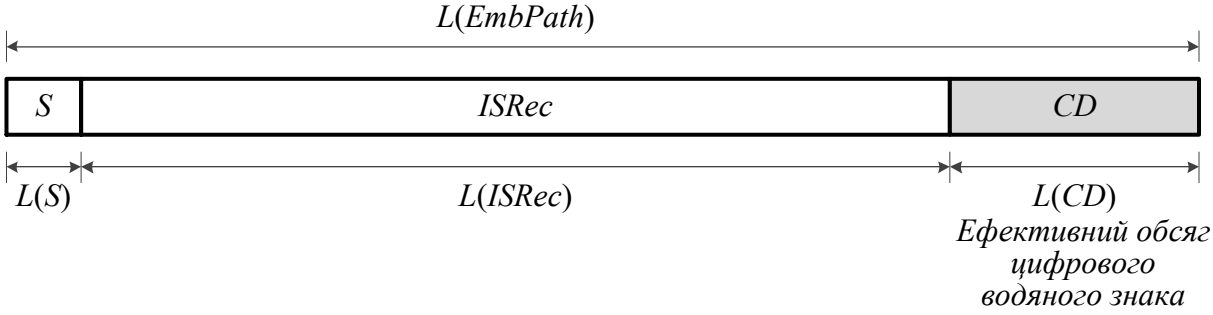


Рисунок 3.2 – Базовий формат цифрового водяного знака, який містить контрольні дані моніторингу

Цифровий водяний знак, що вбудовується у програмний код FPGA в стеганографічний спосіб, розміщується в просторі коду у вигляді сукупності розрядів, які утворюють шлях вбудовування *EmbPath*. Довжина шляху вбудовування *L(EmbPath)*, виражена в кількості розрядів, залежить як від розміру програмного коду FPGA, так і від параметрів стего-ключа. При цьому ефективний обсяг цифрового водяного знака *L(CD)*, це обсяг доступний для зберігання даних моніторингу в його складі:

$$L(CD) = L(EmbPath) - L(ISRec) - L(S); \qquad (3.8)$$

де *L(EmbPath)* – кількість розрядів шляху вбудовування;

L(ISRec) – кількість розрядів даних, що використовуються для відновлення початкового стану інформаційного об’єкта програмного коду FPGA ;

L(S) – довжина поля *S* цифрового водяного знака.

В процесі вбудовування цифрового водяного знака в інформаційний об’єкт програмного коду FPGA змінам піддаються тільки програмний код блоків LUT, що знаходяться на шляху вбудовування. Та частина інформаційного контейнера програмного коду, яка знаходиться поза

шляхом вбудовування, не піддається змінам. В силу цього для відновлення початкового стану інформаційного контейнера необхідно відновити тільки початковий стан блоків LUT, які містяться на шляху вбудовування.

Основним відомим підходом [139], що застосовується для відновлення початкового стану, є метод, заснований на стисненні сукупності розрядів програмного коду блоків LUT, які знаходяться на шляху вбудовування та збереженні результатів стиснення в полі *ISRec* цифрового водяного знака (рис. 3.3).

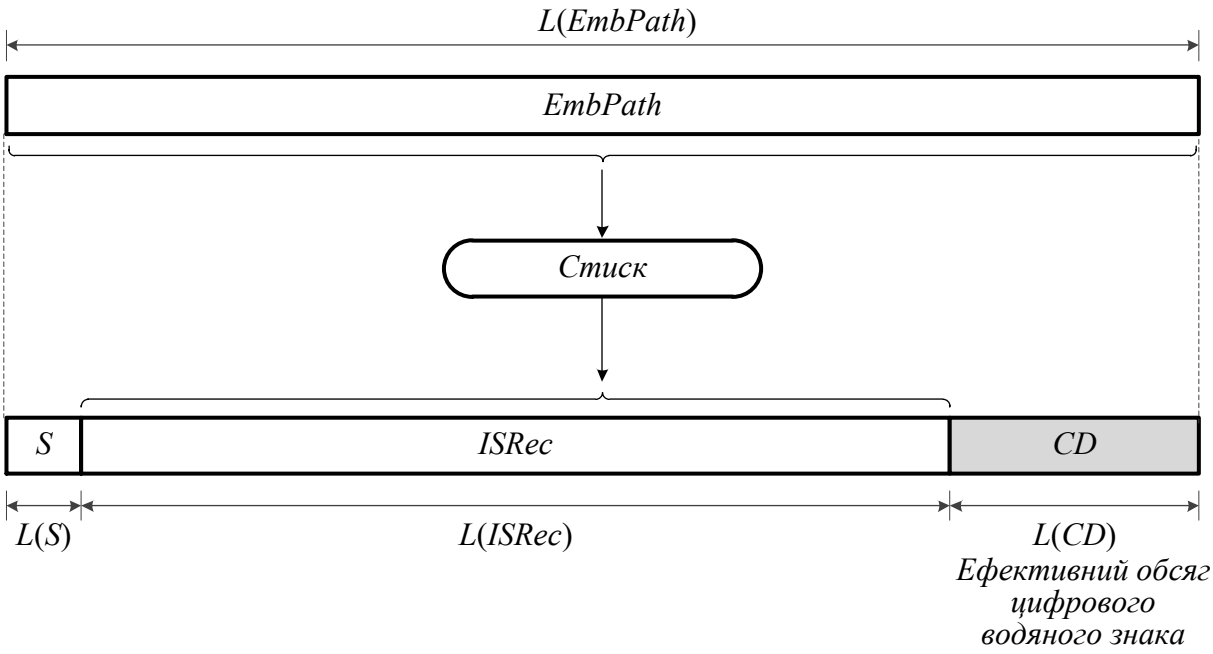


Рисунок 3.3 – Принцип використання стиску без втрат для збереження початкового стану программного коду FPGA

При використанні такого підходу вираз (3.8) набуває наступного вигляду:

$$L(CD) = L(EmbPath) - L(Com(EmbPath)) - L(S) \qquad (3.9)$$

де $L(Com(EmbPath))$ – обсяг результатів стиснення без втрат сукупності розрядів шляху вбудовування.

Нехай необхідний для виконання моніторингу програмного коду FPGA ефективний обсяг цифрового водяного знака повинен становити *LN* розрядів. Тоді має місце наступне співвідношення:

$$L(EmbPath) - L(Com(EmbPath)) - L(S) \geq LN \qquad (3.10)$$

Виконання цього співвідношення залежить від довжини шляху вбудовування та ступеня стиснення даних, що знаходяться на цьому шляху. Довжина шляху вбудовування істотно залежить від властивостей стего-ключа. Збільшення довжини шляху вбудовування призводить до збільшення заповненої частини стего-контейнера, що потенційно може негативно позначитися на ступені здатності контейнера приховувати вбудовану в нього інформацію. Виходячи з цього, параметри стего-ключа повинні бути підібрані таким чином, щоб при виконанні співвідношення (3.10) мінімізувати довжину шляху вбудовування.

Базовий стего-ключ для LUT-контейнера [121], яким є програмний код FPGA, є сукупністю трьох компонентів:

EnumRule – правила, яке задає порядок залучення блоків LUT-контейнера для формування шляху вбудовування;

DThreshold – обмеження на кількість підключень блоків LUT до виходу кожного блоку, що входить у шлях вбудовування;

AddrRule – правило, що визначає адресу розряду програмного коду блоків LUT, який використовується для формування шляхів вбудовування.

Параметр *EnumRule* стего-ключа задає нумераційну відстань між блоками LUT, які мають бути включені до шляху вбудовування. Параметр *DThreshold* – числовий поріг, який визначає можливість включення блоку LUT у шлях вбудовування залежно від кількості зв’язків цього блоку з іншими блоками LUT-контейнера. Зменшення значення параметра *EnumRule* і порога *DThreshold* приводить до збільшення довжини шляху вбудовування, а також збільшення обсягу змін програмного коду FPGA.

Можливі два режими використання стого ключа.

1) Режим спеціалізованого стего-ключа. У цьому режимі стего-ключ формується для кожного окремого FPGA-контейнера і цифрового водяного знака, що вбудовується в нього. У разі використання такого режиму стего-ключ повинен бути переданий системі моніторингу для кожного контрольованого об’єкта деяким надійним каналом зв’язку. Очевидно, що при використанні цього режиму можна підібрати параметри ключа, які забезпечують найбільш відповідну довжину шляху вбудовування. Однак для завдань оперативного моніторингу характеристик безпеки програмного коду такий режим не знайшов застосування. Це пояснюється складністю розподілу ключів та ускладненням системи моніторингу.

2) Режим універсального стего-ключа. У цьому режимі стего-ключ формується одноразово, після чого використовується модулями вбудовування контрольних даних і модулями оперативного моніторингу характеристик безпеки програмного коду. Такий режим є найчастіше використовуваним у моніторингу характеристик безпеки програмного коду. Проблема формування стего-ключ у цьому режимі полягає в наступному. Можлива ситуація за якої під час використання універсального ключа для чергового контейнера неможливо забезпечити виконання умови (3.10). У цьому випадку виконується перехід до використання контрольних даних меншої розрядності, що знижує стійкість системи контролю до атак на контрольну інформацію.

Таким чином, можна констатувати, що ефективний обсяг цифрового водяного знаку та виконання умови (3.10) залежить від довжини шляху вбудовування. Ця довжина залежить від значень компонентів стего-ключа. При використанні режиму універсального стего-ключа має місце суперечність між ефективним обсягом цифрового водяного знаку і часткою задіяних на шляху вбудовування блоків LUT.

3.4.3 Основні положення та послідовність виконання методу

Грунтуючись на наведених вище факторах, пропонується метод формування стего-ключа, що дозволяє адаптувати ефективний обсяг цифрового водяного знака до структури LUT-контейнера. Основні положення запропонованого методу полягають у наступному.

Перше положення методу полягає в тому, що замість точкових значень компонентів універсального стего-ключа пропонується використовувати інтервали значень. Традиційні методи вбудовування цифрового водяного знака у програмний код FPGA використовують стего-ключ із точковими параметрами. В даному методі пропонується використовувати інтервальні параметри стего-ключа. При цьому кожен параметр являє собою інтервал значень, з якого за правилами методу здійснюється вибір параметрів, адаптованих під конкретний стего-контейнер.

В рамках цього положення запропонованого методу для всіх компонентів C_i стего-ключа формується інтервал значень (C_i^{min} ... C_i^{max}) в такий спосіб, що значення компонента $C_i = C_i^{min}$ дає шлях вбудовування, найменшої довжини для даної задачі вбудовування цифрового водяного знака. При цьому, застосування кожного наступного значення з даного інтервалу у якості компоненти C_i стего-ключа дає збільшення довжини шляху вбудовування. При виконанні процедури вбудовування цифрового водяного знака, послідовно, починаючи з мінімальних, обробляються значення інтервалів. (C_i^{min} ... C_i^{max}) та перевіряється виконання співвідношення (3.10). При використанні інтервального методу до декількох компонентів стего-ключа в ключ вводиться додатковий компонент *priority*, який визначає, в якому порядку збільшуються компоненти з декількох інтервалів. Відповідно до зазначеного положення пропонується перейти від точкових значень компонентів *EnumRule* та *DThreshold* стего-ключа до їх інтервальних версій.

Якщо в результаті виконання дій, регламентованих першим положенням запропонованого методу, співвідношення (3.10) не виконається, далі застосовується друге положення методу.

Друге положення методу полягає в тому, що для вбудовування та витягання цифрового водяного знаку може використовуватись послідовність методів $Methods = < met_1, met_2, \ ... \ , met_n >$. Для вбудовування цифрового водяного знака потенційно можна використати декілька методів, що дають різні довжини шляхів вбудовування. При цьому кожен із зазначених методів має індивідуальний набір параметрів (C_i^{min} ... C_i^{max}), розглянутих у попередньому положенні методу. Дане положення запропонованого методу передбачає впорядкування методів вбудовування у вигляді послідовності $Methods$ та виконання вбудовування за допомогою компонентів цієї послідовності з перевіркою співвідношення (3.10). Методи цієї послідовності повинні бути розташовані в порядку потенційного збільшення довжини шляху вбудовавання або ефективного обсягу цифрового водяного знака. При першій спробі застосування використовується перший метод послідовності. Якщо результат вбудовування (з урахуванням першого положення запропонованого методу) не приводить до виконання співвідношення (3.10), то здійснюється перехід до наступного методу послідовності $Methods$. Якщо всі методи послідовності $Methods$ були застосовані з урахуванням першого положення запропонованого методу, але параметри цифрового водяного знака не задовольняють співвідношенню (3.10), застосовуються дії, регламентовані третім положенням запропонованого методу.

Третє положення методу визначає спосіб прийняття рішення про подальші дії, у разі якщо використання жодного з методів послідовності $Methods$ не призвело до виконання співвідношення (3.10). Якщо використання першого та другого положення запропонованого методу не привело до отримання необхідного ефективного обсягу цифрового водяного знака, використовується додатковий однорозрядний компонент стего-ключа

m_{prep}. Цей компонент визначає можливість застосування методу [140] попередньої підготовки інформаційного об’єкта програмного коду FPGA для даних, які містяться в стего-контейнері на шляху вбудовування. Таким чином, параметр m_{prep} визначає допустимість втрати ініціального стану інформаційного об’єкта шляхом переходу до функціонального еквіваленту цього стану. Якщо параметр m_{prep} дорівнює нулю, використання методу попередньої підготовки вважається неприпустимим в даному стего-ключі. Якщо за умов задачі вбудовування цифрового водяного знака використання методу попередньої підготовки не є допустимим, виконується редукція контрольних даних, що змінює вимоги до ефективного обсягу цифрового водяного знака.

Четверте положення методу визначає спосіб витягання цифрового водяного знака зі стего-контейнера в процесі виконання моніторингу характеристики безпеки програмного коду FPGA у разі застосування інтервального стего-ключа. При витяганні цифрового водяного знака, який було вбудовано в програмний код FPGA відповідно до зазначених вище положень методу, виконується процедура визначення точкових значень з інтервальних компонентів стего-ключа. Точкові значення визначаються відповідно до положень пропонованого методу, використаних при вбудовуванні. Після цього виконується витягання цифрового водяного знака з програмного коду FPGA і розкладання його на компоненти. На основі компонента *ISRec* здійснюється відновлення початкового стану інформаційного об’єкта програмного коду. Далі цей інформаційний об’єкт та контрольні дані, отримані із цифрового водяного знака, приймаються системою моніторингу характеристик безпеки програмного коду.

Послідовність виконання запропонованого методу відповідно до наведених положень, представлена на рис. 3.4 у вигляді блок-схеми. Блок-схема показана для двокомпонентної послідовності методів вбудовування $Methods = < met_{Base}, met_{ALM} >$, що відповідає другому положенню запропонованого методу. Компоненти цієї послідовності – базовий метод

вбудовування цифрового водяного знака у програмний код FPGA [140] та метод вбудовування, орієнтований на додаткове використання спеціалізованих блоків Adaptive Logic Module (ALM) [141] FPGA.

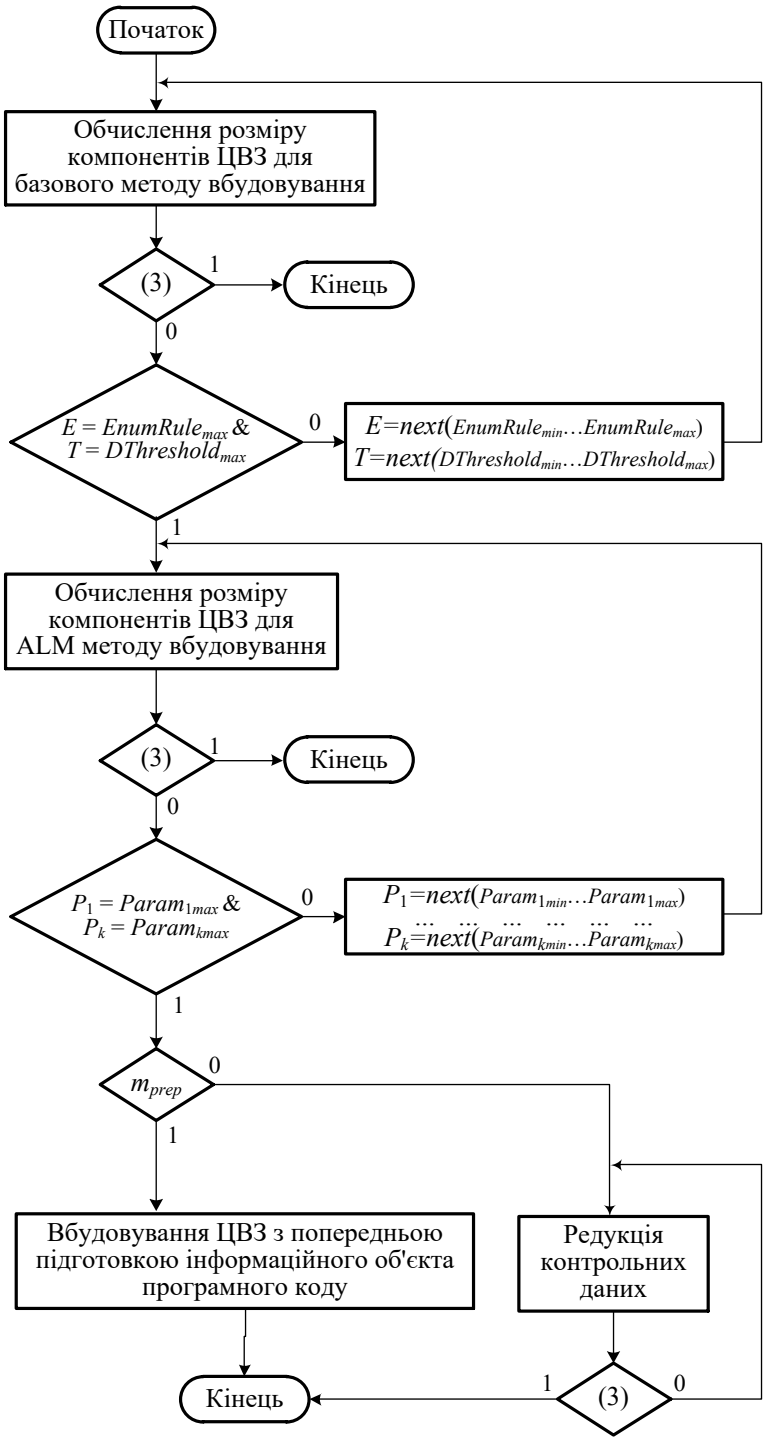


Рисунок 3.4 – Блок схема виконання інтервального методу отримання стего-ключа для вбудовування цифрового водяного знака (ЦВЗ)

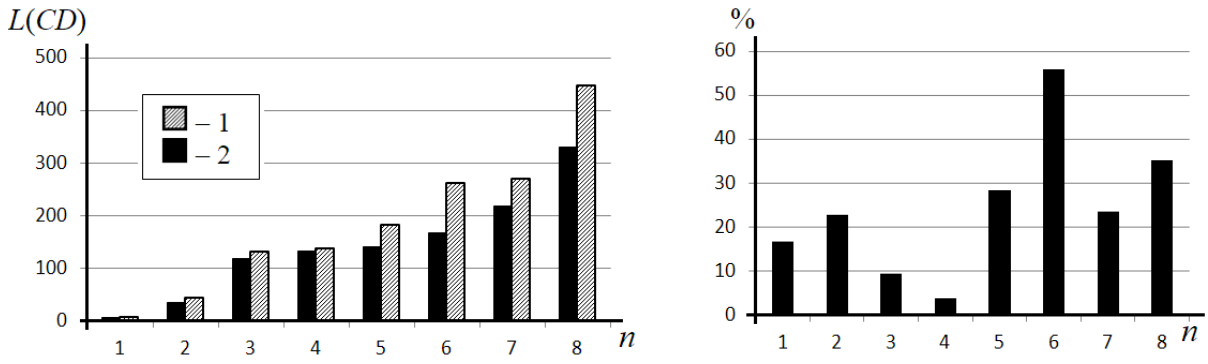
3.4.4 Експериментальне дослідження запропонованого інтервального методу

Для експериментального дослідження запропонованого методу було розроблено відповідне програмне забезпечення. Дане програмне забезпечення використовує в якості компонентів програмні модулі, які реалізують базовий метод вбудовування цифрового водяного знака в програмний код FPGA [140], ALM-базований метод [141] вбудовування та метод вбудовування з попередньою підготовкою інформаційного об’єкта програмного коду FPGA.

Розроблений програмний додаток приймає інтервальний стего-ключ, ітераційно, відповідно до послідовності дій методу, перебирає інтервальні компоненти цього ключа та отримує розміри компонентів цифрового водяного знака для застосовуваних методів вбудовування. На основі розробленого програмного додатку було виконано експериментальне дослідження запропонованого методу. Під час проведення експерименту було задіяно вісім FPGA-проєктів різного розміру та призначення. Синтез проєктів та отримання низькорівневого програмного коду FPGA виконувались у системі Intel Quartus Prime [142].

Методика проведення експерименту полягала в порівнянні результатів, отриманих з використанням звичайного та інтервального стего-ключів. В якості компонентів звичайних стего-ключів при цьому використовувались мінімальні значення параметрів інтервальних стего-ключів. За допомогою обох ключів формувався цифровий водяний знак у просторі кожного з FPGA-проєктів, що беруть участь в експерименті. При достатньому для вирішення задачі моніторингу ефективному обсязі цифрового водяного знака виконувалося вбудовування водяного знака з контрольними даними в інформаційний контейнер програмного коду FPGA. Після цього за стего-ключами проводилося витягання цифрового водяного знака з інформаційного контейнера, поділ цього цифрового водяного знака на компоненти та отримання контрольних даних.

На рис. 3.5 наведено порівняння, отриманих в результаті експерименту, ефективних обсягів цифрових водяних знаків, при застосуванні звичайного підходу до формування стего-ключа та підходу, запропонованого в даному підрозділі дисертаційної роботи.



а) б)

Рисунок 3.5 – Результати експериментального дослідження запропонованого методу

1 – ефективний обсяг цифрового водяного знака при застосуванні запропонованого методу; 2 – обсяг при застосуванні звичайного стего-ключа

На рис. 3.5 а) на горизонтальній осі показані номери експериментальних FPGA-проєктів, впорядкованих за зростанням обсягу апаратних ресурсів, на вертикальній осі відкладено отриманий ефективний обсяг цифрового водяного знака, виражений у кількості розрядів.

Оскільки FPGA-проєкти на рис. 3.5 а) впорядковані за зростанням обсягу апаратних ресурсів, має місце збільшення ефективного обсягу цифрового водяного знака відповідно до кількості блоків LUT в проєкті. Однак, як видно, ефективний обсяг цифрового водяного знака, отриманий при застосуванні запропонованого підходу, для кожного експериментального проєкту є більшим за обсяг, отриманий традиційним шляхом. На рис. 3.5 б) показано відносне збільшення ефективного обсягу

цифрового водяного знака, отримане за рахунок застосування пропозицій даного підрозділу дисертаційної роботи. Відносне збільшення ефективного обсягу не має прямої залежності від обсягу ресурсів проєкту, але для великих, за кількостью блоків LUT, проєктів (проєкти 5 – 8) збільшення обсягу перевищує 20 %. Дуже мале збільшення, отримане для проєкту 4 є результатом специфіки його структури, в якій на множині блоків LUT має місце значно більше зв'язків, ніж для інших проєктів, що були використані в експерименті.

В цілому результати експериментів показують, що запропонований метод дає збільшення ефективного обсягу цифрового водяного знака. В середньому за результатами проведеного експериментального дослідження використання запропонованого методу дозволило на 22,8 % збільшити ефективний обсяг цифрового водяного знака. Це дає можливість використати контрольні дані більшої сукупної розрядності для прихованого моніторингу характеристик безпеки програмного коду FPGA. Таким чином, експериментальне дослідження запропонованого підходу показало, що розробки методу, представленого в даному підрозділі дисертаційної роботи, досягнуто, метод дозволяє збільшити ефективний обсяг цифрового водяного знака, який використовуються для прихованого зберігання контрольних даних.

3.5 Висновки до третього розділу

В третьому розділі дисертації вирішено задачу розробки методу гібридного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA. Метод поєднує еквівалентний та нееквівалентний підхід до перетворення програмного коду FPGA-компонентів та процеси стеганографічного вбудовування та відновної обфускації даних.

Гібридність методу, полягає в двох видах комбінування властивостей відомого еквівалентного підходу та запропонованого в даній роботі

нееквівалентного підходу до стеганографічного зберігання контрольних даних в програмному коді FPGA:

1) комбінування стеганографічних ресурсів, забезпечених еквівалентними та нееквівалентними перетвореннями програмного коду FPGA для досягнення потрібного обсягу даних в середовищі програмного коду та послідовному виділенні цих ресурсів для мінімізації обчислювальної складності їх застосування;

2) комбінування стеганографічного вбудовування (з використанням обох підходів) даних в програмний код FPGA та зворотної (відновної) обфускації програмного коду FPGA, базованої на моделі еквівалентних перетворень, та призначеної для утруднення стегоаналізу і виявлення прихованих в програмному коді даних.

Метод складається з п’яти етапів, кожен з яких містить певну послідовність кроків. Деякі з етапів методу є опціональними та виконуються в залежності від параметрів стего-ключа та конкретної задачі стеганографічного збереження контрольних даних в програмному коді FPGA.

В розділі виконано експериментальне дослідження ефективності розробленого гібридного методу для задач прихованого стеганографічного зберігання контрольних даних в процесі моніторингу програмного коду FPGA. В ході експерименту реалізовано два напрямки оцінки практичного ефекту запропонованого методу. Перший напрямок – оцінка збільшення обсягу, доступного для прихованого стеганографічного зберігання додаткових даних в середовищі програмного коду FPGA за рахунок використання гібридного методу вбудовування даних порівняно з вбудовуванням, забезпеченим традиційним підходом. Другий напрямок – оцінка ймовірності виявлення відомими методами та засобами стегоаналізу наявності вбудованих даних в середовищі програмного коду FPGA.

За першим напрямком експериментального дослідження отримано наступні результати. Для експериментальних FPGA проєктів збільшення

обсягу потенційно доступних додаткових даних за рахунок використання стего-ресурсу несуттєвих блоків LUT при застосуванні швидкого способу локалізації цих блоків склало від 7,1% до 40,8% та в середньому 22,6%, а при застосуванні базового способу локалізації зазначене збільшення склало від 9,2% до 73,4% та в середньому 34,0%.

Збільшення обсягу потенційно доступних додаткових даних за рахунок використання крім цього також стего-ресурсу несуттєвих розрядів суттєвих блоків LUT при застосуванні швидкого способу локалізації цих блоків склало від 31,7% до 138,9% та в середньому 93,7%. За рахунок використання стего-ресурсу несуттєвих розрядів суттєвих блоків LUT при застосуванні базового способу локалізації зазначене збільшення склало від 33,8% до 157,9% та в середньому 105,1%.

За другим напрямком експериментального дослідження в межах стегоналізу програмного коду FPGA-проєктів з вбудованими в них додатковими даними, отримано наступні результати. Оцінка ймовірності наявності додаткових даних при вбудовуванні гібридним методом без виконання етапу відновної обфускації збільшується в середньому на 2,31%, порівняно з застосуванням еквівалентного підходу, та зменшується на 6,43% в разі виконання етапу відновної обфускації.

Також в розділі розроблено метод формування стеганографічного ключа для прихованого збереження контрольних даних в програмному коді FPGA, який дозволяє здійснити балансування між обсягом модифікованої частини програмного коду та ефективним обсягом стеганографічного контейнера, потрібним для збереження контрольних даних. Особливість запропонованого методу полягає у використанні інтервальних компонентів стего-ключа замість точкових компонентів. Це дозволяє адаптувати ключ до кожного конкретного стего-контейнера програмного коду FPGA з метою збільшення ефективного обсягу цифрового водяного знака, який вбудовується в контейнер. Метод може бути застосований якості доповнення

як до традиційних методів стеганографічного вбудовування даних в програмний код FPGA, так і до гібридного методу.

Виконано експериментальне дослідження ефективності розробленого методу формування стеганографічного ключа. В середньому за результатами проведеного експерименту використання запропонованого методу дозволило на 22,8 % збільшити ефективний обсяг цифрового водяного знака. Це дає можливість використати контрольні дані більшої сукупної розрядності для прихованого моніторингу характеристик безпеки програмного коду FPGA.

За результатом досліджень, описаних в даному розділі дисертаційної роботи, отримано наступні пункти наукової новизни:

– *вперше* розроблено метод гібридного замаскованого зберігання даних в середовищі програмного коду FPGA, *який відрізняється* поєднанням еквівалентного та нееквівалентного підходів до перетворення програмного коду FPGA, а також поєднанням процесів стеганографічного вбудовування та відновної обфускації даних, *що дозволяє* збільшити обсяг контрольних даних, які можуть бути приховано вбудованими в програмний код FPGA, не збільшуючи при цьому здатність традиційних методів стегоаналізу до виявлення цих даних в програмному коді.

– *дістав подальшого розвитку* метод формування стеганографічного ключа для замаскованого збереження даних в програмному коді FPGA, *який відрізняється* можливістю адаптації до структури зв’язків між елементарними одиницями FPGA та до необхідного обсягу контрольних даних, *що дозволяє* виконувати балансування між обсягом частини програмного коду, яка модифікується в результаті вбудовування даних та ефективним обсягом стеганографічного контейнера, потрібним для збереження контрольних даних.

Основні результати розділу опубліковані у працях [143–149].

4 РОЗРОБКА ПІДСИСТЕМИ ГІБРИДНОГО ЗАМАСКОВАНОГО ЗБЕРІГАННЯ КОНТРОЛЬНИХ ДАНИХ В СКЛАДІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ПРИХОВАНОГО МОНІТОРИНГУ ПРОГРАМНОГО КОДУ FPGA-КОМПОНЕНТІВ

4.1 Мета та задачі розробки підсистеми гібридного замаскованого зберігання контрольних даних

Мета розробки підсистеми гібридного замаскованого зберігання контрольних даних полягає в забезпеченні процедур прихованого збереження еталонних контрольних даних та доступу до таких даних, необхідних для виконання прихованого моніторингу низькорівневого програмного коду FPGA-компонентів комп'ютерних та інформаційних систем. Підсистема є складовою частиною систем прихованого моніторингу програмного коду FPGA-компонентів, що здійснює моніторинг характеристик безпеки, легітимності використання та шляхів розповсюдження програмного коду FPGA-компонентів.

Задачами розробки є:

– визначення інформаційних потоків підсистеми;

– визначення функціональних та нефункціональних вимог до розроблюваної підсистеми;

– проєктування діаграми логічного уявлення розроблюваної підсистеми;

– визначення розгортання підсистеми;

– визначення структури програмного проєкту;

– розробка та аналіз програмного коду підсистеми;

– тестування розроблюваної підсистеми.

Система прихованого моніторингу програмного коду FPGA-компонентів, складовою частиною якої є розроблювана підсистема використовується на протязі життєвого циклу FPGA-компонентів для захисту від втручання в функціонування зазначених компонентів шляхом зловмисних маніпуляцій програмним кодом.

4.2 Визначення інформаційних потоків підсистеми гібридного замаскованого зберігання контрольних даних

Визначено інформаційні потоки які передаються від головної системи моніторингу програмного коду FPGA до підсистеми гібридного замаскованого зберігання контрольних даних та в зворотному напрямку.

Визначено наступні вхідні інформаційні потоки даних, які передаються від головної системи до підсистеми гібридного замаскованого зберігання контрольних даних:

– об’єкт низькорівневого програмного коду FPGA, призначений для вбудовування в нього контрольних даних – структуроване подання низькорівневого програмного коду FPGA-компонента, передане в підсистему з головної системи для вбудовування в нього контрольних даних з використанням гібридного методу замаскованого зберігання;

– еталонні контрольні дані – контрольні дані, обчислені або призначені системою прихованого моніторингу програмного коду FPGA-компонентів на підготовчому етапі моніторингу певного компонента;

– параметри збереженні контрольних даних – сукупність параметрів, які утворюють стего-ключ та визначають локалізацію вбудовування відповідно до методу гібридного замаскованого зберігання даних;

– карта відповідності програмного коду FPGA – дані про відповідність розрядів низькорівневого програмного коду FPGA-компонента та розрядів його інформаційного об’єкта;

– об’єкт низькорівневого програмного коду FPGA, призначений для діставання з нього контрольних даних – структуроване подання низькорівневого програмного коду FPGA-компонента, передане в підсистему з головної системи для діставання з нього контрольних даних, які було вбудовано з використанням гібридного методу замаскованого зберігання;

– параметри діставання контрольних даних – сукупність параметрів, які утворюють стего-ключ та визначають локалізацію вбудованих даних відповідно до методу гібридного замаскованого зберігання;

– параметри обфускації або деобфускації програмного коду – сукупність параметрів, які визначають локалізацію дії зворотної обфускації в просторі програмного коду FPGA-компонента відповідно до гібридного методу замаскованого зберігання даних.

Визначено наступні вихідні потоки даних, які передаються від підсистеми гібридного замаскованого зберігання контрольних даних до головної системи:

– об’єкт низькорівневого програмного коду FPGA, в який замасковано збережені контрольні дані – структуроване подання низькорівневого програмного коду FPGA-компонента, в яке виконано замасковане вбудовування даних;

– зчитані контрольні дані – еталонні контрольні дані, які було зчитано з програмного коду FPGA-компонента.

На рис. 4.1 наведено діаграму інформаційних потоків підсистеми гібридного замаскованого зберігання контрольних даних.

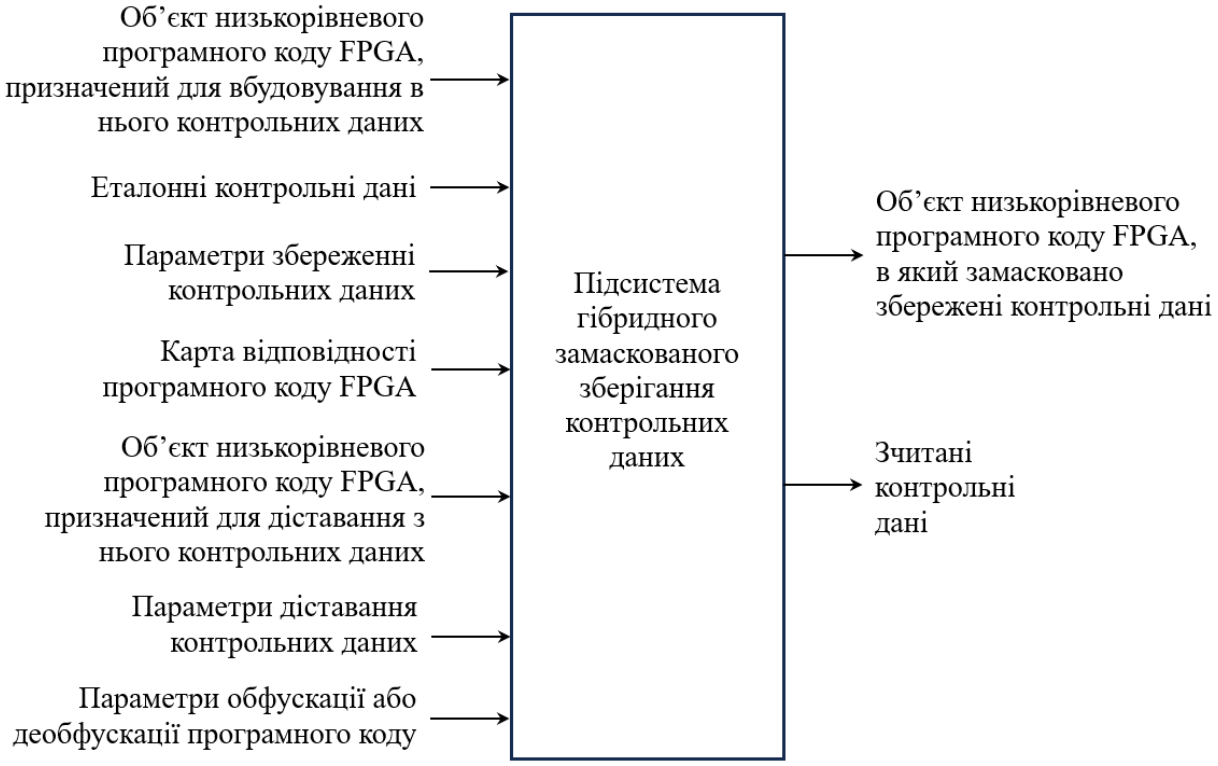


Рисунок 4.1 – Діаграма інформаційних потоків підсистеми

4.3 Визначення функціональних вимог до підсистеми гібридного замаскованого зберігання контрольних даних

Для визначення функціональних вимог до підсистеми виконується виділення акторів, що взаємодіють з підсистемою. Єдиним актором визначається головна система, яка виконує прихований моніторинг програмного коду FPGA та використовує розроблювану підсистему для замаскованого збереження еталонних контрольних даних.

Виконується визначення сценаріїв взаємодії між головною системою та підсистемою замаскованого зберігання даних. Сукупність таких сценаріїв подано у вигляді UML-діаграми прецедентів (рис. 4.2).

На основі наведених на діаграмі варіантів використання підсистеми та сценаріїв її взаємодії з головною системою, виконується опис функціональних вимог до інформаційної підсистеми.

ФВ1. Отримання інформаційного об'єкта програмного коду FPGA для вбудовування контрольних даних. Підсистема повинна забезпечити прийом від системи моніторингу інформаційного об'єкта, що містить бінарний код FPGA, у який буде здійснюватися вбудовування еталонних контрольних даних.

ФВ2. Отримання еталонних контрольних даних від системи моніторингу. Підсистема повинна приймати еталонні контрольні дані, які необхідно вбудувати в об'єкт програмного коду FPGA-компонента використовуючи гібридний метод замаскованого зберігання. Дані мають бути перевірені на коректність формату та готовність до вбудовування.

ФВ3. Виконання гібридного збереження еталонних контрольних даних в програмний код FPGA. Підсистема повинна реалізувати механізм вбудовування контрольних даних у програмний код FPGA із застосуванням гібридного методу замаскованого збереження даних. Процес вбудовування має використати в якості вхідних параметрів:

– карту відповідності програмного коду FPGA-компонента;

– параметри вбудовування, які в сукупності містять стего-ключ.

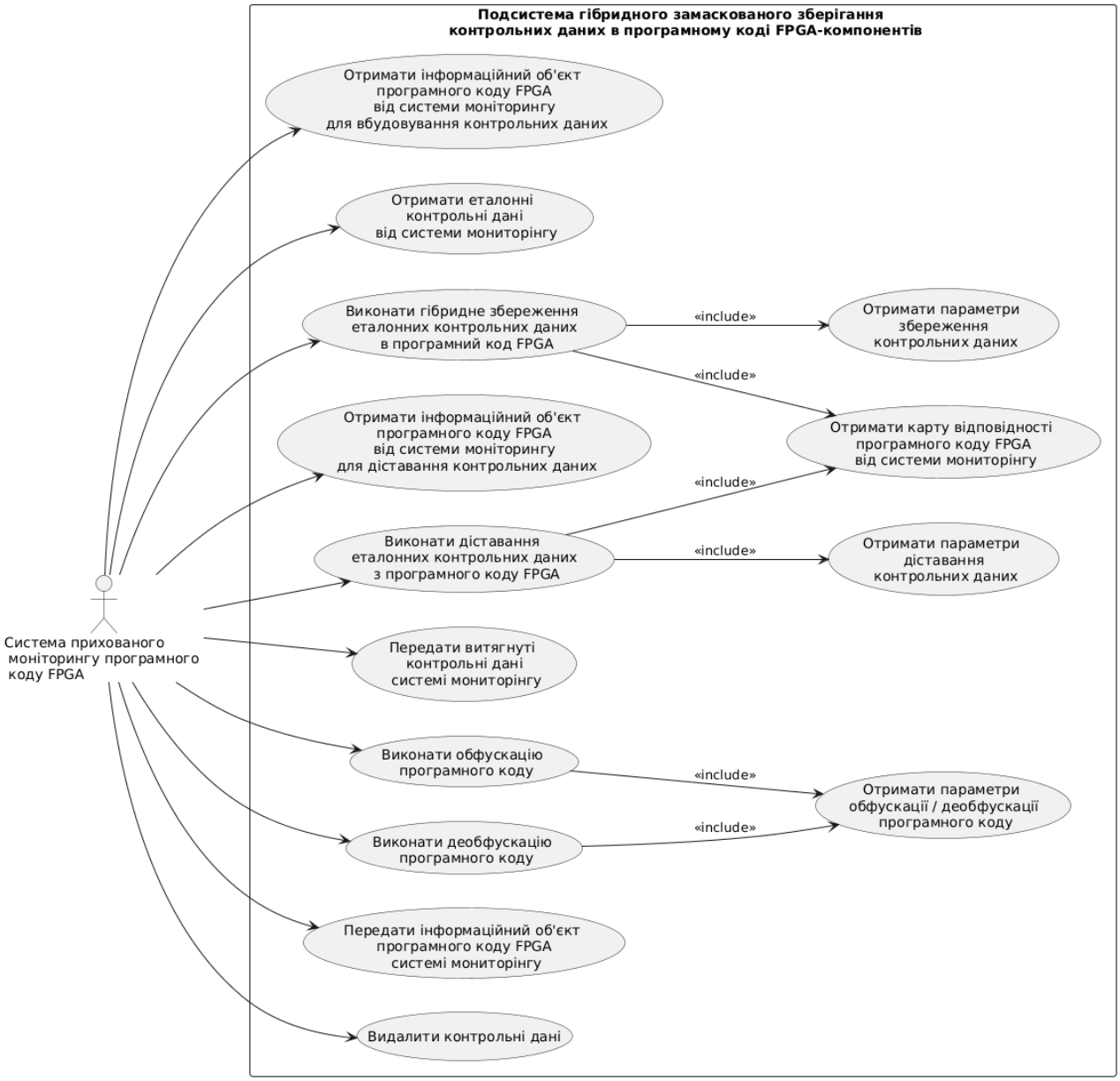


Рисунок 4.2 – Діаграма прецедентів інформаційної підсистеми

ФВ4. Отримання інформаційного об’єкта програмного коду FPGA для діставання контрольних даних. Підсистема повинна отримувати від системи моніторингу програмний код FPGA-компонента, з якого необхідно витягнути раніше вбудовані контрольні дані. Об’єкт програмного коду має бути ідентичний або сумісний із форматом, який використовувався при вбудовуванні.

ФВ5. Виконання діставання еталонних контрольних даних з програмного коду FPGA. Підсистема повинна забезпечити діставання контрольних даних із інформаційного об’єкта програмного коду FPGA-компонента. Процес діставання еталонних контрольних даних використовує:

– карту відповідності програмного коду FPGA-компонента;

– параметри діставання даних, які в сукупності містять стего-ключ, що використовувався при вбудовуванні.

ФВ6. Передача витягнутих контрольних даних системі моніторингу. Після успішного діставання підсистема повинна передати отримані еталонні контрольні дані у систему моніторингу для подальшого порівняння зі знов обчисленими в процесі моніторингу контрольними даними.

ФВ7. Отримання карти відповідності програмного коду FPGA. Підсистема повинна отримувати карту відповідності, яка визначає дані про відповідність розрядів низькорівневого програмного коду FPGA-компонента та розрядів його інформаційного об’єкта. Ця карта використовується як при вбудовуванні, так і при діставанні контрольних даних з програмного коду FPGA-компонента.

ФВ8. Отримання параметрів збереження контрольних даних. Підсистема повинна отримувати параметри, що визначають процес вбудовування: сукупність параметрів, які утворюють стего-ключ та визначають локалізацію вбудовування відповідно до методу гібридного замаскованого зберігання даних.

ФВ9. Отримання параметрів діставання контрольних даних. Підсистема повинна отримувати параметри, необхідні для витягування даних, які відповідають параметрам вбудовування: сукупність параметрів, які утворюють стего-ключ та визначають локалізацію вбудованих даних відповідно до методу гібридного замаскованого зберігання.

ФВ10. Виконання обфускації програмного коду FPGA-компонента. Підсистема повинна забезпечити обфускацію програмного коду FPGA з метою ускладнення аналізу та виявлення прихованих контрольних даних. Обфускація має бути сумісною з процесами вбудовування та діставання відповідно до положень гібридного методу замаскованого зберігання даних в програмному коді FPGA-компонентів. Також обфускація повинна відповідати вимогам відновлення, які полягають в можливості відкату програмного коду в первісний стан після виконання деобфускації, процедура якої збігається з процедурою обфускації.

ФВ11. Отримання параметрів обфускації / деобфускації програмного коду FPGA-компонента. Підсистема повинна отримувати від головної системи параметри, що визначають налаштування процедури обфускації та деобфускації програмного коду FPGA-компонента: сукупність параметрів, які визначають локалізацію дії зворотної обфускації в просторі програмного коду FPGA-компонента відповідно до гібридного методу замаскованого зберігання даних.

ФВ12. Виконання деобфускації програмного коду. Підсистема повинна забезпечити відновну деобфускацію програмного коду FPGA до стану, придатного для витягування контрольних даних відповідно до положень гібридного методу замаскованого зберігання даних в програмному коді FPGA-компонентів.

ФВ13. Передача інформаційного об'єкта програмного коду FPGA системі моніторингу. Після виконання операцій (вбудовування еталонних контрольних даних, обфускація, деобфускація) підсистема повинна передати змінений або оброблений програмний код у головну систему моніторингу.

ФВ14. Видалення контрольних даних. Підсистема повинна забезпечити можливість видалення у разі необхідності раніше вбудованих контрольних даних із програмного коду FPGA.

4.4 Специфікація нефункціональних вимог

Визначення нефункціональних вимог для підсистеми гібридного замаскованого зберігання контрольних даних є необхідним для забезпечення якісних характеристик цієї програмної системи. Зазначені вимоги не визначаються функціональністю підсистеми, але безпосередньо впливають на ефективність та надійність її роботи. Вони встановлюють чіткі критерії оцінювання продуктивності, безпеки, масштабованості та інших властивостей системи, що дозволяє об’єктивно перевіряти відповідність реалізації поставленим цілям.

До підсистеми гібридного замаскованого зберігання контрольних даних в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів визначаються наступні нефункціональні вимоги.

НФВ1. Продуктивність. Підсистема повинна забезпечувати виконання операцій вбудовування, діставання, обфускації та деобфускації за обмежений час. Час вбудовування контрольних даних не повинен перевищувати допустимий поріг, який визначається системою моніторингу. Час діставання еталонних контрольних даних має бути достатньо малим для використання в режимі моніторингу. Обробка великих файлів низькорівневого програмного коду FPGA-компонентів повинна масштабуватися без суттєвого зростання затримок.

НФВ2. Масштабованість. Підсистема повинна підтримувати: обробку різних розмірів низькорівневого програмного коду FPGA-компонентів; збільшення обсягу еталонних контрольних даних без критичного падіння продуктивності.

НФВ3. Сумісність. Підсистема повинна бути сумісною з: різними форматами низькорівневого програмного коду FPGA-компонентів; головною системою моніторингу; різними платформами виконання.

НФВ4. Модульність і розширюваність. Підсистема повинна мати модульну архітектуру та можливість заміни або додавання алгоритмів: замаскованого вбудовування даних та обфускації. Такі розширення повинні здійснювати мінімальний вплив на інші компоненти системи.

НФВ5. Керованість і конфігурованість. Підсистема повинна підтримувати гнучке налаштування: параметрів вбудовування та діставання даних; алгоритмів вбудовування та обфускації; рівня захисту та надлишковості. Налаштування мають змінюватися без необхідності модифікації програмного коду підсистеми.

НФВ6. Відтворюваність процесів. Процеси вбудовування та діставання повинні бути детермінованими: за однакових вхідних параметрів результат функціонування підсистемимає бути ідентичним.

НФВ7. Логування та аудит. За необхідності в підсистемі повинна бути можливість активації журналу дій, в який заносяться: операції вбудовування та діставання контрольних даних; помилки та винятки. Це необхідно для аудиту та аналізу інцидентів.

НФВ8. Зручність інтеграції. Підсистема повинна надавати чіткий інтерфейс для взаємодії та забезпечувати простоту підключення до системи моніторингу. Крім того підсистема повинна мати документовані інтерфейси та формати даних.

НФВ9. Відмовостійкість. Підсистема повинна коректно обробляти помилки: некоректні вхідні дані, відсутність параметрів.

У таких випадках система повинна: повертати зрозумілі повідомлення про помилки та не порушувати загальну роботу головної системи прихованого моніторингу.

НФВ10. Прихованість. Вбудовані дані не повинні: впливати на функціональність FPGA-компонентів; бути помітними при стандартному аналізі; змінювати статистичні властивості коду настільки, щоб їх було можливо виявити.

4.5 Створення діаграми логічного уявлення розроблюваної підсистеми

Підсистема гібридного замаскованого зберігання контрольних даних в програмному коді FPGA-компонентів складається з сукупності програмних модулів. На рис. 4.3 показано діаграму логічного уявлення підсистеми, яка показує зв’язки цих модулів між собою та з головною системою моніторингу.

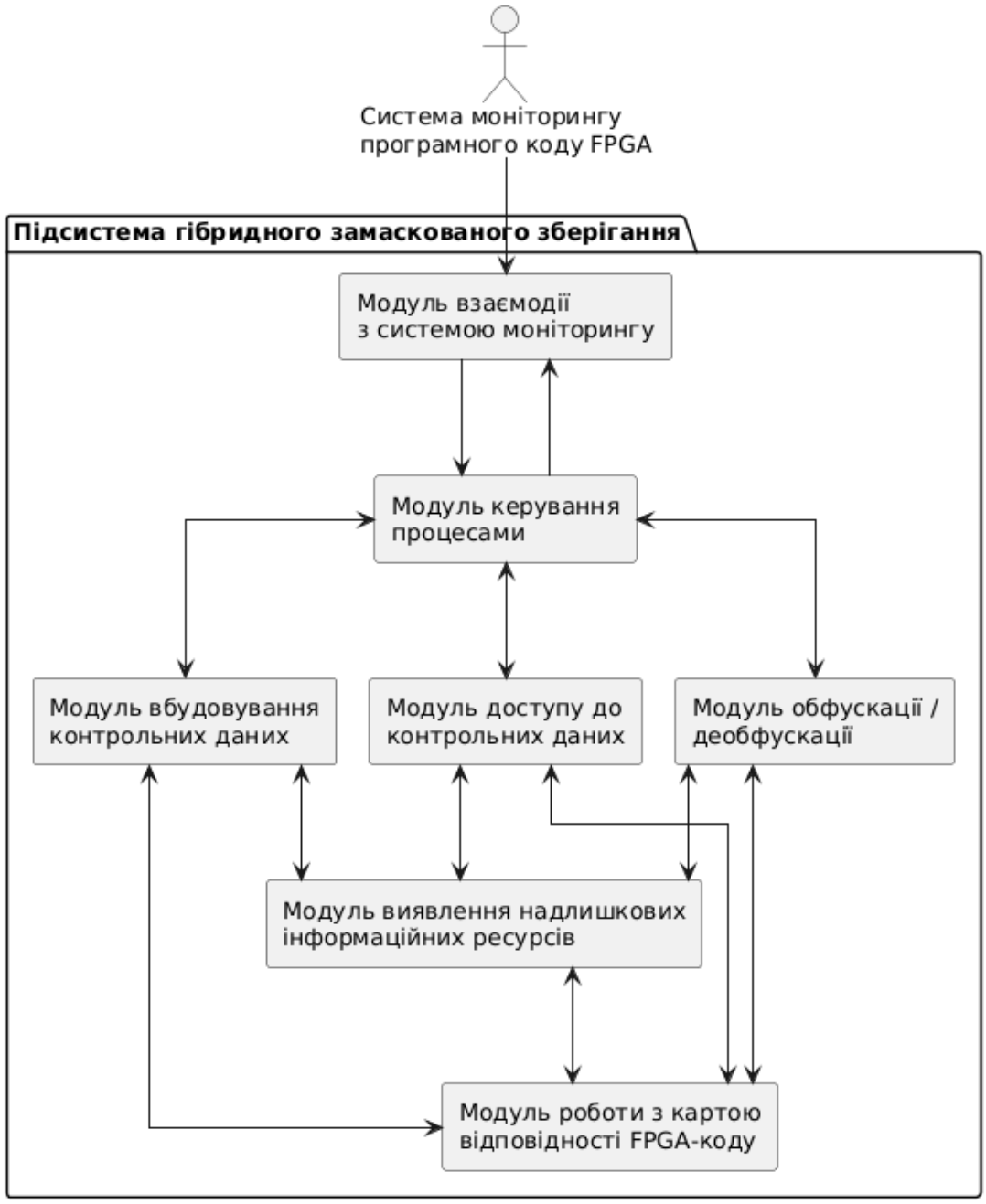


Рисунок 4.3 – Діаграма логічного уявлення підсистеми

Модуль взаємодії з системою моніторингу є єдиною точкою входу та виходу підсистеми і забезпечує прийом FPGA-коду, параметрів операцій і команд від головної системи, а також передачу результатів обробки до головної систем.

Модуль керування процесами виконує координаційну функцію, визначаючи послідовність виконання операцій та агрегуючи отримані результати для подальшої передачі.

Модуль вбудовування контрольних даних реалізує приховане зберігання еталоннних контрольних даних в низькорівневому коді FPGA-компонентів відповідно до розробленого гібридного методу. Модуль використовує модуль виявлення надлишкових інформаційних ресурсів для локалізації надлишкових осередків програмного коду FPGA-компонентів та виконує узгодження вбудовування з картою відповідності програмного коду.

Модуль доступу до контрольних даних виконує зворотну операцію, забезпечуючи витягування прихованих даних із програмного коду з урахуванням способу їх розміщення.

Модуль обфускації та деобфускації змінює структуру FPGA-коду для ускладнення його аналізу сторонніми засобами та відновлює первинний стан програмного коду перед процедурою витягання даних відповідно до запропонованого методу гібридного замаскованого вбудовування даних.

Модуль виявлення надлишкових інформаційних ресурсів здійснює аналіз FPGA-коду з метою визначення осередків, придатних для прихованого зберігання додаткових даних відповідно до моделей, запропонованих в дисертації.

Модуль роботи з картою відповідності FPGA-коду обробляє інформацію про відповідність елементів низькорівневого програмного коду FPGA-компонента та розрядів його інформаційного об’єкта.

Взаємодія між модулями побудована наступним чином: модуль керування процесами забезпечує звернення до всіх функціональних модулів; модуль взаємодії забезпечує прийом даних з головної системи моніторингу

та передачу даних в головну систему. Модулі вбудовування, доступу до даних та обфускації взаємодіють із модулем виявлення надлишкових ресурсів і модулем карти відповідності, використовуючи їх як спільні сервіси для визначення допустимих областей вбудовування.

4.6 Визначення розгортання підсистеми гібридного замаскованого зберігання контрольних даних

Визначено розгортання підсистеми гібридного замаскованого зберігання контрольних даних. На рис. 4.4 показано діаграму розгортання, отриману за результатами такого визначення.

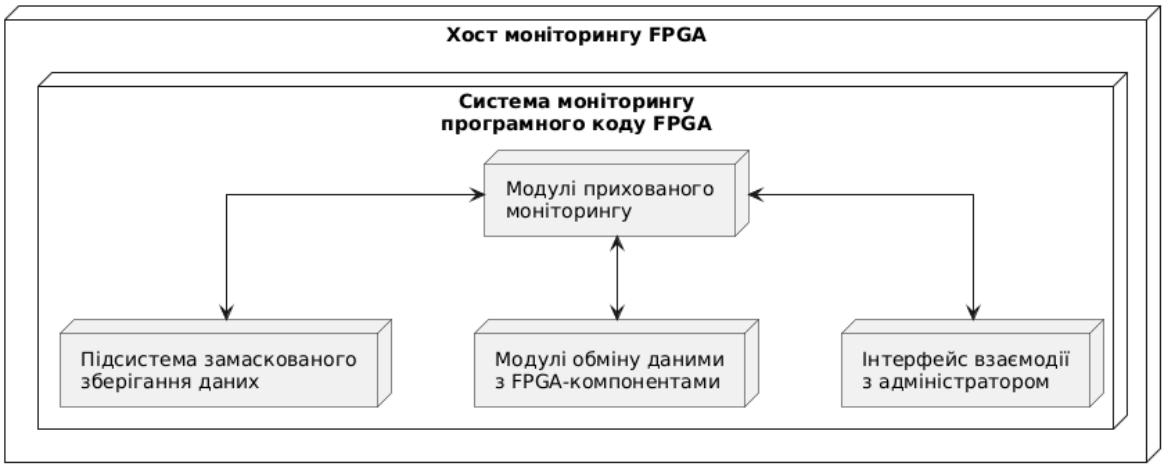


Рисунок 4.4 – Діаграма розгортання підсистеми

Діаграма відображає фізичну архітектуру системи моніторингу цілісності програмного коду FPGA-компонентів. У межах одного обчислювального вузла розміщено головну систему моніторингу та її підсистему замаскованого зберігання даних. Центральним елементом логічної структури є модулі прихованого моніторингу, які використовують інші компоненти системи: підсистему замаскованого зберігання еталонних контрольних даних, модулі обміну даними з FPGA-компонентами та інтерфейс взаємодії з адміністратором.

4.7 Структура програмного проєкту підсистеми

Підсистема замаскованого зберігання контрольних данимих в середовищі програмного коду FPGA-компонентів складається з декількох програмних компонентів, реалізованих на мовах програмування C#, TCL та Python. Ядро підсистеми та її основі модулі реалізовані мовою C#. Далі розглядаються структура проєкту ядра підсистеми. Архітектура підсистеми організована у вигляді трьох основних частин на відповідних рівнях абстракції: доменний рівень, рівень прикладної логіки та точка входу програми, що дозволяє розділити модель даних, логіку та механізм запуску підсистеми. Структура проєкту показана на рис. 4.5.

Папка Domain містить опис предметної області системи. В цій папці файл FPGACode.cs визначає модель інформаційного об’єкта програмного коду FPGA-компонента у вигляді бінарного масиву. Файл ControlData.cs описує контрольні дані, які зберігаються та використовуються системою. Файл EmbeddedControlData.cs представляє контрольні дані, що вже асоційовані з FPGA-кодом. Файл Mapping.cs містить опис логічних відповідностей між елементами низькорівневого програмного коду FPGA та інформаційний об’єктом програмного коду. Файл Parameters.cs визначає параметри операцій вбудовування, витягування та обфускації, основними з яких є стего-ключі. Інтерфейс IControlDataStorage.cs задає контракт для збереження та зчитування контрольних даних, не прив’язуючись до конкретної реалізації сховища даних.

Папка Application містить реалізацію прикладної логіки системи. Файл ControlDataService.cs є головним сервісом, який координує роботу з контрольними даними та реалізує сценарії їх збереження і відновлення. Файл EmbeddingService.cs відповідає за процес вбудовування контрольних даних у програмний код FPGA-компонентів. Файл ExtractionService.cs виконує зворотну операцію – витягування вбудованих контрольних даних з програмного коду FPGA. Файл ObfuscationService.cs забезпечує додаткову

обробку FPGA-коду для ускладнення його аналізу. Файл ControlDataRepository.cs реалізує механізм зберігання контрольних даних і забезпечує фізичне зчитування та запис відповідно до інтерфейсу, визначеного в доменному шарі.

```
FpgaControlDataSaveSubsystem.sln
│
├── Domain/
│   ├── Entities/
│   │   ├── FPGACode.cs
│   │   ├── ControlData.cs
│   │   ├── EmbeddedControlData.cs
│   │   └── Mapping.cs
│   │
│   ├── Parameters.cs
│   └── IControlDataStorage.cs
│
├── Application/
│   ├── Services/
│   │   ├── ControlDataService.cs
│   │   ├── EmbeddingService.cs
│   │   ├── ExtractionService.cs
│   │   └── ObfuscationService.cs
│   │
│   └── Storage/
│       └── ControlDataRepository.cs
│
├── Program.cs
│
└── tests/
    └── Tests.cs
```

Рисунок 4.5 – Структура проєкту підсистеми

Файл Program.cs розташований є точкою входу в підсистему. Він відповідає за ініціалізацію всіх компонентів, налаштування залежностей між сервісами, а також запуск основних компонентів підсистеми. Така організація проєкту забезпечує чітке розділення відповідальностей, спрощує супровід системи та дозволяє змінювати, в разі потреби, механізми зберігання контрольних даних.

4.8 Розробка діаграми класів

Розроблена діаграма класів описує підсистему замаскованого збереження контрольних даних у програмному коді FPGA-компонентів. Діаграма класів основної програмної частини наведена на рис. 4.6.

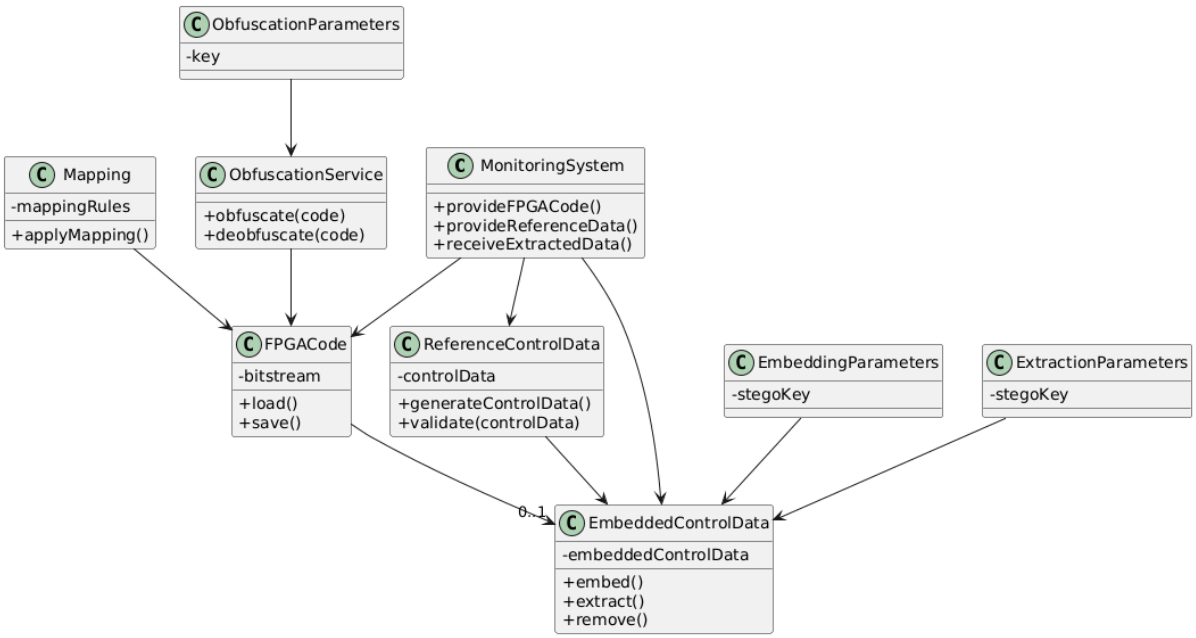


Рисунок 4.6 – Діаграма класів основної програмної частини підсистеми

Центральним елементом діаграми є клас MonitoringSystem, який взаємодіє з іншими сутностями, отримуючи та передаючи дані для процесів вбудовування і діставання контрольних даних. Клас FPGACode представляє бінарний код FPGA, який може бути отриманим від головної системи для

збереження в ньому контрольних даних. Клас ReferenceControlData зберігає еталонні контрольні дані, отримані від головної системи.

Клас EmbeddedControlData відповідає за представлення контрольних даних, які вбудовуються у програмний код FPGA. Він підтримує операції вбудовування, діставання та видалення цих даних. Клас ReferenceControlData пов’язаний з EmbeddedControlData, що відображає віднесення вбудованих даних до еталонних.

Клас Mapping визначає правила відповідності, які використовуються для правильного розміщення або пошуку контрольних даних у низькорівневому програмному коді FPGA-компонентів. Класи EmbeddingParameters та ExtractionParameters містять параметр stegoKey, який використовується для керування процесами вбудовування та діставання відповідно. Це забезпечує узгодженість між операціями приховування та відновлення даних.

Клас ObfuscationParameters містить ключ, що використовується для обфускації програмного коду. Відповідні операції виконуються класом ObfuscationService, який забезпечує обфускацію та деобфускацію програмного коду FPGA-компонентів.

У цілому діаграма відображає взаємодію між компонентами підсистеми, необхідними для прихованого збереження та діставання контрольних даних, а також виконання обфускації та деобфускації програмного коду FPGA-компонентів відповідно до запропонованого в дисертації гібридного методу.

4.9 Функціональне тестування підсистеми

Функціональне тестування підсистеми замаскованого збереження контрольних даних в середовищі програмного коду FPGA-компонентів проводиться з метою перевірки відповідності реалізованої функціональності підсистеми встановленим вимогам. Основною метою

тестування є підтвердження коректної роботи операцій збереження, зчитування, обфускації та деобфускації контрольних даних, а також перевірка узгодженості параметрів обробки.

Тестування виконується без врахування внутрішньої реалізації алгоритмів і спрямоване виключно на перевірку зовнішньої поведінки підсистеми.

У процесі функціонального тестування перевіряються наступні компоненти підсистеми:

– сервіс керування контрольними даними;

– сервіс вбудовування контрольних даних;

– сервіс витягування контрольних даних;

– репозиторій збереження контрольних даних;

– доменні сутності FPGACode, ControlData, EmbeddedControlData;

– інтерфейс сховища IControlDataStorage.

Для виконання функціонального тестування підсистеми були розроблені наступні тестові сценарії.

TC1. Перевірка збереження та зчитування контрольних даних. Метою сценарію є перевірка коректності операцій збереження та подальшого зчитування контрольних даних. У ході тесту створюється об’єкт контрольних даних, який передається до сервісу збереження. Після цього виконується операція зчитування збережених контрольних даних за відповідним стего-ключем. Результат вважається успішним, якщо зчитані контрольні дані повністю збігаються з початковими.

TC2. Перевірка створення інформаційного об’єкта програмного коду FPGA-компонента. Сценарій передбачає створення об’єкта програмного коду FPGA-компонента на основі вхідного бітового потоку. Перевіряється коректність ініціалізації внутрішньої структури об’єкта. Результатом тесту є успішне створення об’єкта та відповідність збережених даних вхідному бітовому потоку.

ТС3. Перевірка вбудовування контрольних даних. Сценарій перевіряє процес асоціювання контрольних даних із програмним кодом FPGA-компонента. Після створення об’єктів програмного коду FPGA- компонента та контрольних даних виконується операція вбудовування. Результатом є успішне формування зв’язку між програмним кодом FPGA-компонента та вбудованими контрольними даними.

ТС4. Перевірка витягування контрольних даних. У межах сценарію використовується програмний коду FPGA- компонента, який містить вбудовані контрольні дані. Після виконання операції витягування перевіряється коректність відновлених даних. Результатом є повна відповідність витягнутих даних тим, що були вбудовані раніше.

ТС5. Перевірка узгодженості параметрів. Сценарій призначений для перевірки залежності результатів операцій від параметра stegoKey. Виконується вбудовування даних із одним ключем та спроба їх витягування з іншим. Результатом є некоректне відновлення даних або відсутність результату, що підтверджує правильність механізму вбудовування.

ТС6. Перевірка обробки відсутніх даних. Сценарій передбачає спробу отримання контрольних даних при відсутності вбудованих додаткових даних в програмний код FPGA-компонента. Результатом вважається повернення контрольованої помилки без аварійного завершення роботи системи.

ТС7. Перевірка обробки неіснуючого стего-ключа. Сценарій передбачає спробу отримання контрольних даних за неіснуючим стего-ключем. Результатом вважається повернення порожнього значення або контрольованої помилки без аварійного завершення роботи системи.

В табл. 4.1 представлено протокол з результатами виконання функціонального тестування підсистеми замаскованого збереження даних в програмний код FPGA-компонентів, отриманий за результатом реалізації зазначених тестових сценаріїв.

Таблиця 4.1 – Протокол з результатами функціонального тестування підсистеми замаскованого збереження даних

Номер сценарію	Назва сценарію	Очікуваний результат	Фактичний результат	Статус
TC1	Збереження та зчитування контрольних даних	Дані ідентичні	Дані ідентичні	PASS
TC2	Створення інформаційного об’єкта програмного коду FPGA-компонента	Об’єкт створено коректно	Об’єкт створено коректно	PASS
TC3	Вбудовування контрольних даних	Дані успішно вбудовані	Дані успішно вбудовані	PASS
TC4	Витягування контрольних даних	Дані відновлені коректно	Дані відновлені коректно	PASS
TC5	Перевірка неідентичних stegoKey	Некоректний результат витягання даних	Некоректний результат витягання даних	PASS
TC6	Перевірка обробки відсутніх даних	Повернення контрольованої помилки	Повернення контрольованої помилки	PASS
TC7	Перевірка обробки неіснуючого stegoKey	Повернення контрольованої помилки	Повернення контрольованої помилки	PASS

Функціональне тестування вважається успішно виконаним, оскільки за основними та додатковими тестовими сценаріями в результаті тестування отримано статус PASS. Додаткові сценарії використовуються для перевірки стійкості системи до некоректних вхідних даних та порушення умов виконання операцій.

4.10 Висновки до четвертного розділу

В четвертому розділі дисертаційної роботи розроблено підсистему гібридного замаскованого зберігання контрольних даних в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів. Сформульовано мету та задачі функціонування зазначеної підсистеми у складі головної інформаційної системи.

Специфіковано сукупність інформаційних потоків підсистеми, яка визначає вхідні та вихідні дані, інформаційну взаємодію з іншими компонентами систем, зовнішнім програмним середовищем та користувачами. Визначено перелік функціональних та нефункціональних вимог до підсистеми. Для розроблюваного програмного забезпечення створено діаграму логічного уявлення, діаграму розгортання, діаграму класів, визначено структуру програмного проєкту та виконано аналіз програмного коду, що в сукупності формалізувало архітектуру підсистеми. Проведено комплексне тестування розробленої підсистеми. Тестування було успішно пройденим, за основними та додатковими тестовими сценаріями. В результаті обробки сукупності тестових сценаріїв отримано статус успішного проходження тестування. Це довело готовність підсистеми гібридного замаскованого зберігання контрольних даних до використання в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів.

ВИСНОВКИ

Дисертаційна робота містить нові наукові положення та отримані автором наукові результати, які в сукупності вирішують актуальну науково-прикладну задачу збільшення доступного обсягу для замаскованого зберігання контрольних даних в процесі прихованого моніторингу програмного коду FPGA-компонентів інформаційних систем шляхом розробки моделей та методів гібридного стеганографічного зберігання цих даних. Це дає змогу підвищити розрядність контрольних даних, які можуть бути замасковано збережені в програмному коді FPGA та збільшити кількости різновидів прихованого моніторингу програмного коду, контрольні дані яких зберігаються в такий спосіб. В дисертаційній роботі отримано наступні наукові та практичні результати.

1. Встановлено, що задача збільшення доступного обсягу для замаскованого зберігання контрольних даних в процесі прихованого моніторингу програмного коду FPGA-компонентів інформаційних систем є актуальною через наявний дефіцит такого обсягу при застосуванні існуючих підходів до стеганографічного зберігання контрольних даних в програмному коді FPGA-компонентів. Виявлено протиріччя між тим фактом, що програмний код FPGA-компонентів являє собою точні дані, спотворення яких не є можливим та неможливістю забезпечення достатнього для задач прихованого моніторингу обсягу зберігання даних за рахунок еквівалентних перетворень програмного коду FPGA. Виконано обґрунтування напрямку досліджень, направленого на збільшення доступного обсягу для замаскованого зберігання контрольних даних в процесі прихованого моніторингу програмного коду FPGA-компонентів інформаційних систем шляхом розробки моделей та методів гібридного стеганографічного зберігання цих даних що дає змогу розірвати вказане протиріччя.

2. Сформульовано твердження, про те, що при реалізації наближеної обробки даних на FPGA-компонентах, ця наближеність переноситься на точно поданий програмний код FPGA. Наслідком цього є можливість замаскованого стеганографічного вбудовування в нееквівалентний спосіб (подібне до того,

що використовується стосовно інформаційних контейнерів з наближено поданими елементарними одиницями) додаткових даних в точно поданий інформаційний контейнер, яким є програмний код FPGA-компонентів.

3. Для визначення множини надлишкових інформаційних ресурсів програмного коду FPGA, які виникають в процесі виконання наближених обчислень, та можуть бути використані для стеганографічного зберігання контрольних даних, розроблено модель замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, яка основана на використанні нееквівалентних перетворень програмного коду FPGA та враховує особливості виконання наближених арифметичних операцій в середовищі FPGA. Експериментальне дослідження підтвердило наявність надлишкових інформаційних ресурсів програмного коду FPGA:

– отримано верхню оцінку кількості несуттєвих блоків LUT в діапазоні 39.35 % … 42.52 % та нижню оцінку кількості зазначених блоків в діапазоні 16.67 % … 29.91 % від загальної кількості блоків LUT в проєкті;

– отримано оцінку частки несуттєвих розрядів програмних кодів блоків LUT, яка склала 16.8 % … 26.3 % від сукупної кількості розрядів програмних кодів блоків LUT у відповідних FPGA-проєктах.

4. Для збільшення обсягу, доступного для прихованого зберігання додаткових даних в середовищі програмного коду FPGA-компонентів та необхідного в задачах моніторингу характеристик безпеки програмного коду FPGA, розроблено метод нееквівалентного замаскованого зберігання даних в середовищі програмного коду FPGA-компонентів, який характеризується використанням надлишковості програмного коду FPGA, яка виникає при виконанні наближеної обробки даних. Експериментальне дослідження методу показало збільшення обсягу замаскованого зберігання даних: за верхньою оцінкою: від 12 до 145 біт при використанні одного розряду програмного коду несуттєвих блоків LUT та від 192 до 2320 біт при використанні всіх розрядів програмного коду зазначених блоків; та за нижньою оцінкою від 5 до 102 біт при використанні одного розряду програмного коду блоків LUT та від 80 до 1632 біт при використанні всіх розрядів програмного коду зазначених блоків.

5. Для збільшення обсягу контрольних даних, які можуть бути приховано вбудованими в програмний код FPGA, не збільшуючи здатність традиційних методів стегоаналізу до виявлення цих даних в програмному коді, розроблено метод гібридного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA, який відрізняється поєднанням еквівалентного та нееквівалентного підходів до перетворення програмного коду FPGA, а також поєднанням процесів стеганографічного вбудовування та відновної обфускації даних. Експериментальне дослідження запропонованого методу показало, що його застосування дозволило збільшити порівняно з традиційним підходом обсяг потенційно доступних додаткових даних, які можуть бути приховано вбудовані в програмний код FPGA в середньому від 22,6% до 34,0% за рахунок використання надлишкового ресурсу несуттєвих блоків LUT FPGA та збільшити цей обсяг в середньому від 93,7% до 105,1% за рахунок використання двох видів виявлених надлишкових ресурсів: несуттєвих блоків LUT FPGA та несуттєвих розрядів суттєвих блоків LUT FPGA.

Виконано експериментальну оцінку ймовірності виявлення відомими засобами стегоаналізу наявності даних, приховано вбудованих в програмний код FPGA, запропонованим гібридним методом. Оцінка ймовірності наявності додаткових даних при їх вбудовуванні гібридним методом без виконання етапу відновної обфускації збільшується в середньому на 2,31%, порівняно з застосуванням еквівалентного підходу, та зменшується на 6,43% в разі виконання відновної обфускації.

6. Для введення можливості балансування між обсягом частини програмного коду, яка модифікується в результаті замаскованого вбудовування даних та ефективним обсягом стеганографічного контейнера, потрібним для збереження контрольних даних, розроблено метод формування стеганографічного ключа для прихованого збереження контрольних даних в програмному коді FPGA, який відрізняється можливістю адаптації до структури зв’язків між елементарними одиницями

FPGA та до необхідного обсягу контрольних даних. Виконано експериментальне дослідження ефективності розробленого методу, які показали середнє збільшення на 22,8% ефективного обсягу цифрового водяного знака, який містить контрольні данні та може бути приховано вбудований в програмний код FPGA.

7. На базі запропонованих моделей та розроблених методів створено програмну підсистему гібридного замаскованого зберігання контрольних даних в складі інформаційної системи прихованого моніторингу програмного коду FPGA-компонентів. Використання розробленої підсистеми, а також запропонованих моделей та методів дозволило збільшити порівняно з традиційним підходом обсяг потенційно доступних додаткових даних, які можуть бути приховано вбудовані в програмний код FPGA в стеганографічний спосіб.

8. Отримані в роботі наукові результати, а також програмні засоби впроваджено в навчальний процес європейських університетів за міжнародним ERASMUS+ проєктом “ALIOT – Internet of Things: Emerging Curriculum for Industry and Human Applications” 573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP, а також в навчальний процес Національного університету «Одеська політехніка» при викладанні дисципліни “Технології захисту інформації”.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hennessy J. L., Patterson D. A., Kozyrakis C. Computer Architecture: A Quantitative Approach. 7th ed. Cambridge: Morgan Kaufmann, 2025. 936 p. ISBN: 9780443154065. URL: https://shop.elsevier.com/books/computer-architecture/hennessy/978-0-443-15406-5.

2. Roy S. Advanced Digital System Design: A Practical Guide to Verilog Based FPGA and ASIC Implementation. Cham: Switzerland: Springer Nature, 2024. 451 p. DOI: 10.1007/978-3-031-41085-7

3. Harris D., Harris S. Digital Design and Computer Architecture: RISC-V Edition. Cambridge: Morgan Kaufmann, 2021. 584 p.

4. Stallings W. Computer Organization and Architecture: Designing for Performance. 11th ed. Boston: Pearson, 2019. 864 p.

5. Patterson D. A., Hennessy J. L. Computer Organization and Design RISC-V Edition: The Hardware/Software Interface. 2nd ed. Cambridge: Morgan Kaufmann, 2020. 912 p.

6. Kirk D. B., Hwu W.-m. W. Programming Massively Parallel Processors: A Hands-on Approach. 3rd ed. Cambridge: Morgan Kaufmann, 2017. 576 p.

7. Yiu J. The Definitive Guide to ARM Cortex-M23 and Cortex-M33 Processors. Oxford: Newnes, 2020. 928 p. ISBN: 978-0128207352.

8. Maxfield C. The Design Warrior’s Guide to FPGAs: Devices, Tools and Flows. Oxford: Newnes, 2004. 544 p.

9. Cardoso J., Hübner M. Reconfigurable Computing: From FPGAs to Hardware/Software Codesign. Cham: Springer, 2014. 311 p. ISBN: 978-1489988591.

10. Tessier R., Pocek K., DeHon A. Reconfigurable Computing Architectures. *Proceedings of the IEEE*. 2015. Vol. 103, No. 3. P. 332–354.

11. Null L., Lobur J. The Essentials of Computer Organization and Architecture. 5th ed. Burlington: Jones & Bartlett Learning, 2019. 640 p.

12. Shen J. P., Lipasti M. A. Modern Processor Design: Fundamentals of Superscalar Processors. Long Grove: Waveland Press, 2016. 640 p.

13. Englander I., Wong W. The Architecture of Computer Hardware, Systems Software, and Networking: An Information Technology Approach. 6th ed. Hoboken, NJ: Wiley, 2021. 672 p. ISBN 978-1-119-49520-8.

14. Koch D., Hannig F., Ziener D. FPGAs for Software Programmers. Cham: Springer, 2016. 265 p.

15. Wijtvliet M., Corporaal H., Kumar A. Blocks, Towards Energy-efficient, Coarse-grained Reconfigurable Architectures. Cham: Springer, 2022. 237 p. URL: https://link.springer.com/book/10.1007/978-3-030-79774-4. ISBN 978-3-030-79773-7.

16. Blaiech A. G., Khalifa K., Halima M. B., Gigoux P. A survey and taxonomy of FPGA-based deep learning accelerators. *Elsevier Journal of Systems Architecture*. 2019. Vol. 98. P. 331–345. DOI: 10.1016/j.sysarc.2019.01.007.

17. Chu P. P. FPGA Prototyping by SystemVerilog Examples: Xilinx MicroBlaze MCS SoC Edition. 2nd ed. Hoboken, NJ : Wiley, 2018. 656 p. ISBN 978-1-119-28266-2.

18. Trimberger S. M. Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. *Proceedings of the IEEE*. 2015. Vol. 103, No. 3. P. 318–331.

19. Mittal S. A Survey of CPU–GPU–FPGA Heterogeneous Computing. ACM Computing Surveys. 2015. Vol. 47, No. 4. P. 1–35. DOI: https://doi.org/10.1145/2788396.

20. Putnam A. *et al*., A reconfigurable fabric for accelerating large-scale datacenter services. *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, Minneapolis, MN, USA, 2014. P. 13-24, DOI: 10.1109/ISCA.2014.6853195.

21. Bailey D. G. Design for Embedded Image Processing on FPGAs. 2nd ed. Hoboken: Wiley, 2023. 496 p.

22. Sun K., Koch M., Wang Z. et al. FPGA-Based Residual Recurrent Neural Network for Real-Time Video Super-Resolution. *IEEE Transactions on Circuits and Systems for Video Technology*. 2022. Vol. 32, № 4. P. 1739–1750. DOI: 10.1109/TCSVT.2021.3080241. URL: https://ieeexplore.ieee.org/document/9430531

23. Meyer-Baese U., Schmid A. Digital Signal Processing with Field Programmable Gate Arrays. 4th ed. Cham: Springer, 2014. 930 p.

24. Singh G., Alser M., Cali D. S. та ін. FPGA-Based Near-Memory Acceleration of Modern Data-Intensive Applications. *IEEE Micro*. 2021. Vol. 41, No. 4. P. 39–48. DOI: 10.1109/MM.2021.3088396. URL: https://ieeexplore.ieee.org/document/9451578.

25. Xilinx Inc. UltraScale Architecture GTY Transceivers User Guide. San Jose: Xilinx, 2019. 450 p.

26. Intel Corporation. P-Tile Avalon Streaming Intel FPGA IP for PCI Express User Guide. San Jose: Intel Corporation, 2022. URL: https://www.intel.com/content/www/us/en/docs/programmable/683059/22-2-8-1-0/about-the-p-tile-fpga-ips-for-pci-express.html

27. Forencich A., Snoeren A. C., Porter G., Papen G. Corundum: An Open-Source 100-Gbps NIC. *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2020. P. 38–46. DOI: 10.1109/FCCM48280.2020.00015.

28. Paar C., Pelzl J. Understanding Cryptography: From Established Symmetric and Asymmetric Ciphers to Post-Quantum Algorithms. Berlin: Springer, 2024. 543 p. ISBN: 978-3662690062.

29. Shahbazi K., Ko S. High Throughput and Area-Efficient FPGA Implementation of AES for High-Traffic Applications. *IET Computers & Digital Techniques*. 2020. Vol. 14, No. 6. P. 344–352. DOI: 10.1049/iet-cdt.2019.0179. URL: https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/iet-cdt.2019.0179.

30. Sklavos N. et al. Hardware Security and Trust. Cham: Springer, 2017. 420 p. ISBN: 978-3-319-44316-4. URL: https://link.springer.com/book/10.1007/978-3-319-44318-8.

31. Santos Junior C. E. de B. et al. SHA-256 Hardware Proposal for IoT Devices in the Blockchain Context. *MDPI Sensors*. 2024. Vol. 24, No. 12. Art. 3908. DOI: 10.3390/s24123908.

32. Martino R., Cilardo A. SHA-2 Acceleration Meeting the Needs of Emerging Applications: A Comparative Survey. *IEEE IEEE Access*. 2020. Vol. 8. P. 28415–28436. DOI: 10.1109/ACCESS.2020.2972265.

33. Kieu-Do-Nguyen B. et al. Low-Cost Area-Efficient FPGA-Based Multi-Functional ECDSA/EdDSA. *Cryptography*. 2022. Vol. 6, No. 2. Art. 25. DOI: 10.3390/cryptography6020025.

34. Ktari J., Frikha T., Hamdi M., Hamam H. Enhancing Blockchain Consensus with FPGA: Accelerating Implementation for Efficiency. *IEEE Access*. 2024. Vol. 12. P. 44773–44785. DOI: 10.1109/ACCESS.2024.3379374.

35. Pham H. L. et al. Double SHA-256 Hardware Architecture With Compact Message Expander for Bitcoin Mining. *IEEE Access*. 2020. Vol. 8. P. 171613–171624. DOI: 10.1109/ACCESS.2020.3012581.

36. Klaisoongnoen M., Brown N., Brown O. I Feel the Need for Speed: Exploiting Latest Generation FPGAs in Providing New Capabilities for High Frequency Trading. *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART '21)*. 2021. Art. 15. DOI: 10.1145/3468044.3468059.

37. Denholm S., Inoue H., Takenaka T., Becker T., Luk W. Network-Level FPGA Acceleration of Low Latency Market Data Feed Arbitration. *IEICE Transactions on Information and Systems*. 2015. Vol. E98-D, № 2. P. 288–297. DOI: 10.1587/transinf.2014RCP0011.

38. Tang Q., Su M., Jiang L., Bai X. A Scalable Architecture for Low-Latency Market-Data Processing on FPGA. *IEEE Symposium on Computers and Communication (ISCC)*. 2016. P. 1053–1058. DOI: 10.1109/ISCC.2016.7543802.

39. Puranik S., Barve M., Rodi S., Patrikar R. Acceleration of Trading System Back End with FPGAs Using High-Level Synthesis Flow. *Electronics*. 2023. Vol. 12, № 3. Art. 520. DOI: 10.3390/electronics12030520.

40. Cooke R. A., Fahmy S. A. Characterizing Latency Overheads in the Deployment of FPGA Accelerators. *Proceedings of the 30th International Conference on Field-Programmable Logic and Applications (FPL)*. 2020. P. 1–6. DOI: 10.1109/FPL50879.2020.00064.

41. Nurvitadhi E. et al. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks? *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, P. 5–14.

42. Zhang C. et al. Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks. *Proceedings of the 2015 ACM/SIGDA FPGA Symposium, 2015.* P. 161–170.

43. Jouppi N. P., Young C., Patil N. et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 2017. P. 1–12. DOI: 10.1145/3079856.3080246.

44. Guo K. et al. Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA. *IEEE Transactions on CAD*. 2018. Vol. 37(1). P. 35–47.

45. Guo K., Zeng S., Yu J. та ін. A Survey of FPGA-based Neural Network Inference Accelerators. *ACM Transactions on Reconfigurable Technology and Systems*. 2019. Vol. 12, No. 1. Art. 2. P. 1–26. DOI: 10.1145/3289185.

46. Boutros A., Betz V. FPGA Architecture: Principles and Progression. *IEEE Circuits and Systems Magazine*. 2021. Vol. 21, No. 2. P. 4–29. DOI: doi.org/10.1109/MCAS.2021.3071607.

47. Bobda C., Mbongue J. M., Ahmad H. et al. The Future of FPGA Acceleration in Datacenters and the Cloud. *ACM Transactions on Reconfigurable Technology and Systems*. 2022. Vol. 15, № 3. Art. 25. P. 1 – 42. DOI: 10.1145/3506713.

48. Vanderbauwhede W., Benkrid K. High-Performance Computing Using FPGAs. Cham: Springer, 2016. 814 p. ISBN: 978-1493943104.

49. Samayoa W. F., Crespo M. L., Cicuttin A. та ін. A Survey on FPGA-based Heterogeneous Clusters Architectures. *IEEE Access*. 2023. Vol. 11. P. 67679–67706. DOI: 10.1109/ACCESS.2023.3288431.

50. Podobas A., Sano K., Matsuoka S. A Survey on Coarse-Grained Reconfigurable Architectures from a Performance Perspective. *IEEE Access*. 2020. Vol. 8. P. 146719–146743. DOI: 10.1109/ACCESS.2020.3012084.

51. Lu Y., Liu L., Zhu J., Yin S., Wei S. Architecture, Challenges and Applications of Dynamic Reconfigurable Computing. *Journal of Semiconductors*. 2020. Vol. 41, № 2. Art. 021401. DOI: https://doi.org/10.1088/1674-4926/41/2/021401.

52. Koch D., Torresen J., Beckhoff C. *FPGA* Computing Systems: Background, Status, and Future Directions. *Proceedings of the IEEE*. 2021. Vol. 109, № 8. P. 1313–1333. DOI: https://doi.org/10.1109/JPROC.2021.3076059.

53. Mai J., Wang J., Di Z., Lin Y. Multi-Electrostatic FPGA Placement Considering SLICEL-SLICEM Heterogeneity, Clock Feasibility, and Timing Optimization. *ACM Transactions on Design Automation of Electronic Systems*. 2024. Vol. 29, № 2. Art. 24. DOI: https://doi.org/10.1145/3636531.

54. Athanas P., Pnevmatikatos D., Sklavos N. (eds.). Embedded Systems Design with FPGAs. New York: Springer, 2013. DOI: https://doi.org/10.1007/978-1-4614-1362-2.

55. Choi S., Yoo H. Fast Logic Function Extraction of LUT from Bitstream in Xilinx FPGA. *Electronics*. 2020. Vol. 9, № 7. Art. 1132. DOI: https://doi.org/10.3390/electronics9071132.

56. Murray K. E., Petelin O., Zhong S., Wang J. M., Eldafrawy M., Legault M., Sha E., Graham A. G., Wu J., Walker M. J. P., Betz V. VTR 8: High-Performance CAD and Customizable FPGA Architecture Modelling. *ACM Transactions on Reconfigurable Technology and Systems*. 2020. Vol. 13, № 2. Art. 9. P. 1–55. DOI: https://doi.org/10.1145/3388617.

57. Zohouri H. R., Maruyama N. In-Memory Computing on FPGAs Using True Dual-Port Block RAMs. *IEEE Access*. 2020. Vol. 8. P. 14001–14012. DOI: https://doi.org/10.1109/ACCESS.2020.2966382.

58. Kuon I., Rose J. Measuring the Gap Between FPGAs and ASICs // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2007. Vol. 26, № 2. P. 203–215. DOI: https://doi.org/10.1109/TCAD.2006.884574

59. Dheeraj Punia. FPGA Design, Architecture and Applications. URL: https://www.logic-fruit.com/blog/fpga/fpga-design-architecture-and-applications

60. Simpson P. A. FPGA Design: Best Practices for Team-based Reuse. 2nd ed. Cham: Springer, 2016. 257 p. DOI: https://doi.org/10.1007/978-3-319-17924-7

61. Duan S., Wang W., Luo Y., Xu X. A Survey of Recent Attacks and Mitigation on FPGA Systems. *IEEE Computer Society Annual Symposium on VLSI*. 2021. P. 284–289. DOI: https://doi.org/10.1109/ISVLSI51109.2021.00059.

62. Ashenden P., Lewis J. The Designer's Guide to VHDL. 4th ed. San Francisco: Morgan Kaufmann, 2018. 600 p.

63. Bhasker J. A Verilog HDL Primer. 3rd ed. CreateSpace Independent Publishing Platform, 2017. 420 p.

64. Bergeron J. Writing Testbenches Using SystemVerilog. 2nd ed. New York: Springer, 2016. 350 p.

65. Unterstein F., Jacob N., Hanley N., Gu C., Heyszl J. SCA Secure and Updatable Crypto Engines for FPGA SoC Bitstream Decryption: Extended Version. *Journal of Cryptographic Engineering*. 2021. Vol. 11, No. 3. P. 257–272. DOI: https://doi.org/10.1007/s13389-020-00247-2.

66. Szefer J., Tessier R. Security of FPGA-Accelerated Cloud Computing Environments. Cham: Springer International Publishing, 2023. 328 p. DOI: https://doi.org/10.1007/978-3-031-45395-3. ISBN 978-3-031-45394-6.

67. Koch D. FPGA Computing Systems: Background, Applications and Trends. Cham: Springer, 2018. 320 p.

68. Zhang T., Wang J., Guo Sh., Chen Z. A Comprehensive FPGA Reverse Engineering Tool-Chain: From Bitstream to RTL Code. *IEEE Access*. 2019. Vol. 7. P. 38379–38389. DOI: https://doi.org/10.1109/ACCESS.2019.2901949.

69. Murray K. E., Luu J., Walker M. J. P., McCullough C. та ін. Optimizing FPGA Logic Block Architectures for Arithmetic. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2020. Vol. 28, No. 6. P. 1388–1401. DOI: https://doi.org/10.1109/TVLSI.2020.2965772.

70. Haibo Wang. FPGA Logic Cells and Architecture. ECE Department. URL: https://www.engr.siu.edu/haibo/ece428/notes/ece428_logcell.pdf

71. Brown S., Vranesic Z. Fundamentals of Digital Logic with FPGA Design. 3rd ed. New York: McGraw-Hill Education, 2017. 960 p.

72. Mehta A. ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies. Cham: Springer International Publishing, 2017. 328 p. ISBN 978-3-319-59417-0.

73. Stallings W., Brown L. Computer Security: Principles and Practice. 4th ed. Boston: Pearson, 2018. 840 p.

74. Bishop M. Computer Security: Art and Science. 2nd ed. Boston: Addison-Wesley, 2018. 1136 p.

75. Ponta S. E., Plate H., Sabetta A. Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering*. 2020. Vol. 25. P. 3175–3215. DOI: https://doi.org/10.1007/s10664-020-09830-x.

76. Anderson R. Security Engineering. 3rd ed. Indianapolis: Wiley, 2020. 1232 p.

77. Taylor R. N., Medvidovic N., Dashofy E. M. Software Architecture: Foundations, Theory, and Practice. Boston: Addison-Wesley, 2016. 736 p.

78. Meng M., Khoo W. An Analysis of Secure Software Development Lifecycle from an Automotive Development Perspective. SAE Technical Paper. 2016. No. 2016-01-0040. DOI: https://doi.org/10.4271/2016-01-0040.

79. Kizza J. M. Guide to Computer Network Security. 4th ed. Cham: Springer, 2020. 600 p.

80. Ansari M. T. J., Pandey D., Alenezi M. STORE: Security Threat Oriented Requirements Engineering Methodology. Journal of King Saud University – Computer and Information Sciences. 2022. Vol. 34, № 2. P. 191–203. DOI: doi.org/10.1016/j.jksuci.2018.12.005.

81. Easttom C. Network Defense and Countermeasures: Principles and Practices. 3rd ed. Burlington: Jones & Bartlett Learning, 2022. 520 p.

82. Goodrich M., Tamassia R. Introduction to Computer Security. 2nd ed. Boston: Pearson, 2019. 600 p.

83. Jon R. Lindsay. Stuxnet revisited: From cyber warfare to secret statecraft. *Journal of Strategic Studies*. 2025. Vol. 48, no. 4. P. 834–873. DOI: doi.org/10.1080/01402390.2025.2481447

84. Langner R. Stuxnet: Dissecting a Cyberwarfare Weapon. *IEEE Security & Privacy*. 2011. Vol. 9, № 3. P. 49–60. DOI: doi.org/10.1109/MSP.2011.67.

85. Gjesvik L., Szulecki K. Interpreting cyber-energy-security events: experts, social imaginaries, and policy discourses around the 2016 Ukraine blackout. *Journal of Strategic Studies*. 2022. Vol. 46, № 1. P. 104–124. DOI: https://doi.org/10.1080/09662839.2022.2082838.

86. Cherdantseva Y., Burnap P., Blyth A., Eden P. та ін. A Review of Cyber Security Risk Assessment Methods for SCADA Systems. *Computers & Security*. 2016. Vol. 56. P. 1–27. DOI: https://doi.org/10.1016/j.cose.2015.09.009.

87. Hassanzadeh A., Rasekh A., Galelli S., Aghashahi M. та ін. A Review of Cybersecurity Incidents in the Water Sector. *Journal of Environmental Engineering*. 2020. Vol. 146, No. 5. Art. 03120001. DOI: https://doi.org/10.1061/(ASCE)EE.1943-7870.0001686.

88. Presekal A., Stefanov A., Rajkumar V. S., Semertzis I., Palensky P. Advanced Persistent Threat Kill Chain for Cyber-Physical Power Systems. *IEEE Access*. 2024. Vol. 12. P. 177746–177771. DOI: https://doi.org/10.1109/ACCESS.2024.3507386.

89. Makrakis G. M., Kolias C., Kambourakis G., Rieger C., et al. Industrial and Critical Infrastructure Security: Technical Analysis of Real-Life Security

Incidents. IEEE Access. 2021. Vol. 9. P. 165295–165325. DOI: https://doi.org/10.1109/ACCESS.2021.3133348.

90. Habinc S. Functional Safety for FPGA-Based Systems. 2nd ed. Cham: Springer, 2016. 250 p.

91. Streit F., Wituschek S., Pschyklenk M. et al. Data acquisition and control at the edge: a hardware/software-reconfigurable approach. *Production Engineering*. 2020. Vol. 14. P. 365–371. DOI: doi.org/10.1007/s11740-020-00964-x.

92. Tessier R. Reconfigurable Computing for Mission-Critical Systems. *IEEE Design & Test*. 2022. Vol. 39, No. 5. P. 15–23.

93. Mohammed N. M., Niazi M., Alshayeb M., Mahmood S. Exploring Software Security Approaches in Software Development Lifecycle: A Systematic Mapping Study. Computer Standards & Interfaces. 2017. Vol. 50. P. 107–115. DOI: doi.org/10.1016/j.csi.2016.10.001.

94. Seacord R. Secure Coding in C and C++. 2nd ed. Boston: Addison-Wesley, 2017. 600 p.

95. Ferguson N., Schneier B., Kohno T. Cryptography Engineering. Indianapolis: Wiley, 2016. 384 p.

96. Katz J., Lindell Y. Introduction to Modern Cryptography. 3rd ed. Boca Raton: CRC Press, 2020. 600 p.

97. Mead N. R., Woody C. Cyber Security Engineering: A Practical Approach for Systems and Software Assurance. Boston: Addison-Wesley Professional, 2016. 384 p. ISBN 978-0134189871.

98. Feng D. Trusted Computing: Principles and Applications. Berlin; Boston: De Gruyter, 2018. 311 p. ISBN 978-3110476040.

99. Dang Q. Secure Hash Standard (SHS). NIST FIPS PUB 180-4. 2015. 36 p.

100. Aumasson J.-P. Serious Cryptography: A Practical Introduction to Modern Encryption. San Francisco: No Starch Press, 2017. 312 p. ISBN 978-1593278267.

101. Boneh D., Shoup V. A Graduate Course in Applied Cryptography. 2020. 600 p.

102. Howard M., LeBlanc D. Writing Secure Code. 3rd ed. Redmond: Microsoft Press, 2016. 800 p.

103. Nguyen T., Reddi V. Attacks on Integrity Verification Mechanisms. *IEEE Security & Privacy*. 2022. Vol. 20, No. 2. P. 70–78.

104. Bornholt J., Kaufmann A., Li J. et al. Specifying and Checking File System Crash-Consistency Models. *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. New York: ACM, 2016. P. 83–98. DOI: doi.org/10.1145/2872362.2872406.

105. Ge C., Susilo W., Fang L. et al. Revocable Attribute-Based Encryption With Data Integrity in Clouds. Institute of Electrical and Electronics Engineers IEEE Transactions on Dependable and Secure Computing. 2022. Vol. 19, No. 5. P. 2864–2872. DOI: doi.org/10.1109/TDSC.2021.3065999.

106. He S., Xing X., Wang G., Sun Z. A Data Integrity Verification Scheme for Centralized Database Using Smart Contract and Game Theory. *IEEE Access*. 2023. Vol. 11. P. 62462–62475. DOI: doi.org/10.1109/ACCESS.2023.3284850.

107. Cappelli D., Moore A., Trzeciak R. The CERT Guide to Insider Threats. 2nd ed. Boston: Addison-Wesley, 2016. 800 p.

108. Stallings W. Cryptography and Network Security. 8th ed. Boston: Pearson, 2023. 768 p.

109. Fridrich J. Steganography in Digital Media. 2nd ed. Cambridge: Cambridge University Press, 2009. 400 p.

110. Cheddad A., Condell J., Curran K., McKevitt P. Digital Image Steganography: Survey and Analysis of Current Methods. *Signal Processing*. 2010. Vol. 90, No. 3. P. 727–752. DOI: 10.1016/j.sigpro.2009.08.010.

111. Drozd A., Drozd M., Kuznietsov M. Use of Natural LUT Redundancy to Improve Trustworthiness of FPGA Design. *CEUR Workshop Proceedings*. 2016. Vol. 1614. P. 322-331.

112. Zashcholkin K., Ivanova O. The Control Technology of Integrity and Legitimacy of LUT-Oriented Information Object Usage by Self-Recovering Digital Watermark. *CEUR-WS*. 2015. Vol. 1356. P. 486 – 497.

113. Защелкин К.В., Иванова Е.Н. Метод внедрения цифровых водяных знаков в аппаратные контейнеры с LUT-ориентированной архитектурой. *Інформатика та математичні методи в моделюванні*. Одеса, 2013. Том. 3. № 4. С. 369 – 384.

114. Cox I., Miller M., Bloom J. Digital Watermarking and Steganography. 3rd ed. Burlington: Morgan Kaufmann, 2008. 600 p.

115. Drozd A., Antoshchuk S., Drozd J., Zashcholkin K., Drozd M., Kuznietsov N., Al-Dhabi M., Nikul V. Checkable FPGA Design: Energy Consumption, Throughput and Trustworthiness. *Studies in Systems, Decision and Control*: Green IT Engineering: Social, Business and Industrial Applications. Switzerland, Cham: Springer International Publishing, 2019. Vol. 171. P. 73 – 94.

116. Subramanian N., Elharrouss O., Al-Maadeed S., Bouridane A. Image Steganography: A Review of the Recent Advances. *IEEE Access*. 2021. Vol. 9. P. 17112–17128. DOI: https://doi.org/10.1109/ACCESS.2021.3053998.

117. Mazurczyk W., Wendzel S. Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures. Wiley-IEEE Press, 2016. 296 p.

118. Дрозд О.В. Теоретичні основи, методи та засоби функціонального діагностування вузлів обчислювальних пристроїв з використанням природної надмірності при виконанні приблизних обчислень: дис. … д-ра техн. наук, Одеса, 2003. 327 с.

119. Overton M. Numerical Computing with IEEE Floating Point Arithmetic. 2nd ed. Philadelphia: SIAM, 2025. 146 p. ISBN: 978-1611978407.

120. Muller J.-M. Handbook of Floating-Point Arithmetic. 2nd ed. Cham: Springer, 2018. 600 p.

121. Защолкін К.В., Іванова О.М. Забезпечення контролю цілісності програмного коду FPGA-базованих пристроїв. *Вісник Херсонського*

національного технічного університету. Херсон, 2018. № 3 (66). Том. 1. С. 307 – 312.

122. Intel Corporation. Intel Quartus Prime Standard Edition User Guide. San Jose: Intel, 2021. 500 p.

123. Intel Corporation. Cyclone IV Device Handbook. San Jose: Intel, 2018. 600 p.

124. Welch B., Jones K., Hobbs J. Practical Programming in Tcl and Tk. 4th ed. Upper Saddle River: Prentice Hall, 2003. 832 p.

125. Антощук С.Г., Іванова О.М. Швидка локалізація осередків програмного коду FPGA, придатних для стеганографічного зберігання додаткових даних. *Вісник Херсонського національного технічного університету*. Херсон, 2025. № 4 (95) Ч. 3. С. 9 – 14. DOI https://doi.org/10.35546/kntu2078-4481.2025.4.3.1.

126. Антощук С.Г., Іванова О.М. Підхід до збільшення обсягу прихованого зберігання даних, призначених для моніторингу FPGA-компонентів комп'ютерних систем. *Вчені записки Таврійського національного університету. Серія: «Технічні науки»*. Київ, 2023. Том 34 (73), № 6. DOI: 10.32782/2663-5941/2023.6/08.

127. Zashcholkin K., Drozd O., Antoshchuk S., Ivanova O., Sachenko O. Steganographic Resources of FPGA-based Systems for Approximate Data Processing. *CEUR-WS*. 2021. Vol. 2864. P. 324-333.

128. Zashcholkin K., Drozd O., Ivanova O., Shaporin R., Kuznietsov M. An Approach to Stego-Container Organization in FPGA Systems for Approximate Data Processing. *CEUR-WS*. 2021. Vol. 2853. P. 527–536.

129. Патент на винахід № 122276 Україна, МПК G06F 11/263 (2006.01), G06F 7/544 (2006.01). Програмований пристрій для обчислення логічної функції N змінних / К.В. Защолкін, О.В. Дрозд, Р.О. Шапорін, О.М. Іванова, Ю.В. Дрозд; заявник Одеський національний політехнічний університет. – № a201811671; заявлено 27.11.2018; опубліковано 12.10.2020; Бюл. № 19/2020.

130. Іванова О.М., Дрозд О.В., Защолкін К.В. Особливості стеганографічного нееквівалентного вбудовування цифрових водяних знаків в програмний код FPGA. *Інформатика, управління та штучний інтелект, ІУШІ-2021*: тези VIII Міжнародної науково-технічної конференції. Харків, 2021.

131. Drozd O., Maevsky D., Maevskaya O., Martynyuk O., Parkhomenko A., Gladkova O., Drozd M., Ivanova O., Surkov S., Zashcholkin K. Internet of Things for Smart Building and City / Edited by D. Maevsky. Ministry of Education and Science of Ukraine, Odessa National Polytechnic University, Zaporizhzhia National Technical University, 2019. 156 p.

132. Zashcholkin K., Drozd O., Ivanova O., Sulima Y. Compositional method of FPGA program code integrity monitoring based on the usage of digital watermarks. *Applied Aspects of Information Technology*. Odessa, 2019. Vol. 2, № 02. P. 138 – 152.

133. Защолкін К.В., Іванова О.М. Інформаційна технологія вбудовування самовідновлюваних цифрових водяних знаків в LUT-орієнтовані контейнери. *Електротехнічні та комп'ютерні системи*. Київ, 2014. Вип. 16 (92). С. 78 – 84.

134. Intel Corporation. Cyclone III Device Handbook. San Jose: Intel, 2017. 550 p.

135. Aletheia: open-source tool for steganalysis. URL: https://github.com/daniellerch/aletheia.

136. Binwalk. URL: https://github.com/ReFirmLabs/binwalk

137. Sealwatch: Python package, containing implementations of modern steganalysis algorithms. URL: https://github.com/uibk-uncover/sealwatch

138. Fridrich J., Goljan, M., Du, R. Lossless Data Embedding – New Paradigm in Digital Watermarking. *EURASIP Journal on Advanced Signal Processing*. 2002. P. 185-196.

139. Goljan, M., Fridrich, J., Du, R. Distortion-Free Data Embedding for Images. *Proc. of the 4th International Workshop on Information Hiding (IHW-01)*, USA, Pittsburg, 2001. P. 27-41.

140. Zashcholkin K., Drozd O, Shaporin R, Ivanova O, Sulima Y. Increasing the Effective Volume of Digital Watermark Used in Monitoring the Program Code Integrity of FPGA-Based Systems. *IEEE East-West Design and Test Symposium (EWDTS)*. 2019. P. 53 – 58.

141. Zashcholkin K., Drozd O., Ivanova O., Shaporin R., Veselska O., Stepova H. Embedding the Digital Watermarks into FPGA-Projects Containing the Adaptive Logic Modules. *Dependable Systems, Services and Technologies (DESSERT'2019)*: Proceedings of 10th IEEE International Conference. United Kingdom, Leeds, 2019. P. 179 – 183.

142. Intel Corporation. Intel Quartus Prime Handbook. San Jose: Intel, 2022. 700 p.

143. Антощук С.Г., Іванова О.М. Метод формування стего-ключа для збільшення обсягу прихованого зберігання даних в середовищі програмного коду FPGA. *Вісник Херсонського національного технічного університету*. Херсон, 2025. № 1 (92). С. 9 – 16. DOI https://doi.org/ 10.35546/kntu2078-4481.2025.1.2.1.

144. Іванова О.М., Дрозд О.В., Защолкін К.В., Кузнєцов М.О. Підхід до нееквівалентного стеганографічного вбудовування додаткових даних в програмний код блоків LUT FPGA. *Вісник Кременчуцького національного університету імені М. Остроградського*. Кременчук, 2021. № 6 (131). С. 60–65. DOI: 10.30929/1995-0519.2021.6.60-65.

145. Ivanova O., Drozd O., Zashcholkin K., Sulima Y. Combined Use of Equivalent and Non-Equivalent Transformations of FPGA Program Code to Embedding Additional Security Data. *IEEE East-West Design and Test Symposium (EWDTS)*. 2021. P. 191 – 195. DOI: 10.1109/EWDTS52692.2021.9580984.

146. Zashcholkin K., Drozd O., Ivanova O., Bykovyy P. Formation of the Interval Stego Key for the Digital Watermark Used in Integrity Monitoring of FPGA-based Systems. *CEUR-WS*. 2020. Vol. 2623. P. 267 – 276.

147. Zashcholkin K., Drozd O., Shaporin R., Ivanova O., Drozd M. Co-Embedding Additional Security Data and Obfuscating Low-Level FPGA Program Code. *IEEE East-West Design and Test Symposium (EWDTS)*. 2020. P. 115 – 119. DOI: 10.1109/EWDTS50664.2020.9225111.

148. Антощук С.Г., Іванова О.М., Защолкін К.В. Стеганографічне вбудовування додаткових даних в програмний код мікросхем FPGA. *Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем, MEICS-2023*: матеріали VIII Всеукраїнської науково-практичної конференції. Дніпро, 2023.

149. Іванова О.М., Михайлов Д.О., Защолкін К.В. Підхід до 3D стеганографічного вбудовування даних та його програмна реалізація. *Сучасні інформаційні технології, МІТ-2021*: матеріали XI Міжнародної наукової конференції. Одеса, 2021. С. 36 – 37.

ДОДАТОК А
СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

Праці, які відображають основні наукові результати дисертації

1. Антощук С.Г., Іванова О.М. Швидка локалізація осередків програмного коду FPGA, придатних для стеганографічного зберігання додаткових даних. *Вісник Херсонського національного технічного університету*. Херсон, 2025. № 4 (95) Ч. 3. С. 9 – 14. DOI: https://doi.org/10.35546/kntu2078-4481.2025.4.3.1. *Видання включено до переліку наукових фахових видань України (категорія Б).*

https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/1265/1215

2. Антощук С.Г., Іванова О.М. Метод формування стего-ключа для збільшення обсягу прихованого зберігання даних в середовищі програмного коду FPGA. *Вісник Херсонського національного технічного університету*. Херсон, 2025. № 1 (92). С. 9 – 16. DOI: https://doi.org/10.35546/kntu2078-4481.2025.1.2.1. *Видання включено до переліку наукових фахових видань України (категорія Б).*

https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/836/803

3. Антощук С.Г., Іванова О.М. Підхід до збільшення обсягу прихованого зберігання даних, призначених для моніторингу FPGA-компонентів комп'ютерних систем. *Вчені записки Таврійського національного університету. Серія: «Технічні науки»*. Київ, 2023. Том 34 (73), № 6. DOI: https://doi.org/10.32782/2663-5941/2023.6/08. *Видання включено до переліку наукових фахових видань України (категорія Б)*.

https://www.tech.vernadskyjournals.in.ua/journals/2023/6_2023/8.pdf

4. Іванова О.М., Дрозд О.В., Защолкін К.В., Кузнєцов М.О. Підхід до нееквівалентного стеганографічного вбудовування додаткових даних в програмний код блоків LUT FPGA. *Вісник Кременчуцького національного університету імені М. Остроградського*. Кременчук, 2021. № 6 (131). С. 60–65.

DOI: https://doi.org/10.30929/1995-0519.2021.6.60-65. *Видання включено до переліку наукових фахових видань України (категорія Б).*

https://visnikkrnu.kdu.edu.ua/statti/2021_6_2021-6-60-65.pdf

5. Zashcholkin K., Drozd O., Antoshchuk S., Ivanova O., Sachenko O. Steganographic Resources of FPGA-based Systems for Approximate Data Processing. *CEUR-WS*. 2021. Vol. 2864. P. 324-333. *Наукове періодичне іноземне видання, Німеччина, ISSN 1613-0073. Видання включено до наукометричної бази* ***SCOPUS***

https://ceur-ws.org/Vol-2864/paper28.pdf

6. Zashcholkin K., Drozd O., Ivanova O., Shaporin R., Kuznietsov M. An Approach to Stego-Container Organization in FPGA Systems for Approximate Data Processing. *CEUR-WS*. 2021. Vol. 2853. P. 527–536. *Наукове періодичне іноземне видання, Німеччина, ISSN 1613-0073. Видання включено до наукометричної бази* ***SCOPUS***

https://ceur-ws.org/Vol-2853/paper55.pdf

7. Ivanova O., Drozd O., Zashcholkin K., Sulima Y. Combined Use of Equivalent and Non-Equivalent Transformations of FPGA Program Code to Embedding Additional Security Data. *IEEE East-West Design and Test Symposium (EWDTS)*. 2021. P. 191 – 195. DOI: https://doi.org/10.1109/EWDTS52692.2021.9580984. *Наукове періодичне іноземне видання, США, ISSN: 2373-826X. Видання включено до наукометричної бази* ***SCOPUS***

https://ieeexplore.ieee.org/abstract/document/9580984

8. Zashcholkin K., Drozd O., Ivanova O., Bykovyy P. Formation of the Interval Stego Key for the Digital Watermark Used in Integrity Monitoring of FPGA-based Systems. *CEUR-WS*. 2020. Vol. 2623. P. 267 – 276. *Наукове періодичне іноземне видання, Німеччина, ISSN 1613-0073. Видання включено до наукометричних баз* ***SCOPUS*** *та* ***Web of Science Core Collection***

https://ceur-ws.org/Vol-2623/paper23.pdf

9. Zashcholkin K., Drozd O., Shaporin R., Ivanova O., Drozd M. Co-Embedding Additional Security Data and Obfuscating Low-Level FPGA Program Code. *IEEE East-West Design and Test Symposium (EWDTS)*. 2020. P. 115 – 119. DOI: https://doi.org/10.1109/EWDTS50664.2020.9225111. *Наукове періодичне іноземне видання, США, ISSN: 2373-826X. Видання включено до наукометричної бази* ***SCOPUS***

https://ieeexplore.ieee.org/document/9225111

10. Патент на винахід № 122276 Україна, МПК G06F 11/263 (2006.01), G06F 7/544 (2006.01). Програмований пристрій для обчислення логічної функції N змінних / К.В. Защолкін, О.В. Дрозд, Р.О. Шапорін, О.М. Іванова, Ю.В. Дрозд; заявник Одеський національний політехнічний університет. – № a201811671; заявлено 27.11.2018; опубліковано 12.10.2020; Бюл. № 19/2020.

https://sis.nipo.gov.ua/uk/search/detail/1458192

Наукові праці апробаційного характеру

11. Антощук С.Г., Іванова О.М., Защолкін К.В. Стеганографічне вбудовування додаткових даних в програмний код мікросхем FPGA. *Перспективні напрямки сучасної електроніки, інформаційних і комп'ютерних систем, MEICS-2023*: матеріали VIII Всеукраїнської науково-практичної конференції. Дніпро, 2023.

https://www.dnu.dp.ua/docs/ndc/2023/materiali%20konf/25_MEICS-2023.pdf

12. Іванова О.М., Дрозд О.В., Защолкін К.В. Особливості стеганографічного нееквівалентного вбудовування цифрових водяних знаків в програмний код FPGA. *Інформатика, управління та штучний інтелект, ІУШІ-2021*: тези VIII Міжнародної науково-технічної конференції. Харків, 2021.

https://web.kpi.kharkov.ua/ai/wp-content/uploads/sites/249/2024/10/TEZY_YUYY_2021.pdf

13. Іванова О.М., Михайлов Д.О., Защолкін К.В. Підхід до 3D стеганографічного вбудовування даних та його програмна реалізація. *Сучасні інформаційні технології, МІТ-2021*: матеріали XI Міжнародної наукової конференції. Одеса, 2021. С. 36 – 37.

Публікації, які додатково висвітлюють результати дисертації

14. Drozd O., Maevsky D., Maevskaya O., Martynyuk O., Parkhomenko A., Gladkova O., Drozd M., Ivanova O., Surkov S., Zashcholkin K. Internet of Things for Smart Building and City. Ministry of Education and Science of Ukraine, 2019. 156 p.

https://aliot.eu.org/wp-content/uploads/2020/01/ALIOT_ITM2_IoT-for-Smart-Build-and-City_web.pdf

ДОДАТОК Б
ДОВІДКА ПРО ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ

ДОВІДКА

про впровадження результатів
дисертаційної роботи
Іванової Олени Миколаївни
на тему «Моделі та методи гібридного маскування даних у процесі моніторингу програмного коду FPGA-компонентів інформаційних систем»

Довідка видана про те, що у діяльності Товариства з обмеженою відповідальністю "ТЕХНО.КОМ" (м. Одеса) використані результати, отримані у дисертаційній роботі здобувачки кафедри Інформаційних систем Національного університету «Одеська політехніка» Іванової Олени Миколаївни.

Впровадження дістали наступні результати, отримані в дисертації:

– методи формування стеганографічного ключа та гібридного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA-компонентів інформаційних систем;

– програмна підсистема гібридного замаскованого зберігання контрольних даних яка реалізує зазначені методи, а також методика застосування програмних засобів цієї підсистеми при виконанні процедури моніторингу програмного коду FPGA-компонентів.

Результати використання запропонованих теоретичних положень та програмних засобів в межах процесів розробки інформаційних систем в складі яких присутні FPGA-компоненти в цілому підтверджують висновки дисертації Іванової О.М. Практичне застосування зазначених програмних засобів показує ефективність запропонованого в дисертації підходу до замаскованого зберігання та доступу до контрольних даних при виконанні процесів прихованого моніторингу програмного коду FPGA-компонентів.

Керівник ТОВ "ТЕХНО.КОМ" Павло СЕМЧУК

УКРАЇНА * м. ОДЕСА * ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ "ТЕХНО.КОМ+" ІДЕНТ. КОД 40157211

ДОДАТОК В

ДОВІДКА ПРО ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ У НАУКОВО-ДОСЛІДНИЦЬКІЙ ДІЯЛЬНОСТІ НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ОДЕСЬКА ПОЛІТЕХНІКА»

Україна
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

КОД ЄДРПОУ 43861328

пр.Шевченка, 1, м.Одеса, 65044
Україна
тел. +38 048 7223474
факс. +38 048 7221992
E-mail: mail@op.edu.ua

24.04.2026 № 703/61-07
на №

Довідка № 703/61-07

про використання результатів дисертаційної роботи
Іванової Олени Миколаївни
«Моделі та методи гібридного маскування даних у процесі моніторингу програмного коду FPGA-компонентів інформаційних систем»,
у науково-дослідницької діяльності
Національного університету «Одеська Політехніка»

Довідка видана в тому, що у науково-дослідницькій діяльності Національного університету «Одеська політехніка» використані наступні наукові результати, отримані у дисертаційній роботі Іванової Олени Миколаївни:

– метод гібридного замаскованого зберігання контрольних даних в середовищі програмного коду FPGA;

– метод формування стеганографічного ключа для прихованого збереження контрольних даних в програмному коді FPGA.

Дисертацію виконано згідно тематичних планів НДР Одеської політехніки за період 2016 - 2025 рр.

Іванова Олена Миколаївна була виконавцем у наступних проєктах:

– держбюджетній НДР № 99-145 (ДР №0115U000833) “Розробка методів кратного ефекту та його фокусування для проєктування та діагностики комп’ютерних інформаційних систем і їх цифрових компонентів”;

– у європейському освітньому проєкті ERASMUS+ “ALIOT” Internet of Things: Emerging Curriculum for Industry and Human Applications, 573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP.

Проректор з наукової та
науково-педагогічної роботи,
д.т.н., професор Дмитро ДМИТРИШИН

ДОДАТОК Г
ДОВІДКА ПРО ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ У НАВЧАЛЬНОМУ ПРОЦЕСІ НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ «ОДЕСЬКА ПОЛІТЕХНІКА»

Україна
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

КОД ЄДРПОУ 43861328

пр.Шевченка, 1, м.Одеса, 65044
Україна
тел. +38 048 7223474
факс. +38 048 7221992
E-mail: mail@op.edu.ua

20.12.2025 №2101/61-07
на №

ДОВІДКА

про впровадження результатів дисертаційної роботи
Іванової Олени Миколаївни
«Моделі та методи гібридного маскування даних у процесі моніторингу програмного коду FPGA-компонентів інформаційних систем»,
у освітній діяльності
Національного університету «Одеська Політехніка»

Довідка видана для підтвердження того, що в програмі та навчально-методичних матеріалах (лекціях та лабораторних роботах) з дисципліни «Технології захисту інформації», яка вивчається на кафедрі Інформаційних систем Навчально-наукового інституту комп'ютерних систем бакалаврами спеціальності 122, використовуються наукові результати, отримані у дисертації Іванової Олени Миколаївни, а саме теми «Цифровий підпис» та «Гібридне стеганографічне вбудовування даних в програмний код».

Використані результати дисертації Іванової Олени Миколаївни на тему «Моделі та методи гібридного маскування даних у процесі моніторингу програмного коду FPGA-компонентів інформаційних систем» свідчать про глибоке опрацювання наукової літератури за тематикою дисертації. Висновки та пропозиції відзначаються науковою новизною, можливістю реалізації в інформаційних системах, де є необхідність усунути можливість зловмисних маніпуляцій з програмним кодом, тому істотно поліпшують матеріал означених дисциплін і сприяють підвищенню якості підготовки фахівців в галузі комп'ютерних наук та інформаційних технологій.

Проректор з науково-педагогічної
та виховної роботи,
д.т.н., професор Сергій НЕСТЕРЕНКО

Завідувачка кафедри
інформаційних систем,
д.т.н., професор Олена АРСІРІЙ

04.12.2025 р.

ДОДАТОК Д
ФРАГМЕНТИ ВИХІДНОГО КОДУ ПІДСИСТЕМИ ГІБРИДНОГО ЗАМАСКОВАНОГО ЗБЕРІГАННЯ КОНТРОЛЬНИХ ДАНИХ

```
using FpgaDataEmbeddingSubsystem.Core.Interfaces;
using FpgaDataEmbeddingSubsystem.Core.Models;
using FpgaDataEmbeddingSubsystem.Stego;

namespace FpgaDataEmbeddingSubsystem.Services
{
public class EmbedderService : IEmbedder
{
  public Bitstream Embed(Bitstream bitstream, ControlData data,
                         Mapping map, EmbedParameters parameters)
  {

  Dictionary<int, string> newCellLutDic = Dictionary<int, string>();

  public void insertData(NodeCollections nodecoll, int[]
                         data, int[] key)
  {
    int k = 0;
    int key_position = key[0];

    for (int i = 0; i < nodecoll.fRankCells.Count; i++)
    {
      int current_cell = nodecoll.fRankCells[i];
      string current_mask=nodecoll.LUTInfo[current_cell];

      if (MaskTransform.GetBitInMask(current_mask, key_position) ==
          data[k])
          Console.WriteLine(current_cell);
      else
      {
```

```
        string invertMask =
                          MaskTransform.invertLUTMask(current_mask);
        newCellLutDic.Add(current_cell, invertMask);
        Console.WriteLine(invertMask);

        foreach (var item in nodecoll.CellInfo)
        {
          foreach (var p in item.Value)
             {
               if (p.Value == current_cell)
               {
                 string maskForRecomb = nodecoll.LUTInfo[item.Key];
                 string recombMask=null;
                 switch (p.Key)
                 {
                   case "DATAA":

recombMask=MaskTransform.recombination_InA_LUTMask(maskForRecomb);
break;

case "DATAB":
recombMask = MaskTransform.recombination_InB_LUTMask(maskForRecomb);
break;

case "DATAC":
recombMask = MaskTransform.recombination_InC_LUTMask(maskForRecomb);
break;

case "DATAD":
recombMask = MaskTransform.recombination_InD_LUTMask(maskForRecomb);
break;
}
            nodecoll.LUTInfo[item.Key] = recombMask;
            newCellLutDic[item.Key] = recombMask;
```

```
       }
     }
   }
}

  if (k < data.Length) k++;
  else break;
  }

  foreach (var item in newCellLutDic)
  {
     Console.WriteLine(item.Value);
  }
 }
}
        }
    }}
using FpgaDataEmbeddingSubsystem.Core.Interfaces;
using FpgaDataEmbeddingSubsystem.Core.Models;
using FpgaDataEmbeddingSubsystem.Stego;

namespace FpgaDataEmbeddingSubsystem.Services
{
    public class ExtractorService : IExtractor
    {
        public ControlData Extract(Bitstream bitstream, Mapping map,
ExtractParameters parameters)
        {
           int[] result = new int[nodecoll.fRankCells.Count];
           int k = 0;

     foreach (var item in nodecoll.LUTInfo)
     {
       if (nodecoll.fRankCells.Contains(item.Key))
```

```
        {
        result[k] = MaskTransform.GetBitInMask(item.Value, key);
        k++;
      }

      //Console.WriteLine("{0}: {1}", item.Key,
MaskTransform.GetBitInMask(item.Value, key));

            return result;
         }
    }
}


namespace FpgaDataEmbeddingSubsystem.Core.Models
{
    public class Bitstream
    {
        public byte[] Data { get; }

        public Bitstream(byte[] data)
        {
            Data = data;
        }
    }
}


namespace FpgaDataEmbeddingSubsystem.Core.Models
{
    public class ControlData
    {
        public byte[] Data { get; }
```

```
        public ControlData(byte[] data)
        {
            Data = data;
        }
    }
}

using System.Collections.Generic;

namespace FpgaDataEmbeddingSubsystem.Core.Models
{
    public class Mapping
    {
        public List<int> mapingRules { get; }
    }
}

namespace FpgaDataEmbeddingSubsystem.Core.Models
{
    public class EmbedParameters
    {

       public Dictionary< string, string> EmbedingStegoKey = new
                                        Dictionary<string,string>();

    }

    public class ExtractParameters
    {
        public Dictionary< string, string> ExtractingStegoKey = new
                                        Dictionary<string,string>();

    }
```

```
    public class ObfuscationParameters
    {
        public public Dictionary< string, string> ObfKey = new
                                        Dictionary<string,string>();

    }
}


namespace FpgaDataEmbeddingSubsystem.Core.Models
{
    static class MaskTransform
    {

    //returns the value located in the LUT mask mask at position pos
    public static int GetBitInMask(string mask, int pos)
    {
    string bin_mask = null;
    for (int i = 0; i < mask.Length; i++)
    {
      bin_mask = bin_mask +
      (Convert.ToString(Convert.ToInt32(mask[i].ToString(), 16),2)).
      PadLeft(4, '0'); //PadLeft - to top up 0 to 4 characters
            }

         return Convert.ToInt32(bin_mask[15-pos].ToString());
        }


   public static string invertLUTMask(string mask)
   {
     string result = null;

     for (int i = 0; i < mask.Length; i++)
```

```
      {

        int m = Convert.ToInt32(mask[i].ToString(), 16);

        m = 15 - m;
        result = result + m.ToString("X");
      }
      return result;}
         public static string recombination_InD_LUTMask(string mask)
         {
           string result = mask[2].ToString() + mask[3].ToString() +
           mask[0].ToString() + mask[1].ToString();
           return result;
         }


         public static string recombination_InC_LUTMask(string mask)
         {
           string result = mask[1].ToString() + mask[0].ToString() +
           mask[3].ToString() + mask[2].ToString();
           return result;
         }


         public static string recombination_InB_LUTMask(string mask)
         {
           string result = null;
           for (int i = 0; i < mask.Length; i++)
           {

             string tetrada = (Convert.ToString(Convert.ToInt32
(mask[i].ToString(), 16), 2)).PadLeft(4, '0');
            tetrada = tetrada[2].ToString() + tetrada[3].ToString() +
                      tetrada[0].ToString() + tetrada[1].ToString();
```

```
            result = result + tetrada;
         }
         return ((Convert.ToString(Convert.ToInt32(result, 2), 16)).
          ToUpper()).PadLeft(4, '0');
        }


        public static string recombination_InA_LUTMask(string mask)
        {
            string result = null;
            for (int i = 0; i < mask.Length; i++)
            {

            string tetrada = (Convert.ToString(Convert.ToInt32
(mask[i].ToString(), 16), 2)).PadLeft(4, '0');
            tetrada = tetrada[1].ToString() + tetrada[0].ToString()
                    + tetrada[3].ToString() + tetrada[2].ToString();

             result = result + tetrada;
        }

        return ((Convert.ToString(Convert.ToInt32(result, 2), 16)).
               ToUpper()).PadLeft(4, '0');
     }
    }
}


namespace FpgaDataEmbeddingSubsystem.Core.Models
{
    class NodeCollections
    {
    string path_cellsinfo = LUT_links_path;
```

```
string path_cells = LUT_list_path;

List<int> cellsList = new List<int>();
public List<int> fRankCells = new List<int>();

Dictionary<int, string> cellLutDic = new
                                    Dictionary<int,string>();
Dictionary<int, Dictionary<string, int>> cellPortDic = new
                  Dictionary<int, Dictionary<string, int>>();

Dictionary<string, int> port_atom_Dic = new
                                    Dictionary<string, int>();



public List<int> CellsList
{
  get
  {
    return cellsList;
  }
}

public Dictionary<int, Dictionary<string, int>> CellInfo
{
  get
  {
     return cellPortDic;
  }
 }

 public Dictionary<int, string> LUTInfo
 {
   get
   {
```

```
        return cellLutDic;
      }
    }



public void getCellsList()
{
  string str;
  StreamReader fcells = new StreamReader(path_cells);

  while ((str = fcells.ReadLine()) != null)
  {
     cellsList.Add(Convert.ToInt32(str));
  }

     fcells.Close();
 }


 public void getCellInfo()
 {
   string str;
   Char[] delimit = new Char[] { ' ', '=', ';' };
   bool first = true;
   int current_node = 0;

   StreamReader flinks = new StreamReader(path_cellsinfo);

   while ((str = flinks.ReadLine()) != null)
   {
     string[] split = str.Split(delimit);

     if (split[0] == "node" && first == true)
```

```
    {
      cellLutDic.Add(Convert.ToInt32(split[1]), split[4]);
             //split[1] – node number, split[4] - LUT mask

      current_node = Convert.ToInt32(split[1]);
    }

    if (split[0] == "node" && first == false)
    {
      cellLutDic.Add(Convert.ToInt32(split[1]), split[4]);
      first = true;

      if (checkFirstRank(port_atom_Dic) == false)

        cellPortDic.Add(current_node, new Dictionary<string,int>
                                                (port_atom_Dic));
       else
       {
         fRankCells.Add(current_node);

         current_node = Convert.ToInt32(split[1]);
         port_atom_Dic.Clear();
       } }

   if (split[0] == "port_name")
   {
     port_atom_Dic.Add(split[1], Convert.ToInt32(split[4]));
     first = false;
   }
}

   if (checkFirstRank(port_atom_Dic) == false)
   {
      cellPortDic.Add(current_node, new Dictionary<string, int>
```

```
                                                    (port_atom_Dic));
        else
        {
           fRankCells.Add(current_node);
           flinks.Close();
        }


        bool checkFirstRank(Dictionary<string, int> dic)
        {
          bool result = false;
          foreach (var item in dic)
          {
            result = result || cellsList.Contains(item.Value);
          }

            return !result;
          }

      }
}
```