МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ АЕРОНАВІГАЦІЇ, ЕЛЕКТРОНІКИ ТА ТЕЛЕКОМУНІКАЦІЙ
КАФЕДРА ЕЛЕКТРОНІКИ, РОБОТОТЕХНІКИ І ТЕХНОЛОГІЙ МОНІТОРИНГУ ТА ІНТЕРНЕТУ РЕЧЕЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Шутко В.М.
«____»_____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА
ЗІ СПЕЦІАЛЬНОСТІ 171 «ЕЛЕКТРОНІКА»
ОСВІТНЬО-ПРОФЕСІЙНОЇ ПРОГРАМИ «ЕЛЕКТРОННІ СИСТЕМИ»

Тема: «Апаратно-програмний модуль стеганографічного захисту інформації»

Виконавець
студент групи ЕС-238М _____ Полторацький Дмитро Анатолійович

Керівник
к.т.н., доцент _____ Сініцин Рустем Борисович

Консультант розділу
«Охорона праці» _____ Радомська Маргарита Мирославівна

Консультант розділу
«Охорона навколишнього
середовища» _____ Козлітін Олексій Олександрович

Нормоконтролер _____ Сініцин Р.Б.

КИЇВ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки і телекомунікацій

Кафедра електроніки, робототехніки і технологій моніторингу та інтернету речей

Напрям (спеціальність) 171 «Електроніка»
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Шутко В. М.
«_____»_____2022р.

ЗАВДАННЯ

на виконання дипломної роботи

Полторацькому Дмитру Анатолійовичу
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи: «Апаратно-програмний модуль стеганографічного захисту інформації» затверджена наказом ректора від « 09 » вересня 2022р. № 1351/ст.
2. Термін виконання роботи : з « 05 » вересня 2022р. по « 30 » листопада 2022р.
3. Вихідні дані до роботи: створення апаратно-програмної реалізації стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера.
4. Зміст пояснювальної записки: Теоретичний опис способу. Розробка програмного рішення. Розробка апаратного рішення. Результати аналізу та тестування.
5. Перелік обов’язкового ілюстративного матеріалу: Структура BMP-файлу. Згенерована псевдовипадкова послідовність. Загальний приклад таблиці перестановок. Згенерована таблиця перестановок. Таблиця розрахунків для перевірки на незвідність. Обрахунки мультиплікативних груп. Прототип інтерфейсу програми у режимі приховування інформації. Прототип інтерфейсу програми у режимі зчитування інформації. Діаграма виклику основних функцій.

Діаграма виклику основних під-функцій при приховуванні повідомлення. Блок діаграма модуля додаткової пам’яті. Блок-схема перевірки полінома на незвідність. Приклад зображення із заміненими молодшими бітами. Приклад зображення із заміненими старшими бітами. Схема генератора на базі матриці G_8. Структура GIF файлу. Структура PNG файлу. Блок-схема алгоритму наповнення контейнера. Блок-схема алгоритму зчитування контейнера. Команди для керування модулем додаткової пам’яті. Блоковий розподіл програми.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Написання заяви на написання кваліфікаційної роботи	01.09.2022	
2	Ознайомлення та обґрунтування актуальності обраної теми	02.09.2022-04.09.2022	
3	Ознайомлення з загальними положеннями про дипломне проектування та оформлення	05.09.2022-07.09.2022	
4	Формулювання мети і завдання кваліфікаційної роботи	08.09.2022-09.09.2022	
5	Бібліографічний пошук за темою кваліфікаційної роботи	10.09.2022	
6	Написання вступу та загальних відомостей	11.09.2022-19.09.2022	
7	Розробка апаратного-програмного модуля	20.09.2022-15.10.2022	
8	Аналіз та тестування апаратно-програмного модуля	16.10.2022-20.10.2022	
9	Написання розділів, базуючись на отриманих результатах дослідження	20.10.2022-06.11.2022	
10	Оформлення роботи. Подання на кафедру.	16.11.2022	

7. Консультація з окремого(мих) розділу(ів):

Назва розділу	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Теоретичний опис способу	к.т.н., доцент, Сініцин Р. Б.	05.09.2022	19.09.2022

8. Дата видачі завдання: «05» вересня 2022р.

Керівник дипломної роботи (проекту)_____ Сініцин Р. Б.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання_____ Полторацький Д. А.
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Апаратно-програмний модуль стеганографічного захисту інформації»: 119 с., 33 рис., 20 табл., 8 літературних джерел, 5 додатків.

Об’єкт дослідження: апаратно-програмний модуль стеганографічного захисту інформації.

Мета роботи: дослідження апаратно-програмного модуля стеганографічного захисту інформації.

Методи дослідження: аналіз практичного застосування апаратно-програмного модуля стеганографічного захисту інформації.

Даний спосіб дозволяє вирішувати сучасні задачі, що ставляться перед стеганографією.

СТЕГАНОГРАФІЯ, ЗАХИСТ ІНФОРМАЦІЇ, ПРИХОВАНА ПЕРЕДАЧА ДАНИХ, ПРИХОВАНЕ ЗБЕРІГАННЯ ІНФОРМАЦІЇ, РАСТРОВЕ ЗОБРАЖЕННЯ.

ЗМІСТ

ПВП – псевдовипадкова послідовність;

НП – незвідний поліном;

BMP – BitMap-формат растрових зображень;

GIF – Graphics Interchange Format;

JPEG – Joint Photographic Experts Group;

PNG – Portable Network Graphics;

TIFF – Tagged Image File Format;

CRC – контрольна сума, розрахована за допомогою циклічного надлишкового коду;

ПК – персональний комп’ютер;

LSB – Last Significant Bit, алгоритм приховування інформації;

ARM – Advanced RISC Machine;

UART – Universal Asynchronous Receiver/Transmitter;

COM – communication port;

SPI – Serial Peripheral Interface;

SD – Secure Digital;

JTAG – Joint Test Action Group;

MISRA – Motor Industry Software Reliability Association;

RTOS – Real-Time Operating System;

HTML – HyperText Markup Language;

CSS – Cascading Style Sheets;

HTTP – HyperText Transfer Protocol;

TCP – Transmission Control Protocol;

IP – Internet Protocol;

MC/DC – критерій покриття коду тестами, Modified Condition/Decision Coverage.

ВСТУП

У сучасному світі зв'язок є основною потребою кожної сфери, що розвивається. Кожен хоче зберегти таємницю та безпеку даних, якими він обмінюється. У нашому повсякденному житті ми використовуємо багато безпечних шляхів, таких як Інтернет або телефон для передачі та обміну інформацією, але вони не є безпечними на певному рівні. Для того, щоб обмінюватися інформацією у прихований спосіб, можна використовувати два методи. Це криптографія та стеганографія. У криптографії повідомлення приводиться у зашифрований вигляд за допомогою ключа шифрування, який відомий відправнику та одержувачу. Повідомлення не може бути отримано ніким без використання ключа шифрування. Як би там не було, передача зашифрованого повідомлення може без особливих зусиль стимулювати сумніви зловмисника, а закодоване повідомлення може бути захоплене, атаковане або розкодоване. З конкретною кінцевою метою подолання слабких місць криптографічних систем були створені стратегії стеганографії. Стеганографія - це майстерність і мистецтво передачі інформації таким чином, що вона приховує наявність секрету. Відповідно до цього, стеганографія приховує наявність інформації з метою, щоб ніхто не зміг визначити її суть. За допомогою стеганографії можна приховати текст, відео, зображення або навіть аудіо дані. Питання стеганографічного захисту інформації є актуальним ще з давніх часів та з грецької мови перекладається як приховане письмо. Перші згадки про приховану передачу інформації датуються V ст. до н.е. З часом технології розвивались і тягнули за собою розвиток нових методів стеганографічного захисту інформації. На сьогоднішній день стеганографія налічує кілька основних напрямків:

- класична стеганографія;
- комп’ютерна стеганографія;
- мережева стеганографія;
- цифрова стеганографія.

Даний спосіб належить до напрямку цифрової стеганографії. Цифрова стеганографія базується на прихованні секретного повідомлення у контейнері, що є цифровими даними. До таких даних можна віднести:

- текстові файли;
- растрові зображення;
- аудіо-файли;
- відео-файли.

Текстова стеганографія - це приховування інформації всередині текстових файлів. Вона включає в себе такі речі, як зміна формату існуючого тексту, зміна слів у тексті, генерування випадкових послідовностей символів або використання контекстно-вільних граматик для створення читабельних текстів.

Приховування даних шляхом використання об'єкта прикриття в якості зображення називається стеганографією зображень. У цифровій стеганографії зображення широко використовуються як контейнер, оскільки в цифровому представленні зображення присутня величезна кількість бітів. Існує багато способів приховування інформації всередині зображення. Найпоширеніші підходи включають в себе:

- заміна молодшого біта;
- накладення та фільтрування;
- надлишкове приховування.

В аудіостеганографії секретне повідомлення вбудовується в аудіосигнал, який змінює двійкову послідовність відповідного аудіофайлу. Приховування секретних повідомлень у цифровому звуці є набагато складнішим процесом у порівнянні з іншими, такими як стеганографія зображень. До різних методів аудіо стеганографії відносяться:

- заміна молодшого біта;
- кодування парності;
- фазове кодування;
- розширення спектру.

За допомогою відеостеганографії можна приховати різного роду дані в цифровий відеоформат. Перевагою цього типу є велика кількість даних, які можуть бути приховані всередині, а також той факт, що це рухомий потік зображень і звуків. Ви можете думати про це як про комбінацію стеганографії зображень та аудіо стеганографії. Два основних класи відеостеганографії включають в себе:

- вбудовування даних у нестиснене відео з подальшим стисненням;
- вбудовування даних безпосередньо в стиснений потік даних.

На сьогоднішній день є як і комерційні використання стеганографії, так і некомерційні. Стеганографію можна використовувати для збереження даних про місцезнаходження. Наприклад, кілька джерел інформації, таких як приватна банківська інформація, деякі військові таємниці, можуть бути збережені у цифрових даних. Стеганографія, також, може бути використана для нанесення водяних знаків, щоб захистити авторські права на самі цифрові дані. На сьогоднішній день захист авторських прав шляхом використання цифрових водяних знаків використовується багатьма компаніями, особливо тими, які використовують Інтернет як ринок збуту. Фірми, які публікують зображення і відео в Інтернеті, наприклад, електронні журнали, використовують стеганографію для вбудовування свого водяного знаку в матеріал таким чином, що людське око не може його побачити. Існують також компанії, які спеціалізуються на створенні бази даних ваших зображень, а потім використовують форму веб для пошуку в Інтернеті, порівнюючи зображення, які він знаходить, з тією базою даних. з цією базою даних. Таким чином вони стверджують, що можуть забезпечити захист авторських прав своїм клієнтам. Існує й класичне використання стеганографії для прихованого обміну інформацією по відкритому каналу зв’язку. Але використання стеганографії може бути й некомерційним, у такому випадку використання стеганографії буде залежати тільки від фантазії людини, яка планує її застосувати.

Спосіб, що розглядається у роботі, оперує над самим цифровим сигналом. Такий спосіб передбачає наявність мінімальних змін у контейнері. Одним із методів модифікації контейнера для приховування секретного повідомлення є метод «LSB». Цей метод полягає у заміні наймолодшого біта в байті контейнера на біт секретного

повідомлення. Нехай байт контейнера $A = 254$, тоді двійковий формат такого байту $A_{bin} = 1111\ 1110$. А також потрібно приховати ненульовий біт секретного повідомлення. Після підміни молодшого біту на біт секретного повідомлення утвориться байт $C_{bin} = 1111\ 1111$. При заміні молодшого біта ніяких візуальних змін не прослідковується. Отже, людські органи чуття не в змозі розпізнати, наприклад, різницю між пікселем, який містить стороній молодший біт. Окрім опису самого способу стеганографічного захисту в роботі розглядається створений апаратно-програмний модуль, мета створення якого полягає в експериментальному досліджені, щоб підтвердити можливість використання даного способу стеганографічного захисту інформації для вирішення відомих проблем у захисті інформації. Важливим є саме можливість апаратної реалізації, так як спектр задач, які вирішуються інтегруванням апаратного додатку стеганографічного захисту, набагато більший ніж простої програмної реалізації. Це може бути як інтеграція у канал зв'язку БПЛА для забезпечення прихованої передачі інформації, що є однією з важливих задач у сфері авіації. Чи, наприклад, інтеграція у систему цифрового фотоапарату, який зможе одразу маркувати зроблені зображення водяними знаками, які дозволять власнику фотографій підтвердити свої авторські права на тій чи іншій фотографії. Або в системі відеоспостереження розумного будинку, яка зможе маркувати відео-потік за допомогою стеганографії чим унеможливити вирізання чи підміну фрагменту відео-потоку. Прикладів використання апаратної інтеграції в електроніці, насправді, набагато більше. Щодо програмної частини, то вона також підтверджує можливість інтеграції, але програмної, у різні цифрові сервіси. А також може використовуватись у навчальних та демонстраційних цілях.

Так як стеганографія приховує сам факт, що контейнер має секретні дані, основною небезпекою є відкриття факту наявності секрету. Тоді, якщо дані не були зашифровані криптографічним алгоритмом, то разом із фактом наявності секрету отримуються і секретні дані. Якщо ж секретні дані були попередньо зашифровані криптографічним алгоритмом, то подальший захист даних буде повністю залежати від крипостійкості алгоритму, яким дані шифрувалися. Існують спеціальні методи,

що дозволяють проаналізувати контейнер на наявність повідомлення. Проста заміна молодшого біта є нестійкою до усіх, наведених нижче, типів атак:

- атака на основі відомого заповненого контейнера;
- атака на основі обраного заповненого контейнера;
- атака на основі відомого порожнього контейнера;
- атака на основі обраного порожнього контейнера;
- атака на основі відомої математичної моделі контейнера або його частини;
- атака на основі відомого прихованого повідомлення;
- атака на основі обраного прихованого повідомлення;
- адаптивна атака на основі обраного прихованого повідомлення.

Тому дана робота пропонує модифікацію методу заміни молодшого біту, яка робить спосіб стійким до атак.

Модифікація полягає у, додатковому, криптографічному захисті самого секретного повідомлення, а також у захисті самого контейнера за допомогою стохастичних перестановок. У якості контейнера використовуються растрові зображення. Для захисту контейнера від атак, пікселі зображення стохастично переставляються блоками за роздільною схемою, а повідомлення заноситься у переставлені блоки. Після повернення пікселів на місця, біти повідомлення стохастично перемішані по зображенню. Саме повідомлення шифрується перед приховуванням, за допомогою гамування байтів секретного повідомлення. Процес гамування полягає у накладанні псевдовипадкових чисел на байти відкритого тексту. Зворотний процес розшифрування отримується шляхом накладання байтів зашифрованого тексту на ту ж саму послідовність псевдовипадкових чисел. Завдяки використанню генератора псевдовипадкової послідовності на базі матриць Галуа в якості генератора псевдовипадкових чисел, що використовуються і для перестановок і для гамування, алгоритм не потребує у збережені таблиць перестановок та гама-послідовностей для подальшого використання їх у якості ключів. Ключами для алгоритму виступають три бінарні вектори, а саме: незвідний поліном, примітивний утворюючий елемент, вектор ініціалізації. Основний ключ – незвідний поліном. Саме складність підбору НП великої степені, серед всієї

множини можливих поліномів, робить алгоритм захищеним. Адже при прямому переборі або як це називають «атака в лоб» необхідно перевірити чи має поліном нетривіальні дільники, перебравши всі можливі поліноми молодшої степені. По складності цю задачу можна порівняти із задачею факторизації простих чисел, з якої відомо, що факторизація великого числа перебором при наявних у людства потужностей для обрахунку займе довгий час.

РОЗДІЛ 1
ТЕОРЕТИЧНИЙ ОПИС СПОСОБУ СТЕГАНОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ

1.1 Алгоритм шифрування секретного повідомлення

Для захисту від виявлення секретного повідомлення, стеганографічний алгоритм поєднується з криптографічним алгоритмом. Криптографічний алгоритм перетворює повідомлення у шифр за певною схемою, використовуючи спеціальний ключ, що дає змогу здійснювати обернену дію розшифрування. Для криптографічного захисту повідомлення, його байти гамуються зі стохастично отриманими байтами. Процес гамування байтів відбувається у полі GF(2) тому є нічим іншим як звичайна бінарна виняткова диз'юнкція (XOR) або додавання за модулем двійки, що описується виразом (1.1).

$$C_i = A_i \oplus B_i \tag{1.1}$$

Де C_i – результуючий байт шифрограми, A_i – байт відкритого повідомлення, B_i – псевдовипадковий байт гами. Використовуючи властивості виняткової диз'юнкції, обернена дія розшифрування являє собою таку ж саму операцію, але операндами виступають байт шифрограми та байт гами. Вираз (1.2) описує дію розшифровки.

$$A_i = C_i \oplus B_i \tag{1.2}$$

Розглянемо дуже простий приклад. Нехай повідомлення складається з чотирьох символів латинського алфавіту, утворюючи слово «Word», тоді для шифрування нам знадобиться чотири байти гами. Застосувавши формулу (1.1) ми отримаємо результат, що наведено у таблиці 1.1.1.

Таблиця 1.1.1

Результат шифрування у шістнадцятковій системі числення

№	A	B	C
1	57	1C	4B
2	6F	4B	24
3	72	85	F7
4	64	EE	8A

Отже, алгоритм шифрування секретного повідомлення є доволі простим, але теоретично є абсолютно стійким. Тобто таким, що не піддається статистичній обробці зашифрованого повідомлення, а передбачає тільки можливість прямого перебору. Але абсолютну стійкість можна отримати, якщо довжина періоду гами є більшою або еквівалентна довжині повідомлення. При гарних статистичних властивостях гами якість шифрування визначається тільки довжиною періоду гами та крипостійкістю ключа.

1.2 Алгоритм заміни молодшого біта

Алгоритм заміни молодшого біта, також відомий як LSB, є основою для даного способу стеганографічного захисту інформації. Молодший біт найменше вливає на візуалізацію байту в аудіо чи фото та відео форматах тому його заміну неможливо розпізнати за допомоги людського ока. Алгоритм LSB проілюстровано на рис. 1.2.1.

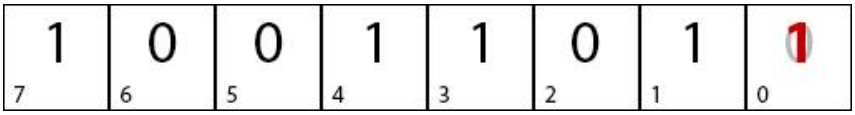


Рис. 1.2.1. Приклад заміни молодшого біту

Де молодший біт, що позначений індексом «0», змінює значення з нуля на одиницю. Аби відобразити наскільки заміна молодшого біта не несе за собою великих змін було підготовлено два ілюстративні приклади. Перший, що зображено на рис. 1.2.2, демонструє результат заміни молодшого біта. У той же час на рис. 1.2.3 зображено результат заміни старшого біта для порівняння.

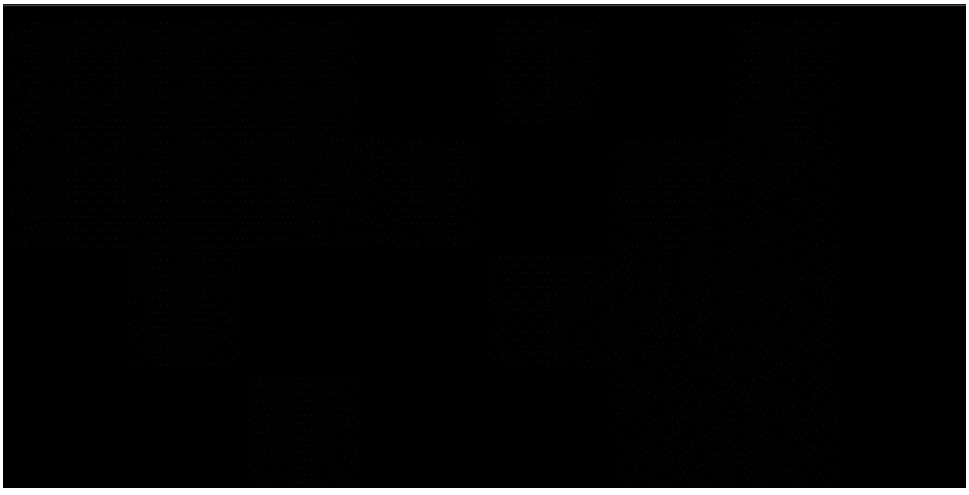


Рис. 1.2.2. Приклад зображення із заміненими молодшими бітами

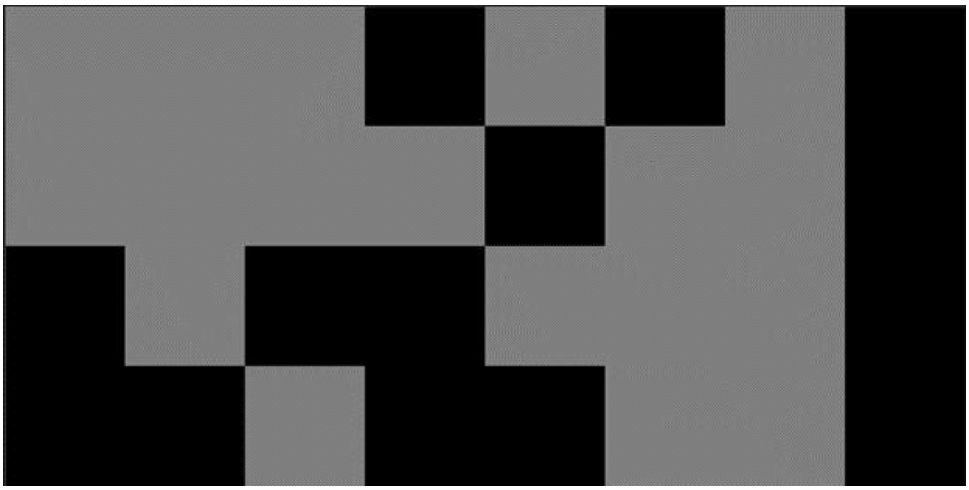


Рис. 1.2.3. Приклад зображення із заміненими старшими бітами

У якості основи для експерименту було використано растрове зображення з шириною у вісім та висотою у чотири пікселі. Оригінальне зображення містило заливку чорного кольору. У форматі зображення Grayscale, а наше зображення мало саме такий формат, кожен байт містить інформацію про відтінок сірого кольору в пікселі зображення. Зображення з прикладу містять зашифроване слово «Word», з цього і маємо відповідний розмір зображення.

Отже, порівнюючи зображення, ми бачимо, що там де був змінений з нуля на одиницю молодший біт візуальної різниці немає, а де заміні підлягав старший біт різниця помітна одразу. Реальні фото, коли це не заливка одного кольору, містять доволі великий спектр кольорів та відтінків, тому візуально розпізнати заміну молодшого біта є абсолютно неможливим. Аналогічно це працює з аудіо та відео даними.

1.3 Алгоритм стохастичного заповнення контейнера

Для уникнення можливості виявлення прихованих даних, перед наповненням, растровий контейнер шифрується шляхом стохастичної перестановки байтів блоками розміром N, який визначається виразом (1.3).

$$N = 2^k, k \in \mathbb{N} \tag{1.3}$$

У випадках, коли кількість байтів контейнера некратна розміру блока, до контейнера додаються, так звані, «баластні» байти, щоб вирівняти розміри контейнера. Але

кращим варіантом є підбор розміру зображення або розміру одного блоку таким чином, щоб уникати баластних байтів. Після повернення пікселів на свої місця, біти повідомлення будуть стохастично розподілені по зображенню. Таким чином руйнується зв'язок між бітами секретного повідомлення. Ілюстрація розподілу пікселів растрового зображення при здійсненні стохастичних перестановок зображена на рис. 1.3.1.

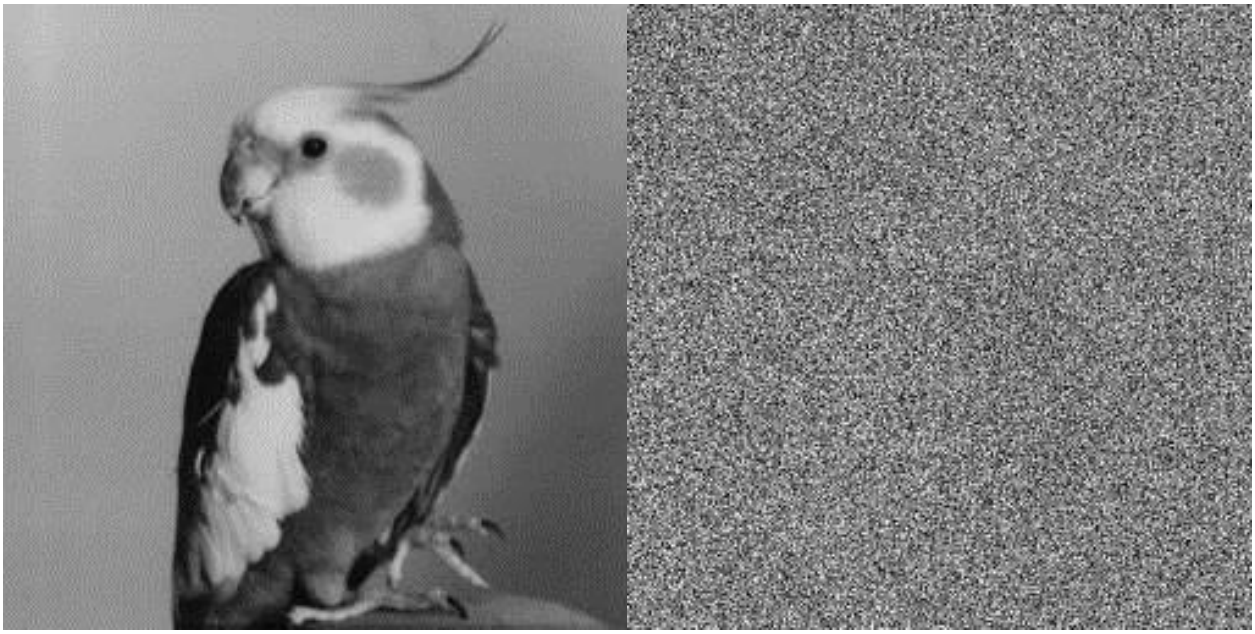


Рис. 1.3.1. Приклад зображення із стохастично переставленими пікселями

З прикладу бачимо, що при стохастичній перестановці пікселів наше зображення перетворюється на білий шум. Отже, при повернені пікселів назад, коли секретне повідомлення було приховане, то біти повідомлення будуть розподілені стохастично і представляти не зв'язану послідовність, а білий шум.

Ідея генерації перестановок зводиться до наповнення, так званої, таблиці перестановок. Таблиця перестановки зберігає відповідність індексу байту та адреси, куди його потрібно переставити. Приклад таблиці перестановок наведено в таблиці 1.3.1.

Таблиця 1.3.1.

Приклад таблиці для стохастичних перестановок пікселів

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**…**	**N – 1**
S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	…	S_{N-1}

Де адреси S_i заповнюються псевдовипадковими числами, що не містять повторень. Так як таблиця перестановок, навідміну від гами не має містити повторення, коли отримане з генератора число вже міститься у таблиці, то воно пропускається і генерується наступне, доки не заповняться всі індекси. Приклад заповненої таблиці наведено в таблці 1.3.2.

Таблиця 1.3.2.

Приклад заповненої таблиці для стохастичних перестановок пікселів

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	13	15	0	10	3	2	9	7	6	4	5	8	11	12	14

1.4 Алгоритм генерації псевдовипадкових чисел

Важливим елементом є генератор псевдовипадкових чисел. Генератор псевдовипадкових чисел – це алгоритм, що створює послідовність, кожне число якої майже незалежне одне від одного і підкорюються рівномірному розподілу. У якості генератора псевдовипадкових чисел використовується генератор псевдовипадкових послідовностей на базі матриць Галуа та Фібоначчі n-ї степені над полем $GF(2)$, а також спряжених матриць Галуа та Фібоначчі. Такі генератори ПСП базуються на лінійних регістрах зсуву з лінійним зворотнім зв'язком. Узагальнені матриці Галуа утворюються способом діагонального заповнення. Спосіб діагонального заповнення полягає у тому, що утворюючий елемент записується у правій частині нижнього рядка матриці, а всі відсутні елементи цього рядка заповнюються нулями. Всі наступні рядки формуються зсувом попереднього рядка вліво на один розряд, якщо після зсуву старший розряд не дорівнює нулю, тоді рядок приводиться до залишку по модулю незвідного полінома. Отже, зі способу формування матриці можна виділити два параметри – утворюючий елемент та незвідний поліном. Як приклад візьмемо матрицю Галуа восьмої степені G_8 , яка сформована утворюючим елементом $\varphi = 11100$ та незвідним поліномом $f_8 = 100011011$.

$$G_8 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix} \quad (1.4)$$

Генерація псевдовипадкової послідовності відбувається шляхом множення вектору ініціалізації на матрицю. Результатом такої дії є новий n-розрядний вектор, що використовується для у якості вектору ініціалізації для наступного такту генератора, а також стає частиною псевдовипадкової послідовності. Щоб генератор мав максимальний період $2^n - 1$, утворюючий елемент матриці має бути примітивним. Утворюючий елемент вважається примітивним, коли він породжує мультиплікативну групу максимального порядку. Таким чином для роботи генератора потрібні три ключі: незвідний поліном степені n, примітивний утворюючий елемент, вектор ініціалізації. Такий генератор здатний працювати у чотирьох основних конфігураціях: генератор на базі матриці Галуа, Фібоначчі, спряженої матриці Галуа, спряженої матриці Фібоначчі. Матриця Фібоначчі утворюється з матриці Галуа шляхом зворотного (правостороннього) транспонування $\perp$, що виконується відносно другорядної діагоналі матриці. Приклад розрахунку матиці Фібоначчі F_8 наведено у формулі (1.5).

$$F_8 = G_8^{\perp} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix} \quad (1.5)$$

Спряжені матриці Галуа та Фібоначчі утворюються шляхом прямого (лівостороннього) транспонування $\top$, але прямо транспонована матриця Галуа утворює спряжену матрицю Фібоначі (1.6), а прямо транспонована матриця Фібоначчі – спряжену матрицю Галуа (1.7).

$$F_8^* = G_8^\top = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (1.6)$$

$$G_8^* = F_8^\top = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (1.7)$$

Поміж основних конфігурацій генератора існують ще додаткові конфігурації, що отримуються за рахунок перетворення подібності базових матриць. Для перетворення подібності використовуються матриці перестановки, це квадратні бінарні матриці, що налічують рівно один ненульовий елемент в кожному рядку та стовпці. До того ж для матриць перестановки легко обраховуються обернені матриці. Перетворення подібності для матриці G_8 отримується за формулою (1.8).

$$\hat{G_8} = P^{-1} * G_8 * P \quad (1.8)$$

Восьма степінь матриць Галуа використовується тільки у якості прикладу, реальні генератори базуються на матрицях високих степенів: 64, 256, 1024, 2048. Саме високі степені забезпечують складність підбору ключа. Таким чином, для генерації перестановок та гами використовується одна ПСП. Послідовність, що згенерована із використанням матриці G_8 та вектору ініціалізації $IV_{G_8} = 00000001$ зображений у таблиці 1.4.1. Для зручності ПСП представлена у вигляді байт в шістнадцятковій системі числення.

Таблиця 1.4.1

Згенерована від G_8 псевдовипадкова послідовність

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	00	01	1C	4B	85	EE	C6	8B	46	09	FC	25	C1	DF	AC	BF
1	40	41	5D	16	93	7D	BB	30	76	7F	83	A6	67	B8	14	AB
2	EB	AA	F7	E1	72	0F	B4	84	F2	8D	0E	A8	CF	77	63	C8
3	23	89	7E	9F	ED	E2	56	D2	20	AD	A3	0B	C4	B3	D0	18
4	3B	B2	CC	53	BE	5C	0A	D8	F8	55	F6	FD	39	8A	5A	42
5	79	CB	07	54	EA	B6	BC	64	9C	C9	3F	C2	FB	71	2B	69
6	10	DB	DC	88	62	D4	68	0C	90	59	66	A4	5F	2E	05	6C
7	7C	A7	7B	F3	91	45	2D	21	B1	E8	8E	2A	75	5B	5E	32
8	4E	E9	92	61	F0	B5	98	B9	08	E0	6E	44	31	6A	34	06
9	48	A1	33	52	A2	17	8F	36	3E	DE	B0	F4	C5	AF	9B	9D
A	D5	74	47	15	B7	A0	2F	19	27	F9	49	BD	78	D7	4C	D1
B	04	70	37	22	95	35	1A	03	24	DD	94	29	51	86	CA	1B
C	1F	6F	58	7A	EF	DA	C0	C3	E7	3A	AE	87	D6	50	9A	81
D	9E	F1	A9	D3	3C	E6	26	E5	02	38	96	11	C7	97	0D	8C
E	12	E3	4A	99	A5	43	65	80	82	BA	2C	3D	FA	6D	60	EC
F	FE	1D	57	CE	6B	28	4D	CD	4F	F5	D9	E4	1E	73	13	FF

При використані матриці більшої степені на виході утворюється вектор, більший за довжину одного байту. Тоді вихідний вектор ділиться на частини по вісім біт, які додаються за модулем двійки, утворюючи один байт.

Такий генератор відноситься до лінійних регістрів зсуву з лінійним зворотнім зв’язком (LFSR). LFSR генератори мають легку реалізацію як програмно, так і апаратно. Схема генератора зображена на рис. 1.4.1.

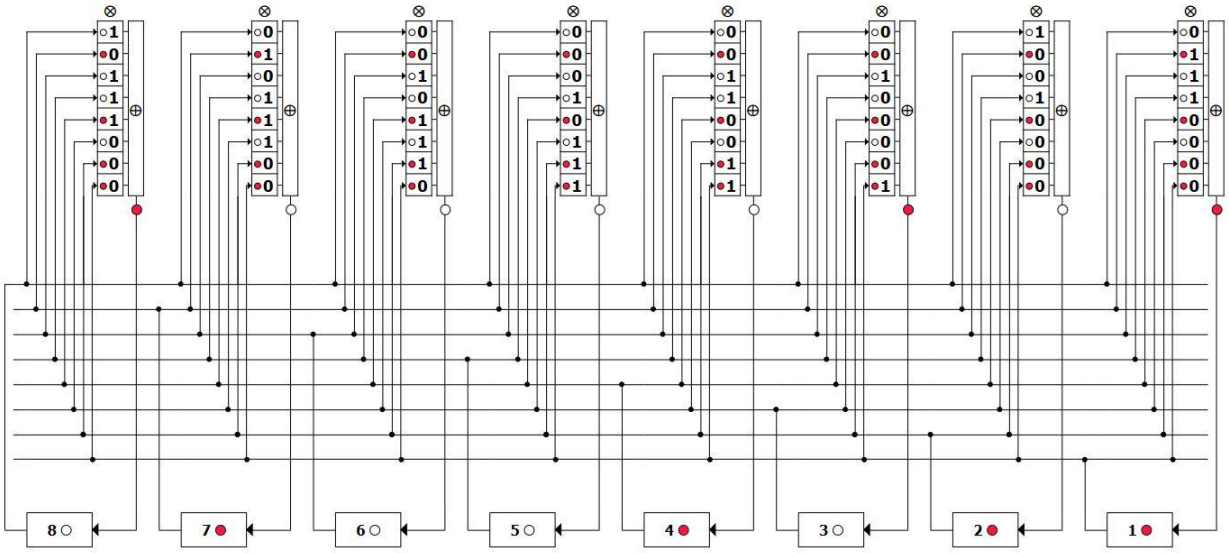


Рис. 1.4.1. Схема генератора на базі матриці G_8

На схемі зображений генератор восьмого порядку. На кожному такті генератора утворюється вектор відповідної довжини, який також використовується для отримання нового вектору, активуючи потрібні верхні регістри, на схемі він записується у нижні регістри. Верхні регістри представляють матрицю Галуа, Фібоначчі або спряжену їм, яка утворена незвідним поліномом та примітивним утворюючим елементом. Верхні вертикальні регістри поділяються на два типи. Регістри першого типу виконують дію порозрядного множення по модулю двійки, такі регістри позначені символом ⊗. А регістри другого типу виконують операцію порозрядного додавання по модулю двійки, вони позначені символом ⊕.

Генератори на базі лінійних регістрів зсуву з лінійним зворотнім зв'язком здатні утворювати псевдовипадкову послідовність великого періоду з гарними статистичними властивостями. Але у таких генераторів є один недолік, який полягає у тому, що, теоретично, існує можливість знайти поліном, яким утворюється послідовність. Відомий алгоритм Берлекемпа-Мессі дає змогу знайти мінімальний поліном, що подається на вхід лінійної рекурентної послідовності над довільним полем. Після проведеного аналізу можливості знаходження полінома, за допомогою якого генерувалась псевдовипадкова послідовність. Було виявлено емпіричним шляхом, що результат алгоритму Берлекемпа-Мессі є завжди примітивним поліномом. А це означає, що для кожного примітивного полінома існує сімейство слабких ключів, примітивних утворюючих елементів. Приклад для примітивних

поліномів восьмої степені наведено у таблиці 1.4.2. Всі слабкі ключі приведені у вісімковій системі числення.

Таблиця 1.4.2

Таблиця слабких ключів для примітивних поліномів восьмої степені

№	ПрП	1	2	3	4	5	6	7	8
1	100011101	002	004	020	035	114	137	205	235
2	100101011	002	004	020	053	301	351	356	373
3	100101101	002	004	020	055	135	165	345	366
4	101001101	002	004	020	115	253	343	353	370
5	101011111	002	004	020	137	206	225	232	300
6	101100011	002	004	020	143	223	341	363	364
7	101100101	002	004	020	145	213	214	231	355
8	101101001	002	003	004	005	020	021	150	151
9	101110001	002	004	020	057	133	161	340	363
10	110000111	002	004	020	035	064	157	207	326
11	110001101	002	004	020	075	140	177	215	270
12	110101001	002	004	020	030	202	251	315	351
13	111000011	002	004	020	037	226	240	303	375
14	111001111	002	004	020	126	130	166	240	317
15	111100111	002	004	020	054	113	175	347	352
16	111110101	002	004	020	050	147	176	323	365

Отже, слабкі ключі мають тільки ті незвідні поліноми, що являються примітивними, а це означає, що при використанні незвідного полінома, який не є примітивним, отримати цей його через алгоритм Берлекемпа-Мессі неможливо. Таке твердження дає можливість використання генератора псевдовипадкової

послідовності, коли незвідний поліном не є примітивним. Тільки у такому випадку можна отримати максимальну крипто-стійкість алгоритму.

1.5 Алгоритм генерації ключа для стеганогфічного захисту

Для генератора псевдовипадкової послідовності на базі класичних та спряжених матриць Галуа, Фібоначі використовується три ключі: незвідний поліном, примітивний утворюючий елемент та вектор ініціалізації. Цей підрозділ присвячений опису ключів, а також алгоритмів для їх створення. Ключ криптографічного алгоритму – це певна кількість цифрових даних, що тримається у секреті та використовується алгоритмом для операції шифрування, а також зворотної дії розшифрування. Алгоритм може використовувати як один ключ, так і набір ключів. Ключ може бути будь-яким типом цифрової інформації, бо незалежно від типу цифрової інформації її можна представити у вигляді послідовності бітів. Згідно принципу Керхгоффса, ключі обираються такими, щоб складність підбору ключа була теоретично дуже високою, а практично, неможливою за короткий проміжок часу. Таким чином сам криптографічний алгоритм не потребує тримання його у секреті, а є відкритим.

Головний ключ до нашого алгоритму це незвідний поліном степені n. Незвідний поліном – це поліном, що має тільки тривіальні дільники. Тому незвідні поліноми є подібними до простих чисел. Синтез незвідних поліномів відбувається за допомогою алгоритму синтезу незвідних поліномів лінійної складності. Такий алгоритм дає змогу виявити незвідність полінома за лінійну складність, в той час як класичні методи синтезу незвідних поліномів мають квадратичну складність. Лінійна складність алгоритму синтезу не тягне за собою стрімке збільшення часових витрат при збільшені степені, на відміну від квадратичної складності. Степінь полінома є одним з основних параметрів полінома, вона дорівнює максимальній степені старшого ненульового монома. Реальний генератор використовує поліноми високих степенів, де степінь n береться з виразу (1.9).

$$n = 2^k \tag{1.9}$$

Приклад полінома восьмої степені наведено у виразі (1.10).

$$f_8(x) = x^8 + x^4 + x^3 + x^1 + 1 \tag{1.10}$$

Поліном з виразу (1.10) наведений у алгебраїчній формі, але існує, також, і векторна форма представлення поліномів, яка в нашому випадку буде зручнішою, так як ми будемо мати справу тільки з бінарними поліномами. Бінарні поліноми можуть мати коефіцієнти, що дорівнюють тільки нулю або одиниці. Векторна форма поліному (1.10) наведена у виразі (1.11).

$$f_8 = 100011011 \tag{1.11}$$

Ще одним важливим параметром полінома є його період. Період полінома є найменшим натуральним числом m, при якому поліном є дільником двочлена (1.12).

$$x^m - 1 \tag{1.12}$$

Період незвідного полінома дорівнює періоду елемента φ=10 поля GF(2^n), яке утворене даним поліномом. Ця властивість є критерієм перевірки незвідного полінома на примітивність, бо незвідний поліном є примітивним, коли його період є максимальним.

Отже, розібравши основні параметри незвідних поліномів, перейдемо до опису алгоритму синтезу. Спочатку потрібно обрати параметри полінома, який будемо генерувати, а саме степінь полінома та вага полінома. Вага полінома – це кількість ненульових коефіцієнтів цього полінома. Вага незвідного полінома завжди є непарним числом. Алгоритм генерації незвідного полінома зводиться до наступних кроків:

1. Обрання початкового полінома для перевірки, такий поліном може бути або підібраний вручну, або за допомогою стохастичного моделювання. Обов’язково коефіцієнти старшого та молодшого мономів мають дорівнювати одиниці, а також вага полінома має бути непарним числом.
2. Перевірка обраного полінома на незвідність за допомогою алгоритму синтезу незвідних поліномів лінійної складності.
3. У разі підтвердження незвідності на попередньому кроці, протестований поліном можна використовувати, бо він є незвідним, у іншому випадку, обраховується найближчий можливий поліном та перевіряється на незвідність.

Процес вибору початкового полінома є нескладним, якщо поліном обирається вручну, то достатньо сформувати вектор потрібної довжини, молодший та старший коефіцієнти якого дорівнюють одиниці, а інші ненульові коефіцієнти розташовані у довільному порядку. Формування полінома шляхом стохастичного моделювання є подібним до ручного способу, окрім того, що ненульові коефіцієнти розподіляються за допомогою програмного алгоритму. Початковий поліном потрібен для ініціалізації процесу синтезу незвідного полінома, так звана відправна точка. Коли початковий поліном сформований, він підлягає перевірці на незвідність. В основі алгоритму перевірки лежать прості рекурентні обрахунки залишків координатного вектору по модулю полінома, що тестується. Якщо залишок на ітерації n дорівнює одиниці, тобто виконується умова (1.13), то поліном вважається незвідним. Однак зазначимо, що у разі отримання одиниці на будь-якій попередній ітерації, поліном незвідним не вважається.

$$S_k = ((S_{k-1})^2 \ll 1)\ mod\ f_k \equiv 1 \tag{1.13}$$

Початковий координатний вектор S_0 завжди дорівнює одиниці. Кожен наступний вектор є залишком піднесеного до квадрату попереднього вектору, що доповнений нулем, по модулю полінома. У таблиці 1.5.1 наведено приклад перевірки незвідного полінома $f_8 = 100011011$, який ми вже використовували в попередньому підрозділі для побудови матриці Галуа.

Таблиця 1.5.1

Перевірка полінома на незвідність

№	$f_8 = 100011011$
0	$S_0 = 1$
1	$S_1 = ((S_0)^2 \ll 1)\ mod\ f_8 = (1^2 \ll 1)\ mod\ 100011011 = 10$
2	$S_2 = ((S_1)^2 \ll 1)\ mod\ f_8 = (10^2 \ll 1)\ mod\ 100011011 = 1000$
3	$S_3 = ((S_2)^2 \ll 1)\ mod\ f_8 = (1000^2 \ll 1)\ mod\ 100011011 = 10000000$
4	$S_4 = ((S_3)^2 \ll 1)\ mod\ f_8 = (10000000^2 \ll 1)\ mod\ 100011011 = 101111$
5	$S_5 = ((S_4)^2 \ll 1)\ mod\ f_8 = (101111^2 \ll 1)\ mod\ 100011011 = 1110010$
6	$S_6 = ((S_5)^2 \ll 1)\ mod\ f_8 = (1110010^2 \ll 1)\ mod\ 100011011 = 10101011$
7	$S_7 = ((S_6)^2 \ll 1)\ mod\ f_8 = (10101011^2 \ll 1)\ mod\ 100011011 = 1111101$
8	$S_8 = ((S_7)^2 \ll 1)\ mod\ f_8 = (1111101^2 \ll 1)\ mod\ 100011011 = 1$

Блок-схема алгоритму тестування полінома на незвідність зображена на рис. 1.5.1.

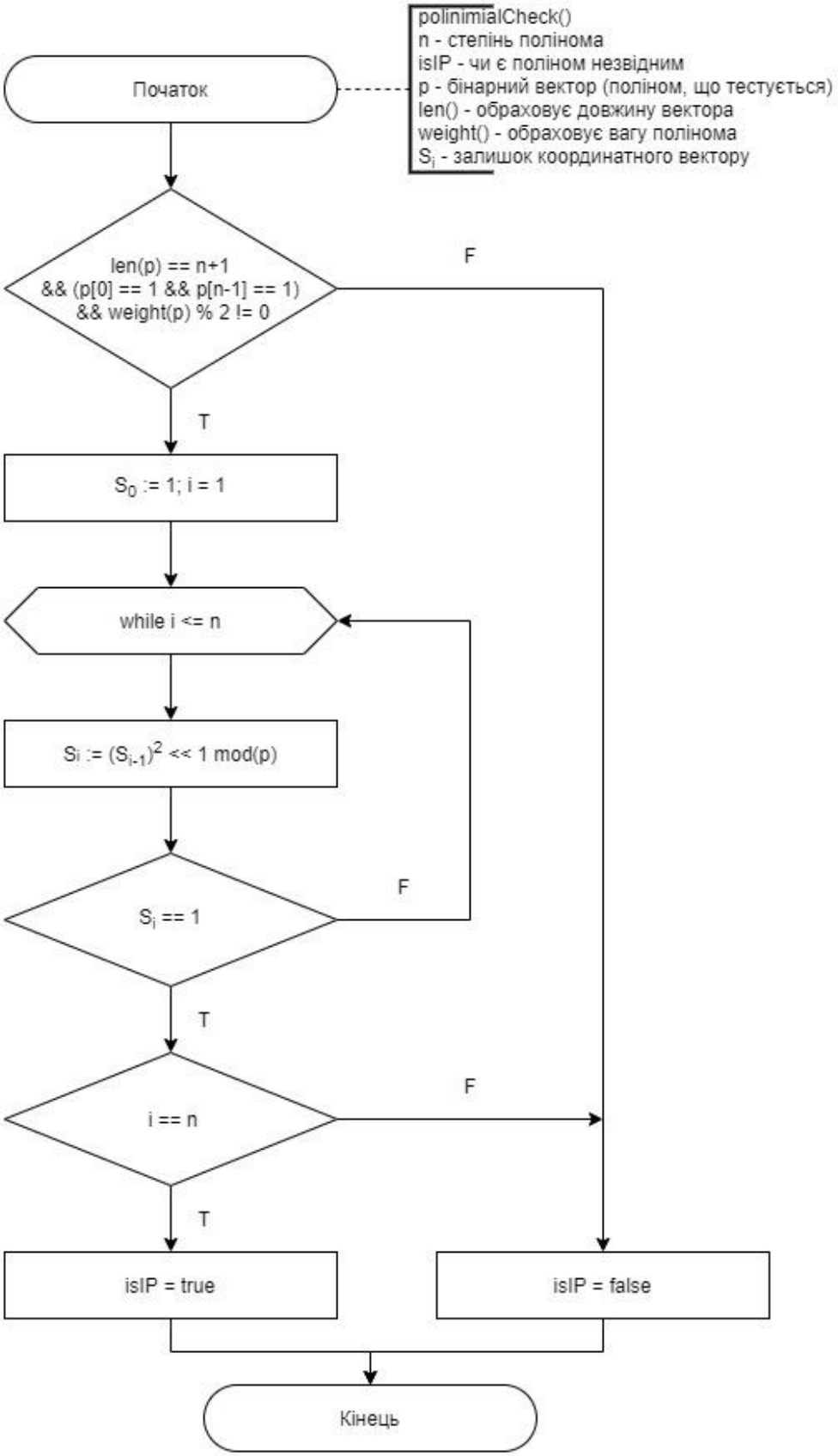


Рис. 1.5.1. Блок-схема перевірки полінома на незвідність

Другорядним ключем алгоритму є примітивний утворюючий елемент. Для генерації псевдовипадкової послідовності максимального періоду утворюючий елемент має бути, обов'язково, примітивним. Утворюючий елемент – це елемент, що утворює деяку множину елементів, базуючись на певній асоціативній арифметичній дії. Таку множину називають групою, якщо в ній існує нейтральний елемент, а також кожен елемент цієї множини має обернений йому елемент, що також належить даній множині. Група може налічувати скінченну кількість елементів, така група називається скінченною, або мати нескінченну кількість елементів. Кожна скінченна група має певну кількість елементів, що називається періодом і є дуже важливою характеристикою. Утворюючий елемент, що може бути використаний у якості ключа утворює скінченну мультиплікативну групу G в полі $GF(2^n)$, утвореним незвідним поліномом f_n. Період $|G|$ мультиплікативної групи G повинен відповідати співвідношенню (1.14).

$$|G| = 2^n - 1 \tag{1.14}$$

Саме співвідношенням (1.14) перевіряється примітивність утворюючого елемента. Для прикладу розглянемо мультиплікативні групи в полі $GF(2^4)$, що утворені елементами $\varphi = 1101$ та $\varphi = 111$ відповідно. У якості незвідного полінома використовується поліном $f_4 = 10011$.

Таблиця 1.5.2

Перевірка утворюючого елемента на примітивність

№	$\boldsymbol{\varphi = 1101}$	$\boldsymbol{\varphi = 111}$
0	$1101^0 = 0001$	$111^0 = 0001$
1	$1101^1 = 1101$	$111^1 = 0111$
2	$1101^2 = 1110$	$111^2 = 0110$
3	$1101^3 = 1010$	$111^3 = 0001$
4	$1101^4 = 1011$	-
5	$1101^5 = 0110$	-
6	$1101^6 = 1000$	-
7	$1101^7 = 0010$	-
8	$1101^8 = 1001$	-
9	$1101^9 = 1111$	-
10	$1101^{10} = 0111$	-
11	$1101^{11} = 0101$	-
12	$1101^{12} = 1100$	-
13	$1101^{13} = 0011$	-
14	$1101^{14} = 0100$	-
15	$1101^{15} = 0001$	-

З прикладу, наведеного у таблиці 1.5.2, бачимо, що елемент $\varphi = 1101$ є примітивним, бо утворює мультиплікативну групу максимального періоду. А елемент $\varphi = 111$, навпаки, не є примітивним.

Отже, алгоритм генерації примітивного утворюючого елемента може бути зведений до наступних кроків:

1. Формування бінарного вектору довжиною n, який і є утворюючим елементом для перевірки на примітивність.
2. Перевірка утворюючого елементу на примітивність шляхом побудови мультиплікативної групи та оцінки її періоду.
3. Збільшення початкового вектору на одиницю, якщо примітивність не підтверджена, у іншому випадку цей крок пропускається.

Формування початкового утворюючого елементу (вектору) може відбуватись як вручну, так і за допомогою стохастичного моделювання.

Третім ключем є вектор ініціалізації, його генерація є найпростішою, адже у ролі вектору ініціалізації може виступати будь-який бінарний вектор довжиною n, окрім нульового вектору. Вектор може бути сформований або вручну, або за допомогою стохастичного моделювання.

РОЗДІЛ 2
ЗАСТОСУВАННЯ АЛГОИТМУ В СТЕГАНОГРАФІЇ ЗОБРАЖЕНЬ

2.1 Приклад заповнення контейнера

Як вже зазначалося, у якості контейнера для стеганографічного захисту можна використовувати різні типи цифрових медіа даних, таких як відео, аудіо, та растрові зображення. У прикладі використано растрове зображення. Такий тип контейнеру був обраний з наступних причин: растрове зображення найкраще візуалізує сам контейнер для читача; розроблений програмно-апаратний модуль був створений першочергово з підтримкою стеганографічного захисту в растрових зображеннях. Перш ніж перейти до алгоритму заповнення контейнера, необхідно описати формати растрових зображень, які підходять для використання. Серед найбільш поширених форматів растрових зображень можна перелічити наступні:

- Bitmap Picture (BMP);
- Joint Photographic Experts Group (JPEG);
- Graphics Interchange Format (GIF);
- Portable Network Graphics (PNG);
- Tagged Image File Format (TIFF).

Бітове зображення BMP є найбільш простішим для використання у стеганографії, тому не дивно, що в багатьох прикладах захисту інформації використовується саме цей формат. Простота використання полягає у відсутності стиснення. Точніше сам формат підтримує стиснення, але стиснення BMP змушує бажати кращого. Тому найчастіше зображення у форматі BMP представлені без стиснення даних. Кожен піксель зображення має свій колір зі стандартної палітри операційної системи або палітра може бути задана індивідуально для зображення. Пікселі зображення можуть бути описані різними типами кодування, що залежить від кількості біт, які виділяються на один піксель. Напряму від кількості біт на піксель залежить спектр кольорів, що може приймати один піксель. Структура файлу BMP складається з чотирьох частин: заголовок файлу (BitMap File Header), інформаційний заголовок (BitMap Info Header), палітра кольорів (Color Table), дані зображення (BitMap Array). Детальна структура контейнера наведена у таблиці 2.1.1.

Таблиця 2.1.1

Структура BMP файлу

Параметр	**Розмір**	**Зміщення**
Слово «BM», яке вказує на BMP формат файлу	2	0
Загальний розмір файлу в байтах	4	2
Зарезервовані нульові байти	4	6
Кількість сервісних байт	4	10
Розмір заголовку в байтах	4	14
Ширина зображення в пікселях	4	18
Висота зображення в пікселях	4	22
Кількість вимірів	2	26
Кількість біт на піксель	2	28
Тип компресії	4	30
Кількість байт в масиві зображення	4	34
Кількість пікселів на метр по осі X	4	38
Кількість пікселів на метр по осі Y	4	42
Кількість використаних кольорів	4	46
Кількість основних кольорів	4	50
Палітра кольорів (якщо використовується)	-	54
Масив зображення	-	54 + розмір палітри кольорів

JPEG - це стандартизований механізм стиснення зображень. JPEG походить від оригінальної назви комітету, який розробив цей стандарт. JPEG стискає як повнокольорові, так і напівтонові зображення, і найкраще працює з фотографіями та ілюстраціями. Для геометричних лінійних малюнків, написів, карикатур, знімків екрану комп'ютера та інших зображень з рівними кольорами і чіткими межами

зазвичай краще використовувати формати PNG і GIF. JPEG використовує метод стиснення з втратами, що означає, що декомпресоване зображення не зовсім схоже на оригінал. Так як алгоритм стиснення буде пошкоджувати приховані дані, то формат JPEG не підходить для використання у стеганографії. Звісно, можна модифікувати алгоритм заповнення контейнера так, щоб інформація була прихована у тих бітах, які не підпадають під стиснення, але така модифікація збільшує складність алгоритму та зменшує його швидкодію.

Формат GIF є розробкою компанії CompuServe і використовується для зберігання декількох растрових зображень в одному файлі для обміну між платформами та системами. Переважна більшість GIF-файлів містять 16- або 256-кольорові зображення майже фотографічної якості. GIF відрізняється від багатьох інших поширених форматів растрових зображень тим, що він є потоковим. Він складається з серії пакетів даних, які називаються блоками, разом з додатковою інформацією протоколу. Через таку організацію GIF-файли повинні читатися так, ніби вони є безперервним потоком даних. Блоки, крім зберігання полів інформації, можуть також містити підблоки. Кожен підблок даних починається з одного лічильного байта, який може бути в діапазоні значень одного байта і вказує на кількість байтів даних, які слідують за лічильним байтом. Послідовність одного або кількох підблоків даних завершується лічильним байтом зі значенням нуль. Формат GIF здатний зберігати растрові дані з глибиною пікселя від одного до восьми біт. Зображення завжди зберігаються з використанням колірної моделі RGB і даних палітри. GIF також здатний зберігати декілька зображень в одному файлі, але ця можливість використовується рідко, і переважна більшість GIF-файлів містить лише одне зображення. Дані зображення, що зберігаються у файлі GIF, завжди стискаються у форматі LZW. Цей алгоритм зводить рядки однакових значень байтів в одне кодове слово і здатний зменшити розмір типових восьми бітних піксельних даних на сорок і більше відсотків. Структура GIF файлу наведена у таблиці 2.1.2.

Таблиця 2.1.2

Структура GIF файлу

Параметр	**Розмір**	**Зміщення**
Підпис заголовку «GIF»	3	0
Версія формату, «87a» чи «89a»	3	3
Значення ширини у пікселях	2	6
Значення висоти у пікселях	2	8
Інформація про кольорову карту	1	10
Колір фону	1	11
Співвідношення сторін у пікселях	1	12
Блоки із даними зображення	N	13
Завершальний байт «0x3B»	1	13 + розмір зображення

Заголовок має розмір шість байт і використовується тільки для ідентифікації файлу як типу GIF. Логічний дескриптор екрану, який може бути окремим від фактичного заголовка файлу, може розглядатися як другий заголовок. Підпис має довжину три байти і містить символи «GIF» як ідентифікатор. Всі GIF-файли починаються з цих трьох байт, і будь-який файл, що не містить їх, не повинен зчитуватися програмою як файл GIF-зображення. Версія також має довжину три байти і містить версію GIF-файлу. Наразі існує лише дві версії GIF: 87a та 89a.

Логічний дескриптор екрану містить інформацію, що описує екран і колірну інформацію, яка використовується для створення і відображення зображення GIF-файлу. Дескриптор містить мінімальну роздільну здатність екрану, необхідну для відображення даних зображення. Якщо пристрій відображення не здатний підтримувати вказану роздільну здатність, то для коректного відображення зображення буде необхідне певне масштабування. Інформація про кольорову карту має побітове кодування, яке наведено у таблиці 2.1.3.

Таблиця 2.1.3

Таблиця кодування кольорової карти у GIF зображені

Індекс біту	**Опис**
0	Розмір глобальної таблиці кольорів
1	
2	
3	Індикатор сортування кольорової таблиці
4	Роздільна здатність кольору
5	
6	
7	Індикатор глобальної таблиці кольорів

Логічний дескриптор екрану може супроводжуватися необов'язковою глобальною таблицею кольорів. Ця таблиця кольорів, якщо вона присутня, є колірною картою, яка використовується для індексації даних про колір пікселів, що містяться в даних зображення. Якщо глобальна колірна таблиця відсутня, кожне зображення, що зберігається у файлі GIF, містить локальну колірну таблицю, яка використовується замість глобальної колірної таблиці. Якщо кожне зображення у файлі GIF використовує власну локальну таблицю кольорів, то глобальна таблиця кольорів може бути відсутня у файлі GIF. Якщо ні глобальної, ні локальної таблиці кольорів немає, то буде використовуватися таблиця кольорів за змовчуванням.

Дані зображення у форматі GIF завжди зберігаються по рядках розгортки і по пікселях. GIF не має можливості зберігати дані зображення у вигляді площини, тому, коли GIF-файли відображаються за допомогою адаптерів відображення, орієнтованих на площину, перед відображенням GIF-зображення необхідно виконати значну буферизацію, зсув і маскування даних зображення. Рядки розгортки, що складають дані растрового зображення GIF, зазвичай зберігаються в послідовному порядку, починаючи з першого рядка і закінчуючи останнім. Формат

GIF також підтримує альтернативний спосіб зберігання рядків растрових даних у чергуванні. Зображення з чергуванням зберігаються у вигляді рядів растрових даних, що чергуються. GIF використовує чотирьох прохідну схему переплетення. Перший прохід починається з нульового рядка і зчитує кожен восьмий рядок растрових даних. Другий прохід починається з четвертого рядка і зчитує кожен восьмий рядок даних. Третій прохід починається з другого рядка і зчитує кожен четвертий рядок. Останній прохід починається з першого рядка і зчитує кожен другий рядок. За такою схемою зчитуються і зберігаються всі рядки растрових даних.

PNG - це формат файлів растрових зображень, який використовує стиснення без втрат. Цей формат був створений як заміна формату Graphics Interchange Format (GIF) і не має обмежень щодо авторських прав. Однак формат PNG не підтримує анімацію. Формат PNG підтримує стиснення зображень без втрат, що робить його популярним серед користувачів. PNG-файл складається з восьми байтового заголовка, за яким слідує довільна кількість фрагментів, що містять керуючі дані, метадані та дані зображення. Кожена частина містить три стандартних поля: розмір, тип, CRC та різні внутрішні поля, які залежать від типу частини. Структура файлу PNG наведено у таблиці 2.1.4.

Таблиця 2.1.4

Структура PNG файлу

Параметр	**Розмір**	**Зміщення**
Підпис заголовку «PNG»	8	0
Розмір даних розділу	4	8
Підпис розділу «IHDR»	4	12
Ширина у пікселях	4	16
Висота у пікселях	4	20
Глибина кольору	1	24
Тип кольору	1	25
Метод компресії	1	26
Метод фільтрування	1	27
Метод суміщення	1	28
Контрольна сума розділу	4	29
Розмір даних розділу	4	33
Підпис розділу «IDAT»	4	37
Дані зображення	N	41
Контрольна сума розділу	4	41 + N
Розмір даних розділу	4	45 + N
Підпис розділу «IEND»	4	49 + N
Контрольна сума розділу	4	53 + N

TIFF представляє растрові зображення, призначені для використання на різних пристроях, які відповідають цьому стандарту формату файлів. Він здатний описувати дворівневі, відтінки сірого, палітру кольорів і повно кольорові дані зображення в декількох колірних просторах. Він підтримує схеми стиснення як з

втратами, так і без втрат. Формат не має обмежень, таких як процесор, операційна система або файлові системи. Файл TIFF починається з восьми байтового заголовка, де байти нумеруються від нуля до максимального індексу, що визначається для кожного зображення. Файл починається з восьми байтового заголовка файлу зображення, який вказує безпосередньо на файл зображення. Опис файлу зображення містить інформацію про зображення, а також вказівники на фактичні дані зображення. TIFF має репутацію складного формату частково через те, що розташування кожного каталогу файлів зображень і даних, на які вказує IFD, включаючи растрові дані, може змінюватися. Насправді, єдиною частиною файлу TIFF, яка має фіксоване розташування, є заголовок файлу зображення, який завжди є першими вісьмома байтами кожного файлу TIFF. Всі інші дані у файлі TIFF знаходяться за допомогою інформації, що міститься в IFD. Кожен IFD і пов'язане з ним растрове зображення відомі як під-файл TIFF. Кількість під-файлів, які може містити файл зображення TIFF, не обмежена. Кожен IFD містить одну або більше структур даних, які називаються тегами. Кожен тег - це дванадцяти байтовий запис, який містить певну інформацію про растрові дані. Тег може містити будь-який тип даних, і специфікація TIFF визначає більше семи десятків тегів, які використовуються для представлення певної інформації. Теги завжди знаходяться в суміжних групах в межах кожного IFD.

Незалежно від формату зображення алгоритм заповнення контейнера буде однаковим. Адже будь-яке растрове зображення буде складатися з пікселів, які кодуються певною кількістю біт. У попередніх розділах зазначалося, що інформація приховується у контейнері шляхом заміни молодшого біта, а це означає, що на один байт секретного повідомлення припадає вісім байт контейнера. Таким чином коефіцієнт заповнення контейнера дорівнює $\frac{1}{8}$. Для виявлення пошкодження контейнера нам також знадобляться так звані сервісні байти у контейнері. У сервісних байтах міститься значення контрольної суми та значення розміру секретного повідомлення. Для контрольної суми відводиться один байт, а для зберігання розміру секретного повідомлення вісім. Враховуючи надану інформацію необхідний розмір контейнера розраховуємо за формулою (2.1).

$$L_{cont} = \left(L_{msg} + 9\right) * 8 \quad (2.1)$$

Де L_{msg} – розмір секретного повідомлення, а L_{cont} – розрахований розмір контейнера.

Розрахувавши необхідний розмір контейнера потрібно перевірити його на кратність розміру блоку перестановки N. Якщо умова кратності виконується, тоді необхідний розмір контейнера може бути використаним, а у іншому випадку необхідний розмір контейнера перераховується за формулою (2.2).

$$L_{cont} = \left(\frac{\left(L_{msg}+9\right)*8}{N} + 1\right) * N \quad (2.2)$$

Слід зазначити, що актуальний розмір контейнера може бути більшим за розрахований необхідний розмір контейнера, але у будь-якому випадку має виконуватися умова кратності розміру блоку перестановки N. Алгоритм наповнення контейнера зводиться до наступних кроків:

1. Визначення необхідних параметрів, таких як розмір блоку перестановки та ключ для крипто-стеганографічного захисту.
2. Заповнення сервісних байтів. На даному етапі розраховується контрольна сума та розмір секретного повідомлення. Далі сервісні байти доповнюють саме повідомлення перед шифруванням.
3. Наповнення контейнера блоками розміру N. Блок піддається стохастичній перестановці перед заміною молодших біт у кожному байті блоку бітами секретного повідомлення. Для заміни молодших бітів, байти секретної інформації шифруються та розкладаються на біти. Після заміни молодших бітів до блоку застосовується обернена стохастична перестановка, що повертає байти блоку на місця, а приховані біти матимуть стохастичний розподіл у середині блоку.

Блок-схема алгоритму заповнення контейнера наведена на рис. 2.1.1.

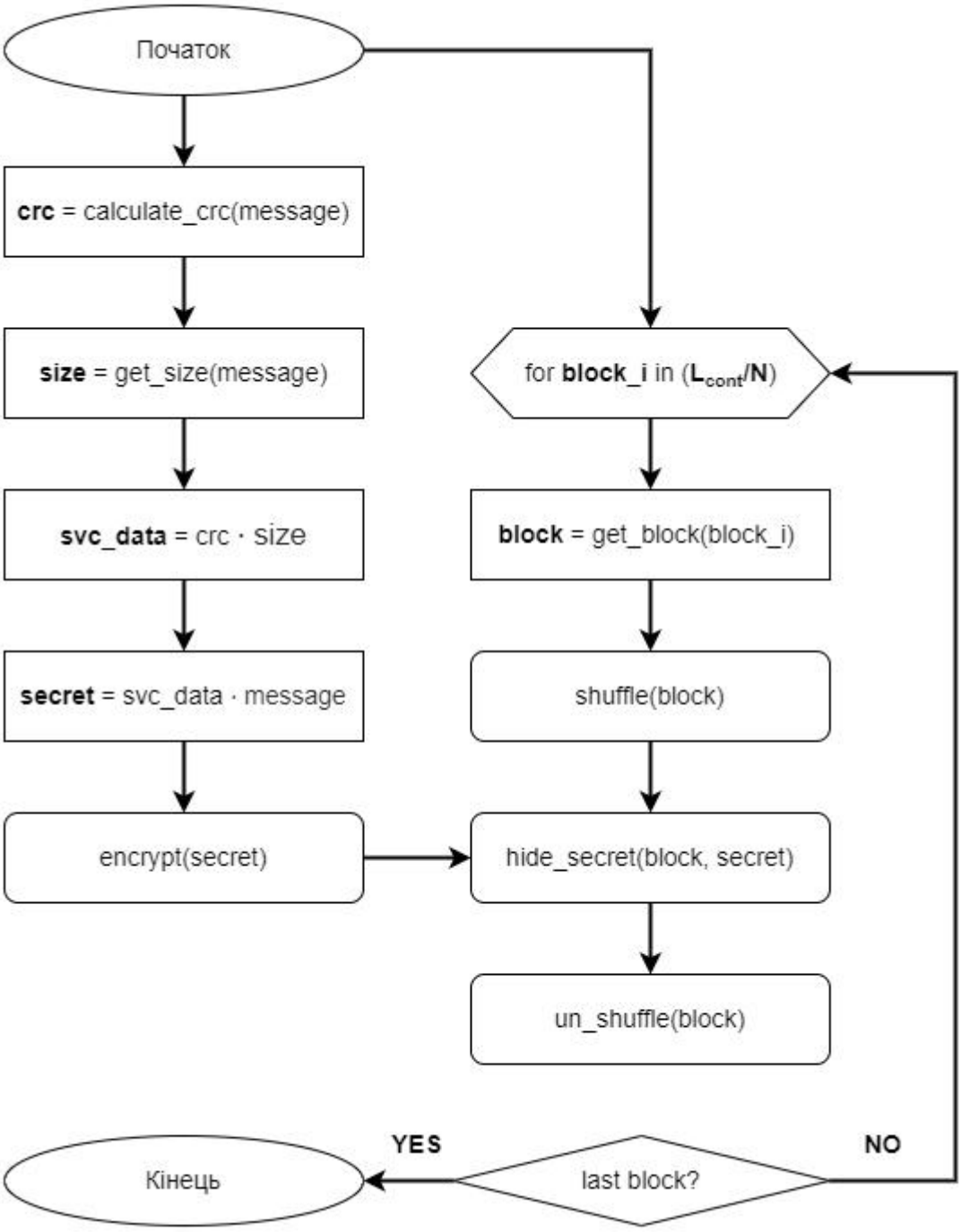


Рис. 2.1.1. Блок-схема алгоритму наповнення контейнера

Серед кроків наведених вище слід окремо звернути увагу на розрахунок контрольної суми. Контрольна сума розраховується за допомогою циклічного надлишкового коду, що полягає у діленні із залишком бінарних поліномів. Значення контрольної суми є залишком від ділення на твірний поліном. У даному випадку

використовується стандарт CRC-8, в основі якого лежить твірний поліном восьмої степені f_{CRC-8}.

$$f_{CRC-8} = 111010101 \quad (2.3)$$

2.2 Приклад зчитування контейнера

Зчитування інформації з контейнера здійснюється аналогічно наповненню, але у зворотному порядку. Перш за все необхідно декодувати зображення відповідно до його формату, щоб отримати доступ до масиву пікселів. Як тільки контейнер готовий для зчитування інформації, необхідно виконати наступні кроки:

1. Задати розмір блоку перестановки та ключ для крипто-стеганографічного захисту, які використовувалися під час наповнення контейнера.
2. Послідовно вичитати блоки розміру N з контейнера. Кожен блок перед цим піддається перестановці байт відповідно до згенерованих з використанням ключа таблиць перестановок. Під час вичитування береться молодші біти та складаються у групи по вісім одиниць, утворюючи цим байти. Кожен сформований байт піддається дешифруванню.
3. Коли байти отримані, необхідно відділити байти сервісної інформації та використовуючи довжину повідомлення розрахувати актуальну контрольну суму. Порівнявши актуальну контрольну суму з оригінальною, що міститься у сервісних байтах, ми отримаємо підтвердження цілісності повідомлення. У випадку невідповідності контрольних сум секретне повідомлення слід вважати пошкодженим.

Блок-схема алгоритму наведена на рис. 2.2.1.

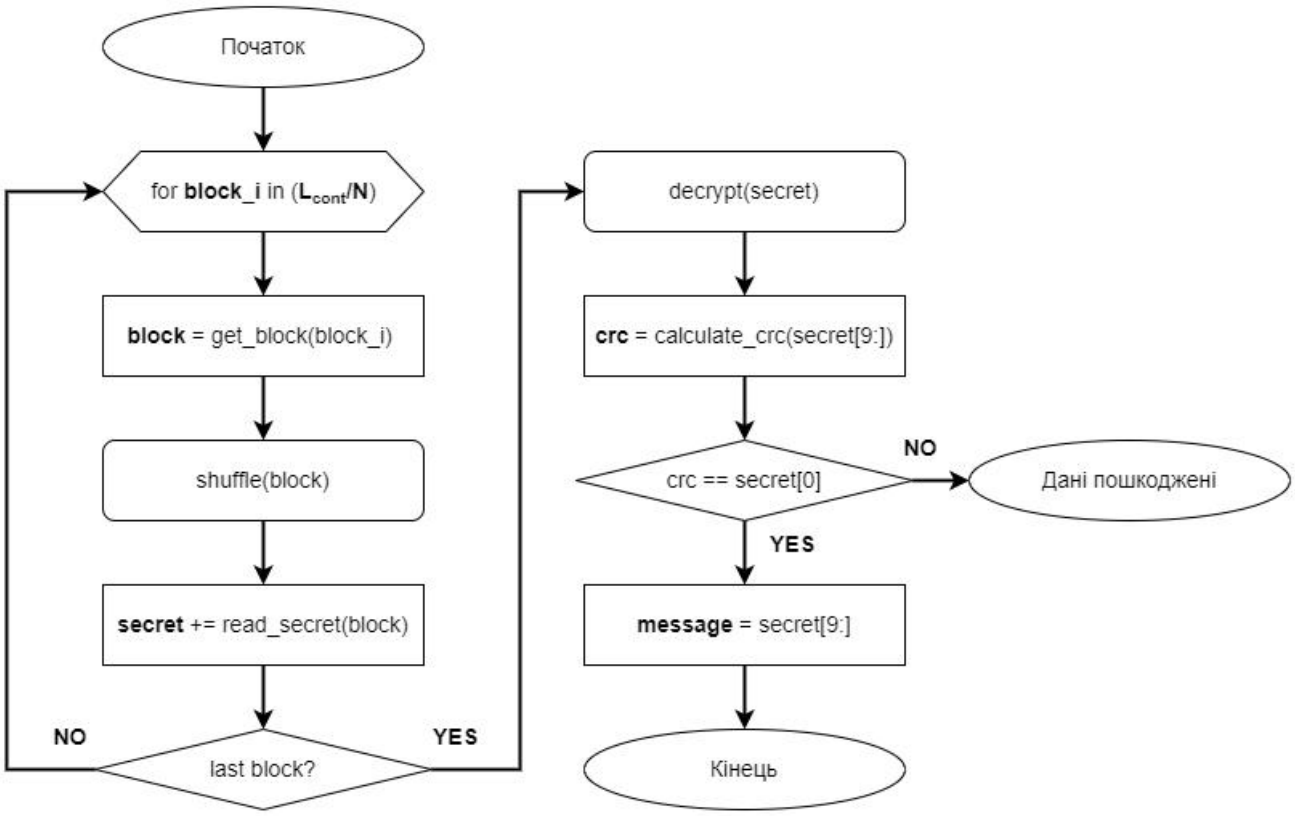


Рис. 2.2.1. Блок-схема алгоритму зчитування контейнера

РОЗДІЛ 3
ОПИС АПАРАТНОЇ ЧАСТИНИ МОДУЛЯ

3.1 Опис мікроконтролера

У якості мікроконтролер використовується інтелектуальна напівпровідникова мікросхема STM32F103. Такі мікросхеми складаються з процесорного блоку, модулів пам'яті, комунікаційних інтерфейсів та периферійних пристроїв. Мікроконтролер має тридцяти двох розрядну архітектуру ARM. Процесори архітектури ARM є вдосконаленою машиною для обчислень зі скороченим набором команд RISC і являє собою 32-розрядний мікроконтролер зі скороченим набором команд RISC. Архітектура ARM налічує наступні компоненти: арифметико-логічний пристрій, мультиплікатор Бута, пристрій швидкого зсуву, блок управління, регістровий файл. Блок-схему архітектури ARM наведено на рис. 3.1.1.

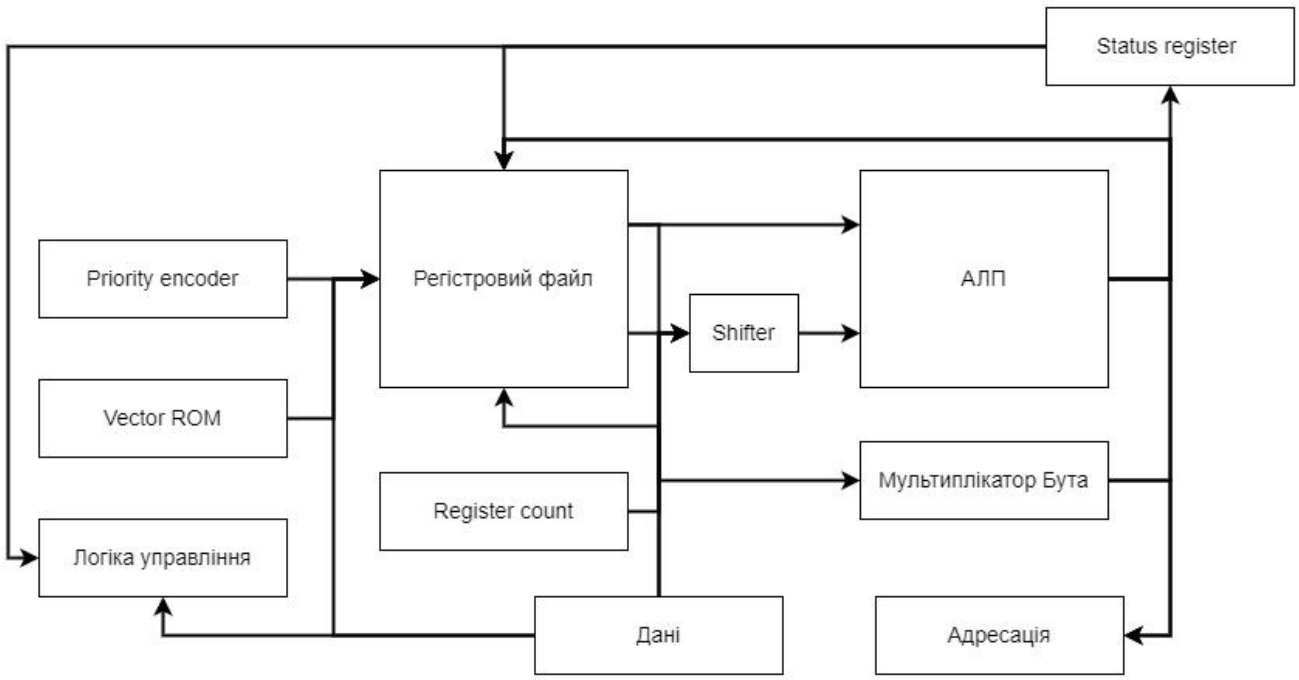


Рис. 3.1.1. Блок-схема архітектури ARM

Арифметико-логічний пристрій має два 32-розрядних входи. Перший надходить з файлу регістрів, а другий з пристрою швидкого зсуву. Регістри стану модифікуються виходами арифметико-логічного пристрою. Арифметико-логічний

пристрій має 4-розрядну функціональну шину, яка дозволяє реалізувати до 16 операційних кодів.

Мультиплікатор Бута має три тридцяти двох розрядних входи, які повертаються з регістрового файлу. Виходом мультиплікатора є тридцять два молодших біта суми. Множення починається щоразу, коли стає активний вхід під четвертим номером. Алгоритм Бута є алгоритмічним правилом множення для двійкових чисел. Алгоритм однаково обробляє додатні та від'ємні числа. Більше того, повторення нулів чи одиниць в множнику пропускаються без виконання додавання або віднімання, тим самим створюючи можливість швидшого множення. Блок схема наведена на рис. 3.1.2.

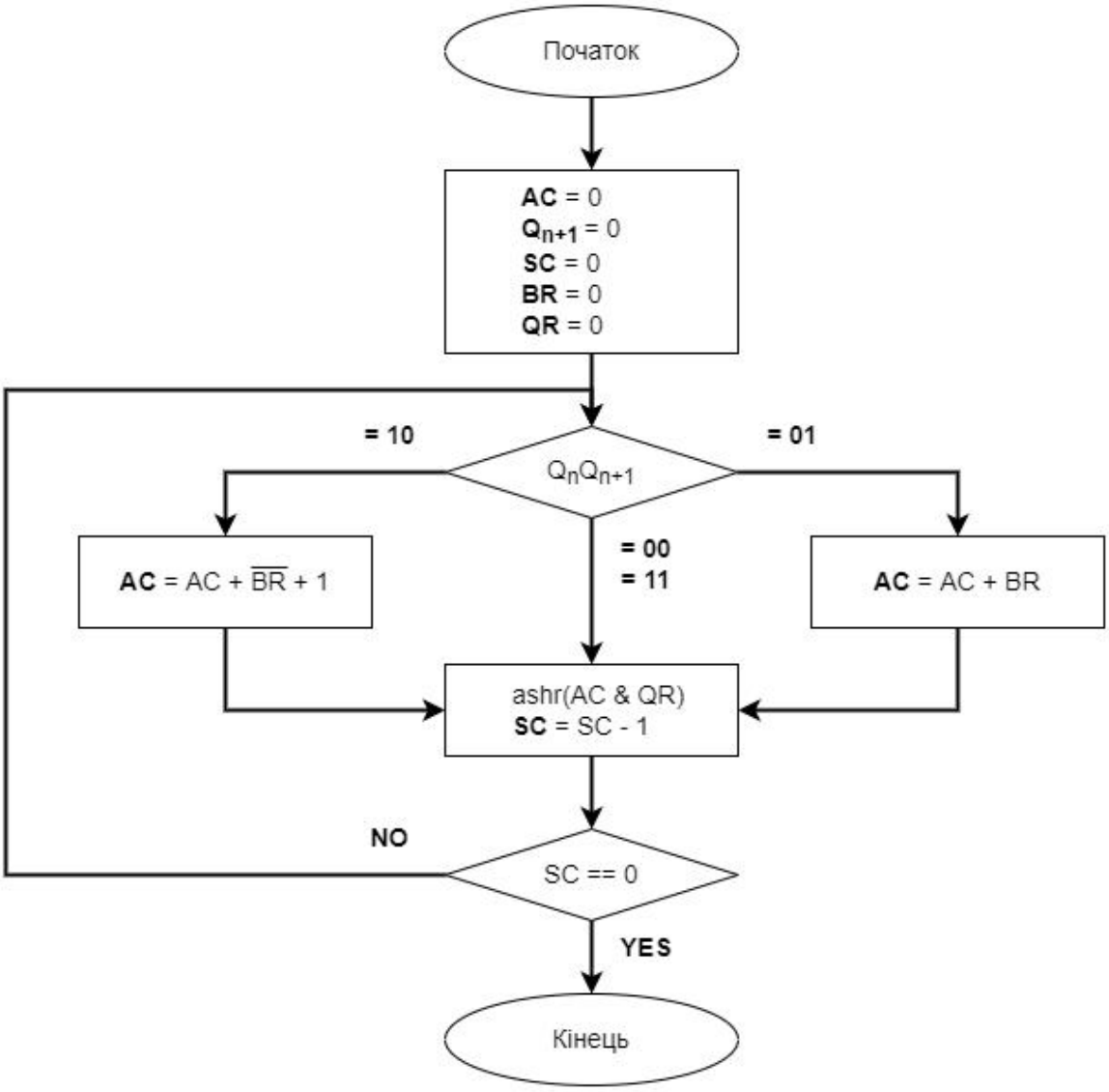


Рис. 3.1.2. Блок-схема алгоритму Бута

Пристрій швидкого зсуву також має тридцяти двох розрядний вхід для зсуву. Вхід під’єднаний до регістрового файлу або це можуть бути безпосередньо дані. Пристрій швидкого зсуву має різні керуючі входи, що повертаються з регістру команд. Поле зсуву в команді керує роботою пристрою швидкого зсуву. Це поле вказує на тип зсуву, який має бути виконаний: логічний зсув вліво або вправо, арифметичний вправо або поворот вправо. Величина, на яку повинен бути зсунутий регістр, міститься в безпосередньому полі інструкції або може бути молодшими шістьма бітами регістра в регістровому файлі. Вхідна шина налічує шість бітів, що дозволяє зсув тридцяти двох бітів. Команди, що вказують на необхідний тип зсуву наведені у таблиці 3.1.1.

Таблиця 3.1.1

Команди для пристрію швидкого зсуву

Тип зсуву	**Бітова комбінація**
Логічний зсув вліво	00
Логічний зсув вправо	01
Арифметичний зсув вправо	10
Поворот вправо	11

Для будь-якого мікропроцесора блок управління є серцем всього процесу і відповідає за роботу системи, тому конструкція блоку управління є найважливішою частиною. Блок управління іноді є чистою комбінаційною схемою. Блок управління у архітектурі ARM реалізований простим автоматом. Хронометраж процесора додатково включений в блок управління. Сигнали від блоку управління підключаються до кожного компонента всередині процесора для спостереження за його роботою.

ARM архітектура відноситься до архітектур зі зменшуваним набором команд, це означає, що ядро не може безпосередньо працювати з пам'яттю. Операції з даними повинні виконуватися регістрами, а інформація зберігається в пам'яті за адресою. Ядро ARM Cortex-M3, що використовується в апаратній реалізації

стеганографічного алгоритму, складається з тридцяти семи наборів регістрів, з яких тридцять один є регістрами загального призначення, а інші шість регістрів є регістрами стану. Для виконання задачі користувача в мікропроцесорі передбачено сім режимів обробки даних: USER, FIQ, IRQ, SVC, UNDEFINED, ABORT, MONITOR. Режим USER є звичайним користувацьким режимом, який має найменшу кількість регістрів. Він не має доступу до регістру збереженого статусу процесора та має обмежений доступ до регістру поточного статусу процесора. Режими швидкого переривання FIQ та звичайного переривання IRQ передбачають негайне реагування процесора на подію, що потребує обробки. Режим швидкого переривання має додаткові п’ять регістрів, що сприяють гнучкості та забезпечують більш високу продуктивність обробки критичних переривань. Режим супервізора SVC є привілейованим користувацьким режимом, що використовується для запуску або скидання. Режим невизначеності UNDEFINED відстежує виконання несанкціонованих інструкцій. Режим відміни ABORT використовується для обробки помилки при доступі до пам’яті. Наприклад, це може статися при спробі отримати дані або інструкції з пам'яті, а програмний лічильник або інструкція завантаження чи збереження містять невірну комірку пам'яті. Режим моніторингу MONITOR є у процесорів, які мають розширення «Security», код в такому випадку може працювати як в захищеному, так і в звичайному стані, а код, який обробляє перехід між захищеним і звичайним станом, працює в режимі моніторингу. Частина регістрів зарезервована в кожному режимі для специфічного використання ядра. До зарезервованих регістрів відносяться: регістр вказівника стеку (SP), регістр адреси повернення (LR), регістр програмного лічильника (PC), регістр збереженого статусу процесора (SPSR), регістр поточного статусу процесора (CPSR). Зарезервовані регістри використовуються для виконання певних функцій. регістр збереженого статусу процесора і регістр поточного статусу процесора містять біти контролю стану, які використовуються для зберігання тимчасових даних. Ці регістри мають ряд властивостей, які визначаються режимами роботи, прапорами дозволу або заборони переривань і прапором стану арифметико-логічного пристрою. Мапа використання регістрів наведена у таблиці 3.1.2.

Таблиця 3.1.2

Мапа використання регістрів

<table>
<tr><th rowspan="2">Регістр</th><th colspan="7">Режими роботи</th></tr>
<tr><th>USER</th><th>FIQ</th><th>IRQ</th><th>SVC</th><th>UNDEFINED</th><th>ABORT</th><th>MONITOR</th></tr>
<tr><td>R0</td><td rowspan="13">Для користування</td><td rowspan="8">Для користування</td><td rowspan="13">Для користування</td><td rowspan="13">Для користування</td><td rowspan="13">Для користування</td><td rowspan="13">Для користування</td><td rowspan="13">Для користування</td></tr>
<tr><td>R1</td></tr>
<tr><td>R2</td></tr>
<tr><td>R3</td></tr>
<tr><td>R4</td></tr>
<tr><td>R5</td></tr>
<tr><td>R6</td></tr>
<tr><td>R7</td></tr>
<tr><td>R8</td><td rowspan="5">Для резерву</td></tr>
<tr><td>R9</td></tr>
<tr><td>R10</td></tr>
<tr><td>R11</td></tr>
<tr><td>R12</td></tr>
<tr><td>R13</td><td colspan="7">Використовується для вказівника стеку</td></tr>
<tr><td>R14</td><td colspan="7">Використовується для адреси повернення</td></tr>
<tr><td>R15</td><td colspan="7">Використовується для програмного лічильника</td></tr>
<tr><td>CPSR</td><td colspan="7">Використовується для поточного статусу процесора</td></tr>
<tr><td>SPSR</td><td>Не викор.</td><td colspan="6">Використовується для збереженого статусу процесора</td></tr>
</table>

Вище було наведено принцип роботи процесора ARM Cortex-M, але ARM Cortex-M3 має також свою периферію, необхідну для роботи, про яку теж слід

згадати, бо вся ця інформація знадобиться для кращого розуміння реалізації стеганографічного алгоритму на високому рівні. Окрім ядра до Cortex-M3 входять: контролер переривань NVIC, контролер переривань WIC, пристрій захисту пам’яті, діагностичні інтерфейси JTAG та Serial Wire, модуль DTW, модуль BPU, відстежування ETM, відстежування ITM, інтерфейс AHB-Lite. Блок-діаграма ARM Cortex-M3 наведена на рис. 3.1.3.

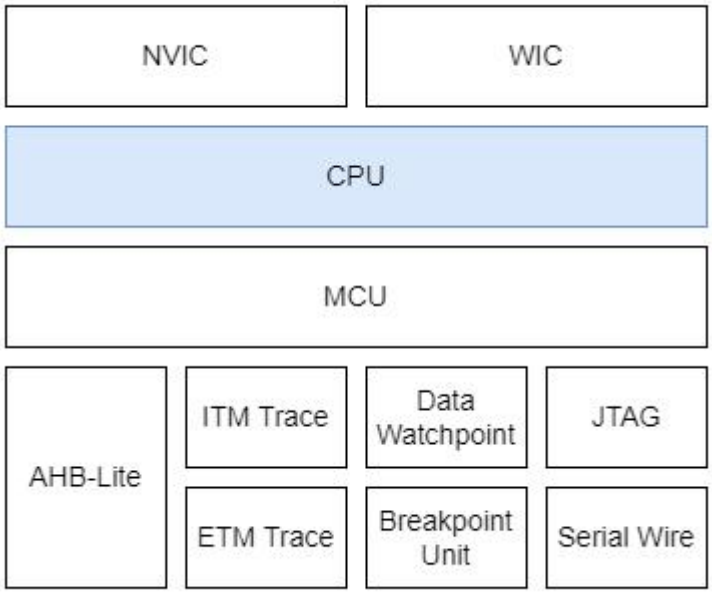


Рис. 3.1.3. Блок-діаграма ARM Cortex-M3

3.2 Опис модуля додаткової пам’яті

Для роботи прототипу необхідно зберігати файл конфігурації пристрою, контейнер, HTML-сторінки для відображення веб-інтерфесу керування пристроєм, та інші файли, що необхідні для роботи пристрою. Всю цю інформацію неможливо вмістити у пам’яті мікросхеми STM32F103, тому є необхідність у модулі додаткової пам’яті. У якості модуля додаткової пам’яті використовується портативна флеш-карта пам’яті, також відома яка флеш-карта SD. Такі пристрої є дуже поширеними носіями інформації. Серед переваг можна виділити мініатюрний розмір, доволі високу швидкість передачі даних, щільність даних та надійний захист. Зв'язок з картою пам'яті Micro SD базується на вдосконаленому восьми контактному інтерфейсі ці контакти відводяться на передачу тактової частоти, команд, даних та

лінію живлення. Такі картки пам’яті класифікують за ємністю як: картки зі стандартною ємністю SDSC, з високою ємністю SDHC та розширеною ємністю SDXC. У прототипі використовується картка SDHC об’ємом у шістнадцять гігабайт. Блок-діаграма модуля додаткової пам’яті зображена на рис 3.2.1.

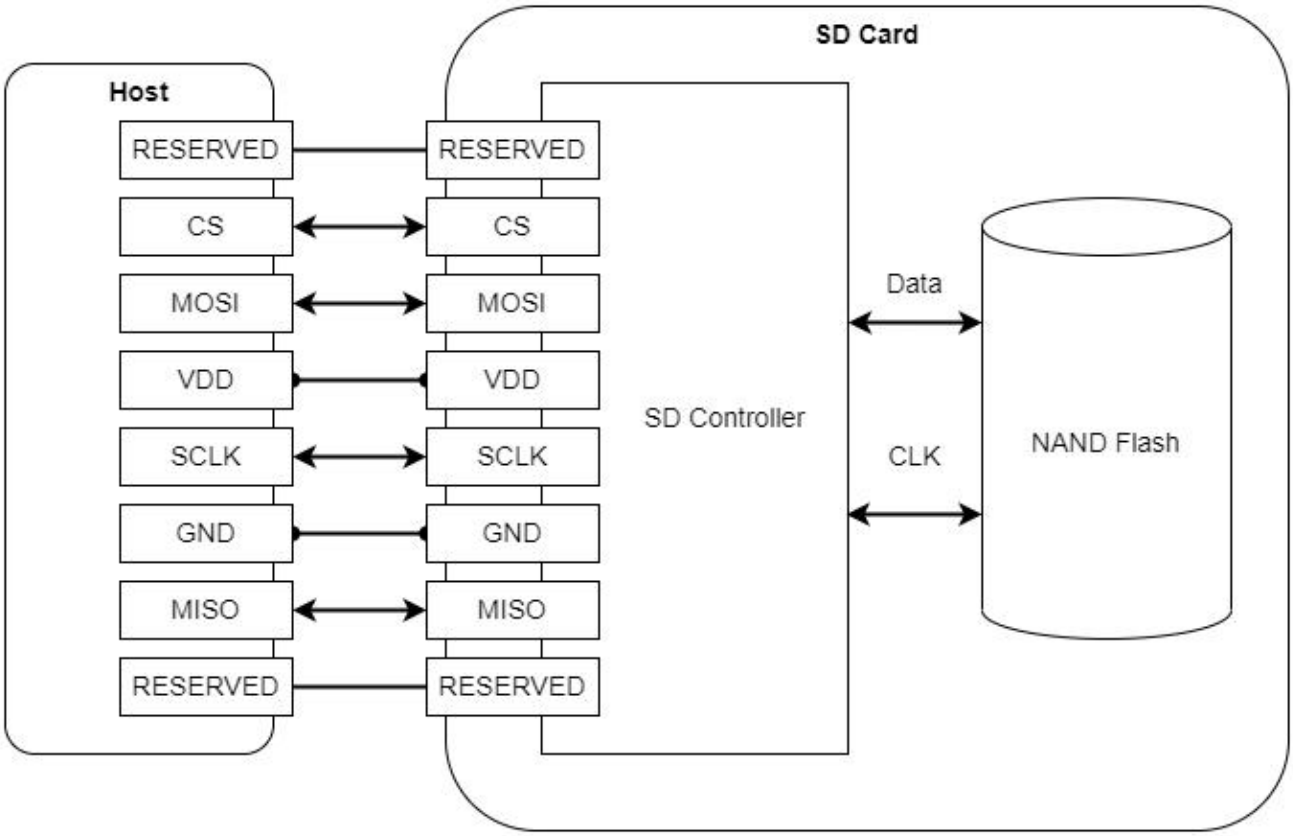


Рис. 3.2.1. Блок діаграма модуля додаткової пам’яті

З наведеної вище діаграми бачимо, що з’єднання модуля додаткової пам’яті з мікросхемою STM32F103 виконується за допомогою послідовного периферійного інтерфейсу SPI. Підключення слідує топології один ведучий та один ведений, що зображена на рис. 3.2.2.

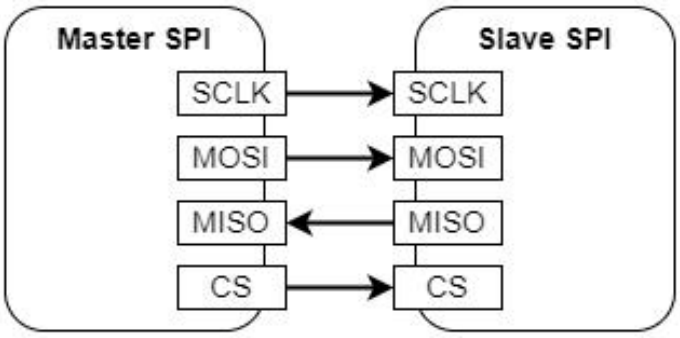


Рис. 3.2.2. Топологія підключення один ведучий та один ведений

Якщо є необхідність працювати з кількома модулями додаткової пам’яті, тоді слід розглянути топології один ведучий та кілька паралельних ведених або один ведучий та кілька послідовних ведених, що зображені на рис. 3.2.3 та рис. 3.2.4 відповідно.

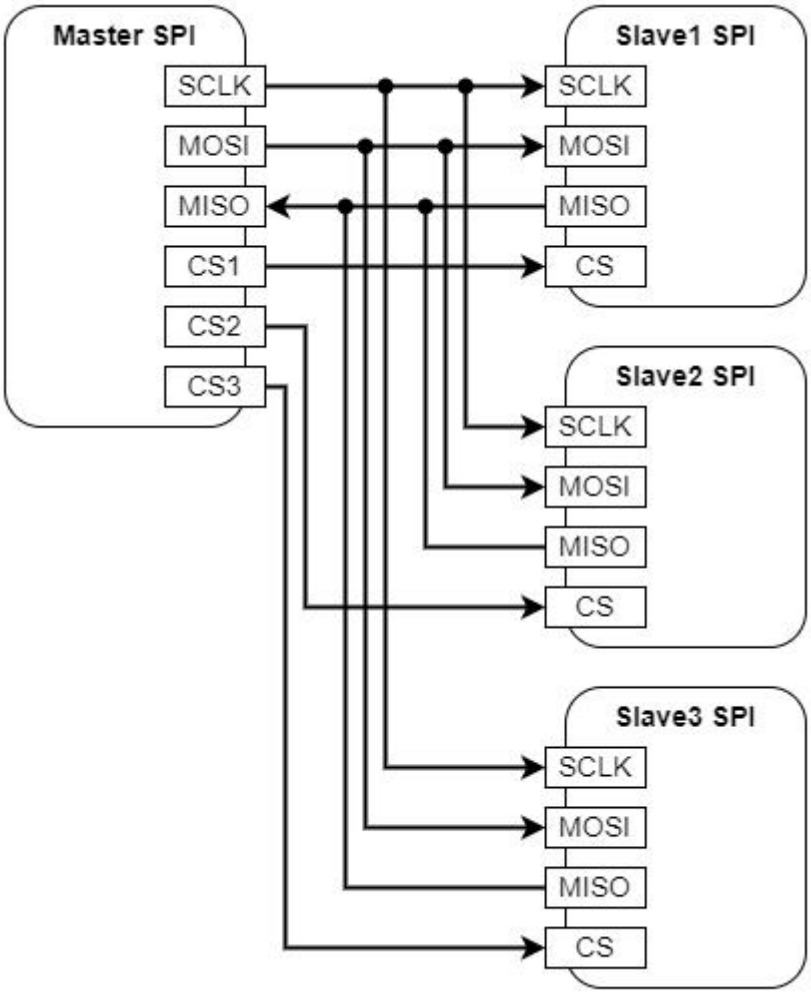


Рис. 3.2.3. Топологія паралельного підключення кількох пристроїв

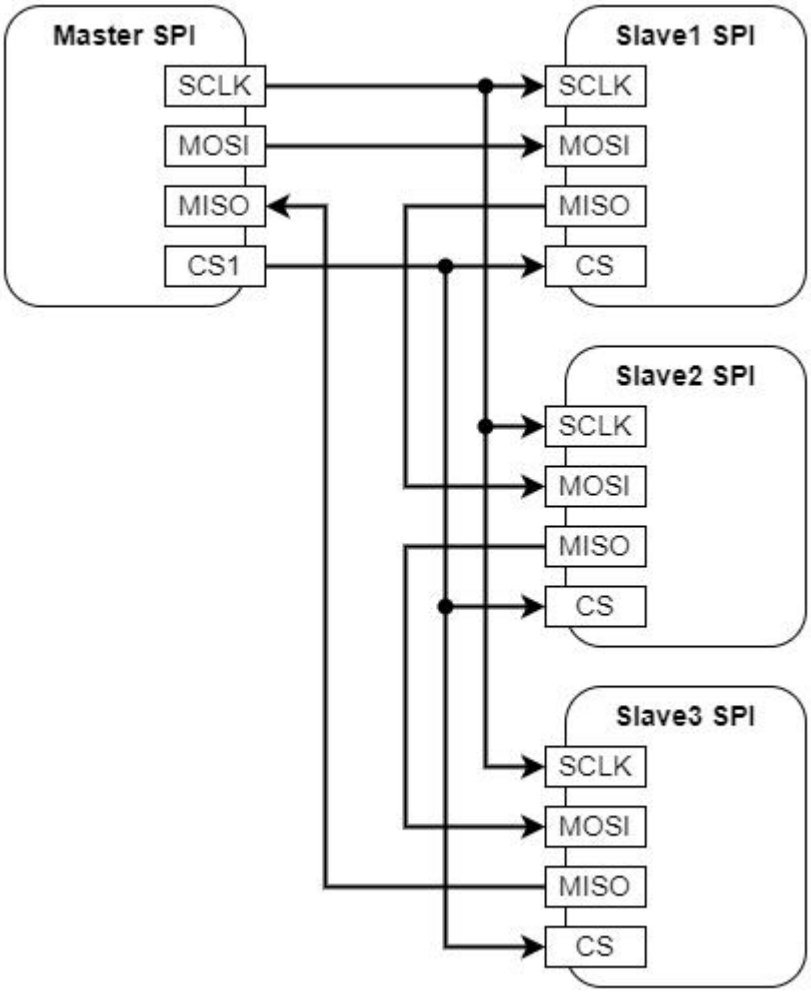


Рис. 3.2.4. Топологія послідовного підключення кількох пристроїв

Але модуль додаткової пам’яті використовується один, тому для підключення достатньо найпростішої топології. Для роботи інтерфейсу та взаємодії з модулем додаткової пам’яті необхідні вісім контактів, що наведені у таблиці 3.2.1.

Таблиця 3.2.1

Опис контактів модуля додаткової пам’яті

#	**Назва контакту**	**Опис контакту**
1	RESERVED	Зарезервовано для SD картки
2	CS	Контакт вибору пристрою, що повідомляє про початок або завершення роботи
3	MOSI	Контакт передачі даних від ведучого пристрою до веденого
4	VDD	Контакт подачі живлення
5	SCLK	Контакт передачі сигналу тактової частоти
6	GND	Контакт нульового потенціалу
7	MISO	Контакт передачі даних від веденого пристрою до ведучого
8	RESERVED	Зарезервовано для SD картки

Окрім з’єднання також необхідно описати принцип комунікації з модулем додаткової пам’яті. Комунікація відбувається шляхом відправки команд, що являють собою шести бітну послідовність. Команди, що використовуються для взаємодії з модулем додаткової пам’яті наведено у таблиці 3.2.2.

Таблиця 3.2.2

Команди для керування модулем додаткової пам’яті

Індекс команди	**Бітова послідовність**	**Назва команди**	**Тип відповіді**
CMD0	000000	GO_IDLE_STATE	R1
CMD1	000001	SEND_OP_COND	R1
CMD6	000110	SWITCH_FUNC	R1
CMD8	001000	SEND_IF_COND	R7
CMD9	001001	SEND_CSD	R1
CMD10	001010	SEND_CID	R1
CMD12	001100	STOP_TRANSMISSION	R1b
CMD13	001101	SEND_STATUS	R2
CMD16	010000	SET_BLOCKLEN	R1
CMD17	010001	READ_SINGLE_BLOCK	R1
CMD18	010010	READ_MULTIPLE_BLOCK	R1
CMD24	011000	WRITE_BLOCK	R1
CMD25	011001	WRITE_MULTIPLE_BLOCK	R1
CMD27	011011	PROGRAM_CSD	R1
CMD28	011100	SET_WRITE_PROT	R1b
CMD29	011101	CLR_WRITE_PROT	R1b
CMD30	011110	SEND_WRITE_PROT	R1
CMD32	100000	ERASE_WR_BLK_START_ADDR	R1
CMD33	100001	ERASE_WR_BLK_END_ADDR	R1
CMD38	100110	ERASE	R1b
CMD42	101010	LOCK_UNLOCK	R1
CMD55	110111	APP_CMD	R1
CMD56	111000	GEN_CMD	R1
CMD58	111010	READ_OCR	R3
CMD59	111011	CRC_ON_OFF	R1

Також слід звернути увагу, що на кожну відправлену команду отримується відповідь, яка відображає результат виконання програми. Відповіді класифікуються за розміром, таким чином відповідь типу R1 займає один байт, а також має підтип R1b, де за маркером відповіді може слідувати нуль або більше байтів, встановлених в нуль. Відповідь типу R2 містить два байти, де перший байт відводиться під бітову комбінацію типу R1, а другий під R2. На тип R3 відводиться п’ять байт, де перший містить бітову комбінацію типу R1, а чотири інших відводяться на регістр стану операції OCR. Для типу R7 необхідні також п’ять байт, але на відміну від R3, чотири останні байти відводяться на результат виконання умови, якщо команда має характер умови, наприклад SEND_IF_COND. Класифікацію відповідей наведено у таблиці 3.2.3.

Таблиця 3.2.3

Класифікація відповідей модуля додаткової пам’яті

<table>
<tr><th rowspan="2">#</th><th colspan="4">Тип відповіді</th></tr>
<tr><th>R1</th><th>R2</th><th>R3</th><th>R7</th></tr>
<tr><td rowspan="8">1</td><td>Режим очікування</td><td rowspan="8">R1</td><td rowspan="8">R1</td><td rowspan="8">R1</td></tr>
<tr><td>Скидання видалення</td></tr>
<tr><td>Несанкціонована команда</td></tr>
<tr><td>Помилка контрольної суми</td></tr>
<tr><td>Помилка у послідовності видалення</td></tr>
<tr><td>Помилка адресації</td></tr>
<tr><td>Помилка параметра</td></tr>
<tr><td>-</td></tr>
<tr><td rowspan="8">2</td><td rowspan="10">-</td><td>Карта заблокована</td><td rowspan="10">OCR</td><td rowspan="10">Результат виконання умови</td></tr>
<tr><td>Помилка розблокування/блокування</td></tr>
<tr><td>Загальна помилка</td></tr>
<tr><td>Помилка контролера</td></tr>
<tr><td>Помилка коригуючого коду</td></tr>
<tr><td>Помилка запису</td></tr>
<tr><td>Помилка видалення</td></tr>
<tr><td>Поза діапазоном значень</td></tr>
<tr><td>3</td><td rowspan="3">-</td></tr>
<tr><td>4</td></tr>
<tr><td>5</td></tr>
</table>

3.3 Опис модуля діагностики та дротового керування пристроєм

Модуль діагностики є дуже важливим елементом пристрою, бо він дає змогу здійснювати моніторинг пристрою у режимі реального часу, а також керувати пристроєм за допомогою системних команд. Модуль діагностики має дротове з'єднання між комп'ютером та пристроєм. Для передачі інформації використовується універсальний асинхронний приймач-передавач, також відомий як інтерфейс UART. Такий інтерфейс може працювати з різними типами послідовних протоколів, які передбачають передачу і прийом даних. При послідовному зв'язку дані передаються біт за бітом з використанням однієї лінії або дроту. Топологія з'єднання пристроїв зображена на рис. 3.3.1.

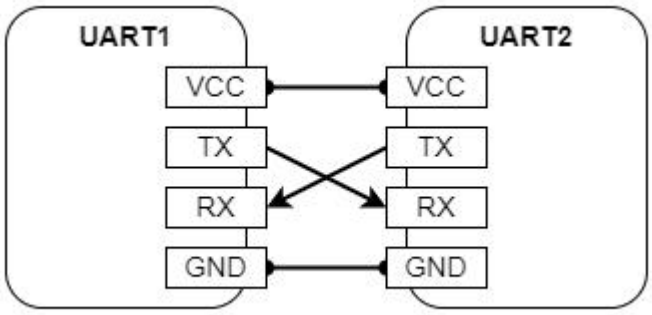


Рис. 3.3.1. Топологія з'єднання UART пристроїв

У двосторонньому зв'язку ми використовуємо два дроти для послідовної передачі даних. Інтерфейс UART, на відміну від раніше розглянутого SPI, не використовує тактовий сигнал для синхронізації пристроїв передавача і приймача, він передає дані асинхронно. Замість тактового сигналу передавач генерує бітовий потік на основі свого тактового сигналу, в той час як приймач використовує свій внутрішній тактовий сигнал для вибірки вхідних даних. Точка синхронізації управляється за допомогою однакової швидкості передачі даних на обох пристроях. Невиконання цієї вимоги може вплинути на час відправлення та отримання даних, що може призвести до розбіжностей при обробці даних. Передача даних являє собою обмін пакетами. Пакет складається зі стартового біта, кадру даних, біта парності та стоп-бітів. Структура пакету проілюстрована на рис. 3.3.2.

Рис. 3.3.2. Структура пакету UART

Лінія передачі даних UART зазвичай утримується на високому рівні напруги, коли передача даних зупинена. Щоб почати передачу даних, передавальний пристрій перемикає лінію передачі з високого рівня на низький протягом одного такту синхронізації. Коли приймаючий пристрій виявляє перехід високої напруги в низьку, він починає зчитувати біти в кадрі даних з частотою, що відповідає швидкості передачі.

Кадр даних містить фактичні дані, що передаються. Він може мати довжину від п'яти біт до восьми біт, якщо використовується біт парності. Якщо біт парності не використовується, кадр даних може мати довжину дев'ять біт. У більшості випадків дані надсилаються починаючи з найменшого значущого біта. Біт парності - це спосіб для приймаючого пристрою визначити, чи були змінені будь-які дані під час передачі. Біти можуть бути змінені електромагнітним випромінюванням, невідповідністю швидкості передачі даних або передачею даних на великі відстані. Після того, як приймаючий пристрій прочитає кадр даних, він підраховує кількість бітів зі значенням, що відповідає логічній одиниці та перевіряє, чи є загальна сума парним або непарним числом. Якщо біт парності дорівнює логічному нулю, то перший або логічний старший біт у кадрі даних повинен дорівнювати парному числу. Якщо біт парності дорівнює логічній одиниці, то ненульові біти у кадрі даних повинні в сумі давати непарне число. Коли біт парності збігається з даними, пристрій знає, що передача була без помилок. Але якщо біт парності дорівнює логічному нулю, а загальна сума непарна, або біт парності дорівнює логічній одиниці, а загальна сума парна, пристрій знає, що біти в кадрі даних змінилися. Щоб сигналізувати про кінець пакета даних, пристрій-відправник переводить лінію передачі даних з низької напруги на високу на час тривалістю від одного до двох біт.

Модуль діагностики та дротового керування пристроєм під’єднаний до мікросхеми STM32F103 напряму, так як серед контактів мікросхеми вже передбачена апаратна реалізація інтерфейсу UART. Але для під’єднання до комп’ютера необхідний адаптер. У якості адаптера використовується інтегрований мостовий контролер USB-UART CP2102. Такий контролер включає в себе швидкісний функціональний контролер USB 2.0, USB-трансивер, частотний генератор, EEPROM або EPROM та асинхронну послідовну шину даних UART. Блок-діаграму контролера CP2102 наведено на рис. 3.3.3.

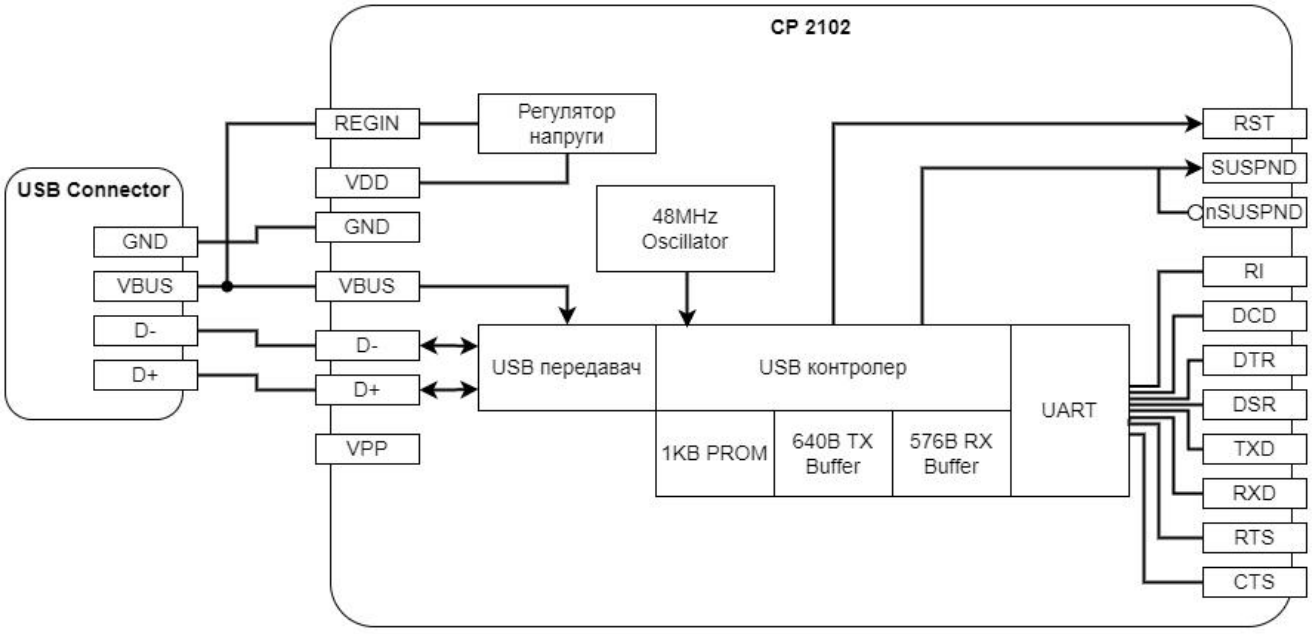


Рис. 3.3.3. Блок діаграма конвертера CP 2102

Для використання контролера CP2102 необхідно встановити безоплатні драйвери пристроїв Virtual COM Port, що надаються компанією Silicon Laboratories, дозволяють продукту на базі CP2102 з'являєтися як COM-порт для додатків для ПК. Таким чином фактична передача даних між комп'ютером і пристроєм CP2102 здійснюється через інтерфейс USB. Також на платі контролера є вбудований стабілізатор напруги від п’яти до трьох вольт. Тому його можна використовувати як пристрій з живленням від шини USB або пристрій з автономним живленням від USB. При увімкненні вихід стабілізатора напруги трьох вольт з'являється на виводі VDD і може бути використаний для живлення зовнішніх пристроїв. Крім того, якщо

живлення подається на вивід VDD, то контролер може функціонувати як USB-пристрій з автономним живленням, а стабілізатор напруги буде відключений. Для цієї конфігурації рекомендується, щоб вхід REGIN був прив'язаний до мережі трьох вольт для відключення стабілізатора напруги. Крім того, VDD або REGIN можуть бути знеструмлені при напрузі VBUS рівній у п’ять вольт.

3.4 Опис модуля бездротового керування пристроєм

Окрім дротового керування пристрою, що здійснюється у режимі реального часу, в прототипі передбачено варіант бездротового керування пристроєм. Бездротове з’єднання є більш зручним для користувача та підтримує можливість роботи з кількома користувачами. За допомогою бездротового керування, користувач може конфігурувати пристрій, а також зчитати зашифроване повідомлення від пристрою. Бездротове з’єднання забезпечується протоколом Wi-Fi «IEEE802.11 b/g/n». Апаратно це реалізовано у мікросхемі ESP8266. Така мікросхема пропонує повне і автономне мережеве рішення Wi-Fi. Блок-діаграма модуля ESP8266 зображена на рис. 3.4.1.

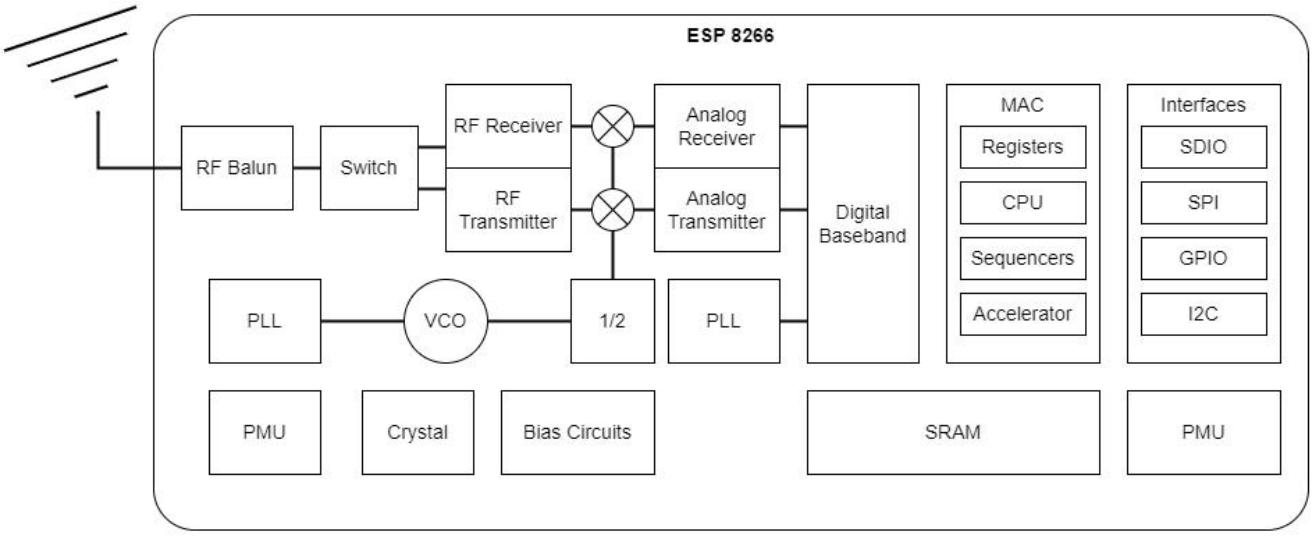


Рис. 3.4.1. Блок-діаграма ESP8266

ESP8266 був розроблений для різних видів електроніки та додатків Інтернету речей з метою досягнення найнижчого енергоспоживання за допомогою комбінації декількох запатентованих технологій. Енергозберігаюча архітектура працює в 3 режимах: активний режим, сплячий режим і режим глибокого сну. Завдяки

використанню передових методів управління енергоспоживанням і логіки для відключення непотрібних функцій і управління перемиканням між сплячим і активним режимами, ESP8266 споживає менше 12 мкА в сплячому режимі щоб залишатися на зв'язку. Тому можна переводити бездротовий зв’язок у режим сну для зменшення витрат енергії, якщо потреби у бездротовому зв’язку тимчасово немає. У сплячому режимі активними залишаються тільки відкалібрований годинник реального часу і сторожовий таймер. Годинник реального часу можна запрограмувати на пробудження ESP8266 через будь-який необхідний інтервал часу. Модуль бездротового зв’язку потребує найбільше енергії у прототипі. Виміряні емпірично показники наведено у таблиці 3.4.1.

Таблиця 3.4.1

Затрати енергії ESP8266

Режим	**Витрати енергії**
Передача протоколом 802.11b зі швидкістю 1Mbps та потужністю передавача +19.5dBm	215mA
Передача протоколом 802.11b зі швидкістю 11Mbps та потужністю передавача +18.5dBm	197mA
Передача протоколом 802.11g зі швидкістю 54Mbps та потужністю передавача +16dBm	145mA
Передача протоколом 802.11n зі швидкістю 72.2Mbps та потужністю передавача +14dBm	135mA
Прийом за протоколом 802.11b з об’ємом в один кілобайт та потужністю приймача -80dBm	60mA
Прийом за протоколом 802.11g з об’ємом в один кілобайт та потужністю приймача -70dBm	60mA
Прийом за протоколом 802.11n з об’ємом в один кілобайт та потужністю приймача -65dBm	62mA
Сплячий режим	0.9mA
Режим глибокого сну	10μA

ESP8266 працює на частоті 2.4GHz, а тому трансивер підтримує до чотирнадцяти каналів, що відповідають протоколу «IEEE802.11 b/g/n». Розподіл частот по каналам наведено у таблиці 3.4.2.

Таблиця 3.4.2

Спектр частот ESP8266

Номер каналу	**Частота**
1	2412MHz
2	2417MHz
3	2422MHz
4	2427MHz
5	2432MHz
6	2437MHz
7	2442MHz
8	2447MHz
9	2452MHz
10	2457MHz
11	2462MHz
12	2467MHz
13	2472MHz
14	2484MHz

Приймач перетворює радіочастотний сигнал в квадратурні сигнали базової смуги частот і переводить їх в цифрову область за допомогою двох високошвидкісних АЦП з високою роздільною здатністю. Для адаптації до мінливих умов сигнального каналу в радіостанцію інтегровані радіочастотні фільтри, автоматичне регулювання підсилення, схеми усунення зсуву постійного струму та фільтри базової смуги. фільтри основної смуги частот інтегровані в

радіостанцію. Передавач перетворює квадратурні сигнали базової смуги частот і керує антеною за допомогою CMOS-підсилювача потужності. Використання цифрового калібрування ще більше покращує лінійність підсилювача потужності, забезпечуючи сучасні показники середньої потужності +19dBm для передачі стандарту 802.11b і +16dBm для передачі стандарту 802.11n. Модуль бездротового керування пристроєм використовує UART з'єднання з мікросхемою STM32F103 для обміну даними. Комунікація відбувається за допомогою спеціально визначених команд, що передаються між STM32F103 та ESP8266.

РОЗДІЛ 4
ПРОГРАМНА ЧАСТИНА НИЗЬКОГО РІВНЯ

До програмної частини низького рівня відноситься програмне забезпечення, що виконується у прототипі на базі STM32F103. Для написання програмного забезпечення було використано мову програмування С. Мова програмування С є дуже поширеною у програмуванні електронних пристроїв. Вона дозволяє програмісту більш просто, у порівнянні з мовою Асемблера, створювати програмне забезпечення. Та на відміну від мов програмування вищого рівня, мова програмування С дозволяє одразу компілювати програму в машинний код.

4.1 Створення бінарного файлу програми

Щоб отримати бінарний файл програми, спочатку необхідно скомпілювати програму, написану мовою програмування С. Процес компіляції в мові С перетворює код, який читається людиною, в машинний код. Для мови програмування С це відбувається перед початком виконання програми для перевірки синтаксису та семантики коду. Процес компіляції в мові С включає в себе чотири етапи: попередню обробку, компіляцію, збірку та компонування, після чого ми отримуємо необхідний файл програми. Блок-діаграма процесу компіляції зображена на рис. 4.1.1.

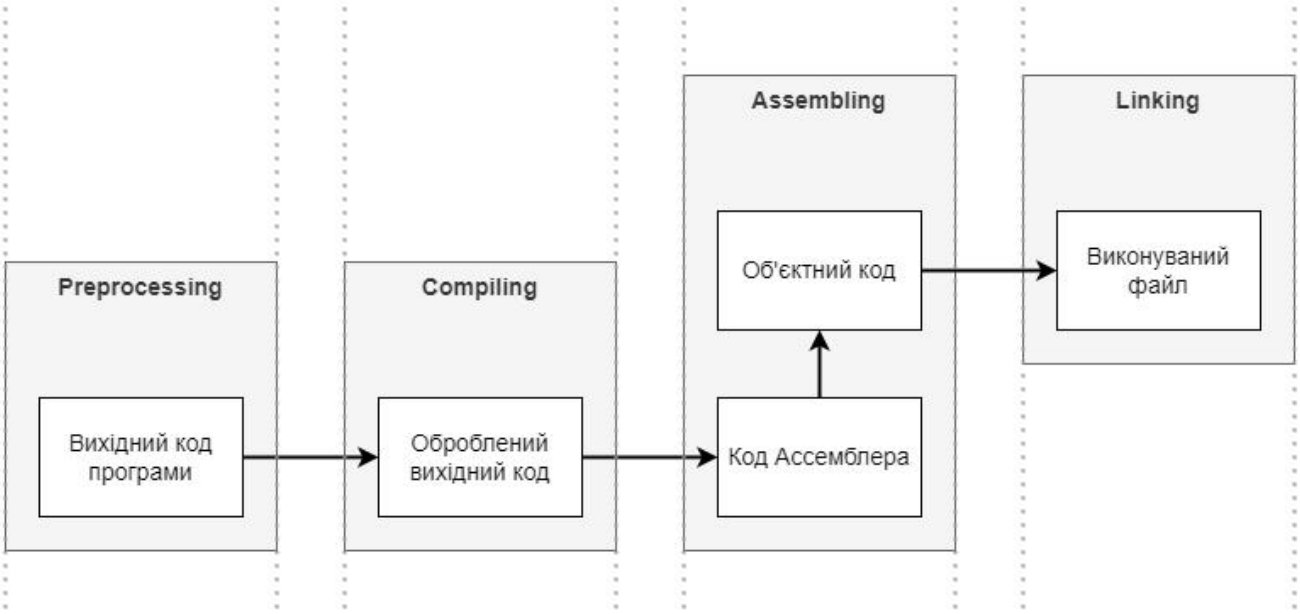


Рис. 4.1.1. Блок-діаграма процесу компіляції

Попередня обробка - це перший крок у процесі компіляції на мові С, що виконується за допомогою інструменту препроцесора (заздалегідь написана програма, яка викликається системою під час компіляції). Всі оператори, що починаються з символу «#» в програмі на мові С, обробляються препроцесором, і він перетворює наш програмний файл в проміжний файл без операторів «#». При препроцесуванні виконуються наступні задачі: видалення коментарів, розширення макросів, включення файлів, умовна компіляція. Коментарі в програмі на мові С використовуються для того, щоб дати загальне уявлення про певний оператор або частину коду, власне, коментарі - це частина коду, яка видаляється в процесі компіляції препроцесором, оскільки вони не несуть особливої користі для машини. Всі коментарі будуть видалені з програми після завершення фази препроцесора. Макроси - це деякі константні значення або вирази, визначені за допомогою директив «#define» у мові С. Виклик макросу призводить до розгортання макросу. Препроцесор створює проміжний файл, в якому деякі заздалегідь написані інструкції на рівні асемблера замінюють визначені вирази або константи. Для того, щоб відрізнити оригінальні інструкції від інструкцій асемблера, отриманих в результаті розширення макросів, до кожного оператора, що розширюється, додається знак «+». Включення файлу в мові С - це додавання іншого файлу, що містить деякий заздалегідь написаний код, в нашу програму на мові С під час її попередньої обробки. Здійснюється з допомогою директиви «#include». Включення файлу під час препроцесування призводить до того, що весь вміст файлу додається до вихідного коду, замінюючи необхідну директиву, створюючи новий проміжний файл. Умовна компіляція запускає або пропускає блок коду після перевірки того, чи макрос, константне значення або вираз, визначений за допомогою «#define». Препроцесор замінює всі директиви умовної компіляції деяким заздалегідь визначеним асемблерним кодом і передає компілятору новий розширений файл. Умовна компіляція може бути виконана за допомогою таких команд, як «#ifdef», «#endif», «#ifndef», «#if», «#else» і «#elif» в програмі на мові С.

На етапі компіляції у мові С вбудований компілятор перетворює проміжний файл у асемблерний, що містить інструкції рівня асемблера. Для підвищення продуктивності програми компілятор транслює проміжний файл в асемблерний. Асемблерний код - це проста мова англійського типу, яка використовується для написання низькорівневих інструкцій. Весь програмний код підлягає синтаксичному аналізу та розбирається програмою-компілятором за один раз, і вона повідомляє нам про будь-які синтаксичні помилки або попередження, присутні у вихідному коді, через вікно терміналу.

Код на рівні асемблера перетворюється в машинний код у двійковій або шістнадцятковій формі за допомогою асемблера. Асемблер - це заздалегідь написана програма, яка переводить асемблерний код у машинний. Він бере основні інструкції з файлу асемблерного коду і перетворює їх у двійковий чи шістнадцятковий код, специфічний для машинного типу, відомий як об'єктний код. Створений файл має ту саму назву, що й асемблерний файл, і відомий як об'єктний файл.

Зв'язування - це процес включення бібліотечних файлів в нашу програму. Бібліотечні файли - це деякі заздалегідь визначені файли, які містять визначення функцій на машинній мові. В об'єктному файлі записані деякі невідомі оператори, які наша операційна система не може зрозуміти. Це можна порівняти з книгою, в якій є деякі слова, яких ви не знаєте, і ви будете використовувати словник, щоб знайти значення цих слів. Аналогічно, ми використовуємо бібліотечні файли, щоб надати значення деяким невідомим твердженням з нашого об'єктного файлу.

Для компіляції програми необхідне спеціальне програмне забезпечення, яке має назву компілятор. Існує велика кількість компіляторів, які використовуються відповідно до середи розробки, операційної системи та цільової архітектури. Так як для розробки ПО прототипу використовувалась операційна система UNIX, а цільова архітектура для програмного забезпечення це ARM, то мова піде про «GNU C/C++» компілятор з набору інструментів розробки «Arm GNU Toolchain». Цей набір інструментів розробки є у вільному доступі та може бути отриманим з офіційної веб-сторінки. До його переваг можна віднести: наявність інтегрованих та перевірених пакетів, підтримку мов програмування С та С++, підтримку процесорів

на базі профілів A, R і M архітектури ARM, підтримка крос-інструментарію для найбіль популярних операційних систем, постійно покращується розробниками та доступний для безоплатного користування.

Після компілювання програми ми отримуємо спеціальний файл, що можна завантажити у флеш пам’ять мікросхеми STM32F103 для його подальшого виконання на мікроконтролері. Але залишається відкритим питання, як саме програма переноситься у флеш пам’ять STM32F103. Для завантаження програми на пристрій нам необхідно мати спеціальний пристрій, що має назву програматор. Найбільш поширеними для STM32 є програматори ST-Link. ST-Link - це внутрішньосхемний відладчик і програматор для мікроконтролерів STM8 і STM32. Модуль однопровідного інтерфейсу SWIM та інтерфейси JTAG, SWD використовуються для зв'язку з будь-яким мікроконтролером STM8 або STM32, розташованим на прикладній платі. Також ST-Link забезпечує цифрову ізоляцію між ПК і цільовою платою. А для використання програматора на ПК можна використовувати спеціальне програмне забезпечення для роботи з програматором, що є доступним на офіційній веб-сторінці у відкритому доступі. Для завантаження необхідно обрати плату на яку завантажується ПО та бінарний файл програми, після чого залишається тільки дочекатись поки програма копіюється у флеш пам’ять та перезавантажити STM32F103.

4.2 Комунікація із апаратною частиною

Щоб керувати модулями, які під’єднані до мікросхеми STM32F103, можна напряму керувати інтерфейсами, що вимагає від програміста великих затрат на написання коду. Але існують спеціальні бібліотеки для роботи з інтерфейсами, що полегшують розробку програмного забезпечення. Одна з таких є бібліотека для роботи з периферією від «STMicroelectronics». Такий драйвер надає простий, загальний багатоекземплярний набір інтерфейсів прикладного програмування для взаємодії з верхнім рівнем. Драйвери поділяються на дві категорії: загальний набір інтерфейсів прикладного програмування, які надають загальні та типові функції для всієї серії STM32 та розширений набір інтерфейсів прикладного програмування, які включають специфічні та налаштовані функції для певної лінії або номера деталі.

Драйвери включають повний набір готових до використання інтерфейсів прикладного програмування, які спрощують реалізацію додатків для користувача. Наприклад, комунікаційні периферійні пристрої містять інтерфейси прикладного програмування для ініціалізації та конфігурації периферійного пристрою, управління передачею даних в режимі опитування, обробки переривань та управління помилками зв'язку. Рівень драйверів реалізує виявлення збоїв під час виконання програми шляхом перевірки вхідних значень усіх функцій. Така динамічна перевірка підвищує надійність мікропрограми. Список драйверів, що використовується у прототипі наведено у таблиці 4.2.1.

Таблиця 4.2.1

Перелік використаних драйверів

Тип драйверу	**Короткий опис**
HAL Core	Загальний драйвер апаратного рівня абстракції
HAL GPIO	Драйвер апаратного рівня абстракції для керування вхідними та вихідними контактами
HAL Cortex	Драйвер апаратного рівня абстракції для керування процесором
HAL DMA	Драйвер апаратного рівня абстракції для керуванням контролера прямого доступу до пам’яті
HAL EXTI	Драйвер апаратного рівня абстракції для керуванням зовнішнім контролером переривань та подій
HAL FLASH	Драйвер апаратного рівня абстракції для керуванням флеш-пам’яттю
HAL PWR	Драйвер апаратного рівня абстракції для керуванням контролером живлення
HAL RCC	Драйвер апаратного рівня абстракції для керуванням контролером скидання та синхронізації
HAL SPI	Драйвер апаратного рівня абстракції для керуванням послідовним периферійним інтерфейсом
HAL UART	Драйвер апаратного рівня абстракції для керуванням універсальним асинхронним приймачем та передавачем

Драйвери описані вище вимагають для коректної роботи драйвери низького рівню, що напряму керують апаратною частиною. До таких драйверів відносяться CMSIS, який пропонує програмний API, що описує інтерфейси драйверів периферійних пристроїв для стеків проміжного програмного забезпечення та користувацьких додатків. Драйвер представлений у вигляді заголовних файлів. Блок-діаграма з'єднань між апаратною частиною та програмним забезпеченням зображена на рис. 4.2.1.

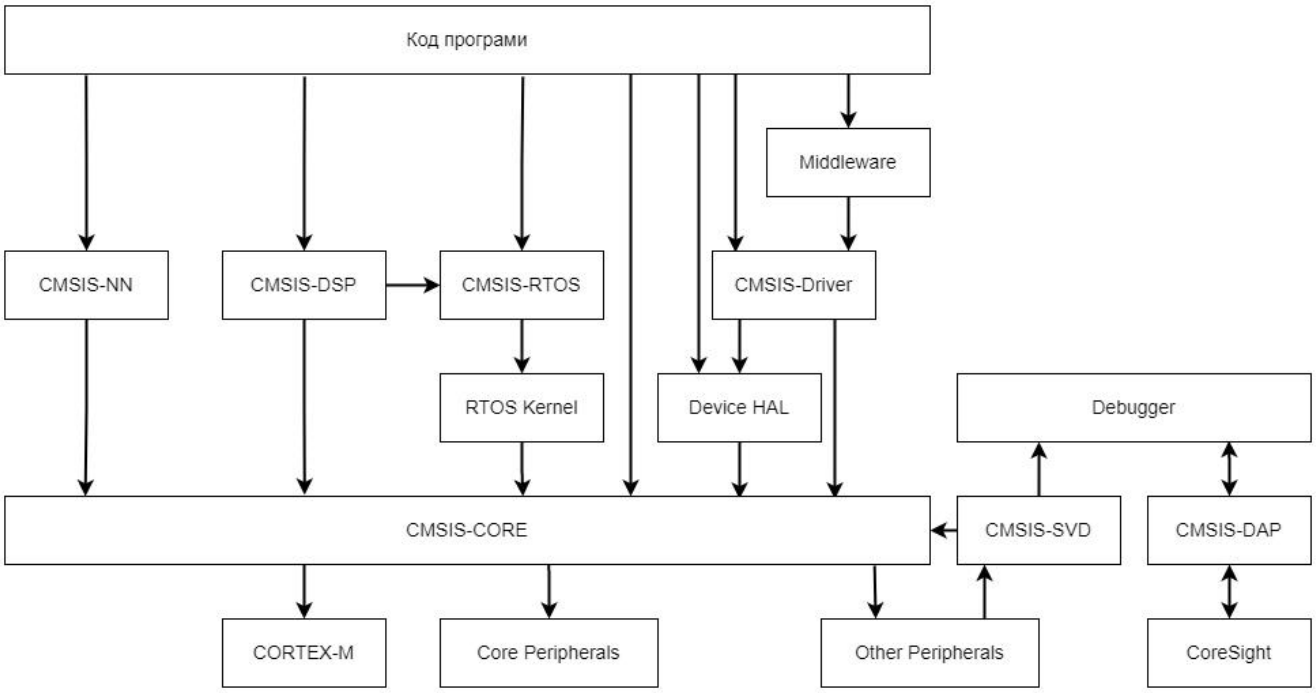


Рис. 4.2.1. Блок-діаграма драйверів

CMSIS-RTOS пропонує детермінований контроль виконання програмного забезпечення в реальному часі для простих вбудованих додатків. Кожен програмний продукт ARM включає його в себе, починаючи від безкоштовного Keil Microcontroller Development Kit (MDK) і закінчуючи найпопулярнішими продуктами ARM Design Studio, що входять до складу MDK. По суті, CMSIS-RTOS є інтерфейсом прикладного програмування, який забезпечує узгодженість програмних шарів з проміжним програмним забезпеченням та бібліотечними компонентами. CMSIS-Driver пропонує програмний API, який описує інтерфейси драйверів

периферійних пристроїв для стеків проміжного програмного забезпечення та користувацьких додатків. CMSIS-Driver API описує інтерфейс за допомогою заголовних файлів та документації. CMSIS-Driver API охоплює широкий спектр підтримуваних периферійних пристроїв для більшості потенційних випадків. Типова структура взаємозв'язків між програмними компонентами та апаратними блоками зображення на рис. 4.2.2.

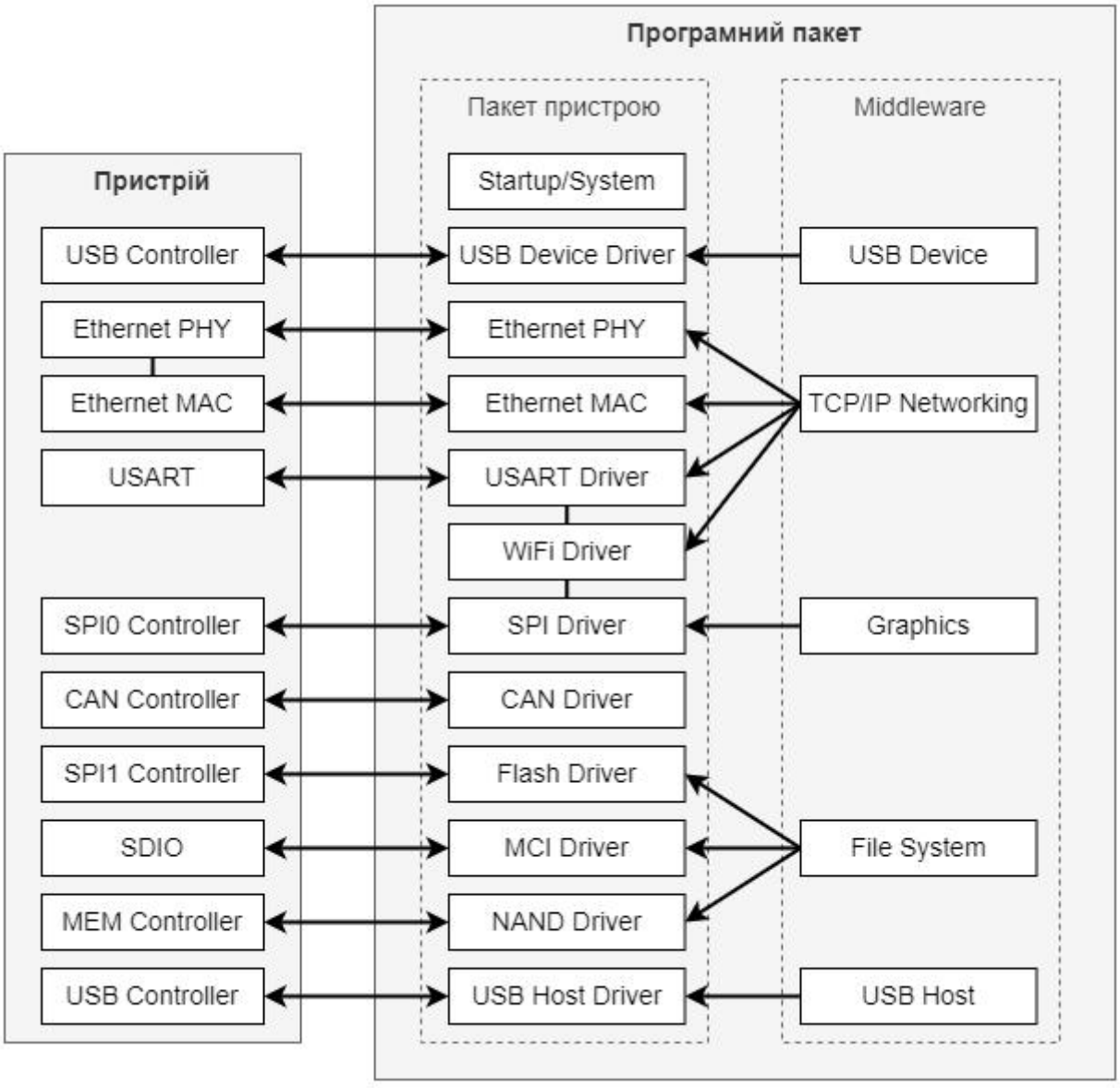


Рис. 4.2.2. Структура CMSIS-Driver

CMSIS-CORE - найважливіша частина, яка керує запуском системи, доступом до ядра процесора і визначеннями периферії для використання в кожному вбудованому

додатку. ARM пропонує загальні системні файли CMSIS-CORE, так звані шаблони. На основі цього шаблону кожен виробник мікроконтролерів створює і пропонує свій власний пакет сімейства пристроїв (DFP). Інженери-програмісти можуть вільно використовувати як загальні ресурси, так і специфічні. CMSIS-Pack - це файл у форматі XML, що описує вміст програмного пакету, який включає вихідний код, заголовні файли, програмні бібліотеки, шаблони вихідного коду та параметри пристроїв, а також код запуску та алгоритми програмування. Він включає приклади проектів для мікроконтролерів та плат для розробки. Повна колекція файлів поставляється у форматі ZIP-архів. Після інсталяції програмного пакету всі програмні компоненти, що входять до його складу, стають доступними інструментальним засобам розробки. CMSIS-DSP дозволяє розробити систему цифрової обробки сигналів (ЦОС) в реальному часі, не будучи тривіальною, як алгоритми ЦОС. Бібліотека CMSIS-DSP - це багата колекція функцій ЦОС, які оптимізовані для різних ядер процесорів Cortex-M. CMSIS-DSP широко використовується в промисловості, а також дозволяє оптимізовано генерувати код на мові C з MATLAB. CMSIS-SVD входить до складу CMSIS-Pack. Він підтримує детальний перегляд периферійних пристроїв мікроконтролера, який відображає поточний стан регістрів у форматі опису системного вигляду (SVD). CMSIS-DAP - це інтегрований інструмент, який використовує стандартизований інтерфейс до порту налагоджувального доступу Cortex (DAP). Таким чином, інженерам не потрібні спеціальні налагоджувальні пристрої типу ULINK, J-Link або подібні для базової оцінки плат.

Для використання бібліотеки HAL необхідно налаштувати спеціальний файл конфігурації, що містить необхідні константи для коректної роботи бібліотеки. Параметри конфігурації наведено у таблиці 4.2.2.

Таблиця 4.2.2

Параметри конфігурації бібліотеки HAL

Параметр	**Опис**	**Значення**
HSE_VALUE	Визначає значення зовнішнього генератора (HSE), виражене в Гц. Користувач повинен відкоригувати це визначення при використанні іншого значення кристала.	25 000 000 Hz на STM3210C-EVAL, в іншому випадку 8 000 000 Hz
HSE_STARTUP_TIMEOUT	Тайм-аут на запуск HSE, виражений в мс	5000 Hz
HSI_VALUE	Визначає значення внутрішнього генератора (HSI), виражене в Гц	8 000 000 Hz
LSE_VALUE	Визначає значення зовнішнього генератора (HSE), виражене в Гц. Користувач повинен відкоригувати це визначення при використанні іншого значення кристала.	32768
LSE_STARTUP_TIMEOUT	Тайм-аут для запуску LSE, виражений в мс	5000
VDD_VALUE	Значення VDD в mV	3300
USE_RTOS	Дозволяє використовувати RTOS	TRUE
PREFETCH_ENABLE	Вмикає функцію попередньої вибірки	TRUE

4.3 Операційна система реального часу

Операційна система - це комп'ютерна програма, яка підтримує основні функції комп'ютера та надає послуги іншим програмам (або додаткам), що працюють на комп'ютері. Додатки забезпечують функціональність, яка потрібна користувачеві комп'ютера. Послуги, що надаються операційною системою, роблять написання додатків швидшим, простішим і зручнішим для обслуговування. Більшість операційних систем дозволяють виконувати декілька програм одночасно. Це називається багатозадачністю. Насправді, кожне ядро процесора може виконувати лише один потік виконання в будь-який момент часу. Частина операційної системи, яка називається планувальником, відповідає за прийняття рішення про те, яку програму коли запускати, і забезпечує ілюзію одночасного виконання шляхом швидкого перемикання між кожною програмою. Тип операційної системи визначається тим, як планувальник вирішує, яку програму коли запускати. Наприклад, планувальник, що використовується в багатокористувацькій операційній системі Unix, забезпечить кожному користувачеві достатню кількість часу обробки. Іншим прикладом є планувальник у настільній операційній системі Windows, який намагатиметься забезпечити швидку реакцію комп'ютера на запити користувача.

Планувальник в операційній системі реального часу (RTOS) призначений для забезпечення передбачуваного, зазвичай описуваного як детермінований, шаблону виконання. Це особливо цікаво для вбудованих систем, оскільки вбудовані системи часто мають вимоги до реального часу. Вимоги реального часу - це вимоги, які визначають, що вбудована система повинна реагувати на певну подію протягом строго визначеного часу. Гарантія виконання вимог реального часу може бути зроблена тільки в тому випадку, якщо поведінка планувальника операційної системи може бути передбачуваною, детермінованою. Традиційні планувальники реального часу, такі як планувальник, що використовується у FreeRTOS, досягають детермінованості, дозволяючи користувачеві призначати пріоритет кожному потоку виконання. Потім планувальник використовує пріоритет, щоб знати, який потік виконання запускати наступним. У FreeRTOS потік виконання називається завданням.

FreeRTOS - це клас RTOS, який розроблений таким чином, щоб бути достатньо малим, щоб працювати на мікроконтролері. Хоча його використання не обмежується застосуванням мікроконтролерів. FreeRTOS надає лише основні функції планування в реальному часі, міжзадачного зв'язку, синхронізації та примітивів синхронізації. Це означає, що її більш точно можна назвати ядром реального часу або виконавчим процесором реального часу. FreeRTOS має ряд переваг, що сприяє її широкому використанню:

- забезпечує єдине і незалежне рішення для багатьох різних архітектур і засобів розробки;
- визнана надійною операційною системою;
- є багатофункціональною і постійно підтримується розробниками;
- має мінімальний обсяг ПЗП, ОЗП і витрати на обробку;
- двійковий образ ядра RTOS буде в районі від 6K до 12K байт;
- ядро ядра RTOS міститься всього в 3 файлах на мові С.

Розберемо принцип роботи FreeRTOS починаючи з ядра. Ядро є основним компонентом операційної системи. Операційні системи, такі як Linux, використовують ядра, які дозволяють користувачам отримувати доступ до комп'ютера нібито одночасно. Кілька користувачів можуть виконувати кілька програм нібито одночасно. Кожна програма, що виконується, є завданням або потоком під контролем операційної системи. Якщо операційна система може виконувати кілька завдань таким чином, то це називається багатозадачністю. Використання багатозадачної операційної системи може спростити розробку того, що в іншому випадку було б складним програмним додатком:

- багатозадачність та міжзадачні комунікаційні можливості операційної системи дозволяють розбити складний додаток на набір менших і більш керованих завдань;
- розподіл може призвести до полегшення тестування програмного забезпечення та повторного використання коду;
- складні часові параметри та деталі послідовності можуть бути вилучені з коду програми і стати відповідальністю операційної системи.

Звичайний процесор може одночасно виконувати лише одне завдання - але завдяки швидкому переключенню між завданнями багатозадачна операційна система може створити враження, що кожне завдання виконується одночасно. Це зображено на рис. 4.3.1, який показує схему виконання трьох завдань у часі. Назви завдань записані зліва. Час рухається зліва направо, а лінії показують, яке завдання виконується в кожен конкретний момент часу. Верхня діаграма демонструє уявну модель одночасного виконання, а нижня - фактичну модель багатозадачного виконання.

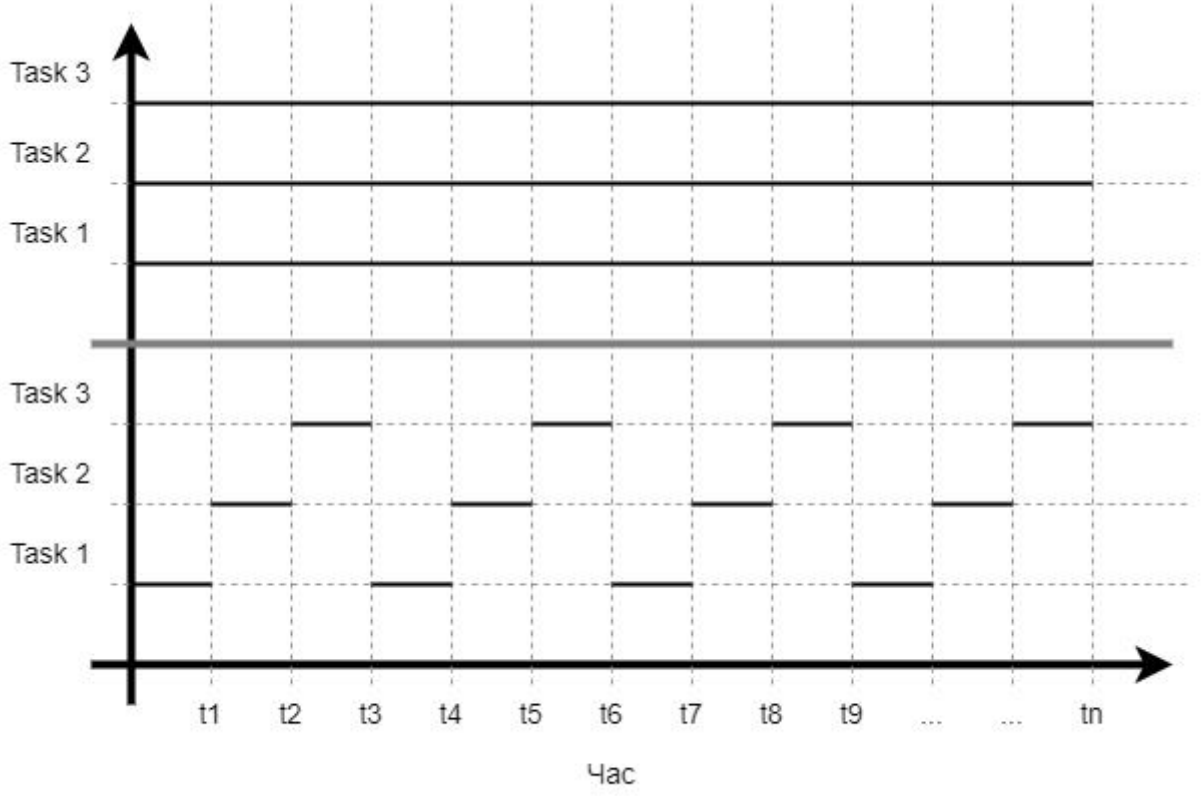


Рис. 4.3.1. Графік виконання задач операційної системи

Планувальник - це частина ядра, яка відповідає за прийняття рішення про те, яка задача повинна виконуватися в конкретний момент часу. Ядро може призупиняти, а потім відновлювати виконання завдання багато разів протягом його життя. Стратегія планування - це алгоритм, який використовується планувальником для прийняття рішення про те, яку задачу виконувати в будь-який момент часу. Стратегія багатокористувацької системи, швидше за все, дозволить кожному

завданню «справедливу» частку процесорного часу. На додаток до примусового призупинення ядром, завдання може призупинити роботу за власним бажанням. Це може бути зроблено, якщо воно хоче або відкласти виконання на певний час, або дочекатися, поки стане доступним певний ресурс або відбудеться певна подія. Заблоковане або спляче завдання не може бути виконане, і йому не буде виділено жодного часу на обробку.

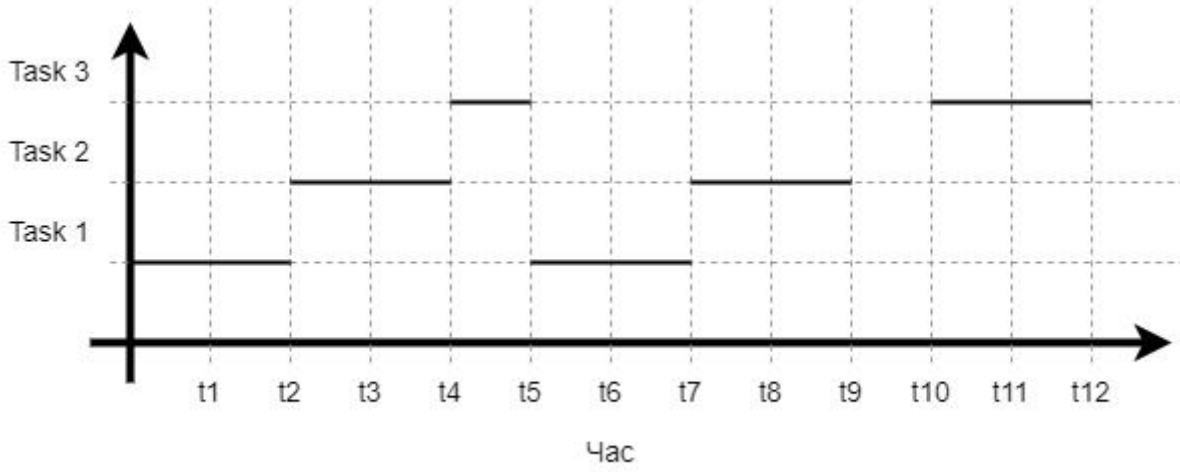


Рис. 4.3.2. Графік виконання задач за планувальником

Розглянемо невеликий приклад роботи планувальника задач, графік якого наведено на рис. 4.3.2. Вертикальна вісь містить три задачі, що виконуються в операційній системі. Горизонтальна вісь позначає абстрактні проміжки часу. Цей приклад можна розписати у вигляді списку подій:

- у момент часу «t1» виконується перша задача;
- з настанням моменту часу «t2», операційна система перемикається на виконання другої задачі;
- у момент часу «t3», подібно до «t1», виконується друга задача, під час якої блокується периферія процесора для ексклюзивного доступу;
- у момент часу «t4» відбувається перемикання на третю задачу;
- «t5», через певний час від початку третьої задачі, задача намагається отримати доступ до периферії процесора, але через блок, накладений другою

задачею, третя задача завершується достроково та перемикається на першу задачу;

- на момент часу «t8», друга задача розблоковує доступ до периферії процесора;
- розблокування доступу в другій задачі дозволяє третій задачі продовжувати роботу в момент часу «t11».

Ще одним важливим елементом роботи операційної системи, який необхідно розглянути, є перемикання контекстів. Під час виконання завдання операційна система використовує регістри процесора та звертається до оперативної та постійної пам'яті, як і будь-яка інша програма. Ці ресурси разом складають контекст виконання завдання. Задача є послідовним фрагментом коду - вона не знає, коли буде призупинена або відновлена ядром, і навіть не знає, коли це сталося. Розглянемо приклад призупинення завдання безпосередньо перед виконанням інструкції, яка підсумовує значення, що містяться в двох регістрах процесора. Поки завдання призупинено, інші завдання будуть виконуватися і можуть змінити значення регістрів процесора. Після відновлення роботи завдання не буде знати, що регістри процесора були змінені. Якби воно використовувало змінені значення, підсумовування призвело б до неправильного значення. Щоб запобігти цьому типу помилок, важливо, щоб після відновлення завдання мало контекст, ідентичний тому, що був безпосередньо перед його призупиненням. Ядро операційної системи відповідає за забезпечення цього і робить це шляхом збереження контексту завдання під час його призупинення. Коли завдання відновлюється, збережений контекст відновлюється ядром операційної системи перед його виконанням. Процес збереження контексту призупиненої задачі та відновлення контексту задачі, що відновлюється, називається перемиканням контексту. Розглянемо невеликий приклад перемикання контексту, що проілюстровано на рис. 4.3.3. У прикладі виконання поточного завдання призупиняється, оскільки воно наближається до виконання інструкції ADD. У попередніх інструкціях вже були встановлені регістри, що використовуються інструкцією ADD. Коли завдання буде відновлено, інструкція ADD буде першою інструкцією, яка буде виконана. Задача не буде знати, чи інша

задача змінила Reg1 і Reg2 за цей час. Тому про перемикання контекстів слід завжди пам’ятати піч час розробки програмного забезпечення.

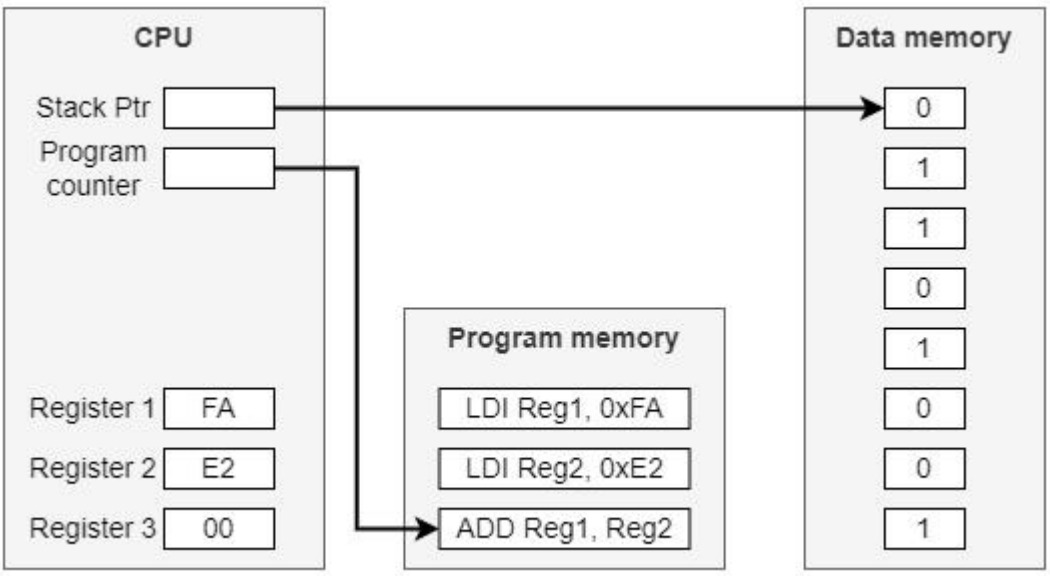


Рис. 4.3.3. Приклад регістрів під час перемикання контексту

4.4 Вимоги до програмного забезпечення пристрою

Розібравши низькорівневу частину програмного забезпечення, що дозволяє абстрагуватися від апаратної частини та програмувати мікросхему STM32F103 на більш високому рівні, слід описати ряд вимог до програмного забезпечення, що описують архітектуру, правила та поведінку, яким має відповідати програмне забезпечення. На початку необхідно зазначити правила для коду програми. Програмний код, що виконується у прототипі, відповідає стандарту MISRA C. MISRA C є еталонним стандартом для розробок з використанням мови програмування C у багатьох галузях промисловості, незалежно від того, чи є безпека першочерговим питанням, чи ні. Він однаково корисний для коду, який має вимогу бути безпечним. Кількість правил кодування для кожної мови варіюється в кожному виданні. Наприклад, MISRA C:2012 включає сто сімдесят п’ять правил кодування, з яких шістнадцять класифікуються як обов'язкові, сто двадцять як необхідні, а тридцять дев’ять як рекомендаційні. Повні матриці відповідності доступні для MISRA C та MISRA C++. Вони детально описують всі правила кодування, а також те, які з них підтримуються інструментами LDRA для MISRA C та MISRA C++

відповідно. Стандарти MISRA містять багато правил. Теоретично можна було б перевірити, чи всі вони були виконані шляхом інспектування, але це була б висококваліфікована, трудомістка і схильна до помилок робота. Програми перевірки MISRA, такі як ті, що надаються LDRA, автоматизують цей процес за допомогою статичного аналізу. Дотримання стандартів кодування MISRA сприяє:

- покращенню переносимості коду за рахунок уникнення специфічних для компілятора або платформи конструкцій;
- уникненню несподіваної поведінки програми;
- виявленню недосяжного або нездійсненного коду, що часто свідчить про дефект і потенційну вразливість системи безпеки;
- зменшенню небезпечних та ненадійних практик кодування шляхом заборони певних мовних конструкцій;
- помітному зменшенню складності програми;
- покращенню тестування програм;
- полегшенню дотримання стандартів, що застосовуються до критично важливих для безпеки та критично важливих додатків.

Наприклад, беручи до уваги два випадкові правила MISRA C:2012, ми можемо побачити два критично важливих пояснення, дотримання яких робить нашу програму біль безпечною та стабільною. Перше правило стосується порядку обробки виразів. Стандарт мови С надає дуже широку свободу дій розробникам щодо порядку обчислень у виразах. Будь-який код, чутливий до порядку обчислення, є, таким чином, залежним від компілятора, а залежний від компілятора код завжди слід вважати небезпечним. Друге правило стосується рекурсії. Час від часу зручним способом вираження алгоритму є використання рекурсії. Однак, якщо рекурсія не контролюється дуже жорстко, існує небезпека переповнення стеку, що може, в свою чергу, призвести до дуже важко локалізованих помилок. У критичному з точки зору безпеки коді рекурсії слід уникати. Це тільки два випадково взятих правила, які вже застерігають від помилок та сприяють створенню безпечного коду.

Також код програми має відповідати певному єдиному стилю. Стиль кодування - це набір умовностей щодо форматування коду. Наприклад, стиль

кодування визначає як необхідно називати змінні, використовувати пробіли або табуляцію для відступів, де розміщувати коментарі, та ін. Послідовне використання одного стилю у всьому коді полегшує його читання. Код, який легко читається, легше зрозуміти як розробнику, так і потенційним співавторам. Тому дотримання стилю кодування знижує ризик помилок і полегшує спільну роботу над програмним забезпеченням. Існує різноманітна кількість різних стилів для різних мов програмування. Також, ніхто не забороняє визначати свій власний стиль. Але найкращі практики радять використовувати найбільш популярні стилі. Програмний код прототипу відповідає стилю кодування Google.

Визначивши правила для програмного коду, необхідно описати архітектуру програми. Архітектура будь-якої системи має описувати принципи її роботи, а також взаємозв'язки її елементів між собою та зовнішньою середою. Етап створення архітектури є важливим і не може бути пропущеним. Адже наявність системної архітектури дає змогу: провести попередній аналіз системи ще до початку процесу розробки, спланувати та вибудувати найбільш оптимальний процес розробки, задокументувати процеси розробки та обслуговування, створити необхідні інструменти для розробки, подальшого аналізу, а також тестування. Згідно специфіки програми, ми можемо розділити її на кілька функціональних блоків: управління, керування додатковою пам’яттю, керування дротовим з’єднанням, керування бездротовим з’єднанням. Блоки зображені графічно на рис. 4.4.1.

Рис. 4.4.1. Блоковий розподіл програми

Відповідно головний функціональний блок управління має пряме з’єднання з кожним іншим функціональним блоком. Це потрібно для обміну даними та прийняття рішень відповідно до отриманих даних. Таким чином блок управління

реалізує необхідні алгоритми для роботи стеганографічного захисту та проводить постійне опитування трьох блоків керування. Блок керування додатковою пам’яттю відповідно реалізує всі необхідні методи для роботи з додатковою пам’яттю. Блок керування дротовим з’єднанням реалізує методи для роботи із дротовим з’єднанням, а блок керування бездротовим – для бездротового з’єднання відповідно. Кожен з блоків керування має інтерфейс через який ведеться комунікація з блоком управління.

Блок управління очікує на команду для початку стеганографічного захисту від блоків керування дротовим та бездротовим з’єднанням, періодично опитуючи їх на наявність активного запиту від клієнта. У разі отримання команди на початок стеганографічного захисту, блок управління звертається до блоку керування додатковою пам’яттю, щоб отримати доступ до секретного зображення та зображення-контейнера. Після встановлення доступу до зображень, блок управління декодує зображення відповідно до їх формату, щоб отримати доступ до значень пікселів. Коли пікселі доступні для обробки, то блок управління послідовно зчитує блоки контейнера, на кожній ітерації паралельно вичитує відповідну кількість пікселів, що може бути прихована у одному блоці, із секретного зображення та піддає пікселі секретного зображення шифруванню. Слід зазначити, що перед початком роботи з пікселями, блок управління обраховує кількість пікселів секретного зображення та контрольну суму всіх пікселів, з цих даних формується сервісна інформація. Під час обробки першого блоку пікселів контейнера сервісна інформація додається до пікселів секретного зображення та утворює зсув на дев’ять байт. Коли байти секретного зображення зашифровані, то блок управління зчитує кожен байт почергово та заміняє молодші біти байтів контейнера, використовуючи таблицю перестановок, щоб визначити необхідний порядковий номер байту, молодший біт якого необхідно замінити. Такий спосіб дозволяє провести уявну перестановку пікселів у блоці контейнера, що не відрізняється за результатом від реальної перестановки, але запобігає використанню великої кількості машинного часу на реальну перестановку. Після закінчення стеганографічного захисту, блок

управління кодує вихідне зображення відповідно до необхідного формату та пересилає результат через блоки керування дротовим чи бездротовим з’єднанями.

Блок керування додатковою пам’яттю реалізує файлову систему використовуючи бібліотеку FatFS. FatFS - це легка програмна бібліотека для мікроконтролерів та вбудованих систем, що реалізує підтримку файлових систем FAT/exFAT. FatFS не залежить від платформи і легко переноситься на багато апаратних платформ, таких як 8051, PIC, AVR, ARM, Z80. FatFS розроблений як рівень файлової системи, який є агностичним до платформи та носіїв інформації, з якими він використовується. Це досягається шляхом надання інтерфейсу доступу до носія, який використовується для зв'язку з модулем управління пристроєм зберігання даних, який надається реалізатором. Це означає, що FatFS може працювати з будь-яким фізичним пристроєм, таким як SD-карта або жорсткий диск на будь-якій платформі, яка може виконувати простий код на мові С, якщо реалізатор надає інтерфейс модуля управління. Архітектура бібліотеки FatFS логічно розділяє абстракції користувацького додатку та платформно-залежного коду. Користувацький додаток та низькорівневий рівень дискового вводу/виводу повинні бути додані реалізатором. Також архітектура бібліотеки передбачає, що в системі може бути декілька пристроїв зберігання даних з різними драйверами і бібліотека може працювати в багатопотоковій операційній системі. На рівні програми приховано, який саме фізичний носій використовується.

Блок керування бездротовим з’єднанням та блок керування дротовим з’єднанням реалізують комунікацію через UART. Але якщо для дротового з’єднання ця реалізація досить проста, бо дротове з’єднання на високому рівні має пряме з’єднання з клієнтом, то для бездротового все набагато складніше через необхідність комунікації із WiFi модулем, що є посередником між клієнтом та пристроєм. WiFi модуль керується через спеціальні AT-команди. AT-команди WiFi модуля ESP8266 відповідають за управління всіма операціями модуля, такими як перезавантаження, підключення до WiFi, зміна режиму роботи та ін. AT-команди ESP8266 можна розділити на чотири типи:

- тестові;

- команди запиту;
- команди набору;
- команди виконання.

Тестові команди використовуються для отримання параметрів команди та діапазону допустимих значень. Команди запиту повертають поточне значення параметрів команди. Команди набору використовуються для задання значень параметрів в командах, а також для запуску команд. Команди виконання запускають команди без параметрів. У таблиці 4.4.1 наведений опис команд з прикладами.

Таблиця 4.4.1

Приклади AT-команд

Тип команди	**Формат команди**	**Опис**
Тестові команди	AT+TEST=?	Повертає діапазон значень параметрів
Команди запиту	AT+TEST?	Повертає поточне значення
Команди набору	AT+TEST=param1, …	Встановлює конфігурацію
Команди виконання	AT+TEST	Виконує дію

РОЗДІЛ 5
ПРОГРАМНА ЧАСТИНА ВИСОКОГО РІВНЯ

До програмної частини високого рівня відноситься програмне забезпечення, що виконується на комп’ютері. Програмне забезпечення поділяється на дві частини. Перша відповідає за комунікацію з пристроєм за допомогою послідовного COM порту комп’ютера. Друга реалізовує користувацький інтерфейс для роботи з пристроєм через WiFi з’єднання.

5.1 Опис програмної реалізації клієнта

Для забезпечення підтримки операційних систем Windows, починаючи з сьомої версії, у якості програмної технології використаний .NET Framework 4.5. Код програми написаний мовою C#. Використана парадигма – об’єктно орієнтоване програмування. Графічний інтерфейс створений за допомогою фреймворку WPF (Windows Presentation Foundation). Код програми поділений на модулі, кожен модуль представлений у вигляді окремого класу. Таке розділення є зручним для розробки програмного забезпечення, а також його налагодження та подальшої підтримки. Прототип інтерфейсу програми в режимі приховування інформації зображено на рис. 5.1.1.

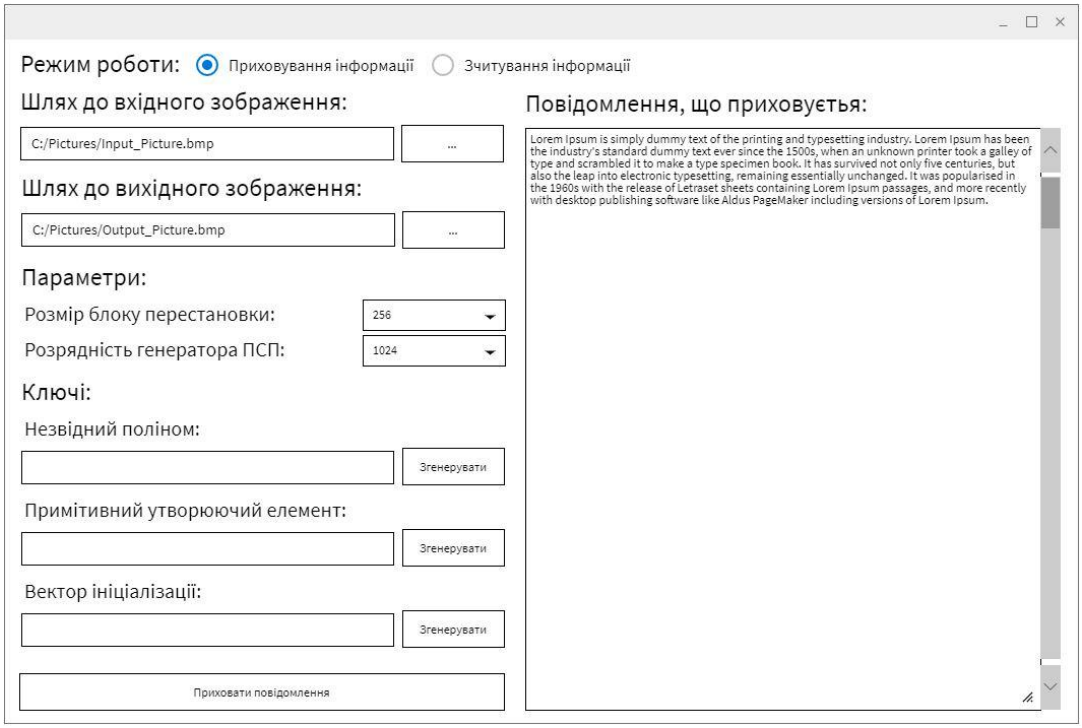


Рис. 5.1.1. Прототип інтерфейсу програми у режимі приховування інформації

Перемикач режиму роботи є елементом під назвою «радіокнопка», який представлений у просторі імен «System.Windows.Controls», класом «RadioButton». Важливо задати властивість «GroupName», де зазначається ім’я групи, до якої входять певні радіокнопки. При перемиканні радіокнопки змінюється властивість «IsChecked», яка є важливою властивістю, бо саме вона дає змогу відслідковувати активне положення радіокнопки. Вікна, де вказуються шляхи до вхідного та вихідного файлів відрізняються лише внутрішньою властивістю, яке зберігає відповідний шлях. Отже, достатньо розглянути тільки одне вікно. Текстове поле, що містить шлях до файлу, представлено класом «TextBox». Поруч знаходиться кнопка, яка відкриває вікно вибору файлу для відкриття. Властивість текстового поля «Text», яка містить строкове значення цього поля, допомагає відслідкувати відкриття користувачем файлу, властивість може змінюватись або при ручному вводі з клавіатури, або програмно після вибору файлу через вікно відкриття файлів. Параметри, такі як, розмір блоку перестановки та розрядність генератора ПВП обираються за допомогою випадаючого списку, клас «ComboBox». Відслідковується зміна вибраного елементу зі списку за допомогою властивості «SelectedItem». Вікна, де вказуються ключі подібні до вікон вибору шляху до файлу, але кнопка

«Згенерувати» виконує дію запуску стохастичного моделювання, яка програмно створює відповідний ключ. У правій частині вікна програми розташоване текстове поле, в яке вводиться користувачем повідомлення, що потрібно приховати. Для такого текстового поля, властивостям «TextWrapping» та «VerticalScrollBarVisibility» задані значення «Wrap» та «Visible» відповідно. Кнопка «Приховати повідомлення» запускає виконання стеганографічного алгоритму та збереження вихідного зображення. В режимі зчитування інформації графічний інтерфейс програми має невеликі відмінності від графічного інтерфейсу в режимі приховання інформації. А саме, приховане вікно вибору шляху до вихідного зображення, бо в режимі зчитування повідомлення ніякого вихідного зображення не очікується. Також відсутні кнопки для стохастичного формування ключів, бо для розшифровки потрібно використовувати попередньо отримані ключі. Текстове поле з повідомленням є недоступним для ручого вводу символів з клавіатури. Його вміст заповнюється програмно після вичитування інформації з контейнера. Прототип інтерфейсу програми в режимі зчитування інформації зображено на рис. 5.1.2.

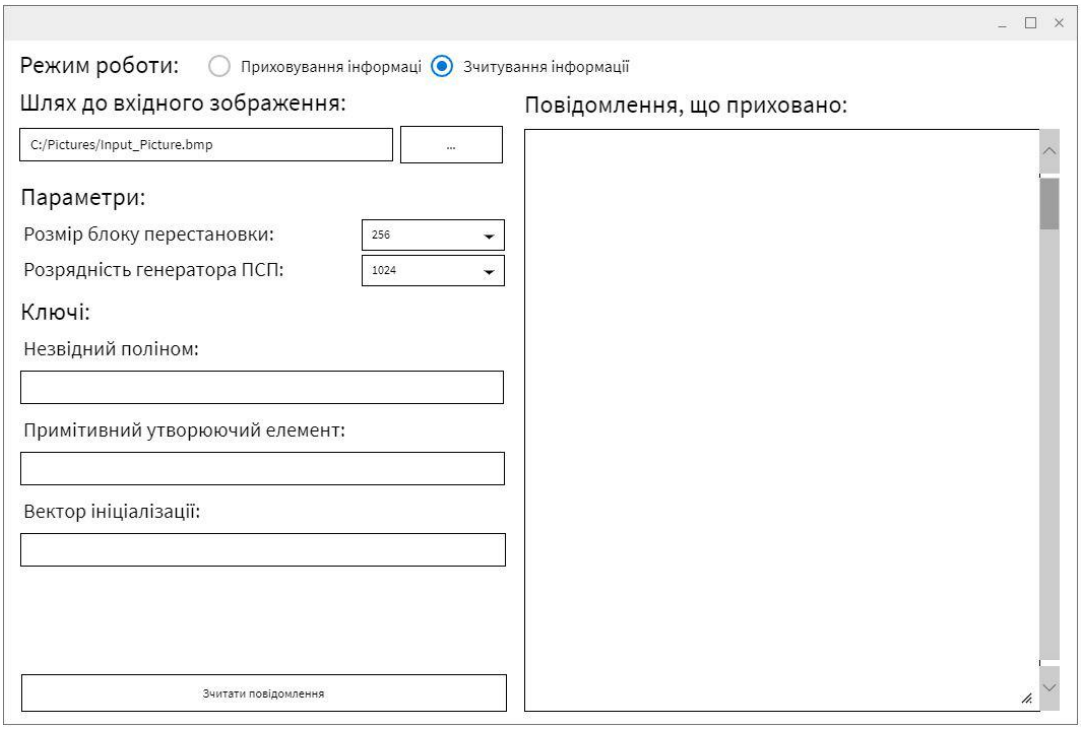


Рис. 5.1.2. Прототип інтерфейсу програми у режимі зчитування інформації

При описі елементів графічного інтерфейсу не раз зазначалося відслідковування змін у властивостях відповідних елементів. Це досягається за допомогою використання шаблону проєктування MVVM. Сам шаблон MVVM налічує три частини: графічний інтерфейс (View), дані програми (Model), які модифікуються певною логікою, а також спеціальний прошарок (ViewModel), що використовується для зв'язку, безпосередньо, графічного інтерфейсу та даних. Зв'язування властивостей елементів з приватними властивостями у коді відбувається за допомогою інтерфейсу «INotifyPropertyChanged», представленого у просторі імен «System.ComponentModel». А опрацювання команд, що подаються користувачем через графічний інтерфейс, наприклад натиск кнопки, можливе через використання інтерфейсу «ICommand» з простору імен «System.Windows.Input». Таким чином кожна властивість елементу графічного інтерфейсу, що зв'язана з даними програми, має відповідну публічну властивість в коді програмі. Ця властивість в коді як раз і є частиною прошарку «ViewModel». Коли користувач, наприклад, змінює вміст текстового поля, то змінюється значення властивості «Text», так як до цієї властивості прив'язана властивість в коді, то зміни чіпляють і другу властивість. Отже, дані введені користувачем доступні у бекенд частині програми. Зв'язок властивостей двосторонній, а це значить, що логіка програми також може змінювати значення в графічному інтерфейсі. Подібно до елементів вводу інформації, таких як текстове поле, працюють і елементи керування, такі як кнопки, тільки при натиску на кнопку запускається прив'язана до неї лямда-функція.

У попередніх абзацах було сказано, що програма поділена на окремі частини. Кожна частину виконує певну функцію, яка є однією з основних чи допоміжних функцій або під-функцій програми. Щоб краще розподілити програму на незалежні частини, представимо процес її виконання у вигляді ланцюга викликів основних функцій, що зображений на рис. 5.1.3.

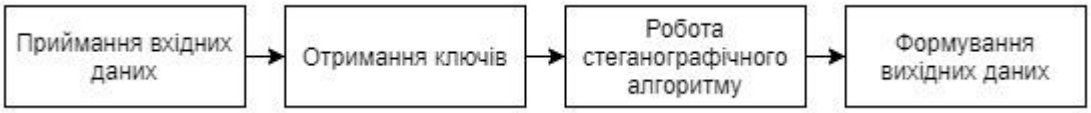


Рис. 5.1.3. Діаграма виклику основних функцій

Розглянемо основні функції та опишемо їх розподіл на незалежні частини. Першою основною функцією є приймання вхідних даних, вхідні дані програми є, завжди, зображеннями. Функція приймання вхідних даних поділяється на дві частини. Перша має необхідний набір методів для відкриття зображення дозволеного типу, а саме растрове зображення у форматі BMP. Після відкриття ми матимемо масив байтів, певної довжини. А друга містить методи для представлення цього масиву байтів у вигляді об'єкту зображення. Коли вхідні дані опрацьовані, програма отримує ключі до стегаографічного алгоритму, ця функція поділена на кілька частин: генератор незвідних поліномів, генератор примітивних утворюючих елементів, генератор векторів ініціалізації, утворювач загального ключа, який створює об'єкт загального ключа з отриманих за допомогою відповідних генераторів або введених вручну ключів. Зі сформованим об'єктом загального ключа та об'єктом зображення ініціалізується стеганографічний алгоритм. Стеганографічний алгоритм поділений на більшу кількість частин, тому на рис. 5.1.4 наведено, додатково, діаграму викликів основних під-функцій.

Рис. 5.1.4. Діаграма виклику основних під-функцій при приховуванні повідомлення

Розберемо під-функції стеганографічного алгоритму та частини на які вони поділені. Зчитування ключів складається з однієї частини, задає ключі для ініціалізації генератора ПВП. Коли ключі до генратора задані, то утворюється, безпосередньо, об'єкт генератора ПВП. Такий об'єкт має можливість видавати псевдовипадкове число певної розрядності. Далі рахується контрольна сума повідомлення. Після отримання можливості генерації псевдовипадкового числа, починається процес шифрування повідомлення. Кожен байт повідомлення шифрується за допомогою гамування його зі згенерованим числом. Після шифрування повідомлення

розраховується розмір котейнера, який зможе вмістити в собі це повідомлення. На виході маємо масив байтів потрібної довжини. Коли контейнер готовий, створюється об'єкт з алгоритмом перестановок. При створені такого об'єкту рахується кількість блоків, що вміщується у контейнері, якщо вміщується нецілe число, то контейнер доповнюється потрібною кількістю байтів. Після поділення контейнера на блоки, для кожного блоку генерується таблиця перестановок, використовуючи генератор псевдовипадкових чисел. За допомогою цього об'єкту відбувається перестановка пікселів, відповідно таблиць перестановок, а також відновлення положень пікселів. Приховується зображення за допомогою звичайного циклу, який перебирає біти зашифрованого повідомлення, а потім підміняє молодший біт байту контейнера. Однак стеганографічний алгоритм може працювати як для приховування, так і для зчитування повідомлення. Діаграма викликів основних під-функцій у режимі зчитування зображена на рис. 5.1.5.

Рис. 5.1.5. Діаграма виклику основних під-функцій при зчитуванні повідомлення

З діаграми бачимо, що частина під-функцій співпадає з тими, які використовувались для приховування. Отже, розберемо тільки ті, що не співпадають. Зчитування повідомлення являє зворотній процес до приховання, а саме, у циклі перебираються байти контейнера і з молодших бітів формується ще один масив байтів, який і є нашим повідомленням. Але зчитане повідомлення зашифроване, для того щоб розшифрувати потрібно піддати кожен байт повторній операції гамування, у результаті отримаємо вихідне повідомлення. Не завжди ми можемо бути впевненими, що зчитане повідомлення відповідає тому, що приховувалось. Тому для зчитаного повідомлення рахується контрольна сума і порівнюється з контрольною сумою прихованого повідомлення, яка була також захована у контейнері.

Розглянувши функцію стеганографічного алгоритму, залишається остання, яка виконує формування вихідних даних. Ця функція поділяється на створення об’єкту вихідного зображення та об’єкту вихідного повідомлення, відповідно до режиму роботи програми. Поміж основних функцій є ще і допоміжні, такі як перетворення типів даних, відкриття та збереження файлів, контроль блокування чи приховування елементів графічного інтерфейсу, перевірка введених даних.

5.2 Архітектура програмної реалізації

Програмний продукт, опис якого наведено у даному розділі, дозволяє провести перевірку способу стеганографічного захисту на вирішення задач прихованої передачі інформації, а також прихованого збереження інформації, які є одними з основних задач, до яких може бути застосована стеганографія. У цьому підрозділі описано архітектуру програмного продукту, яка розділена на три групи опису: функціональну, логічну та фізичну.

До функціональної групи входять описи програмного продукту на найвищому рівні, а саме взаємодію користувача з ним. На цьому етапі визначається задача чи ряд задач, що може вирішити користувач за допомогою системи. Отже формується основна функція програмного продукту. Даний програмний продукт виконує функцію перевірки практичного застосування способу стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера. Програма має показати наскільки даний спосіб підходить для вирішення задач прихованої передачі та збереження інформації, а також допомогти провести аналіз які додаткові задачі можуть бути вирішені.

Результат, як раз, і має сформувати вектори розвитку та вдосконалення програмного продукту. Користувач має змогу зашифрувати певну інформацію та приховати її у растровому зображенні довільного розміру. В роботі програми, теоретично, можливі наступні сценарії:

- розмір секретних даних менший чи дорівнює розміру контейнера, а також розмір контейнера може бути поділений на ціле число блоків;
- розмір секретних даних більший за розмір контейнера;
- контейнер не вміщує цілої кількості блоків.

Програма опрацьовує кожен з наведених сценаріїв. Взаємодія з програмою відбувається через графічний інтерфейс. Програмний продукт має повну підтримку операційних систем Windows, починаючи з версії 7, а також вимоги, не більші ніж характеристики стандартного ПК.

Логічна група описів налічує описи інтерфейсів взаємодії, поведінку системи відносно зовнішніх та внутрішніх даних, а також описує потоки цих даних в системі. До зовнішніх даних програми входять: текст, який приховується, растрове зображення у форматі BMP, яке є контейнером для зберігання секретного повідомлення, розмір блоку перестановки, розрядність генератора псевдовипадкових чисел, ключі відповідної розрядності для генератора, спосіб отримання ключа, шлях для збереження вихідного зображення, яке містить приховане зашифроване повідомлення. Внутрішні дані налічують: ключі, що формуються за допомогою стохастичного моделювання, контрольна сума секретного повідомлення та його розмір, саме зашифроване секретне повідомлення, вихідний контейнер, наповнений бітами секретного повідомлення, а також розмір вихідного контейнера. Користувач має змогу вводити вхідні дані за допомогою спеціально відведених вікнах графічного інтерфейсу. Потік даних та поведінка програми являє наступну послідовність. Першим кроком користувач завантажує зображення у форматі BMP, програма декодує завантажене зображення. Після успішного декодування програма перевіряє зображення на відповідність умовам відсутності алгоритму стиснення та наявності сторонньої палітри кольорів. Якщо зображення не відповідає хоча б одній з умов, то програма повідомляє про це користувача. У випадку, коли зображення не вдалось декодувати, програма також має повідомити користувача про несправність файлу зображення. Наступними даними після вхідного зображення, користувачем обираються розмір блоку перестановки, а також розрядність генератора псевдовипадкових чисел. Коли обрана розрядність генератора псевдовипадкових чисел, користувач обирає спосіб формування ключів, серед яких, ручний спосіб формування або за допомогою стохастичного моделювання. Якщо обраний ручний спосіб формування ключів, то необхідно додатково задати у відповідних вікнах інтерфейсу самі ключі. У випадку

стохастичного моделювання ключів, програма сама сформує потрібні ключі. Останнім кроком користувач вводить текст, що потрібно приховати та обирає шлях для збереження вихідного зображення. Стадія опрацювання зовнішніх даних замінюється стадією формування внутрішніх даних. На другій стадії програма створює внутрішні дані для поточної сесії. Спочатку розраховується розмір вихідного зображення. Якщо вхідне зображення одразу задовольняє умови, то розмір вихідного зображення дорівнює розміру вхідного. Виходячи з отриманого розміру вихідного зображення, формується контейнер. Якщо був обраний спосіб формування ключів, то програма розраховує потрібні ключі. Маючи ключі, повідомлення та контейнер, програма спочатку шифрує зображення, потім контейнер підлягає перестановкам, після чого зашифроване повідомлення підставляється до контейнера. Контейнер повертається у початкове положення, чим стохастично розподіляє повідомлення по контейнера. Коли вихідне зображення готове, то програма зберігає його по заданому шляху. Зворотній процес, отримання прихованих даних, являє обернену процедуру, єдиною відмінністю є те, що на вхід подається зображення, яке містить приховані дані, а ключі мають відповідати початковим.

Фізична група описів представляє програмний продукт з точки зору розробки. Описується саме рішення, використовуючи функціональні вимоги до програмного продукту. Функціональні вимоги описують поведінку програми, базуючись на цих вимогах здійснюється розробка програмного продукту. Функціональні вимоги розподіляються на: вимоги, що описують функції програми, вхідні дані та їх потік, вихідні дані, опис умов, необхідних для виконання функції, опис опрацювання помилок та даних, що не відповідають вимогам. Функції програмного продукту розподілені на функції обробки даних та функції стеганографічного алгоритму. Програма приймає шлях до вхідного зображення, який є строковим типом даних. Якщо вхідне зображення недоступне по заданому шляху, програма оброблює цю помилку та повідомляє користувача. Програма завантажує вхідне зображення та перевіряє його на відповідність умовам, якщо зображення не підходить, то програма повідомляє користувача про неможливість використання поточного зображення.

Програма надає можливість обрати розмір блоку перестановки та розрядність генератора псевдовипадкових чисел за допомогою випадаючого списку, з переліченим у ньому можливими варіантами. Розміри блоку перестановки налічують два варіанти: 256, 65536. Варіантів розрядності генератора псевдовипадкових чисел налічується чотири: 64, 256, 1024, 2048. Вибір способу формування ключів відбувається за допомогою відповідного перемикача в інтерфейсі. Виходячи із заданої розрядності генератора та способу формування ключів, програма має блокувати можливість введення ключів вручну при способі стохастичного моделювання, а при ручному способі перевіряти відповідність введених ключів вимогам. У режимі отримання прихованих даних, на вхід програма очікує шлях до зашифрованого контейнера, а вікно вводу тексту замінюється вікном виводу секретного повідомлення. Перемикач способу формування ключів завжди залишається в положенні ручного формування, адже потрібно використовувати попередньо сформовані ключі. Описавши функції обробки даних, перейдемо до опису функцій стеганографічного алгоритму. Функції стеганографічного алгоритму поділяються на: функцію стохастичного моделювання ключів генератора, функцію генератора псевдовипадкових чисел, функцію перестановок пікселів, функцію шифрування секретного повідомлення, функцію наповнення контейнера, функцію перевірки цілісності повідомлення. Програма має змогу стохастично змоделювати ключі до генератора псевдовипадкових чисел, а також, обов’язково виконується перевірка ключів на відповідність вимогам до кожного ключа. Якщо програма працює з вручну введеними ключами, то вони також проходять таку перевірку. У разі невідповідності вручну введених ключів, програма повідомляє користувача.

5.3 Стратегія тестування

Під час розробки програмного забезпечення велику роль відіграє тестування. Адже саме тестування дає змогу перевірити чи відповідає код програми заданим вимогам. Дуже важливо проробити стратегію тестування, бо стратегія тестування описує мету тестування, а також спосіб, у який буде тестуватись програма. Стратегія тестування даного програмного продукту нескладна. Тестування обмежується лише перевіркою компонентів. Такий тип тестування передбачає розбиття програми на

окремі модулі, одиничні елементи програми, які не залежать один від одного. До кожного такого модуля пишеться окремий тест, який може розділятися на кілька частин, це залежить від кількості варіантів роботи модуля, які потрібно покрити тестами. Метою тестування є перевірка відповідності кожного модуля вимогам. Також важливим є критерій покриття коду тестами. Саме цей критерій дає змогу оцінити наскільки система протестована. Теоретично, щоб протестувати систему повністю, кількість тестів має дорівнювати повній кількості всіх логічних входів системи. Таке покриття на великій кількості логічних входів системи є практично неможливим, приклад розрахунку кількості тестів T_{count} відносно кількості логічних входів I_{count} наведено у виразі (5.1).

$$T_{count} = 2^{I_{count}} = 2^{32} = 4294967296 \quad (5.1)$$

З прикладу бачимо, що практично протестувати систему, у якій є тридцять два логічні входи, неможливо. Тому існують різні критерії достатнього покриття. Один з таких критеріїв називається MC/DC, саме цей критерії застосовується для покриття даного програмного коду. Існує багато різних формулювань критерія покриття MC/DC, ми ж сформулюємо його наступним чином. Якщо вплив кожного класу еквівалентності на логіку, а також кожне граничне значення протестовані, тоді ця логіка вважається покрита. Розглянемо невеликий приклад, нехай потрібно протестувати функцію $F(A, B, C, D)$, яка приведена у виразі (5.2).

$$F(A, B, C, D) = A \mid B \mid C \mid D \quad (5.2)$$

Якщо покривати функцію $F(A, B, C, D)$ за кількістю логічних входів, то потрібно шістнадцять тестів, більша частина з яких не несе реального корисного навантаження, а тільки перевіряє вже перевірене. При використанні достатнього критерію покриття, нам знадобиться лише п’ять тестів. Потрібно протестувати, що при будь-якому вході з ненульовим значенням, функція $F(A, B, C, D)$ повертає також ненульове значення, на це нам потрібно чотири тести. А також ще один, щоб перевірити, що функція $F(A, B, C, D)$ повертає нульове значення, коли всі входи також мають нульові значення. Отже, для даного прикладу бачимо, що при використанні достатнього критерію покриття потрібно майже в три рази менше тестів, при чому результат тестування залишається незмінним.

Розібравши загальний приклад покриття коду тестами, розберемо також реальний приклад, покривши одну з частин коду. Наприклад перевірку полінома на незвідність. Для цього представимо його у вигляді блок-схеми, яка зображена на рис. 1.5.1. З блок-схеми бачимо, що вхідними даними є поліном степені n, а на виході отримуємо булеве значення, що вказує на незвідність полінома. Також є три логічні конструкції, які потрібно покрити тестами. Почнемо з першої, яка перевіряє довжину полінома, непарність його ваги, наявність ненульового молодшого та старшого коефіцієнтів. Для покриття цієї частини логіки потрібно подати чотири поліноми, які будуть задовольняти тільки одну логічну умову, після чого отримати оцінку що поліном не відповідає умові. А для покриття останніх двох логічних конструкцій, потрібно подати ще два поліноми, які задовольняють усі умови самої першої логічної конструкції. Один з цих поліномів має бути завідомо незвідним, а інший зіставним. Результат має показати для незвідного полінома позитивну відповідь, а для зіставного полінома негативну.

Частиною стратегії тестування, також, є фреймворк для тестування. Фреймворк для тестування – це певний набір інструментів та бібліотек, які дозволяють створювати автоматизовані тести для програмного забезпечення. До функцій фреймворку для тестування входять:

- завдання єдиного формату тестів;
- зручне задання тестових наборів даних;
- підключення необхідних бібліотек або їх симуляція;
- автоматизований запуск тестів;
- вивід результатів тестування.

За допомогою фреймворку для тестування збільшується швидкість написання тестів та стандартизується формат самих тестів, за рахунок чого спрощується підтримка тестів у майбутньому і повторне використання коду. Також, при стандартизованій системі створення тестів їх простіше запускати на різних платформах та читати звіт тестування. Кожен фреймворк для тестування поділений на три частини: оперування даними, створення бізнес-логіки, ядро самого фреймворку. Спрощена структура фреймворку зображена на рис. 5.3.1.

Рис. 5.3.1. Спрощена структура фреймворку для тестування

Рівень керування даними включає у себе заготовлені данні для тестування, а також користувацькі файли конфігурації. На цьому рівні задаються вхідні дані, дані, які очікуються, а також описується та налаштовується оточення. На рівні опису логіки створюються самі тести, використовуючи функції, які представлені у ядрі фреймворку. Також на цьому рівні задається симуляція оточення, якщо є потреба у ньому. Останній рівень, ядро фреймворку, містить основні методи, класи, функції для роботи з фреймворком. Існує багато готових рішень, які дозволяють створювати автоматизовані тести. Серед найпопулярніших фреймворків, що підтримують мову C# існують: xUnit, NUnit, MSTest. Для створення тестів до програмного продукту було використано фреймворк MSTest. Цей фреймворк розроблений компанією Microsoft, та підтримується у середовищі розробки Visual Studio, що робить його зручним для інтеграції у проект.

5.4 Опис користувацького інтерфейсу пристрою

Користувацький інтерфейс, що використовується для взаємодії з пристроєм через WiFi з’єднання, використовує протокол HTTP для обміну інформацією. HTTP - це протокол для отримання ресурсів, таких як HTML-документи. Він є основою

будь-якого обміну даними в мережі і являє собою протокол клієнт-сервер, тобто запити ініціюються одержувачем, зазвичай веб-браузером. Повний документ відновлюється з різних отриманих піддокументів, наприклад, тексту, опису макета, зображень, відео, скриптів та ін. Клієнти та сервери спілкуються, обмінюючись окремими повідомленнями. Повідомлення, що надсилаються клієнтом, зазвичай веб-браузером, називаються запитами, а повідомлення, що надсилаються сервером у відповідь, називаються відповідями. З'єднання контролюється на транспортному рівні, і тому принципово не входить до сфери застосування HTTP. HTTP не вимагає, щоб основний транспортний протокол був заснований на з'єднанні. Він лише вимагає, щоб він був надійним або не втрачав повідомлення. Серед двох найпоширеніших транспортних протоколів, TCP є надійним, а UDP - ні. Тому HTTP покладається на стандарт TCP. Перш ніж клієнт і сервер можуть обмінятися парою HTTP-запитів/відповідей, вони повинні встановити TCP-з'єднання, процес, який вимагає декількох етапів. Поведінка HTTP/1.0 за замовчуванням полягає у відкритті окремого TCP-з'єднання для кожної пари HTTP-запиту/відповіді. Це менш ефективно, ніж спільне використання одного TCP-з'єднання, коли кілька запитів надсилаються у великій послідовності. Блок-діаграма компонентів, що використовуються для роботи користувацького інтерфейсу, наведена на рис. 5.4.1.

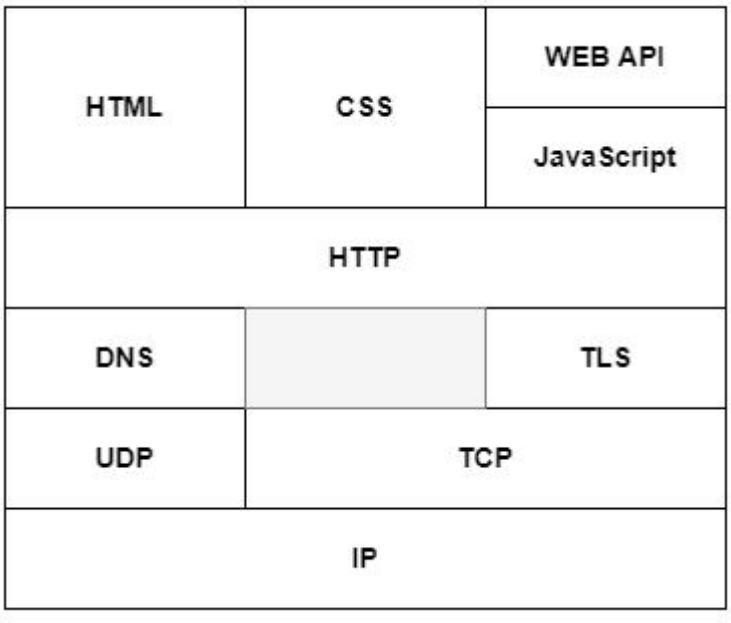


Рис. 5.4.1. Стек компонентів для роботи користувацького інтерфейсу

Користувацький інтерфейс дозволяє керувати пристроєм для зміни параметрів налаштування пристрою, а також отримати секретне повідомлення, приховане у контейнері. Коли отримується секретне повідомлення, то через канал зв’язку передається зашифрований контейнер, який потім розшифровується клієнтом у веб-браузері. У такій системі, пристрій виступає сервером, а веб-браузер клієнтом. Ключі для стеганографічного захисту можна виставити, використовуючи користувацький інтерфейс. Веб-інтерфейс має перевагу, так як він може бути відкритим на будь-якому пристрої із веб-браузером.

РОЗДІЛ 6
ОХОРОНА ПРАЦІ

Робоче місце - місце, де працівник перебуває постійно або тимчасово у зв'язку з трудовою діяльністю. Робоче місце для працівників, які займаються складанням та інтерпретацією деталей для електронного робота. Приміщення, достатньо велике для розміщення монтажного столу площею 1 метр на 2 метри та 3D-принтера. Таке приміщення має вентиляцію та хороше освітлення для контролю якості під час складання.

6.1 Аналіз шкідливих та небезпечних виробничих факторів при виробництві електронного пристрою

При роботі з електронним обладнанням працівники піддаються впливу наступних небезпечних і шкідливих виробничих факторів. Підвищена температура поверхні обладнання, тобто паяльної станції. Температура від 245 до 300 градусів за Цельсієм. Такі температури можуть бути потенційно небезпечними при контакті зі шкірою і викликати серйозні опіки. Підвищення напруги в електричному ланцюзі або поява їх на обладнанні, що знаходиться під напругою, в результаті чого підвищується ймовірність проходження струму через тіло людини. Це може статися

при необережному поводженні з паяльною станцією. Розпечений жало паяльника може розплавити ізоляцію кабелю живлення і викликати коротке замикання. Це також може стати причиною пожежі. Наявність токсичних і канцерогенних факторів при роботі з паяльною станцією. Припій - це метал, сплав або суміш оксидів, що використовується для з'єднання металевих деталей, мінеральної кераміки та інших деталей, з жерстю тощо. До складу припою можуть входити такі хімічні складові, як олово, свинець, цинк, мідь, нікель, фосфор, срібло, кадмій та ін. Флюс - це речовина, що додається в розплавлений метал для видалення оксидів і сторонніх шлаків, або в процесі пайки для запобігання окислення поверхні металу. Кожен з цих елементів, в залежності від складу того чи іншого флюсу, може випаровуватися в легенях і негативно впливати на організм людини.

Монотонність роботи, перевантаження зорових аналізаторів, психічне напруження, дискомфорт і статична поза. Постійна концентрація уваги на дрібних об'єктах під час збирання та паяння виробу. Робота вимагає хорошої зорової концентрації під час контролю якості. Можливе зниження швидкості та якості роботи, швидка втомлюваність, зниження реакції та ослаблення уваги, з часом негативно впливає на якість зору, при тривалому незмінному навантаженні. Тривале сидіння в одній позі на робочому місці, яка не змінюється з плином часу, може негативно вплинути на здоров'я.

Недостатнє штучне та природне освітлення. Недостатнє освітлення цеху, де виготовляється пристрій може викликати проблеми з якістю роботи і здоров'ям працівника. Наприклад, погана концентрація уваги, втома, головні болі, зниження реактивності та уповільнення швидкості роботи. Підвищений ризик порушення техніки безпеки при роботі з паяльною станцією.

6.2 Організаційні та конструктивно-технологічні заходи для зниження впливу шкідливих виробничих факторів

Для забезпечення безпеки працівника роботодавець і працівник повинні дотримуватися вимог з охорони праці при паянні електрифікованим інструментом. Дотримуючись усіх цих правил, досягаються найоптимальніші умови для працівника. Вимоги безпеки праці при паянні електрифікованим інструментом:

1. Паяння електрифікованим інструментом (далі - паяльник) необхідно виконувати відповідно до вимог Інструкції з охорони праці під час виконання монтажних робіт інструментом і пристроями, затвердженої наказом Міністерства праці та соціальної політики України від 05 червня 2001 року № 254, зареєстрованої в Міністерстві юстиції України 20 липня 2001 року за № 616/5807 (НПАОП 0.00-5.24-01), ДСТУ 7237:2011.
2. Електроінструмент для паяння деталей повинен відповідати вимогам ДСТУ IEC 60745-1:2010 «Інструмент ручний електромеханічний». Безпека. Частина 1: Загальні вимоги».
3. Пайку великогабаритних виробів необхідно проводити паяльником з вбудованим відсмоктуванням.
4. Пайку необхідно виконувати закритим об'ємом, використовуючи паяльник з напругою не більше 12 В.
5. Паяльник на робочому місці повинен бути встановлений на вогнетривкій підставці для запобігання його падінню.
6. Паяльник в робочому стані завжди повинен знаходитися в приміщенні з місцевою витяжною вентиляцією.
7. У проміжках між паянням нагрівання жала паяльника слід зменшити до 150-180°С, а якщо робота тимчасово переривається - вимкнути. Робочі місця повинні бути обладнані регуляторами нагріву паяльників.
8. Паяння дрібних виробів у вигляді штепсельних з'єднувачів, наконечників, клем і тому подібних виробів необхідно проводити з їх фіксацією в спеціальних пристосуваннях (затискачах, кліпсах).
9. При пайці інтегральних мікросхем необхідно використовувати бінокулярні стереоскопічні мікроскопи з телевізійними екранами.

Щоб уникнути короткого замикання при роботі з паяльником, необхідно дотримуватися певних норм. Ручні електричні машини повинні відповідати вимогам Правил улаштування електроустановок, ДНАОП 0.00-1.21-98 «Правила безпечної експлуатації електроустановок споживачів», затверджених наказом Держнаглядохоронпраці України від 09.01.98 N 4 (z0093-98), зареєстрованих у

Мін'юсті 10.02.98 за N 93/2533, та ГОСТ 12.2.013.0-91. Вимоги безпеки перед початком роботи:

1. Ви повинні отримати інструктаж на робочому місці.
2. Одержати спецодяг, засоби індивідуального захисту, інструмент та пристосування для виконання робіт і перевірити їх комплектність та неушкодженість.
3. Підготувати робоче місце: прибрати зайві предмети, перевірити достатність освітлення на робочому місці.

Технічний процес паяння радіоелементів супроводжується забрудненням атмосферного повітря у вигляді аерозолів припою, флюсів, парів від рідин, що використовуються для змивання флюсу або для розчинення лаків, що застосовуються для покриття друкованих плат тощо. На працівників також може негативно впливати інфрачервоне випромінювання від нагрітої поверхні електричного паяльника. Клас небезпеки, гранично допустима концентрація в повітрі робочої зони (ГДКрз), токсичність і можлива біологічна дія основних компонентів олов'яно-свинцевого припою, а також флюсів і миючих засобів наведені в ДСН 3.3.6.042-99. Наявність у повітрі робочої зони аерозолю свинцю, який відноситься до шкідливих речовин 1 класу небезпеки, вимагає обов'язкового використання ефективної системи вентиляції у виробничих приміщеннях. Під час технологічного процесу паяння радіоелементів олов'яно-свинцевими припоями концентрація аерозолю свинцю в повітрі робочої зони, як правило, перевищує гранично допустиму концентрацію (ГДКрз), що вимагає застосування на робочих місцях місцевої витяжної вентиляції. Існуючі нормативні документи забороняють використання паяльних постів, які не обладнані місцевою витяжною вентиляцією. Вентиляційні установки повинні бути включені до початку робіт і вимкнені після їх закінчення. Місцеве видалення відпрацьованих газів із зон паяння повинно забезпечуватися незалежною вентиляційною установкою. Трасування вентиляційної мережі та конструкція місцевих відсмоктувачів повинні забезпечувати можливість регулярного очищення повітропроводів. Електропаяльник в робочому стані повинен знаходитися в приміщенні з витяжною вентиляцією. У зоні ручного паяння

швидкість спрямованого потоку, що створюється місцевими відсмоктувачами, повинна бути не менше ніж на 0,2 м/с вищою за швидкість руху повітря в зоні паяння і не менше ніж на 0,5 м/с. При виконанні паяльних робіт олов'яно-свинцевим припоєм на робочих місцях необхідно користуватися «Інструкцією з охорони праці при роботі з малосірчистим олов'яно-свинцевим припоєм» П1 1.4.32-423-2005.

Системи вентиляції виробничих приміщень у поєднанні з технічним обладнанням, що виділяє шкідливі речовини згідно з ГОСТ 12.0.003, надлишкове тепло або вологу, повинні забезпечувати умови мікроклімату і чистоту повітря, що відповідають вимогам ГОСТ 12.1.005, ДСТУ. 3.3.6.042 на постійних і тимчасових робочих місцях у робочій зоні виробничих приміщень. У зоні адміністративно-побутових приміщень та в приміщеннях громадських будинків умови мікроклімату повинні забезпечуватися відповідно до вимог ДСН 3.3.6.042.

Технічні рішення, прийняті при проектуванні систем вентиляції, а також вимоги до їх влаштування та експлуатації, повинні відповідати ДБН А.3.2-2 2009, ДБН В.2.5-67:2013, СНиП 2.09.02. 85, ДБН В.2.2-28-2010. Випробування систем вентиляції проводяться відповідно до вимог нормативних документів та вимог виробника. Розташування вентиляційних систем повинно забезпечувати безпечний і зручний монтаж, експлуатацію та ремонт технічних засобів. Розташування систем вентиляції повинно відповідати нормам освітлення приміщень, робочих місць і проходів згідно з ДСТУ Б А.3.2-15:2011, ДБН В.2.5:28-2018. Для монтажу, ремонту та обслуговування елементів системи вентиляції, а також для проходу через них повинні бути передбачені стаціонарні площадки, проходи, сходи і містки. Розташування вентиляційних установок припливно-витяжної вентиляції в приміщеннях вентиляційного устаткування повинно бути відповідно до ДБН В.2.5-67:2013. У разі виникнення пожежі слід передбачити спеціальні заходи, що забезпечують відключення вентиляційних систем, а за необхідності включення систем аварійної протидимної вентиляції, відповідно до вимог ДБН В.2.5-67:2013. Розміщення і компонування електрообладнання в системах вентиляції, а також контрольно-вимірювальної апаратури, розташування струмоведучих частин і заземлення повинні відповідати вимогам ДСТУ Б В.2.5-82:2016, НПАОП 0.00-1.29,

НПАОП 40.1-1.01, НПАОП 40.1-1.21, НПАОП 40.1-1.32 та чинним стандартам на вибухозахищене обладнання та гірничошахтне устаткування. 1.32, та чинним стандартам на вибухозахищене обладнання та гірничошахтне устаткування. Системи вентиляції, що обслуговують приміщення категорій А і Б згідно з НАПБ Б.03.002, а також системи з місцевими витяжками, які можуть утворювати статичну електрику, повинні бути вибухозахищеними та захищеними від статичної електрики згідно з ГОСТ 12.1.018, ГОСТ 12.4.124, ДБН В.2.5-27, НПАОП 0.00-1.29. Проектування вентиляційного обладнання систем, що обслуговують приміщення категорій А і Б згідно з НАПБ Б.03.002, а також місцевих і районних відсмоктувачів для вибухопожежонебезпечних і вогнетривких сумішей повинно відповідати вимогам ДБН В.2.5-67:2013. Працівники, які працюють з вентиляційними системами, повинні використовувати засоби захисту відповідно до ДСТУ 7239:2011 ССБП Засоби індивідуального захисту. Загальні вимоги та класифікація.

Уникнути проблем психофізіологічного стресу, монотонності роботи, перевантаження зорових аналізаторів, психічного напруження, дискомфорту та статичної пози. Для цього необхідно дотримуватися низки рекомендацій. Потрібно час від часу відпочивати, наприклад, подивившись на хвилину у вікно. Для відпочинку очей можна виконати наступні вправи:

1. Швидко моргнути очима 20 разів, при цьому без зусиль закриваючи і відкриваючи повіки.
2. Із заплющеними очима починайте рухати очними яблуками зліва направо (4 рази), а потім вгору-вниз (4 рази).
3. Із заплющеними очима намалювати очними яблуками кола - спочатку за годинниковою стрілкою, потім проти годинникової стрілки (5 разів).
4. Вигляньте у вікно і знайдіть об'єкт вдалині (наприклад, пташку на дереві або антену на даху сусіднього будинку). Сфокусуйте на ньому погляд на 5 секунд. Тепер переведіть погляд на кінчик власного носа і також затримайте на ньому погляд на 5 секунд.
5. Щільно заплющити очі, а потім широко розплющити (повторити 5 разів).

Ви можете робити ці вправи в тому порядку, який вам підходить, і так часто, як вам зручно, але не менше двох разів на день. Це не займе багато часу, але допоможе вам зберегти зір на довгі роки.

Тривале сидіння не тільки погано впливає на опорно-руховий апарат, але й зменшує приплив крові до мозку, що призводить до зниження когнітивних здібностей, а отже, і ефективності на роботі. Тому лікарі рекомендують робити перерви кожні 20 хвилин. Саме цей проміжок часу є безпечним для сидіння. Спробуйте встати після 20 хвилин сидіння і пройтися по офісу, підійти до колеги, помити руки, випити склянку води - що завгодно. Якщо у вас дуже термінова робота і навіть похід в туалет може зашкодити - спробуйте попрацювати стоячи.

Щоб уникнути несприятливого освітлення, необхідно дотримуватися наступних норм: Основні вимоги до виробничого освітлення:

1. Забезпечити на робочій поверхні освітленість, що відповідає характеру зорової роботи і не нижче встановлених норм;
2. Забезпечити, щоб рівень освітленості у виробничих приміщеннях був достатньо рівномірним і постійним, щоб уникнути частого перенастроювання органів зору;
3. Не створювати ефекту засліплення як від самих джерел світла, так і від інших об'єктів, що знаходяться в полі зору;
4. Не створюйте на робочій поверхні різноманітні та глибокі тіні (особливо тіні, що рухаються);
5. Контраст між освітленими поверхнями повинен бути достатнім для того, щоб можна було розрізнити деталі;
6. Вони не повинні створювати небезпечних і шкідливих виробничих факторів (шум, теплове випромінювання, небезпека ураження електричним струмом, небезпека пожежі та вибуху для ламп): повинні бути надійними і простими в експлуатації.

Нормативні значення освітленості робочих місць для різних видів робіт та відповідних зорових навантажень визначені у ДБН В.2.5:28-2018. «Природне та штучне освітлення».

Методика розрахунку проведеного телескопічного відсмоктування повітря із зон паяння. Світлодіодний телескопічний відсмоктувач може бути як з прямокутним, так і з круглим отвором. У вертикальній площині столу встановлюються відкидні та телескопічні присоски прямокутної форми з гострими краями, що обумовлено прямокутною формою друкованої плати і, як правило, радіоелементи (РЕ) розміщуються в процесі пайки в горизонтальній площині робочої поверхні столу. Кількість повітря, що видаляється із зони паяння РЕ за допомогою відсмоктувального пристрою з прямокутним отвором, визначається за формулою (6.1).

$$L_{вп} = (S + 7.7 * E^{0.63} * X^{1.4}) * v_x \quad (6.1)$$

Розмір меншої сторони прямокутного всмоктувального отвору (В) визначається шляхом отримання оптимального співвідношення між сторонами всмоктувального отвору В і Е.

$$\frac{B}{E} = 0.24 * \left(\frac{X}{E}\right)^{0.36} \quad (6.2)$$

Кількість всмоктуваного повітря визначається за наступною формулою (6.3).

$$L_{вк} = \frac{\pi}{4} * (D^2 + 9.1 * D^{0.6} * X^{1.4}) * v_x \quad (6.3)$$

При використанні керованих телескопічних відсмоктувачів повітря із зон пайки на робочих місцях концентрація аерозолю свинцю в повітрі робочої зони під час технологічного процесу пайки ПЕ олов'яно-свинцевим припоєм визначається за такою формулою (6.4).

$$C_{рз} = 0.6 * \frac{y*n*t*N}{V+L_в*N*t} \quad (6.4)$$

Під час розрахунку розраховувалася поверхня повітря, яка буде видалятися місцевим відсмоктуванням з розмірами, зазначеними в умові. Це значення

відповідає 0,056 м3. Таким чином, при зміні початкових параметрів можна регулювати потрібний розмір локального екстрактора.

6.3 Ризик пожежі та вибуху

Промисловий переїзд можна віднести до категорії Г згідно НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою» (помірно пожежонебезпечні). Негорючі речовини та/або матеріали в гарячому, розпеченому та/або розплавленому стані, процес обробки яких супроводжується виділенням променистого тепла, іскор та/або полум'я; горючі гази, рідини та/або тверді речовини, які спалюються або використовуються як паливо. Оскільки виробничий процес передбачає пайку електронних компонентів. Відповідно до правил експлуатації та типових норм на вогнегасники. ДСТУ 3855-99 Протипожежний захист. Визначення горючості матеріалів і конструкцій. Для приміщень за умови дотримання наступних параметрів (Додаток 4 до Правил експлуатації та типових норм належності вогнегасників): промислові, сільськогосподарські, складські та лабораторні будівлі і приміщення, адміністративно-побутові будівлі і приміщення та споруди промислових підприємств, громадські будівлі і споруди, гаражі та авторемонтні майстерні. Рекомендується використовувати вогнегасники типу:

1. Порошкові (ВП-5, ВП-6, ВП-9, ВП-12). У кількості (додаток 1 до Правил експлуатації та типових норм належності вогнегасників) для паломницького павільйону площею до 50 м2 , та категорії приміщення Г слід використовувати 1 вогнегасник вагою 8-12 кг.
2. З метою оповіщення людей про пожежу встановлюється комплекс технічних засобів, за допомогою яких відповідно до підготовлених планів евакуації передаються сигнали оповіщення одночасно по всьому об'єкту, що захищається, а при необхідності - послідовно або вибірково в окремі його частини (поверхи, секції тощо).

6.4 Інструкція з охорони праці під час паяння

Відповідно до положення про розробку інструкцій з охорони праці (ДНАОП 0.00-4.15-98) ми підготуємо типову інструкцію:

1. До роботи з приладом допускається технічний персонал, який вивчив об'єкт, технічну інструкцію, діючу інструкцію і пройшов перевірку знань з технічної та пожежної безпеки.
2. Робоче місце або ділянка повинні бути обладнані засобами протипожежного захисту - порошковими вогнегасниками або іншими видами вогнегасників.
3. Перед початком роботи переконайтеся, що:
    a. Прилад правильно підключено та заземлено.
    b. Всі з'єднувальні кабелі та точки відключення знаходяться в належному стані.
4. Під час проведення робіт необхідно:
    a. Використовуйте тільки справні інструменти, які використовуються за призначенням.
    b. Переконайтеся, що на робочому місці немає зайвих предметів, які відволікають увагу і можуть призвести до травмування.
    c. При появі іскор, короткого замикання, запаху гару або диму негайно вимкніть прилад і встановіть причину загоряння.
5. По закінченню роботи необхідно:
    a. Вимкніть прилад, коли фахівець залишить своє робоче місце.
    b. Приберіть своє робоче місце.
    c. Переконайтеся, що всі інструменти є в наявності, як описано.
    d. Про будь-яку несправність приладу повідомити керівника.
6. У разі виникнення пожежі негайно викликати пожежну охорону. До їх прибуття приступити до гасіння пожежі підручними засобами, рятувати людей та надавати їм допомогу.
7. У разі ураження електричним струмом вимкніть електроживлення і вживіть необхідних заходів для надання першої медичної допомоги.
8. На робочому місці працівники повинні бути ознайомлені з планом і порядком евакуації з приміщення, який повинен знаходитися на видному місці.

РОЗДІЛ 7
ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Метою законодавства про охорону навколишнього природного середовища є регулювання умов охорони, використання і відтворення природних ресурсів, забезпечення екологічної безпеки, запобігання і ліквідація негативного впливу господарської та іншої діяльності на навколишнє природне середовище, збереження природних ресурсів, генетичного фонду диких тварин, ландшафтів та інших природних комплексів, унікальних територій та природних об'єктів, пов'язаних з історико-культурною спадщиною. Відповідно до Закону України «Про підприємства України» всі роботодавці зобов'язані забезпечити у своїй діяльності дотримання вимог законодавства України про охорону праці та навколишнього середовища. В дипломній роботі основною темою досліджень була розробка прототипу стеганографічного захисту. В роботі розроблено електронний прототип пристрою. Плати та корпуси елементів містять деякий відсоток негативних речовин.

Основними матеріалами, що використовуються у виробництві, є: різновид пластику, текстоліт для друкованих плат, різні електронні компоненти (резистори, транзистори, котушки тощо), батарейки (що містять хімічні речовини), мідь для проводки, припої, флюс для паяння. Багато з цих компонентів є досить токсичними при виробництві та неправильній утилізації.

7.1 Аналіз проблеми впливу електронного прототипу на стан довкілля

АБС - це терполімер, що виробляється полімеризацією стиролу та акрилонітрилу в присутності полібутадієну. Пропорції можуть варіюватися від 15 до 35 % акрилонітрилу, від 5 до 30 % бутадієну і від 40 до 60 % стиролу. В результаті виходить довгий полібутадієновий ланцюг, перехрещений з більш короткими полі(стирол-ко-акрилонітрильними) ланцюгами. Нітрильні групи з сусідніх ланцюгів полярні і притягуються один до одного і зв'язують ланцюги, роблячи АБС міцнішим, ніж чистий полістирол. Стирол надає пластику глянцеву, непроникну поверхню. Полібутадієн, гумоподібна речовина, забезпечує міцність навіть при низьких температурах. При виготовленні АБС необхідно використовувати два ротаційних вакуумних насоси Рутса і рідинний вакуумний насос, при виготовленні АБС - для видалення речовини «повітря + N2 + етилбензол, стирол і акрилонітрил». Через виділення стиролу і акрилонітрилу та інших речовин можливе перевищення нормативів викидів у навколишнє середовище. АБС-пластик не є біорозкладним матеріалом, якщо просто викидати непотрібний пластик, він буде накопичуватися в природі. Відомо, що існує таке стихійне лихо, яке негативно впливає на все живе, що мешкає в морі та на узбережжі. Наприклад, у шлунках деяких морських організмів виявлено велику кількість сміття, в тому числі пластикових виробів. При нагріванні також утворюються пари акрилонітрилу, які є токсичними.

Ще одним компонентом, який є небезпечним, є літій-іонна батарея. Основними проблемами для навколишнього середовища є видобуток літію і кобальту та утилізація батарейок. Літій міститься в солончаках. У солончаках бурять свердловини і викачують солону воду на поверхню для випаровування у ставках. Це дає можливість видобувати карбонат літію за допомогою хімічного процесу. Видобуток літію має значні екологічні та соціальні наслідки, зокрема,

через забруднення та виснаження водних ресурсів. Крім того, для переробки літію потрібні токсичні хімікати. Якщо такі хімічні речовини вивільняються шляхом вилуговування, розливу або викидів у повітря, вони можуть завдати шкоди громадам, екосистемам та виробництву продуктів харчування. Крім того, видобуток літію неминуче призводить до пошкодження ґрунтів та забруднення повітря. Соляні родовища літію розташовані в посушливих регіонах, де доступ до води має вирішальне значення для місцевих громад та їх життєдіяльності, а також для місцевої флори і фауни. У чилійських соляних рівнинах Атакама гірничодобувна промисловість споживає, забруднює і відволікає дефіцитні водні ресурси від місцевих громад. Видобуток літію став причиною конфліктів водних питань з різними громадами, такими як Токонао на півночі Чилі. У Салар-де-Хомбре-Муерто в Аргентині місцеві жителі стверджують, що літієва промисловість забруднила водотоки, які використовуються для зрошення людей, худоби та сільськогосподарських культур. Висловлювалися припущення, що Болівія може стати літієвою наддержавою і, можливо, випередити Чилі, видобуваючи його значні запаси, які можуть перевищувати 100 млн. тонн в соляних озерах країни.

Розвідка літію та інвестиції в нього також здійснюються за межами Андського регіону. Американська компанія Nova Mining Corporation, наприклад, просувається з придбанням ліцензій на видобуток літію в Монголії у відповідь на нинішній сплеск продажів електронних товарів. Болівія поки що виступає проти великомасштабного промислового видобутку літію, хоча планує побудувати пілотний проект як попередник можливого майбутнього розвитку видобутку літію. Однак багате на літій родовище Салар-де-Уюні знаходиться поблизу шахти Сан-Крістобаль, яка з моменту свого відкриття у 2007 році спричинила «екологічну і соціальну катастрофу, що впливає на весь південний захід Потосі», в тому числі через використання 50 000 літрів води на день. Загалом в ЄС виробляється близько 24 кг електричних та електронних відходів на одного мешканця на рік, включаючи літій, який використовується у високотехнологічних галузях промисловості. Що стосується батарейок, то в ЄС діють правила збору, переробки, обробки та утилізації батарейок, які вимагають, щоб рівень збору становив щонайменше 25% до кінця

вересня 2012 року і 45% до кінця вересня 2016 року. Законодавство не регулює питання збору літієвих батарейок. Кількість літій-іонних батарей, зібраних в ЄС в 2010 році, оцінювалася в 1289 тонн, разом з 297 тоннами нових літієвих батарей. За даними бельгійської переробної компанії Umicore, це лише близько 5% іонів літію. Вихід на ринок акумуляторів. Німеччина, Франція, Бельгія та Нідерланди мають найкращі показники зі збору батарейок, в тому числі первинних та вторинних літій-іонних акумуляторів. Однак, навіть у цих країнах рівень стягнення дуже низький. Батареї не тільки ризикують випускати токсичні гази при пошкодженні, але й ключові елементи, такі як літій і кобальт, є обмеженими, а їх видобуток може призвести до забруднення та виснаження водних ресурсів, серед інших екологічних наслідків.

7.2 Аналіз основних джерел впливу та їх наслідків для людини та її оточення

При виготовленні електронного пристрою завжди необхідно використовувати пайку. Однак цей процес не є корисним для людини, а, навпаки, може викликати певні хронічні захворювання. Пайка передбачає постійне нагрівання припою і каніфолі до температури близько 300 градусів. Пайка - це метал, сплав або суміш оксидів, які використовуються для з'єднання металевих деталей, мінеральної кераміки та інших деталей, для потовщення пластин тощо. Припої на основі олов'яно-свинцевих сплавів широко використовувалися в минулому і виробляються досі. Вони особливо корисні для ручної пайки, але свинець, що входить до їх складу, може мати негативний вплив на навколишнє середовище. Тому відбувається поступовий перехід на безсвинцеві припої, які, однак, менш придатні для ручного використання з різних причин. Свинець може викликати серйозні хронічні наслідки для здоров'я. Вплив відбувається в основному через випадковий контакт зі шкірою, при роботі зі свинцем слід носити рукавички. Під час паяння може відбуватися обмежене утворення диму. Свинець впливає на кровотворну і нервову системи, шлунково-кишковий тракт і нирки. Викликає анемію (оскільки є частиною домашнього біосинтезу і скорочує життя еритроцитів), а також енцефалопатія, психічні розлади, шлунково-кишкові розлади, диспепсія, коліки, нефропатія. Як і

більшість подібних сполук, свинець може викликати гострі та хронічні отруєння. Симптомами гострого отруєння є: порушення з боку травного тракту (підвищене слиновиділення, металевий присмак у роті, нудота, блювання, свинцеві коліки в животі, діарея), зниження температури тіла і артеріального тиску, брадикардія або тахікардія, холодний піт, утруднене дихання, колапс. Проковтування 250 мг найтоксичнішої сполуки тетраетилсвинцю (C2H5)4Pb призводить до летального результату. Люди, чия професія вимагає контакту зі сполуками свинцю або які перебувають у забрудненому середовищі, страждають від хронічних отруєнь. Хронічне отруєння проявляється у вигляді анемії, яка є наслідком блокування ферментних систем, що призводять до синтезу гемоглобіну; ураження як центральної, так і периферичної нервової системи. Симптомами такого ураження є порушення розумової діяльності (діти, які відстають у розвитку, мають більш високий рівень свинцю в крові), надмірна рухливість, агресивність, порушення сну, дисфункція нирок. Свинець також вкрай небезпечний для вагітних жінок, оскільки має здатність проникати через плаценту і призводити до серйозних незворотних неврологічних порушень у плода. Каніфоль (колофоній, ерсин) - смола, що входить до складу припою. Флюс виділяє видимі пари, які помітні під час паяння. Вплив каніфолі може викликати подразнення очей, горла та легенів, носові кровотечі та головний біль. Повторний вплив може спричинити сенсибілізацію дихальних шляхів та шкіри, що може спровокувати та загострити астму. Каніфоль є серйозною небезпекою для здоров'я на робочому місці.

Вплив парів припою з флюсу на основі смоли або колоїдного флюсу може призвести до різних ризиків для здоров'я. Професійна астма є однією з багатьох небезпек, спричинених потоками диму; кашель, задишка, хрипи та біль у грудях є одними з симптомів астми. Алергічна гіперчутливість - ще один поширений ризик для здоров'я, алергічна гіперчутливість розвивається в перші кілька місяців впливу і може продовжувати розвиватися протягом багатьох років, викликаючи хрипи і утруднене дихання. Подразнення - в результаті прямого або непрямого контакту з припоєм на основі смоли симптоми можуть варіюватися від простого подразнення очей або носа до більш серйозних шкірних проблем, що передаються повітряно-

крапельним шляхом. Пари припою також можуть викликати інші проблеми зі здоров'ям, такі як хронічний бронхіт, хімічна чутливість, біль у грудях, головний біль і запаморочення. Для виготовлення корпусу необхідно використовувати 3D-друк. У процесі друку використовуються пластмаси, які при нагріванні можуть виділяти токсичні пари. Не тільки ABS, але і PLA може виділяти токсичні пари, які називаються ЛОС (летючі органічні сполуки). Не всі ЛОС токсичні, але деякі можуть бути токсичними, особливо для молодих користувачів. Перш ніж це стане серйозною проблемою для здоров'я, нове дослідження, проведене 3Dsafety.org у співпраці з італійським виробником 3D-принтерів WASP, проаналізувало точну кількість токсичних ЛОС і потенційно небезпечних наночастинок, що виділяються під час екструзії ниток, щоб оцінити потенційні ризики для здоров'я.

Нове дослідження, представлене Фабріціо Мерло і Стефано Маццоні, ґрунтується на інших попередніх дослідженнях, проведених на початку 1990-х років, які показали, що плавлення і переробка пластикових матеріалів вивільняє кілька токсичних частинок у вигляді газів, в тому числі аміак, ціанідронова кислота, фенол і бензол. Лабораторні дослідження показали, що АБС значно токсичніший за полімер на основі кукурудзи, але й полімер на основі кукурудзи не позбавлений небезпечних викидів, особливо якщо він екструдований при температурі понад 200°C. Крім того, одні й ті ж рулони матеріалу, придбані у різних продавців, виробляють дуже різну кількість летких органічних сполук, навіть якщо вони використовуються в одному і тому ж 3D-принтері і мають однакові налаштування швидкості і температури.

Другим критичним аспектом є викид наночастинок, тобто частинок діаметром менше 0,1 мкм, які можуть поглинатися безпосередньо альвеолами та епідермісом легень. При цьому викиди при використанні АБС варіюються від 3 до 30 разів більше, ніж для PLA пряжі. Випробування також показало, що час, необхідний для того, щоб концентрація наночастинок у повітрі повернулася до стандартного рівня, становив від 10 до 30 хвилин після припинення процесів екструзії. Завдяки методу фотоіонізації, дослідження (яке було опубліковане на 3Dsafety.org і буде оновлюватися з подальшою інформацією) також може бути проведене на нейлоні,

полістиролі, ПЕТ та інших матеріалах. Проблема АБС Під час нагрівання АБС в процесі виробництва (для формування або екструзії, а також під час 3D-друку) утворюються пари акрилонітрилу, які є токсичними. Існують побоювання, що акрилонітрил може бути канцерогенним для людини. Бутадієн є відомим канцерогеном. Стирол також має канцерогенні властивості. Стирол легко окислюється, приєднує галогени, полімеризується (утворює тверду склоподібну масу - полістирол) і кополімеризується з різними мономерами. Полімеризація відбувається вже при кімнатній температурі (іноді з вибухом), тому при зберіганні стирол стабілізують антиоксидантами (наприклад, третбутилпірокатехіном, гідрохіноном).

Вдихання парів стиролу викликає багато гострих і хронічних захворювань. Ця речовина негативно впливає на функцію печінки та нирок, систему кровообігу та нервову систему. Тривале потрапляння стиролу в організм людини призводить до катару дихальних шляхів, подразнення шкіри та слизових оболонок, зміни складу крові та порушень в роботі вегетативної системи.

Акрилонітрил відноситься до другого класу небезпеки (дуже небезпечні речовини) за ступенем впливу на організм людини. Необоротно зв'язується з білками, РНК і ДНК у різних тканинах. Небезпечний при вдиханні, токсичний при прийомі всередину - може спричинити смерть. Пари викликають подразнення слизових оболонок та шкіри. Діє через неушкоджену шкіру. При згорянні виділяються токсичні гази.

Симптоми ураження: головний біль, запаморочення, слабкість, нудота, блювання, задишка, пітливість, прискорене серцебиття, зниження температури тіла, послаблення пульсу, судоми, втрата свідомості, почервоніння та печіння шкіри. 1,3-бутадієн класифікований MAIP як канцероген групи 1 «канцероген людини», а також внесений до списку канцерогенів Агентством з реєстрації токсичних речовин і хвороб Агентства з охорони навколишнього середовища США. Американська конференція урядових промислових гігієністів (ACGIH) включила цю хімічну речовину до списку підозрюваних канцерогенів. Рада з охорони природних ресурсів (NRDC) перераховує деякі групи захворювань, які підозрюються у зв'язку з цією

хімічною речовиною. Деякі дослідники дійшли висновку, що він є найпотужнішим канцерогеном у сигаретному димі, вдвічі потужнішим, ніж витікаючий акрилонітрил.

1,3-бутадієн також є підозрюваним тератогеном для людини. Тривалий і надмірний вплив може впливати на багато частин людського тіла; показано, що кров, мозок, очі, серце, нирки, легені, ніс і горло реагують на присутність надлишку 1,3-бутадієну. Дані на тваринах свідчать про те, що жінки більш чутливі до можливого канцерогенного впливу бутадієну на чоловіки, які постраждали від хімічних речовин. Це може бути пов'язано з впливом естрогенних рецепторів. Хоча ці результати свідчать про важливі наслідки для ризиків впливу бутадієну на людину, для остаточної оцінки ризиків необхідно більше даних. Також бракує даних щодо впливу бутадієну на репродуктивну функцію та розвиток людини, виявлених у мишей, але дослідження на тваринах показали, що вдихання бутадієну під час вагітності може збільшити кількість вроджених вад розвитку, а люди мають таку ж ендокринну систему, як і тварини. 1,3-бутадієн визнаний високоактивною летючою органічною сполукою (ЛОС) через його здатність легко утворювати озон, і тому викиди хімічної речовини суворо регулюються TCEQ в озонових районах штату Х'юстон-Бразорія-Галвестон. Зона підвищеної концентрації. З літій-іонними акумуляторами під час пайки слід поводитися обережно. Літій-іонні акумулятори складаються з анода і катода, розділених пористим полімерним сепаратором. Активним матеріалом в катоді зазвичай є оксиди перехідних металів з вбудованими в кристал іонами літію. В якості анода зазвичай використовується графіт. Електроліт, який заповнює електрохімічний елемент, являє собою органічний розчин солей літію. Під час першого заряду, що виконується виробником, при встановленні літію в аноді, на електродах (особливо на аноді) утворюється захисний іонно-провідний шар (ЗІШ), що складається з розкладеного електроліту. Цей шар захищає електроди від паразитних реакцій з електролітом. Найпоширенішою причиною самозайняття акумуляторів є коротке замикання в електрохімічному елементі. Електричний контакт між анодом і катодом може виникнути з багатьох причин. Наприклад, може бути механічне пошкодження клітини. Інше внутрішнє коротке замикання виникає

через порушення технології виробництва, коли електроди нерівномірно обрізані або коли частинки металу потрапляють між анодом і катодом, викликаючи пошкодження. Пористий сепаратор. Причиною внутрішнього короткого замикання може бути також «розтікання» металевих ланцюжків літію (дендритів) по сепаратору. Цей ефект виникає, якщо іони літію не встигають інтегруватися в кристал анода при занадто швидкому заряді або низькій температурі, і якщо ємність катодно-активного матеріалу перевищує ємність анода.

Після того, як сталося коротке замикання, батарея починає нагріватися. При досягненні температури 70-90 °C іонний захисний шар на аноді починає руйнуватися. Потім літій в аноді вступає в реакцію з електролітом і виділяє леткі вуглеводні: етан, метан, етилен тощо. Однак, незважаючи на наявність такої вибухонебезпечної суміші, загоряння не відбувається, оскільки в системі ще немає кисню. Оскільки реакції з електролітом є екзотермічними, температура і тиск в акумуляторі продовжують зростати. Коли температура досягає 180-200 °C, матеріал катода, як правило, оксид перехідного металу з літієм, вбудованим в кристал, починає реакцію диспропорціонування і виділяє кисень. Саме тут відбувається самозайманння і ще більший стрибок температури. Паралельно відбувається термічне розкладання електроліту (200-300 °C), що також призводить до виділення тепла. Нарешті, графіт вступає в реакцію з електролітом (якщо він все ще присутній), і коли температура досягає 660 °C, алюмінієвий колектор плавиться. Температура зазвичай не піднімається вище 900 °C, оскільки розкладатися вже нема чому. Крім внутрішнього короткого замикання, існують й інші причини самозайманння: перегрів акумулятора, неправильна зарядка/розрядка (перевищення максимально допустимої напруги, зарядка великими струмами, занадто глибока розрядка) тощо. Але всі ці причини призводять до одного результату: тепловому прискоренню і деградації електроліту при взаємодії з електродами. Відрізняються лише порядок і швидкість цих реакцій. Такі вибухи можуть завдати значної фізичної шкоди людям.

7.3 Рекомендації щодо зниження цих негативних чинників

Відходи АБС-пластика можуть бути перероблені. Для переробки відходів АБС використовуються різні термічні методи.

Пластмаси піддаються термічній обробці без втрати первинної якості або з невеликою часткою таких втрат, але іноді можуть набувати жовтуватого відтінку (що легко виправляється фарбуванням у потрібний колір). Оскільки АБС є термопластиком, він може регенеруватися при високих температурах. Однак, важливо пам'ятати, що вона містить токсичні речовини. Наприклад, в процесі нагрівання виділяється токсичний газоподібний стирол. Його необхідно вивести з цеху, а на стадії плавки краще використовувати автоматизований контроль. Також варто пам'ятати, що різні марки пластику несумісні, тому відходи з АБС-пластику необхідно попередньо сортувати. Визначити, до якого типу пластику належить полімер, можна за маркуванням його етикетками для вторинної переробки. Матеріал переробляється методом формування або екструзії. Формування передбачає нагрівання пластику до в'язкого стану і подальшу подачу його в закриту форму, де матеріал застигає і приймає форму готового виробу. Вага виготовлених таким чином виробів може варіюватися від декількох грамів до декількох кілограмів, товщина стінок від 1 мм до 20 мм.

При екструзії АБС пластик плавиться в екструдері, а потім «продавлюється» через формувальний пристрій з подальшим охолодженням та калібруванням. Стандартні екструдовані марки характеризуються хорошою ударостійкістю, а деякі мають антистатичні властивості. Матеріал використовується при виготовленні деталей для холодильних установок. Важливо відзначити, що для екструзії АБС необхідний екструдер з водяним кільцем для гранулювання, так як цей матеріал негативно реагує на присутність кисню в місці нагріву. Він окислюється і втрачає свої товарні властивості, а вода захищає пелети від впливу повітря. Кінцева вторинна гранула повинна бути ретельно висушена. Щільність нижча, ніж у вихідного матеріалу, є пори. Завдяки цьому він має здатність вбирати воду і утримувати її тривалий час. Якщо матеріал недостатньо просушений, відлити з нього якісний виріб неможливо.

Стандартні марки лиття характеризуються високою стійкістю до вигоряння і антистатичними властивостями. Зі стандартних марок виготовляють великогабаритні, тонкостінні вироби, корпуси приладів і товари народного споживання.

Класичне застосування вторинної сировини - це зниження собівартості оригінальної продукції, адже перероблений вид сировини дешевший за первинний. Перероблені відходи АБС можна використовувати без обмежень, так при дотриманні всіх умов переробки його структура не змінюється. Простий новий процес переробки відновлює старі елементи літієвих батарей до нового стану, витрачаючи вдвічі менше енергії порівняно з нинішніми процесами. На відміну від існуючих методів переробки, де катоди розщеплюються на окремі елементи, які необхідно збирати заново, нова технологія випльовує з'єднання, готові до включення в нову батарею[33].

Метод працює на літій-кобальт-оксидних акумуляторах, що використовуються в ноутбуках і смартфонах, а також на складних літій-нікель-марганець-кобальтових батареях, які використовуються в електромобілях. Літієві батареї мають аноди з графіту і катоди з оксидів металу літію, де метал являє собою комбінацію кобальту, нікелю, марганцю і заліза. На сьогоднішній день переробляється менше 5% старих літієвих батарейок. Оскільки мільйони великих акумуляторів для електромобілів будуть виведені з експлуатації протягом наступного десятиліття, ми будемо відправляти ще більші гори горючих і токсичних відходів акумуляторів на звалища. Крім того, відходи містять цінні метали. Існує серйозне занепокоєння, що запаси таких важливих металів, як кобальт і літій, скорочуються. Переробка буде мати вирішальне значення, якщо ми хочемо не відставати від попиту на батарейки.

Кілька компаній, в основному в Китаї, вже займаються переробкою батарейок. Стандартний процес передбачає подрібнення батарейок, а потім або плавлення, або розчинення їх у кислоті. Кінцевим результатом є окремі метали, такі як кобальт, літій, нікель і марганець. Крім того, що ці методи використовують багато енергії, вони руйнують найціннішу частину катодних батарей, каже Чжен Чен, професор нанотехнологій Каліфорнійського університету в Сан-Дієго. «Матеріал має форму красивих, добре продуманих частинок зі специфічною мікроскопічною структурою, яка визначає продуктивність батареї», - говорить він. «Щоб побудувати ці споруди, потрібно багато технологій, енергії та часу». Простий метод, розроблений Ченом і його колегами, зберігає цю мікроструктуру. Спочатку дослідники розкручували

комерційні літієві елементи до тих пір, поки вони не втрачали половину своєї здатності зберігати енергію. Вони зняли катодний матеріал з алюмінієвої фольги і замочили його в гарячих солях літієвої ванни. Потім розчин висушували до отримання порошку, який швидко нагрівали до 800 градусів С, а потім дуже повільно охолоджували. Процес відновлює атомну структуру матеріалу катода і знову вводить в нього іони літію. І використовує вдвічі менше енергії, ніж звичайні процеси. З регенерованого катодного матеріалу дослідники виготовили нові елементи батареї. Нові катоди мають таку ж ємність накопичення енергії, час заряджання та термін служби, як і оригінальні. Результати дослідження опубліковані в журналі «Green Chemistry».

Дві інші компанії використовують аналогічну технологію "прямого ресайклінгу", коли весь структурований катодний матеріал переробляється. Компанія Farasis Energy з Сан-Франциско та стартап OnTo Technologies з міста Бенд, штат Орегон, розробляють технологію та намагаються її масштабувати. Кожен процес дещо відрізняється від іншого. Для вирішення проблеми вибуху літій-іонних акумуляторів під час пайки необхідно дотримуватися інструкції з проведення робіт. Або перейти на інший тип батареї. Безпосередньо виробничі приміщення, де розташовані ділянки пайки, повинні бути обладнані постійно діючою місцевою вентиляцією. Місцеві витяжні клапани, що відводять шкідливі речовини від виробничого обладнання, повинні бути заблоковані при включеному обладнанні для запобігання його роботі при вимкненій вентиляції. Безпека осіб, які працюють з 3D-принтером, забезпечується розміщенням принтера в окремому герметичному корпусі та встановленням системи відведення газів. Робочі місця для паяння олов'яно-свинцевими припоями повинні бути обладнані місцевими витяжними системами, що забезпечують швидкість руху повітря безпосередньо на місці паяння не менше 0,6 м/с незалежно від конструкції повітроприймачів. Повітроприймачі повинні легко переміщатися з надійною фіксацією положення під час проведення монтажних робіт з метою отримання максимально можливого наближення до місця паяного з'єднання.

У виробництві електроніки існують деякі проблемні фактори, які можуть негативно впливати як на екологію всієї планети, так і на людей. Наприклад, виробництво літію негативно впливає на цілі регіони, і вирішити цю проблему дуже складно. Але, принаймні, літій-іонні акумулятори можна переробляти і навіть використовувати повторно через вторинну переробку. Крім того, на екологію цілих регіонів, океанів і річок впливають АБС-пластики, що використовуються в 3D-друку. Вирішення цієї проблеми також знайдено. Рішенням стало повторне використання пластику шляхом переплавлення, якщо коротко. Забезпечити безпеку людини під час паяння. Без спеціальних заходів, таких як димові витяжки, працівники можуть піддаватися дуже серйозним професійним захворюванням. Корпус 3D-принтера відокремлений від руху голівки герметичним кожухом, а всередині є вентиляція. Такі рішення в цілому сприяють поліпшенню екологічної ситуації на виробництві та в цілому протягом усього життєвого циклу продукції.

ВИСНОВКИ

Отримані результати в ході перевірки роботи прототипу підтверджують можливість захисту стеганографічного інформації на основі стохастичного переставлення пікселів контейнера, а також можливість роботи за спрощеною стеганографічною моделлю Крістіана Кашена, яка відрізняється від повної тим, що у повній моделі, коли система знаходиться у пасивному режимі, то по відкритому

каналу зв’язку відправляються пусті контейнери. У даному випадку є сенс розглядати систему тільки у активному режимі. Система знаходиться у активному режимі, коли по відкритому каналу зв’язку відправляються контейнери з прихованим повідомленням. В моделі є три актори: адресант, адресат та зловмисник. Задача адресанта непомітно для зловмисника передати секретне повідомлення. Адресат має зчитати секретне повідомлення, а зловмисник не повинен мати змогу виявити наявність прихованої інформації. Модель працює наступним чином: адресант генерує ключ та приховує секретне повідомлення у контейнері за допомогою ключа; готовий контейнер відправляється відкритим каналом зв’язку, а ключ передається іншим, секретним, каналом зв’язку; під час передачі контейнера, зловмисник перехоплює контейнер, але він не підозрює наявність у контейнері прихованої інформації; адресат отримує ключ і контейнер та зчитує секретне повідомлення. У такому випадку повідомлення вважається переданим. Але потрібно зазначити, що зловмисник не повинен виявити наявність секретного повідомлення.

За допомогою апаратно-програмної реалізації даного способу стеганографічного алгоритму було успішно зашифровано та приховано секретне повідомлення. Візуальних змін у зображенні, яке містило приховане повідомлення, не було виявлено. Зчитування та розшифровка секретного повідомлення пройшли успішно. Контрольні суми прихованого та зчитаного повідомлень були однакові. Перевагами даного способу є його простота реалізації, можливість реалізувати алгоритм як програмно так і апаратно, висока ентропія зашифрованого повідомлення, висока крипто-стійкість контейнера.

Окрім цілі розробки апаратно-програмного модуля для перевірки можливості здійснення стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера, була ще ціль отримати можливі вектори подальшого розвитку способу, а також дослідити спектр задач, які можна вирішити за допомогою даного способу. Для початку розберемо основні цілі стеганографії, до цілей стеганографії можна віднести: приховану передачу даних, зберігання цифрових відбитків, зберігання стеганографічних водяних знаків. Перша ціль,

прихована передача даних, є історично найперша ціль для якої застосовується стеганографія. Задача полягає у передачі інформації таким чином, щоб третя особа і не здогадалась, що повідомлення було передано. До практичних застосунків пов'язаних до цієї цілі можна віднести: непомітну передачу інформації, приховане збереження інформації. Приховане збереження даних є одним із основних практичних застосунків стеганографії. Цей практичний застосунок було успішно підтверджено за допомогою розробленого апаратно-програмного модуля, який дав змогу успішно приховати секретне зображення у зображенні-контейнері. Після збереження зображення, яке містить приховані дані, візуальних змін не було виявлено, тому можна вважати, що секретне зображення приховано збережене. Непомітна передача даних схожа на приховане збереження даних, окрім того, що зображення, яке містить приховане повідомлення, передається каналами зв’язку, прототип реалізує два канали зв’язку, дротовий та безлротовий. Наступна ціль, зберігання цифрових відбитків, полягає у збереженні певної ідентифікуючої інформації у контейнері. Ця інформація використовується для захисту контенту від повторного комерційного використання. Наприклад, на торговій площадці, де продаються зображення з умовою заборони на повторне використання у комерційних цілях, користувач купляє зображення, тоді у зображення приховується інформація, яка допомагає ідентифікувати покупця, це може бути електронна адреса, номер телефону, ім’я, номер транзакції. Якщо така людина захоче перепродавати зображення, коли це заборонено, то у торгової площадки, яка має ексклюзивні права на зображення, будуть всі можливості юридично захистити себе від цифрового піратства. Приховування такого цифрового відбитку відбувається за допомогою стеганографії. Також цифровий відбиток не повинен легко виявлятися і підмінятися. Стеганографічний спосіб захисту інформації, який описується у даній роботі, задовольняє ці умови, а це означає, що він здатний вирішувати цю задачу. Зберігання стеганографічних водяних знаків є подібним до зберігання цифрового відбитку, різниця лише в тому, що водяний знак це певна незмінна інформація, що проставляється у кожний екземпляр контейнера.

У результаті перевірки апаратно-програмної реалізації способу стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера, сформовані наступні вектори розвитку:

- реалізація алгоритму з використанням програмованих логічних інтегральних схем, що сприятиме стрімкому підвищенню швидкості роботи;
- розширення спектру типів контейнерів у апаратно-програмній реалізації, включивши аудіо та відео дані.

Отже, в результаті досліджень була підтверджена можливість практичного використання запропонованого способу стеганографічного захисту інформації на основі стохастичного переставлення пікселів контейнера. Були, також, виявлені наступні переваги запропонованого способу:

- швидкість роботи алгоритму;
- простота апаратної та програмної реалізації;
- можливість програмної реалізації на різних платформах;
- можливість апаратної реалізації;
- універсальність використання алгоритму;
- стійкість до атак.

Розроблена апаратно-програмна реалізація дає змогу вирішити задачі прихованого зберігання та передачі інформації, а також виконує демонстраційні функції та може використовуватись у навчальних цілях. Завдяки стохастичній перестановці пікселів, зловмисник не в змозі виявити приховану інформацію, а також здогадатись про її наявність. Щоб отримати оригінальну послідовність, не маючи відповідних ключів, зловмиснику знадобиться перебрати $N!$ варіантів, що є практично неможливим. До того ж сама послідовність знаходиться у зашифрованому стані. Варіант, коли зловмисник може підібрати відповідні ключі, також практично неможливий, адже у якості основного ключа використовується незвідний поліном великої степені. Множина можливих векторів дуже велика для здійснення перебору. Навіть якщо орієнтуватися саме на перебір усіх можливих незвідних поліномів. Та навіть з відомим незвідним поліномом не буде змоги отримати таблицю перестановки та розшифрувати повідомлення, потрібно ще мати примітивний утворюючий та вектор

ініціалізації, які також є практично неможливими для атаки в лоб. Вразливість генераторів псевдовипадкової послідовності на базі лінійних регістрів зсуву з лінійним зворотним зв’язком, яка полягає у можливості отримання незвідного полінома з відрізку згенерованої послідовності за допомогою алгоритму Берлекемпа-Мессі, притаманна тільки для класичних генераторів, такі генератори використовують примітивні незвідні поліноми, які мають ряд слабких ключів. Генератори на основі матриць Галуа, Фібоначчі та спряжених їм здатні працювати з незвідними поліномами, що не є примітивними. Послідовність, яка утворена незвідним поліномом, що не є примітивним, не має слабких ключів, а це означає неможливість знаходження незвідного полінома за допомогою алгоритму Берлекемпа-Мессі.

Окрім перевірених застосувань даного способу для вирішення задач прихованої передачі та прихованого збереження інформації, існує також можливість застосування його для вирішення задач приховання цифрових відбитків та стеганографічних водяних знаків.

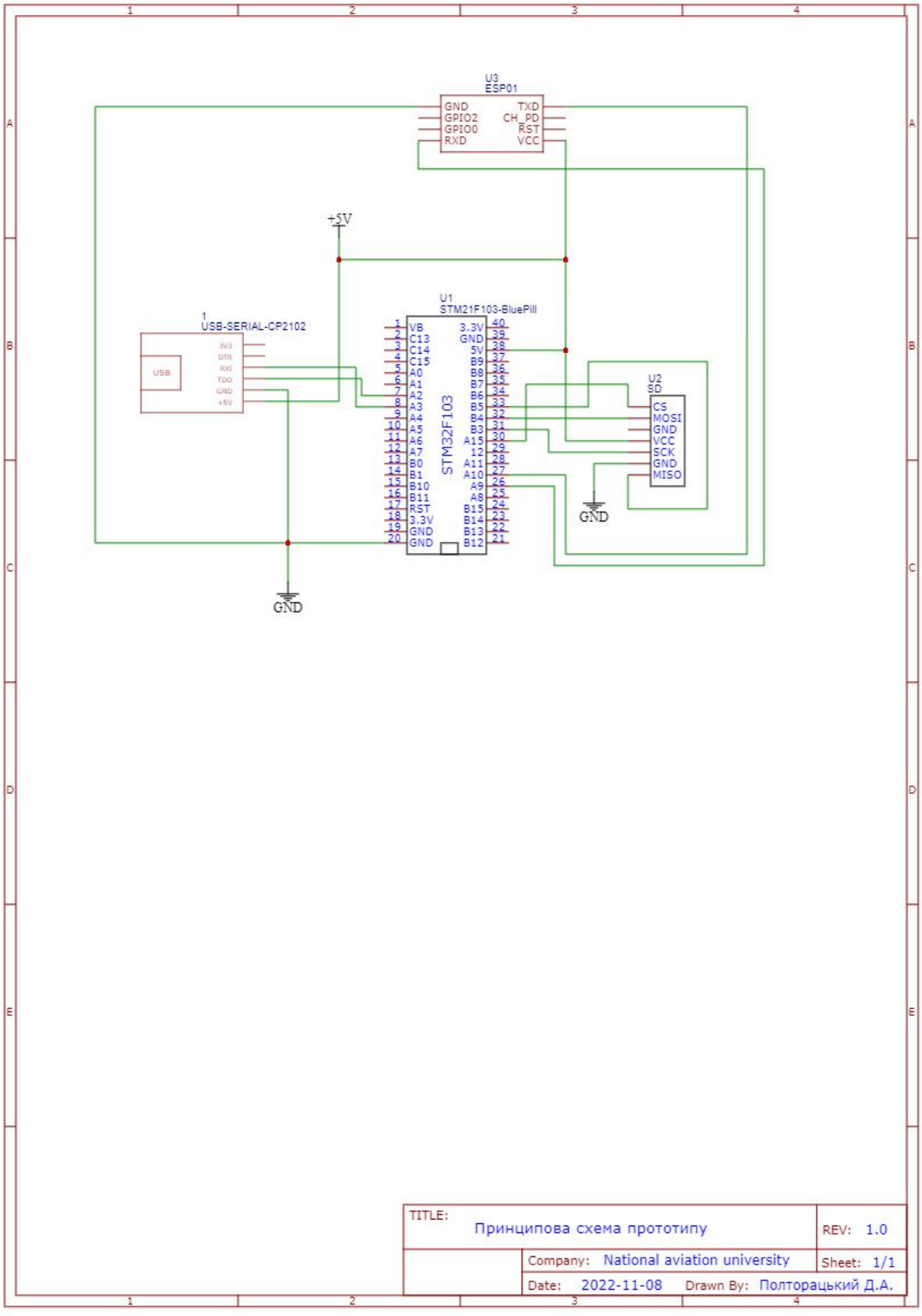
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Білецький А. Я. Алгоритм синтезу незвідних поліномів лінійної складності / А. Я. Білецький, А. В. Коавльчук, К. А. Новіков, Д. А. Полторацький // Захист інформації. – 2020. – № 2. – С. 74–87.

2. Міжнародна науково-технічна конференція «ABIA» ABIA-2019 : тези доп., 23–25 квіт. 2019 р., Київ / Нац. ун-т «Національний авіаційний університет». – Київ : Вид-во Нац. ун-ту «Національний авіаційний університет», 2019. – 13.5 с.
3. Програмний комплекс потокового байт-орієнтованого Галуа шифрування : матеріали XV науково-технічної конференції студентів, аспірантів, докторантів та молодих учених, 21–22 лист. 2018 р., Київ, Україна / нац. ун-т «Національний авіаційний університет». – Київ, 2018. – 72 с.
4. Білецький А. Я. Примітивні матриці Галуа в криптографічних застосуваннях / А. Я. Білецький // Захист інформації. – 2014. – № 4. – С. 274–283.
5. Білецький А. Я. Систематичні байт-орієнтовані коди / А. Я. Білецький, Д. В. Конюший, Д. А. Полторацький // Захист інформації. – 2018. – № 1. – С. 18–31.
6. Белецкий А. Я. Алгебраические основы теории кодирования и криптографии. / А. Я. Белецкий. – Аграр Медиа Груп, 2017. – 176 с.
7. GitHub [Електронне джерело] – 2020. - GammaGenerator/GammaGaloisMatrix/MainWindow.xaml.cs at master · licurg/ GammaGenerator. – режим доступу: https://github.com/licurg/GammaGenerator/blob/master/GammaGaloisMatrix/MainWindow.xaml.cs (дата звертання 01.06.2021). – Назва з екрана
8. GitHub [Електронне джерело] – 2018. – RASTR-ENCRYPTOR/RASTR-ENCRYPTOR/RASTR-2/ at master · licurg/RASTR-ENCRYPTOR. – режим доступу: https://github.com/licurg/RASTR-ENCRYPTOR/tree/master/RASTR-ENCRYPTOR/RASTR-2 (дата звертання 01.06.2021). – Назва з екрана

Додаток 1

Принципова схема прототипу



Додаток 2

Вихідний код графічного інтерфейсу

```
<Window x:Class="Steganography.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Steganography"
        mc:Ignorable="d"
        Title="Steganography"
        Background="#ecf0f1"
        Height="600"
        Width="1200">
    <Window.DataContext>
        <local:ViewModel/>
    </Window.DataContext>
    <Grid Margin="10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <WrapPanel Grid.Row="0">
            <Label Content="Режим роботи:" Style="{StaticResource MainLabel}"/>
            <RadioButton Content="Приховування інформації" GroupName="mode" IsChecked="{Binding
ForwardMode}" Style="{StaticResource MainRadioButton}"/>
            <RadioButton Content="Зчитування інформації" GroupName="mode" IsChecked="{Binding
ReverseMode}" Style="{StaticResource MainRadioButton}"/>
        </WrapPanel>
        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Grid Grid.Column="0" Margin="0,0,5,0">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="*"/>
                </Grid.RowDefinitions>
```

```
        <Label Grid.Row="0" Margin="0,5,0,0" Content="Шлях до вхідного зображення:"
Style="{StaticResource MainLabel}"/>
        <Grid Grid.Row="1">
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="Auto"/>
          </Grid.ColumnDefinitions>
          <TextBox Grid.Column="0" Style="{StaticResource MainTextBox}" Text="{Binding
InputFilePath}"/>
          <Button Grid.Column="1" Content="•••" Style="{StaticResource FileButton}"
Command="{Binding OpenInputFile}"/>
        </Grid>
        <Label Grid.Row="2" Margin="0,5,0,0" Content="Шлях до вихідного зображення:"
Style="{StaticResource MainLabel}" Visibility="{Binding OutputImageVisibility}"/>
        <Grid Grid.Row="3" Visibility="{Binding OutputImageVisibility}">
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="Auto"/>
          </Grid.ColumnDefinitions>
          <TextBox Grid.Column="0" Style="{StaticResource MainTextBox}" Text="{Binding
OutputFilePath}"/>
          <Button Grid.Column="1" Content="•••" Style="{StaticResource FileButton}"
Command="{Binding OpenOutputFile}"/>
        </Grid>
        <Label Grid.Row="4" Content="Параметри:" Style="{StaticResource MainLabel}"/>
        <Grid Grid.Row="5">
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="Auto"/>
          </Grid.ColumnDefinitions>
          <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
          </Grid.RowDefinitions>
          <Label Grid.Row="0" Grid.Column="0" Content="Розмір блоку перестановки:"
Style="{StaticResource SecondaryLabel}"/>
          <ComboBox Grid.Row="0" Grid.Column="1" Style="{StaticResource MainComboBox}"
Margin="0,0,0,5" SelectedItem="{Binding BlockSize}">
            <ComboBoxItem Content="256" IsSelected="True"/>
            <ComboBoxItem Content="65536"/>
          </ComboBox>
          <Label Grid.Row="1" Grid.Column="0" Content="Розрядність генератора ПСП:"
Style="{StaticResource SecondaryLabel}"/>
          <ComboBox Grid.Row="1" Grid.Column="1" Style="{StaticResource MainComboBox}"
SelectedItem="{Binding GeneratorDegree}">
            <ComboBoxItem Content="64" IsSelected="True"/>
```

```
                <ComboBoxItem Content="256" IsEnabled="False"/>
                <ComboBoxItem Content="1024" IsEnabled="False"/>
                <ComboBoxItem Content="2048" IsEnabled="False"/>
            </ComboBox>
        </Grid>
        <Label Grid.Row="6" Content="Ключі:" Style="{StaticResource MainLabel}"/>
        <Button Grid.Row="7" Content="Завантажити файл-ключ" Style="{StaticResource
MainButton}" Command="{Binding OpenKeys}"/>
        <Grid Grid.Row="8">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <WrapPanel Grid.Row="0">
                <Ellipse Width="12" Height="12" VerticalAlignment="Center" Fill="{Binding
IrreduciblePolynomialStatus}"/>
                <Label Content="Незвідний поліном:" Style="{StaticResource SecondaryLabel}"/>
            </WrapPanel>
            <Grid Grid.Row="1">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <Button Grid.Column="0" Margin="0" Grid.ColumnSpan="{Binding
EnterKeyButtonSpan}" Content="Ввести вручну" Style="{StaticResource GenerateButton}"
Command="{Binding EnterIrreduciblePolynomial}"/>
                <Button Grid.Column="1" Content="Згенерувати" Style="{StaticResource
GenerateButton}" Command="{Binding GenerateIrreduciblePolynomial}" Visibility="{Binding
GenerateKeyButtonVisibility}"/>
            </Grid>
            <WrapPanel Grid.Row="2" Margin="0,5,0,0">
                <Ellipse Width="12" Height="12" VerticalAlignment="Center" Fill="{Binding
PrimitiveOEStatus}"/>
                <Label Content="Примітивний утворюючий елемент:" Style="{StaticResource
SecondaryLabel}"/>
            </WrapPanel>
            <Grid Grid.Row="3">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
```

```
                <Button Grid.Column="0" Margin="0" Grid.ColumnSpan="{Binding
EnterKeyButtonSpan}" Content="Ввести вручну" Style="{StaticResource GenerateButton}"
Command="{Binding EnterPrimitiveOE}"/>
                <Button Grid.Column="1" Content="Згенерувати" Style="{StaticResource
GenerateButton}" Command="{Binding GeneratePrimitiveOE}" Visibility="{Binding
GenerateKeyButtonVisibility}"/>
              </Grid>
              <WrapPanel Grid.Row="4" Margin="0,5,0,0">
                <Ellipse Width="12" Height="12" VerticalAlignment="Center" Fill="{Binding
InitVectorStatus}"/>
                <Label Content="Вектор ініціалізації:" Style="{StaticResource SecondaryLabel}"/>
              </WrapPanel>
              <Grid Grid.Row="5">
                <Grid.ColumnDefinitions>
                  <ColumnDefinition Width="*"/>
                  <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <Button Grid.Column="0" Margin="0" Grid.ColumnSpan="{Binding
EnterKeyButtonSpan}" Content="Ввести вручну" Style="{StaticResource GenerateButton}"
Command="{Binding EnterInitVector}"/>
                <Button Grid.Column="1" Content="Згенерувати" Style="{StaticResource
GenerateButton}" Command="{Binding GenerateInitVector}" Visibility="{Binding
GenerateKeyButtonVisibility}"/>
              </Grid>
            </Grid>
            <Button Grid.Row="9" Content="Приховати повідомлення" Style="{StaticResource
MainButton}" Command="{Binding HideMessage}" Visibility="{Binding
HideMessageButtonVisibility}"/>
            <Button Grid.Row="9" Content="Зчитати повідомлення" Style="{StaticResource
MainButton}" Command="{Binding ExtractMessage}" Visibility="{Binding
ExtractMessageButtonVisibility}"/>
            <ProgressBar Grid.Row="10" Style="{StaticResource MainProgressBar}"
Maximum="{Binding MaxProgressValue}" Value="{Binding ProgressValue}"/>
          </Grid>
          <Grid Grid.Column="1" Margin="5,0,0,0">
            <Grid.RowDefinitions>
              <RowDefinition Height="Auto"/>
              <RowDefinition Height="Auto"/>
              <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <Label Grid.Row="0" Margin="0,5,0,0" Content="Повідомлення, що приховується:"
Style="{StaticResource MainLabel}"/>
            <Button Grid.Row="1" Content="Записати повідомлення у файл" Style="{StaticResource
MainButton}" Command="{Binding SaveMessage}" Visibility="{Binding
MessageSaveButtonVisibility}"/>
```

```
            <Button Grid.Row="1" Content="Відкрити повідомлення з файлу" Style="{StaticResource
MainButton}" Command="{Binding OpenMessage}" Visibility="{Binding
MessageOpenButtonVisibility}"/>
            <TextBox Grid.Row="2" Style="{StaticResource MainTextArea}" Text="{Binding
MessageText}" IsEnabled="{Binding MessageTextFieldEnabled}"/>
          </Grid>
       </Grid>
   </Grid>
</Window>
```

Додаток 3

Вихідний код генератора ПВП

```
class GaloisAlgorithm
    {
        private MainKey mainKey;
        private ulong[] matrix;
        private ulong gamma;
        private int degree;
        public GaloisAlgorithm(MainKey mainKey)
        {
            this.mainKey = mainKey;
            degree = mainKey.IrreduciblePolynomial.Length - 1;
            ulong max = 0xFFFFFFFFFFFFFFFF;
            BigInteger polynomial = NumericHelper.BinToDec(mainKey.IrreduciblePolynomial);
            matrix = new ulong[degree];
            matrix[0] = Convert.ToUInt64(mainKey.PrimitiveOE, 2);
            for (int i = 1; i < degree; i++)
            {
                BigInteger a = matrix[i - 1] << 1;
                if (a > max)
                    a = a ^ polynomial;
                matrix[i] = (ulong)a;
            }
        }
        public ulong Next()
        {
            ulong vector = gamma;
            if (gamma == 0)
                vector = Convert.ToUInt64(mainKey.InitVector, 2);
            gamma = 0;
            for (int i = 0; i < degree; i++)
            {
                gamma ^= matrix[i] & matrix[i] * (vector & 0x1);
                vector >>= 1;
            }
            return gamma;
        }
        public void Refresh()
        {
            gamma = 0;
        }
    }
```

Додаток 4

Вихідний код генератора ключів

```
class KeyGenerator
   {
      private static List<BigInteger> D = new List<BigInteger>()
      {
3,5,15,17,51,85,255,257,771,1285,3855,4369,13107,21845,65535,65537,196611,327685,983055,111412
9,3342387,5570645,16711935,16843009,50529027,84215045,252645135,286331153,858993459,143165
5765,4294967295,4294967297,12884901891,21474836485,64424509455,73014444049,219043332147,3
65072220245,1095216660735,1103806595329,3311419785987,5519032976645,16557098929935,18764
712120593,28470681808895,56294136361779,93823560602965,281479271743489,844437815230467,1
407396358717445,4222189076152335,4785147619639313,14355442858917939,23925738098196565,7
1777214294589695,217020518514230019,361700864190383365,723401728380766673,1085102592571
150095,1229782938247303441,3689348814741910323,6148914691236517205
      };
      private int degree;
      public KeyGenerator(int degree)
      {
         this.degree = degree;
      }
      public string GenerateIrreduciblePolynomial()
      {
         PolynomialGenerator polynomialGenerator = new PolynomialGenerator(degree);
         BigInteger polynomial = polynomialGenerator.Generate();
         while (CheckOE(2, polynomial))
         {
            polynomial = polynomialGenerator.Generate();
         }
         return polynomial.ToBinaryString();
      }
      public string GeneratePrimitiveOE(string polynomialString)
      {
         BigInteger OE;
         BigInteger polynomial = NumericHelper.BinToDec(polynomialString);
         using (RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider())
         {
            byte[] numBytes = new byte[degree / 8];
            rng.GetBytes(numBytes);
            OE = new BigInteger(numBytes);
         }
         while(!CheckOE(OE, polynomial))
         {
            OE += 1;
         }
         return OE.ToBinaryString();
      }
      public string GenerateInitVector()
      {
```

```
        string initVector = "";
        using (RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider())
        {
            byte[] numBytes = new byte[degree / 8];
            rng.GetBytes(numBytes);
            initVector = new BigInteger(numBytes).ToBinaryString();
        }
        return initVector;
    }
    private bool CheckOE(BigInteger OE, BigInteger polynomial)
    {
        BigInteger maximum = BigInteger.Pow(2, degree - 1);

        foreach (var divider in D)
        {
            if (ModularArithmetics.ModPower(OE, divider, polynomial, maximum) == 1) return false;
        }
        return true;
    }
}
```

Додаток 5

Вихідний код шифратора повідомлень

```
class MessageEncryptor
    {
        GaloisAlgorithm galoisAlgorithm;
        public MessageEncryptor(GaloisAlgorithm galoisAlgorithm)
        {
            this.galoisAlgorithm = galoisAlgorithm;
        }
        public string Encrypt(string message)
        {
            StringBuilder cipher = new StringBuilder();
            galoisAlgorithm.Refresh();
            foreach (var item in message)
            {
                byte[] gamma = BitConverter.GetBytes(galoisAlgorithm.Next());
                byte addition = 0;
                foreach (var part in gamma)
                {
                    addition ^= part;
                }
                cipher.Append((char)((byte)item ^ (byte)addition));
            }
            return cipher.ToString();
        }
        public string Decrypt(string cipher)
        {
            StringBuilder message = new StringBuilder();
            galoisAlgorithm.Refresh();
            foreach (var item in cipher)
            {
                byte[] gamma = BitConverter.GetBytes(galoisAlgorithm.Next());
                byte addition = 0;
                foreach (var part in gamma)
                {
                    addition ^= part;
                }
                message.Append((char)((byte)item ^ (byte)addition));
            }
            return message.ToString();
        }
    }
```