Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

«До захисту допущено»
Завідувач кафедри
комп'ютерної інженерії
к.т.н., доц. О.М.Березький
_____
"____"_____20__ р.

ДИПЛОМНИЙ ПРОЕКТ

освітньо-кваліфікаційного рівня "Спеціаліст"
зі спеціальності 7.05010201 "Комп'ютерні системи та мережі"
на тему:

АПАРАТНА РЕАЛІЗАЦІЯ ОПЕРАЦІЇ ШИФРУВАННЯ КРИПТОАЛГОРИТМУ TWOFISH

Студент групи КСМзс-51_____Жалко О.М.
(підпис)

Керівник:
викладач _____ Дубчак Л.О.
(підпис)

Нормоконтроль:
к.т.н., доцент _____Васильків Н.М.
(підпис)

Консультант
з охорони праці:
доцент _____Сапожник Г.В.
(підпис)

2012

Міністерство освіти і науки, молоді та спорту України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії
Спеціальність 7.05010201 – "Комп'ютерні системи та мережі"

"Затверджую"
Завідувач кафедри
комп'ютерної інженерії
к.т.н., доц. О.М.Березький
_____
"____"_____20__ р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТА
Жалко Олександр Михайлович

1. **Тема проекту:** "Апаратна реалізація операції шифрування криптоалгоритму Twofish " затверджена наказом університету № ___від "___"_____20__ р.
2. **Термін здачі студентом закінченого проекту** "___"_____20__ р.
3. **Вихідні дані для проекту:**Технічне завдання.
4. **Перелік задач, які мають бути вирішені:**

- провести аналіз існуючих рішень для програмно-апаратного шифрування даних;
- визначити оптимальну структурну організацію програмно-апаратної системи для шифрування даних;
- вибрати метод шифрування даних;
- розробити алгоритм шифрування даних;
- розробити архітектуру апаратної частини системи;
- виконати програмну реалізацію алгоритму шифрування даних;
- виконати тестуванняVHDL-моделі;
- оцінити швидкодію програмно-апаратної системи шифрування даних;
- виконати порівняльний аналіз розробленої системи з існуючими програмно-апаратними засобами для шифрування даних.

5. **Перелік графічного матеріалу** (з точним вказанням обов'язкових креслень)

- Узагальнена *f* функція. Схема структурна
- Модифікована *f* функція. Схема структурна
- Контролер. Схема структурна
- Середовище тестування. Схема структурна

6.Консультанти по проекту (із зазначенням розділів):

Розділ	Консультант	Підпис
Охорона праці	Сапожник Г.В.	

КАЛЕНДАРНИЙ ПЛАН

№	Назва розділів дипломного проекту	Термін виконання	Позначки керівника про виконання завдань
1	Аналіз та оцінка блочних симетричних алгоритмів шифрування	15.09.2011 – 5.11.2011	
2	Проектування шифрування алгоритму Twofish	6.11.2011 – 31.01.2012	
3	Дослідження продуктивності апаратної реалізації	1.02.2012 – 14.03.2012	
4	Охорона праці	15.03.2012 – 12.04.2012	

Завдання прийняв до виконання _____

(підпис)

Керівник дипломного проекту _____

(підпис)

АНОТАЦІЯ

Робота виконана на 111 сторінках, містить 20 рисунків, 11 таблиць, 6 додатків, з них 4 графічного матеріалу.

Метою дипломного проекту є розробка апаратної реалізації операції шифрування блочного симетричного криптоалгоритму Twofish. Результатом є VHDL – модель, що дозволяє дослідити роботу даного алгоритму.

В цьому дипломному проекті розглянуто реалізацію основних функцій шифру Twofish і отримані результати його продуктивності. Детально описано структуру шифру, різні стандартні блоки, які необхідні для апаратної реалізації, остаточні структури, отримані після інтеграції всіх компонентів шифру. Наведено узагальнені результати апаратної реалізації.

Розроблений пристрій може використовуватися для захисту інформації у приватних структурах.

ANNOTATION

Work is executed on 111 pages, contains 20 figures, 11 tables, 6 additions, from them 4 graphic material.

The purpose of the work is implementation of hardware the operation of encrypting of symmetric cryptoalgorithm of Twofish. The result is a VHDL-model, which allows investigating work of this code.

Realization used on basic functions of code of Twofish and got results of his productivity is considered in this project. A code structure, different standard blocks that isneeded for hardware representation, final structures got after integration of all the tools of code, is described in detail. The generalized results over of hardware representation are brought.

The worked out device can be used for a private structures.

ТЕХНІЧНЕ ЗАВДАННЯ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

1.1 Апаратна реалізація операції шифрування криптоалгоритму Twofish.

1.2 Область застосування — державні установи, комп'ютерні фірми та магазини, провайдери Інтернет, які використовують різноманітні системи захисту інформації.

2 ОСНОВА ДЛЯ РОЗРОБКИ

Основою для розробки є індивідуальне завдання на дипломний проект, затверджене кафедрою комп'ютерної інженерії факультету комп'ютерних інформаційних технологій Тернопільського національного економічного університету.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даної розробки є створення VHDL-моделі ядра процесора симетричного блокового шифрування для підвищення продуктивності шифрування даних у спеціалізованих комп'ютерних системах захисту інформації згідно з алгоритмом шифрування, визначеним Twofish.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами даної розробки є матеріали навчальної і реферативної літератури, технічна документація, науково-дослідні роботи, журнали.

5 ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до структури і функцій

5.1.1. Ядро процесора симетричного блокового шифрування повинне здійснювати симетричне блокове шифрування даних за алгоритмом Twofish.

5.1.2. Ядро повинне шифрувати і дешифрувати дані у режимі простої заміни, необхідно передбачити можливість введення ключів.

5.2 Вимоги до апаратної сумісності

Ядро процесора повинно містити простий інтерфейс керування шиною для завантаження даних, ключової інформації, виводу шифрованих

даних.

5.3 Вимоги до надійності

5.3.1 Сценарії підсистеми повинні виключати можливість додавання некоректних даних.

5.3.2 Розроблене програмне забезпечення має бути якомога стійкішим і коректно реагувати на нестандартні ситуації.

5.4 Вимоги безпеки

Будова процесора симетричного блокового шифрування у складі спеціалізованої комп’ютерної системи захисту інформації повинна відповідати вимогам електробезпеки ГОСТ 25.861-85.

6 ВИМОГИ ОХОРОНИ ПРАЦІ

В розділі “Охорона праці” дипломного проекту повинен бути даний аналіз умов праці розробника програмних засобів.

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙОМУ

7.1 Представлення дипломного проекту на попередній захист.

7.2 Представлення дипломного проекту на захист.

ЗМІСТ

ВСТУП

Успішний розвиток електронних комунікацій в значній мірі залежить від можливості безпечної передачі інформації по незахищених каналах зв'язку, збереженні її в електронному вигляді на носіях різних типів. Для обмеження доступу на модифікацію, осмислене читання та фальсифікацію інформації використовуються системи криптографічного захисту інформації. Проектування цих систем направлене на забезпечення високого ступеня захисту інформації, продуктивності роботи, низької споживаної потужності. При цьому існують додаткові вимоги, такі як короткий термін розробки та простота модифікації чи заміни використовуваних алгоритмів (так звана алгоритмічна незалежність).

Прогрес у розвитку теорії побудови алгоритмів симетричного блокового шифрування, доступність обчислювальної техніки та здешевлення технології виробництва надвеликих інтегральних схем зумовлюють широке використання процесорів симетричного блокового шифрування як у складі спеціалізованих систем захисту інформації, так і в складі пристроїв, орієнтованих на використання в універсальних системах, наприклад, персональних комп’ютерах, з’єднаних у глобальні та локальні обчислювальні мережі. Перелік галузей застосування процесорів симетричного блокового шифрування постійно розширюється. Особливо швидко такі процесори впроваджують у засоби телезв’язку, оскільки щоразу збільшуються цінність, об’єми та швидкості передавання даних.

Метою даного дипломного проекту є розробка та дослідження VHDL-моделі операцій шифрування та дешифрування симетричного блочного криптоалгоримту Twofish.

Доцільність розробки VHDL-моделі процесора шифрування за алгоритмом Twofish мотивується такими положеннями: цінність інформації, яка обробляється в системі може перевищувати вартість самої системи, а також швидкодія роботи самої реалізації. Захист інформації від різноманітних загроз здійснюється не лише

адміністративними, але й технічними засобами. Такий підхід дозволить підняти рівень захисту передачі даних, зокрема, в комп'ютерних мережах.

1 ОГЛЯД СУЧАСНИХ БЛОКОВИХ КРИПТОАЛГОРИТМІВ

1.1 Загальна характеристика блокових криптоалгоритмів

Для блокових криптоалгоритмів характерним є те, що вони перетворюють блок вхідної інформації фіксованої довжини й одержують вихідний блок того ж розміру, але недоступний для читання стороннім особам, що не володіють інформацією про ключ. Таким чином, схему роботи блокового шифру можна описати функціями *Z=EnCrypt*(*X*,*Key*) і *X=DeCrypt*(*Z*,*Key*), де ключ *Key* є параметром блокового шифру і являє собою деякий блок двійкової інформації фіксованого розміру. Вихідний *X* і зашифрований *Z* блоки даних також мають фіксовану розрядність, яка є рівною між собою, але необов'язково дорівнює довжині ключа [1].

Блокові шифри є основою, яку беруть для реалізації практично всі криптосистеми (таблиця 1.1). Методика створення ланцюгів із байт, що зашифровані блоковими алгоритмами, дозволяє шифрувати пакети інформації необмеженої довжини. Така властивість блокових шифрів, як швидкість роботи, використовується асиметричними криптоалгоритмами. Відсутність статистичної кореляції між бітами вихідного потоку блокового шифру використовується для обчислення контрольних сум пакетів даних і в хешуванні паролів.

Таблиця 1.1 — Сучасні симетричні криптоалгоритми

Назва алгоритму	Автор	Розмір блоку	Довжина ключа
IDEA	Xuejia Lia and James Massey	64 біта	128 біт
CAST128		64 біта	128 біт
BlowFish	Bruce Schneier	64 біта	128 – 448 біт
ГОСТ	НДІ	64 біта	256 біт
TwoFish	Bruce Schneier	128 біт	128 – 256 біт
MARS	Корпорація IBM	128 біт	128 – 1048 біт

До стійких криптоалгоритмів застосовується дуже важлива вимога, яку вони повинні обов'язково задовольняти: при відомих вхідному і вихіному значеннях блоку ключ, за допомогою якого здійснюється шифрування, може бути знайденим тільки шляхом повного перебору. Часто зустрічаються ситуації, в яких сторонньому спостерігачеві відома частина вихідного тексту. Це можуть бути стандартні написи в електронних бланках, фіксовані заголовки форматів файлів, що досить часто зустрічаються в тексті.

Таким чином, до функції стійкого блокового шифру *Z=EnCrypt*(*X*,*Key*) висуваються наступні вимоги:

— функція *EnCrypt* повинна бути оборотною;

— не повинно існувати інших методів прочитання повідомлення *X* за відомим блоком *Z*, крім повного перебору ключів *Key*;

— не повинно існувати інших методів визначення яким ключем *Key* було зроблене перетворення відомого повідомлення *X* у *Z*, окрім повного перебору ключів.

Розглянемо методи, за допомогою яких розробники блокових криптоалгоритмів досягають одночасного виконання цих трьох умов з дуже великою вірогідністю. Всі дії, які здійсненні над даним блоковим криптоалгоритмом, полягають в тому, що перетворений блок може бути представлений у вигляді цілого невід’ємного числа з діапазону, що відповідає його розрядності. Так, наприклад, 32-бітний блок даних можна інтерпретувати як число з діапазону 0..4'294'967'295. Крім того, блок, розрядність якого є "зі степенем два", можна трактувати як кілька незалежних невід’ємних чисел з меншого діапазону (розглянутий вище 32-бітний блок можна також представити у вигляді 2 незалежних чисел з діапазону 0..65535 або у вигляді 4 незалежних чисел з діапазону 0..255) [2].

Блокові криптоалгоритми дозволять здійснювати над цими числами за визначеною схемою наступні дії (таблиця 1.2) :

Параметр *V* для кожного з цих перетворень може бути використаний:

1) як фіксоване число (наприклад, *X*'=*X*+125);

2) як число, що отримане за допомогою додавання ключа (наприклад, $X'=X+F(Key)$);

3) як число, що отримане із незалежної частини блоку (наприклад, $X_2'=X_2+F(X_1)$).

Таблиця 1.2 — Операції шифрування

Біективні математичні функції	
Додавання	$X'=X+V$
Виключне АБО	$X'=X$ XOR V
Множення за модулем 2^N+1	$X'=(X*V) \bmod (2^N+1)$
Множення за модулем 2^N	$X'=(X*V) \bmod (2^N)$
Бітові зсуви	
Арифметичний зсув вліво	$X'=X$ SHL V
Арифметичний зсув вправо	$X'=X$ SHR V
Циклічний зсув вліво	$X'=X$ ROL V
Циклічний зсув вправо	$X'=X$ ROR V
Табличні підстановки	
S-box (англ. substitute)	X'=Table[X,V]

Останній варіант використовується в схемі, яка називається в честь її винахідника — мережею Фейстеля (Feistel).

Послідовність операцій, що виконуються над блоком, комбінації перерахованих вище варіантів V і самі функції F і складають "ноу-хау" кожного конкретного блокового криптоалгоритму [3].

Один-два рази в рік дослідницькі центри світу публікують черговий блоковий шифр, що під проведенням атак криптоаналітиків або здобуває статус стійкого криптоалгоритму, або навпаки.

Характерною ознакою блокових алгоритмів є багаторазове і непряме використання ключа. Це диктується в першу чергу вимогою неможливості зворотнього дешифрування у відношенні ключа при відомих вихідному і

шифрованому текстах. Для рішення цієї задачі в приведених вище перетвореннях найчастіше використовується не саме значення ключа або його частини, а деяка необоротна (не біективна) функція від значення ключа. Більш того, в подібних перетвореннях той самий блок або елемент ключа використовується багаторазово. Це дозволяє при виконанні умови оборотності функції щодо величини *X* зробити функцію необоротною щодо ключа *Key*.

Оскільки операція шифрування або дешифрування окремого блоку в процесі кодування пакета інформації виконується багаторазово (іноді до сотень тисяч разів), а значення ключа і, отже, функцій $V_i(Key)$ залишається незмінним, то іноді стає доцільно заздалегідь однократно обчислити дані значення і зберігати їх в оперативній пам'яті разом із ключем. Варто зазначити, що дана операція ніяким чином не змінює ні довжину ключа, ні криптостійкість алгоритму в цілому. Тут відбувається лише оптимізація швидкості обчислень шляхом кешування (англ. caching) проміжних результатів. Описані дії зустрічаються практично в багатьох блокових криптоалгоритмах і називаються розширенням ключа (англ. key scheduling).

Мережа Фейстеля є подальшою модифікацією описаного вище методу змішування поточної частини блоку, що шифрується, з результатом деякої функції, яка отримується шляхом обчислення від іншої незалежної частини того ж блоку. Цей метод одержав широке поширення, оскільки забезпечує виконання вимоги про багаторазове використання ключа і вихідного блоку інформації.

Класична мережа Фейстеля має наступну структуру (рисунок 1.1).

Незалежні потоки інформації, що породжені з вхідного блоку, називаються галузями мережі. У класичній схемі їх дві. Величини V_i називаються параметрами мережі, звичайно це функції від значення ключа. Функція *F* називається утворюючою. Дія, що складається з однократного обчислення утворюючої функції і наступного накладення її результату на іншу галузь з обміном їх місцями, називається циклом або раундом (англ. round) мережі Фейстеля [4].

Оптимальне число раундів *K* – від 8 до 32. Важливо те, що збільшення кількості раундів значно збільшує криптостійкість будь-якого блокового шифру

до криптоаналізу. Можливо, ця особливість і вплинула на активне поширення мережі Фейстеля – адже при виявленні, скажімо, якого-небудь слабкого місця в алгоритмі, здебільшого достатньо збільшити кількість раундів на 4-8, не переписуючи сам алгоритм.

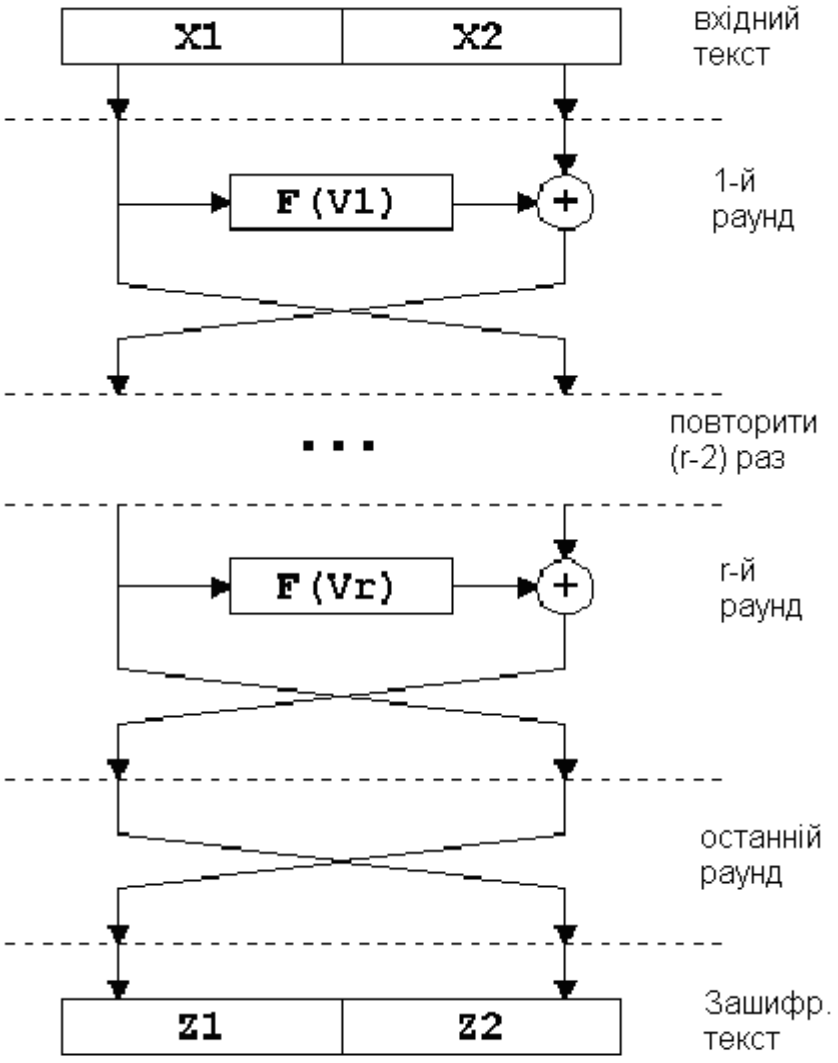

Рисунок 1.1 — Структура мережі Фейстеля

Часто кількість раундів не фіксується розробниками алгоритму, а лише вказуються розумні межі (обов'язково нижній, і не завжди – верхній) цього параметра.

Варто зазначити, що схема є оборотною. Мережа Фейстеля володіє тією властивістю, що навіть якщо утворюючу функцію *F* буде використано як необоротне перетворення, то й у цьому випадку весь ланцюжок буде відновлено.

Це відбувається внаслідок того, що для зворотнього перетворення мережі Фейстеля не потрібно обчислювати функцію F^{-1}.

Більше того, мережа Фейстеля симетрична. Використання операції XOR, оборотною своїм же повтором, і інверсія останнього обміну гілок роблять можливим декодування блоку тією ж мережею Фейстеля, але з інверсним порядком параметрів V_i. Зазначимо, що для оборотності мережі Фейстеля не має значення чи є число раундів парним або непарним числом. У більшості реалізацій схеми, в яких обидві перераховані вище умови (операція XOR і знищення останнього обміну) збережені, пряме і зворотне перетворення виробляються однієї і тією ж процедурою.

З незначними доробками мережу Фейстеля можна зробити й абсолютно симетричною, тобто виконуючі функції шифрування і дешифрування тим самим набором операцій. Математичною мовою це записується як "Функція *EnCrypt* тотожно дорівнює функції *DeCrypt*". Якщо ми розглянемо граф станів криптоалгоритму, на якому крапками відзначені блоки вхідної і вихідної інформації, то при якомусь фіксованому ключі для класичної мережі Фейстеля ми будемо мати картину, зображену на рисунку 1.2 а), а в другому випадку кожна пара точок одержить унікальний зв'язок, як зображено на рисунку 1.2 б).

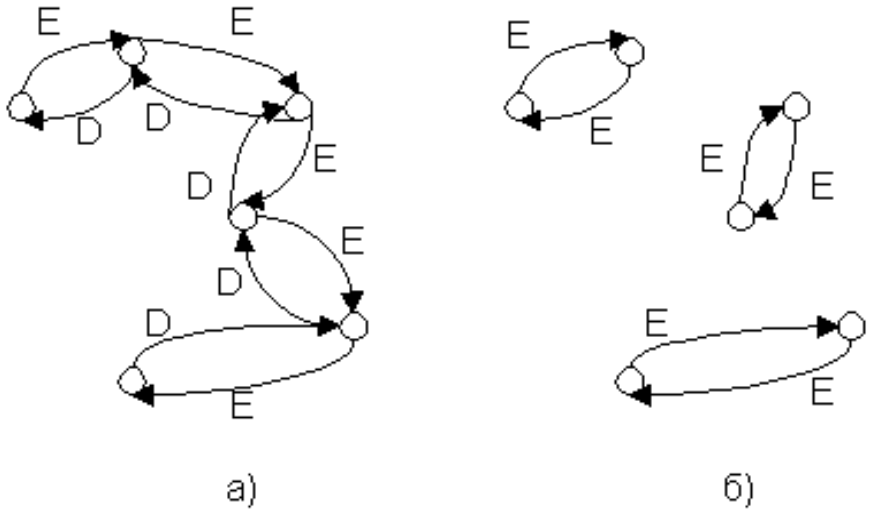

Рисунок 1.2 — Граф станів мережі Фейстеля

Модифікація мережі Фейстеля, що володіє подібними властивостями наводиться на рисунку 1.3. Як бачимо, основна її хитрість у повторному використанні даних ключа в зворотному порядку в другій половині циклу. Варто зазначи, що саме через цю недостатньо досліджену специфіку такої схеми (тобто потенційної можливості ослаблення зашифрованого тексту зворотними перетвореннями) її використовують у криптоалгоритмах з великою обережністю.

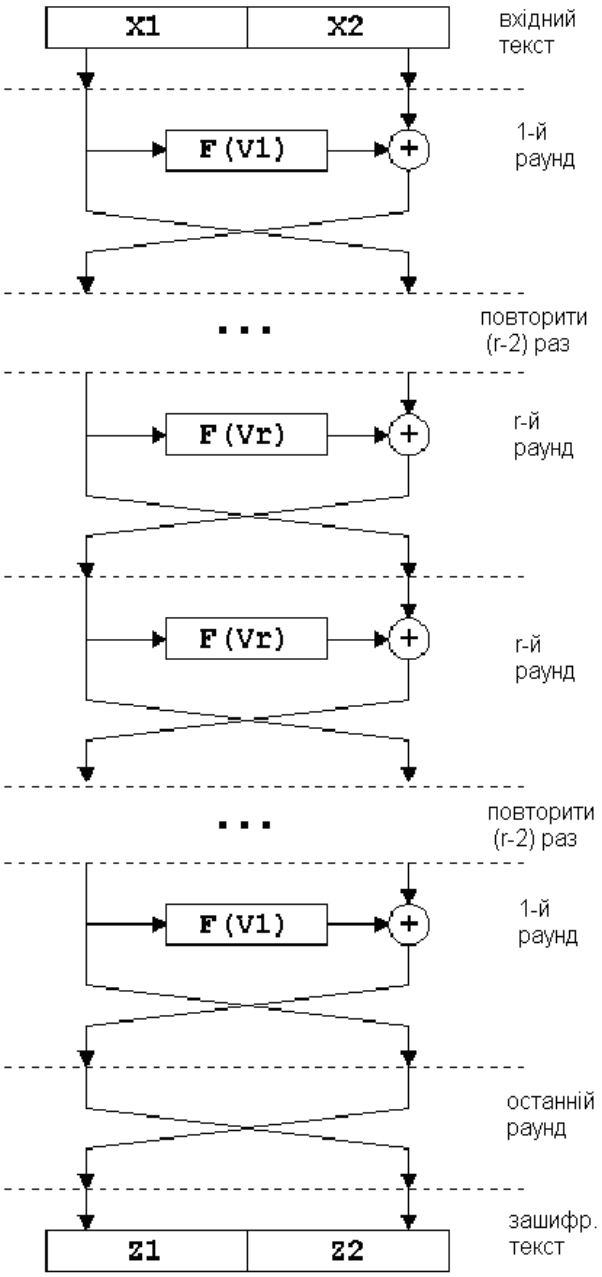

Рисунок 1.3 — Модифікація мережі Фейстеля

Модифікацію мережі Фейстеля для більшого числа віток застосовують набагато частіше. Це в першу чергу пов'язане з тим, що при великих розмірах блоків (128 і більше біт), що шифруються, стає незручно працювати з математичними функціями за модулем 64 і вище [5]. Як відомо, основні одиниці інформації, що обробляються процесорами на сьогоднішній день – це байт і подвійне машинне слово 32 біта. Тому все частіше й частіше в блокових криптоалгоритмах зустрічається мережа Фейстеля з 4-ма вітками. Найпростіший принцип її модифікації зображений на рисунку 1.4 а). Для більш швидкого перемішування інформації між вітками (а це основна проблема мережі Фейстеля з великою кількістю віток) застосовуються дві модифіковані схеми, що називаються "type-2" і "type-3" [4]. Вони зображені на рисунках 1. 4 б) і в), відповідно.

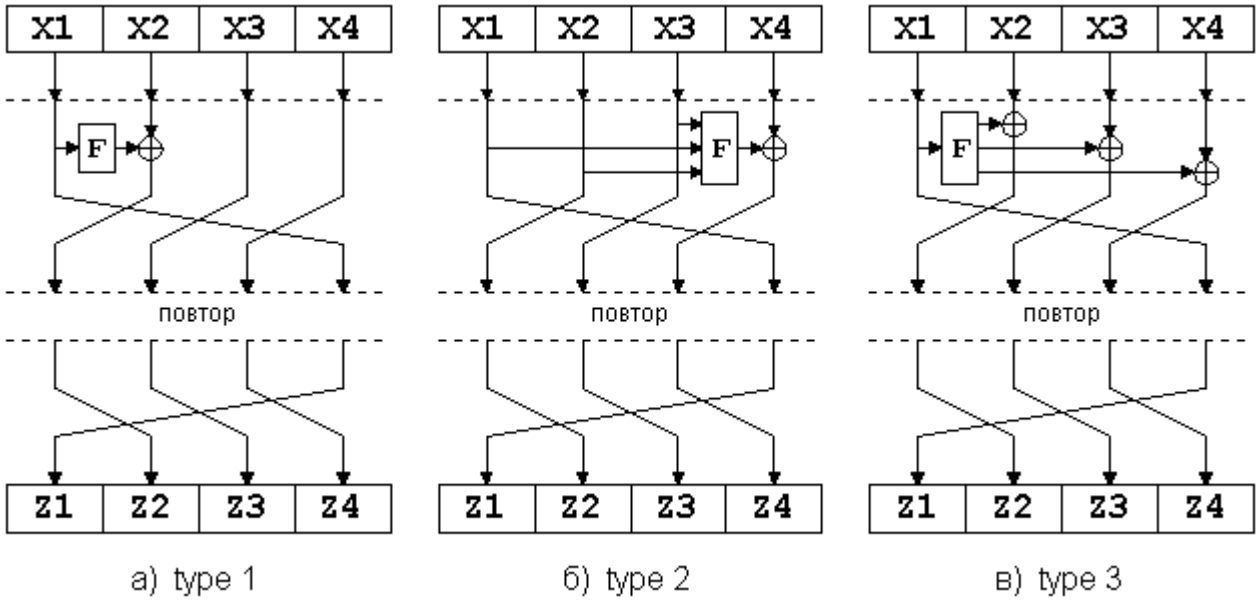

Рисунок 1.4 — Принципи модифікації мережі Фейстеля

Мережа Фейстеля надійно зарекомендувала себе як криптостійка схема добутку перетворень і її можна знайти практично в будь-якому сучасному блоковому шифрі. Незначні модифікації стосуються звичайно додаткових початкових і останніх перетворень (англомовний термін – whitening) над блоком, що шифрується. Подібні перетворення, що виконуються, звичайно, також за допомогою "виключного АБО" або додавання, мають за мету підвищити

початкову рандомізацію вхідного тексту. Таким чином, криптостійкість блокового шифру, що використовує мережу Фейстеля, визначається на 95% функцією F і правилом обчислення V_i із ключа. Ці функції і є об'єктом сучасних досліджень фахівцями в області криптографії.

1.2 Особливості блокового алгоритму шифрування Twofish

Симетричний блоковий шифр Twofish є одним із найбільш вдалих розробок компанії Counterpane Systems (нині Counterpane Internet Security, Inc.), яку заснував і очолив Брюс Шнайер — автор бестселера «Прикладна криптографія». Широку популярність Twofish здобув після виходу у фінал конкурсу на новий американський урядовий стандарт шифрування. У створенні криптоалгоритму взяли участь: Брюс Шнайер (Bruce Schneier), Джон Келсі (John Kelsey), Даг Уайтинг (Doug Whiting), Девід Вагнер (David Wagner), Кріс Хол (Chris Hall) і Нілс Фергюсон (Niels Ferguson) [5].

Основні характеристики криптоалгоритму Twofish:

— довжина блоку шифрування 128 біт;

— допустимі довжини ключів шифрування 128, 192 і 256 біт;

— передбачено можливість використання ключів шифрування з довжинами, відмінними від припустимих, що не перевищують 256 біт;

— відсутність «слабких» ключів;

— складний алгоритм для реалізації.

Twofish використовує 16-раундову архітектуру Файстеля (Feistel Network) з біективною (взаємно однозначною) функцією F і додатковими «відбілюваннями» (Whitenings) на вході і виході. Єдина відмінність від «чистої» Файстелевої структури полягає в наявності функціональних блоків, що виконують циклічні однобітові зсуви вправо і вліво.

Вхідний 128-бітовий блок P відкритого тексту (16 байт $p_0,\ldots,p_{15}$) розбивається на чотири 32-бітових слова P_0, P_1, P_2 і P_3 зі збереженням прямого порядку байтів (Little-Endian Convention)

На етапі вхідного «відбілювання» виконується операція XOR між цими словами і чотирма ключами K_0, K_1, K_2, K_3: $R_{0,i} = P_i \oplus K_i \quad i = 0,\ldots,3$.

Після цього відбувається 16 раундів шифрування. У кожному раунді два «лівих» слова є вхідними для функцій g (біти одного з вхідних слів спочатку циклічно зсуваються на 8 позицій вліво). Для отриманих вихідних слів функції g застосовується псевдоперетворення Адамара (PHT – Pseudo-Hadamard Transform) і додаються два раундових ключі K_{2r+8} і K_{2r+9}, де r – номер раунду шифрування [5]. Далі між модифікованими в такий спосіб «лівими» словами і двома «правими» словами (біти одного з яких циклічно зсуваються на одну позицію вліво) виконується операція XOR, після чого циклічному зсуву на 1 біт вправо піддається інше з тепер вже видозмінених «правих» слів. «Ліва» і «права» пари слів потім міняються місцями для наступного раунду шифрування. Таким чином:

$$(F_{r,0}, F_{r,1}) = F(R_{r,0}, R_{r,1}, r)$$

$$R_{r+1,0} = ROR(R_{r,2} \oplus F_{r,0}, 1)$$

$$R_{r+1,1} = ROL(R_{r,3}, 1) \oplus F_{r,1})$$

$$R_{r+1,2} = R_{r,0}$$

$$R_{r+1,3} = R_{r,1}$$

де r = 0,…,15, а абревіатурами ROR і ROL позначені функції двох аргументів, що виконують побітовий циклічний зсув першого аргументу вправо і вліво, відповідно, на число позицій, що рівне другому аргументові.

Після реалізації всіх 16-ти раундів шифрування останній обмін місцями «лівої» і «правої» пар слів скасовується, і між отриманими 32-бітовими словами і

ключами K_4, K_5, K_6, K_7 виконується операція XOR (формула — етап вихідного «відбілювання»):

$$C_i = R_{16,(i+2) \bmod 4} \oplus K_{i+4}, \quad i = 0,\ldots,3 \tag{1.1}$$

Отримані слова C_0, C_1, C_2, C_3 потім об'єднуються в 128-бітовий блок С (16 байт $c_0,\ldots,c_{15}$) шифрованого тексту.

Функція *F* є функцією трьох аргументів: двох вхідних слів R_0, R_1 і номера раунду шифрування *r*, необхідного для вибору належних раундових ключів. Перетворення R_0 функцією g дає T_0. Біти R_1 спочатку циклічно зсуваються вліво на 8 позицій, далі результат зсуву перетворюється за допомогою функції g в T_1. Після цього до T_0 і T_1, що пройшли РНТ, додаються два раундових ключі [6]:

$$\begin{aligned} T_0 &= g(R_0) \\ T_1 &= g(ROL(R_1,8)) \\ F_0 &= (T_0 + T_1 + K_{2r+8}) \bmod 2^{32} \\ F_1 &= (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}, \end{aligned}$$

де F_0 і F_1 - виходи функції *F*.

Функція g складає основу алгоритму шифрування. Вхідне 32-бітове слово *X* розбивається на чотири байти. Кожен байт проходить через відповідний *S*-box, що залежить від раундового ключа. Кожен *S*-box описується біективною функцією, що перетворить «вхідний» байт у «вихідний». Чотири результуючі байти інтерпретуються як вектор довжини 4 над полем Галуа $GF(2^8)$, що збільшується ліворуч на MDS-матрицю розмірів 4×4 (обчислення також проводяться над полем $GF(2^8)$). Отриманий в ході перетворень вектор розглядається як 32-бітове слово, що і є «виходом» функції g.

Елементи MDS-матриці є байтові величини, що записані в шістнадцятковій системі числення. У ході шифрування одного блоку відкритого тексту алгоритм

генерації ключів необхідно сформувати сорок 32-бітових ключів $K_0,\ldots,K_{39}$, а також чотири залежних від вигляду раундового ключа *S*-box'a, що використовується функцією *g*. Twofish приймає ключі з довжинами *N*, рівними 128, 192 і 256 біт. Ключі із довжинами, відмінними від зазначених і меншими 256 біт, також можуть бути використані за допомогою доповнення їх нулями до наступної більшої зі згаданих основних довжин. Наприклад, 80-бітовий ключ $m_0,\ldots,m_9$ буде доповнений до 128-бітового ключа в такий спосіб: $m_0,\ldots,m_9$, m_i, де $i=10,\ldots,15$.

Нехай $k=N/64$ (можливі значення *k* рівні 2, 3 і 4). Ключ *M* складається з 8*k* байт $m_0,\ldots,m_{8k-1}$. Ці байти насамперед перетворяться в 2*k* слів довжиною в 32 біта кожне. Третій вектор довжини *k*, *S*, також формується з ключа в такий спосіб: ключ розбивається на групи по 8 байт у кожній; кожна з груп розглядається як вектор довжини 8 над полем Галуа $GF(2^8)$, на який збільшується праворуч матриця розмірів 8×4, одержувана з RS-коду (Reed-Solomon Code). Результат множення (розміром у чотири байти) розглядається як 32-бітове слово. З цих слів і складається вектор *S*:

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,4} \end{pmatrix} = \begin{pmatrix} \\ RS \\ \\ \end{pmatrix} \bullet \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix}, \qquad (1.2)$$

де $i=0,\ldots,k-1$.

Варто звернути увагу на зворотний порядок запису слів у векторі *S*.

Для визначення операції множення RS-матриці праворуч на вектор-стовпець *m* представимо $GF(2^8)$ як $GF(2)[x]/\omega(x)$, де $\omega(x)=x^8+x^6+x^3+x^2+1$ - інший

примітивний поліном степеня 8 над полем $GF(2)$. Відповідність між байтовими величинами й елементами поля $GF\left(2^8\right)$ задається подібно тому, як це було зроблено в попередньому розділі у випадку з MDS-матрицею. RS-матриця записується в такий спосіб:

$$RS = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix}$$

Таким чином, вектори $M_e, M_0,$ і S складають основу алгоритму генерації ключів.

Розробники алгоритму вже були впевнені в стійкості свого витвору до криптоатак. Під час першого конкурсу на новий американський державний стандарт шифрування Б. Шнайером був оголошений конкурс на кращу криптоатаку проти Twofish із призовим фондом у 10 тисяч доларів. Задача була непростою, оскільки Twofish має високу складність алгоритму, яка і стала однією з причин, по якій він не став новим стандартом шифрування США.

Проте, визначні успіхи в криптоаналізі Twofish все-таки були досягнуті. Один із відомих методів Impossible Differential Attack, що належить Ларсу Нудсену (Lars Knudsen), для 6-раундового шифрування криптоалгоритму Twofish (без «відбілювань») має складність 2^{128} — для 128-бітового ключа, 2^{160} — для 192-бітового ключа і 2^{192} — для 256-бітового ключа.

Також була запропонована атака на 7-раундовий Twofish (знову ж без «відбілювань») зі складністю 2^{256} для 256-бітового ключа. Таким чином, очевидним є значний запас криптостійкості алгоритму шифрування Twofish.

1.3 Формування вимог і постановка задачі

Для того, щоб алгоритм став гідною заміною DES, його архітектура повинна задовольняти декілька критеріїв: високий ступінь захисту, мати просту структуру і високу продуктивність.

Очевидно, що найвищим пріоритетом для алгоритму є захист. Уже на рівні внутрішньої архітектури він повинен мати надійність, яка достатня для того, щоб протистояти спробам його взлому.

Разом з тим структура алгоритму, на відміну від традиційних поглядів, повинна бути настільки простою, щоб гарантувати ефективну процедуру шифрування.

Наступною вимогою, запропонованою до алгоритмів стандартизації, — висока продуктивність. Широке поширення алгоритму на ринку потребує прийнятної продуктивності при роботі на різних платформах: від смарт-карт до великих серверів. Висока продуктивність алгоритму припускає високу швидкість роботи при шифруванні і дешифруванні, а також при реалізації графіка ключа.

Щоб бути затвердженим як стандарт, алгоритм повинен:

- реалізувати надійне шифрування;
- являти собою блоковий шифр;
- працювати з 128-розрядними блоками даних і ключами трьох розмірів (128, 192 і 256 розрядів);
- використовувати операції, легко реалізовані як апаратно (у мікрочіпах), так і програмно (на персональних комп'ютерах і серверах);
- орієнтуватися на 32-розрядні процесори;
- не ускладнювати без необхідності структуру шифру для того, щоб всі зацікавлені сторони могли самостійно провести незалежний криптоаналіз алгоритму і переконатися, що в ньому не закладено ніякі недокументовані можливості.

Крім того, алгоритм, що претендує на те, щоб стати стандартом, повинен поширюватися по всьому світі на неексклюзивних умовах і без плати за користування патентом.

При остаточному тестуванні враховувалися наступні дев'ять характеристик: дві стосувалися захисту (загальний рівень захисту і захищеність реалізації), одна характеризувала гнучкість, а інші були зв'язані з ефективністю реалізації.

Важливою характеристикою цього алгоритму є його високий рівень захисту, проте гнучкість при реалізації та використанні в різноманітних застосуваннях обробки даних є трудомісткою та надзвичайно складною.

Для розробки апаратної реалізації необхідно використовувати сучасні засоби автоматизованого проектування вузлів обчислювальної техніки. Одним із таких потужних засобів є Active-HDL фірми Altera.

2 ПРОЕКТУВАННЯ ОПЕРАЦІЇ ШИФРУВАННЯ НА ОСНОВІ БЛОКОВОГО КРИПТОАЛГОРИТМУ TWOFISH

2.1 Структура основних компонентів криптоалгоритму Twofish

Із розвитком електронних комунікацій в кожній сфері сучасного життя, криптографія стала необхідним компонентом керування автентичності, цілісності, конфіденційності особистих даних, що знаходяться в публічних мережах. Зі збільшенням продуктивності, необхідною вимогою стає удосконалення в технології захисту, тобто замінити старі алгоритмів, які виявилися дуже слабкими або занадто повільними для поточної програми.

В цьому дипломному проекті пристрій ALTERA FPGA використовується для реалізації перспективного шифру цього покоління — Twofish. Як згадувалося вище, Twofish — 128-бітний шифр, який підтримує ключі довжиною 128, 192 або 256 біт. Для простоти впроваджено лише ключі довжиною 128 біт. Програмне забезпечення реалізації алгоритму було використане для випробування оперативної сумісності. Були реалізовані всі функції, включаючи sub-key, шифрування та дешифрування.

Twofish є одним з п'яти фіналістів для додаткового шифрування стандарту AES. Конкурс проводив Національний інститут стандартів і технологій (NIST). AES замінив застарілий стандарт шифрування та дешифрування криптоалгоритм DES. Twofish є наступником Blowfish: явних недоліків у шифрі не було виявлено і він є більш ефективним, ніж DES. Алгоритм є доступний для затвердження патентів і, таким чином, широко використовується.

Багато позитивних характеристик притаманно для Twofish, які роблять його цікавим для реалізації. Він може бути реалізований апаратно, таким чином є ефективним для застосування таких апаратних пристроїв як FPGAs. Його реалізація передбачає вирішити важливі питання, які цікавлять розробників: використання об’єму пам’яті і забезпечення необхідної швидкодії, і може бути використаний в якості pipelined [10].

В цьому дипломному проекті буде розглянута реалізація шифру Twofish і отримані результати його продуктивності. Варто розглянути і структуру шифру, різні стандартні блоки, які необхідні для апаратної реалізації, остаточні структури, отримані після інтеграції всіх компонентів шифру. Нарешті, необхідно буде навести узагальнені результати апаратної реалізації.

На рисунку 2.1 показано загальний вигляд структури криптоалгоритму Twofish.

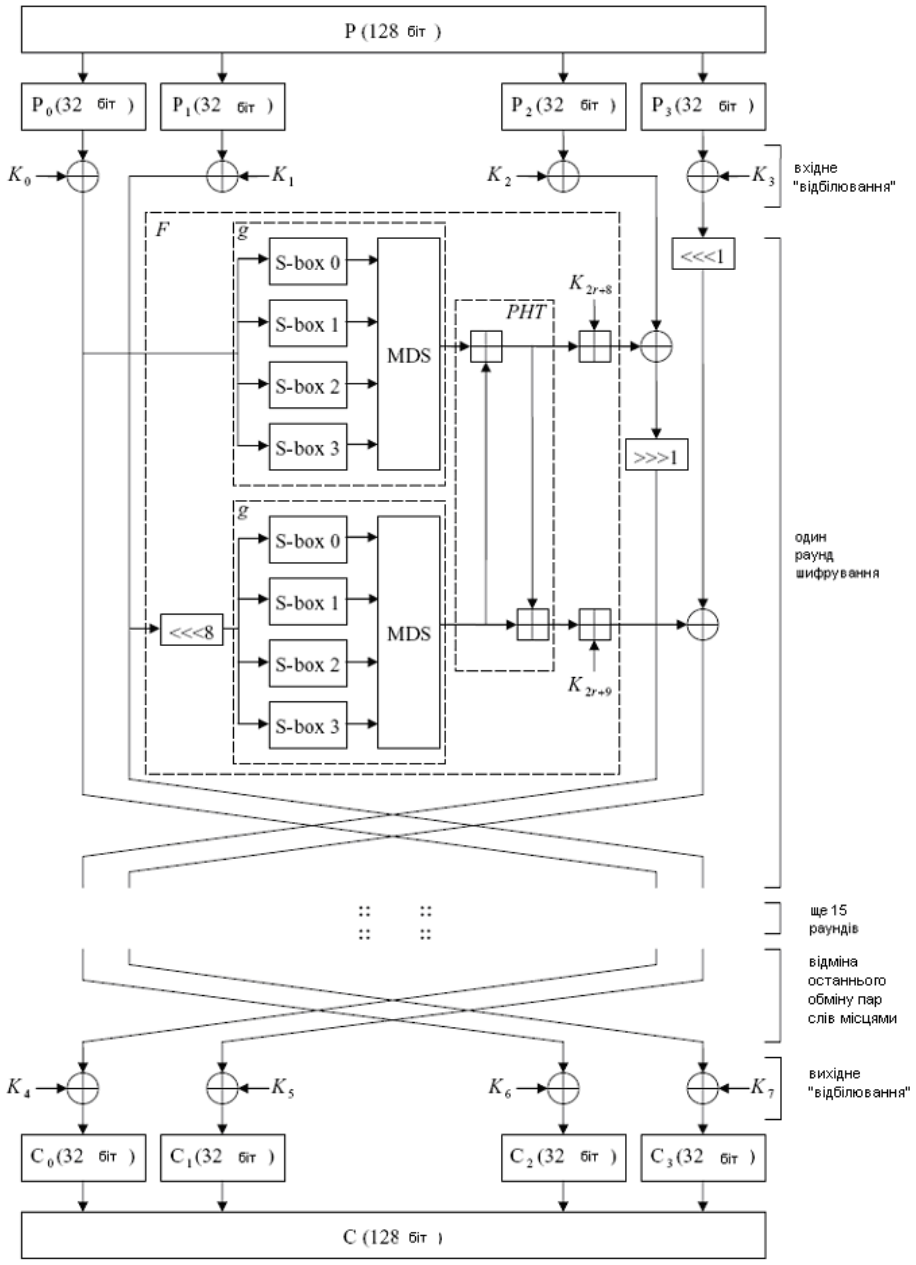

Рисунок 2.1 — Стуктурна схема шифрування даних алгоритму Twofish

Як видно з рисунку 2.1, вхідні дані записуються у регістрі. Потім відбувається поділ вхідної інформації на чотири слова, які додаються разом з підключами $K_0 \ldots K_3$ за допомогою операції xor. Цей крок алгоритму називається вхідне відбілювання. Після цього дані проходять через модуль, який називається функція *f*, де застосовуються різні перетворення і перестановки до тексту, що надходить. Ця функція *f* складається з двох *g* функцій, які містять ключі *S*-box, а також MDS-матрицю та псевдоперетворення Адамара (PHT – Pseudo-Hadamard Transform). Варто зазначити, що елементи MDS-матриці є байтові величини, що записані в шістнадцятковій системі числення. Після 16 раундів застосування цієї функції чотири слова даних ще раз додаються з використанням операції xor з підключами $K_4 \ldots K_7$ (рисунок 2.2). Цей етап називається відбілювання виводу. Нарешті, зашифровані або дешифровані дані приєднуються до вихідного регістру.

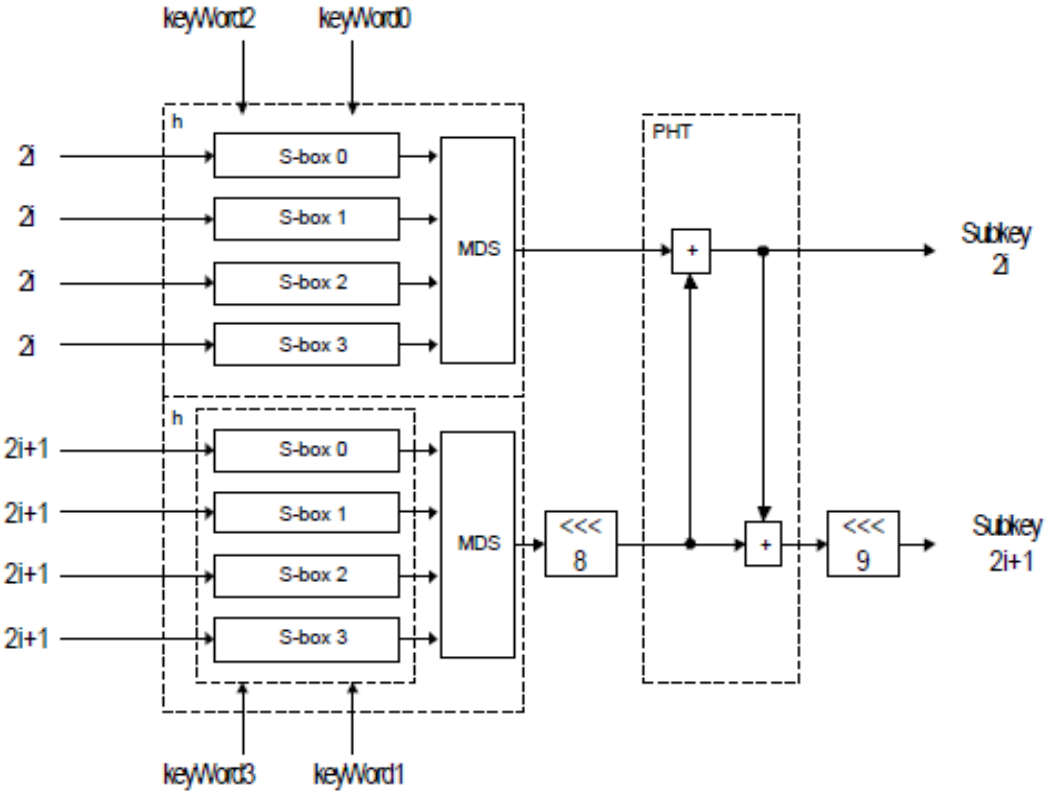

Рисунок 2.2 — Структурна схема модуля генерування підключів

З рисунку 2.2 можна побачити яким чином утворюються ключі. Цей модуль використовується для генерації сорока *K*-підключів, що необхідні для процесу

шифрування даних. Він приймає на вході непарне число від 0 до 39 і залежить від загального секретного ключа користувача. Можна помітити, що саме для генерації підключів використовується функція *g*, яка входить до складу функції *f*. Це важливо, оскільки вона дозволяє використовувати ту ж функцію для створення підлючів і здійснювати операцію шифрування, що дозволяє зменшити елементи устаткування, необхідні для реалізації цього алгоритму.

2.2 Розробка структури модулів функції *g* та *q*-перестановок

Існує ще один набір підключів, які використовуються в алгоритмі для створення *S*- boxів. У ході шифрування одного блоку відкритого тексту алгоритму для генерації ключів необхідно сформувати сорок 32-бітових ключів $K_0,\ldots,K_{39}$, а також чотири залежних від вхідного раундового ключа S-box'a, що використовується функцією *g*. Подальші дані отримуються шляхом множення відповідної частини секретного ключа користувача з RS-матриці, що описана в першому розділі. Варто зазначити, що вони обчислюються один раз для секретного ключа і залишаються фіксованими протягом всього процесу шифрування та дешифрування.

Структура Twofish володіє достатньою гнучкістю в плані об’єму, проте поступається швидкодією. Тому, для апаратної реалізації було використано тільки одну *g* функцію замість двох, що буде використовуватися для обчислення *K*-підключів і шифрування даних.

Важливою проблемою залишається обчислення *K*-підключів. Розглядалися два варіанти: попередньо обчислити *K*-підключі і зберігати в пам'яті RAM (повний ввід), або для обчислення *K*-subkeys на “льоту” в міру необхідності (нуль вводу). Було вирішено, що нульове введення підходить краще, ніж занесення інформації в RAM, оскільки це призведе до того, що необхідно буде споживати занадто багато місця на FPGA.

На рисунку 2.3 зображено *q*-перестановки, що є важливою операцією алгоритму Twofish.

Перестановка виконується побайтово, яку розбито на два блоки, над якими будуть проведені зміни. Найбільш важливі операції перестановки виконуються з чотирьма таблицями пошуку підстановки, позначеними t0, t1, t2, t3.

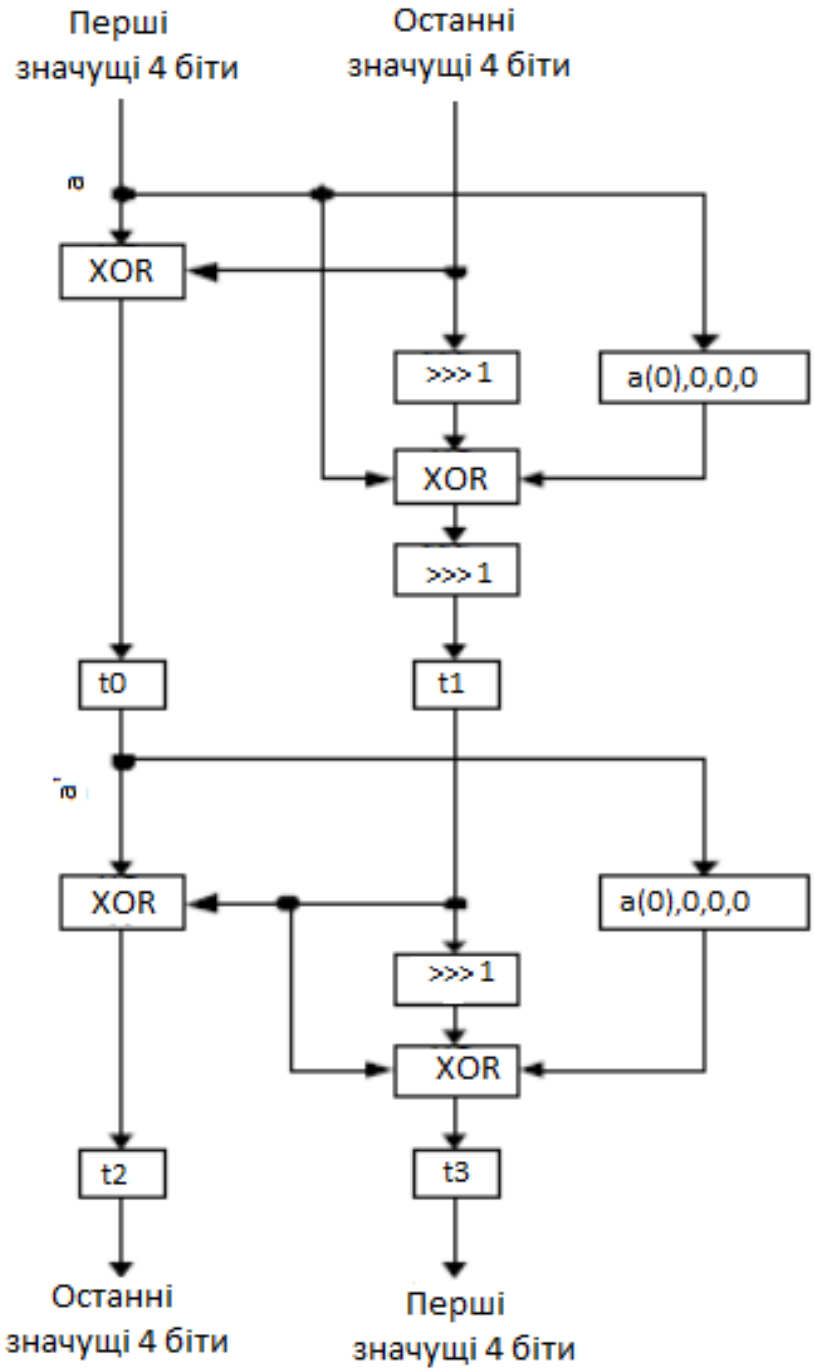

Рисунок 2.3 — Структурна схема q-перестановок

Кожна таблиця підстановки займає 4 біти вводу і видає 4-бітне значення. Таким чином, кожна таблиця підстановки має 16 записів по 4 біти. Існують чотири таблиці підстановки за q - перестановками і є дві різні q - перестановки — q_0 і q_1, кожна з яких міститься в таблиці підстановки.

Таблиці підстановки можна було запрограмувати у ROM, оскільки велика кількість q - перестановок повинні працювати паралельно. Таким чином, окремі ROM були б необхідні для кожного екземпляра q - перестановок.

Проте це зайняло б занадто багато ресурсів для невеликих реалізацій Altera, яка використовується для даного криптоалгоритму. Тому таблиці підстановки були реалізовані за допомогою мови VHDL, що дозволить дослідити роботу шифру.

Як показано на рисунку 2.4 S-Box працює з 32-бітним словом. Кожний байт слова проходить через три q-перестановки.

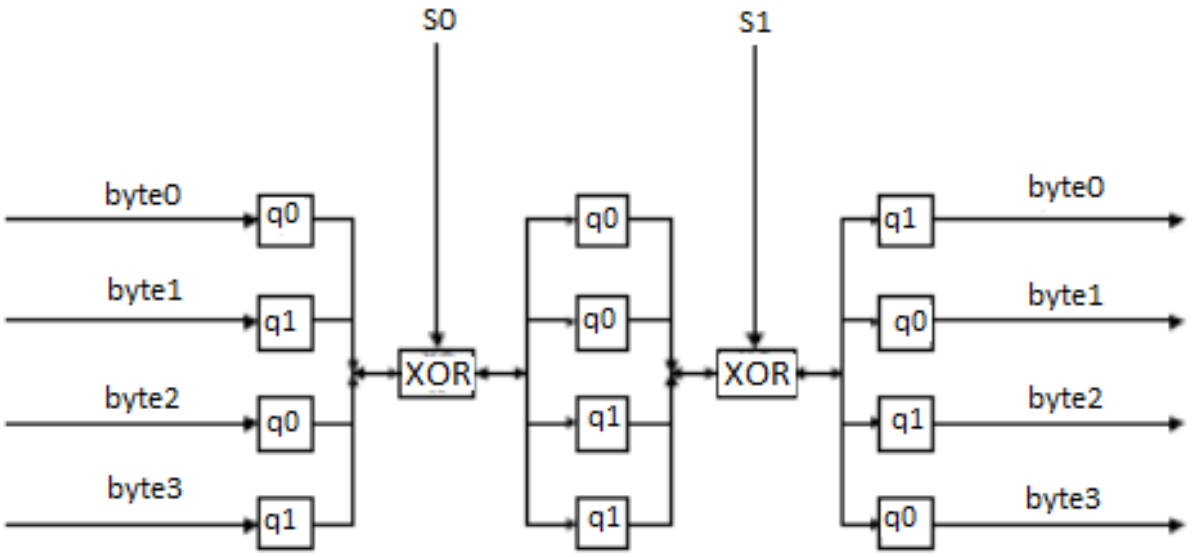

Рисунок 2.4 — Структурна схема S-Boxів

Вихід кожного банку даних q - перестановок перетворюється в слово і додається за допомогою операції XOR з 32-бітним значенням. Ці два значення, що отримані з секретного ключа користувача і S-Boxів алгоритму Twofish, називають

як «залежні ключі» від S-Вохів.

Вхідне 32-бітове слово розбивається на чотири байти. Кожен байт проходить через відповідний S-box, що залежить від раундового ключа. Кожен S-box описується біективною функцією, що перетворить «вхідний» байт у «вихідний». Чотири результуючі байти інтерпретуються як вектор довжини 4 над полем Галуа $GF(2^8)$, що збільшується ліворуч на MDS-матрицю розмірів 4×4 (обчислення також проводяться над полем $GF(2^8)$). Отриманий в ході перетворень вектор розглядається як 32-бітове слово, що і є «виходом» функції g.

$$x_i = \left\lfloor \frac{X}{2^{8i}} \right\rfloor \bmod 2^8, \quad i = 0,\ldots,3$$

$$y_i = s_i[x_i], \quad i = 0,\ldots,3$$

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} . & \cdots & . \\ \vdots & MDS & . \\ . & \cdots & . \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \qquad (2.1)$$

$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

де s_i - функції S-box'ів, *Z*- вихід функції *g*.

Кожен байт перетворюється на многочлен, я кому кожен степінь *p* від *x* представлений в тому випадку, якщо *p*-й біт рівний 1. Множення в полі *GF* зводиться до множення поліномів шляхом ділення на простий многочлен. Результат повертається назад у вигляді бітового вектора, встановлюючи 0, якщо відповідний степінь вхідного x має непарний коефіцієнт, в іншому випадку — встановлюється 1 (ділення відбувається за модулем 2).

В цьому випадку справедливим є обчислювати тільки три коефіцієнти: 0x01, 0xEF and 0x5B. Результат множення зменшується до рядів з операцією додавання

XOR для кожного вихідного біта. Наприклад, якщо помножити на 5B, то результат запишеться в байтах у змінну *b*:

b0 = a2 xor a0
b1 = a3 xor a1 xor a0
…
b7 = a7 xor a1

Матриця Reed-Solomon (RS) подібна до MDS - матриці (див. формулу (1.2)). В цьому випадку множення виконуються над матрицею 8x4 і вектор складається з 8 байт. Обчислення здійснюється над полем $GF(2^8)$ з поліномом x8+x6+x3+x2+1.

На відміну від MDS-матриці RS-матриця має велике число різних співмножників. Для того, що мінімізувати ресурси, які використовуються для операції XOR було запропоновано, щоб кожний 23-й співмножник призводив до рівності, яка б відповідала значенням MDS- матриці.

Ці рівності не були задані і їх необхідно виводити. Це дозволяє обчислити загальну рівність для множення двох поліномів над полем $GF(2^8)$. Для цього необхідно виконати обчислення для 23 випадків.

2.3 Розробка структури модуля вибору операцій шифрування та дешифрування

Оскільки Twofish є симетричний криптоалгоритм, то операції шифрування та дешифрування можна реалізувати апаратно. Першою вимогою є те, що підключі використовуються у зворотньому порядку, а друга вимога полягає у вихідній послідовності кожного раунду. Реалізація шифру Twofish, використовуючи FPGA пристрої, вимагає реалізації наступних важливих функцій:

- перестановки;
- додавання за модулем 2 (побітово);
- додавання за модулем 232 слів довжиною 32-біти;
- залежність ключів від S-boxів;

- MDS-матриці;
- PHT перетворення.

Ці операції можна реалізувати апаратно, проте вони не є простими в реалізації. Важливо впорядкувати з'єднання між логічними комірками схеми шифру.

Операція додавання за модулем 2 (XOR) є досить гнучкою для апаратної реалізації. Кожний вихідний біт залежить від значення двох вхідних бітів. Очевидно, що така операція може бути використана для функції f та g у таблиці пошуку FPGA.

Як видно з рисунку 2.5 вихідний етап складається з чергування операції зсуву та XOR.

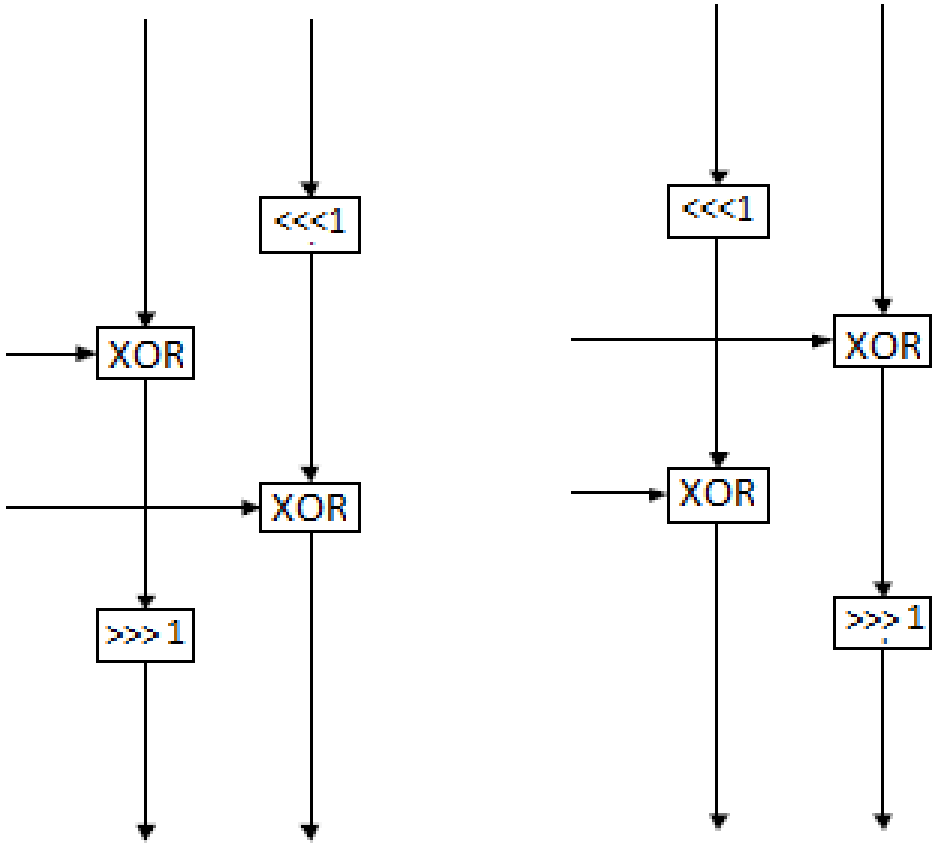

Рисунок 2.5 — Структура вибору операцій шифрування та дешифрування

Вихідний етап є віддзеркаленням операцій шифрування та дешифрування і

може бути побудований таким чином як показано на рисунку 2.6.

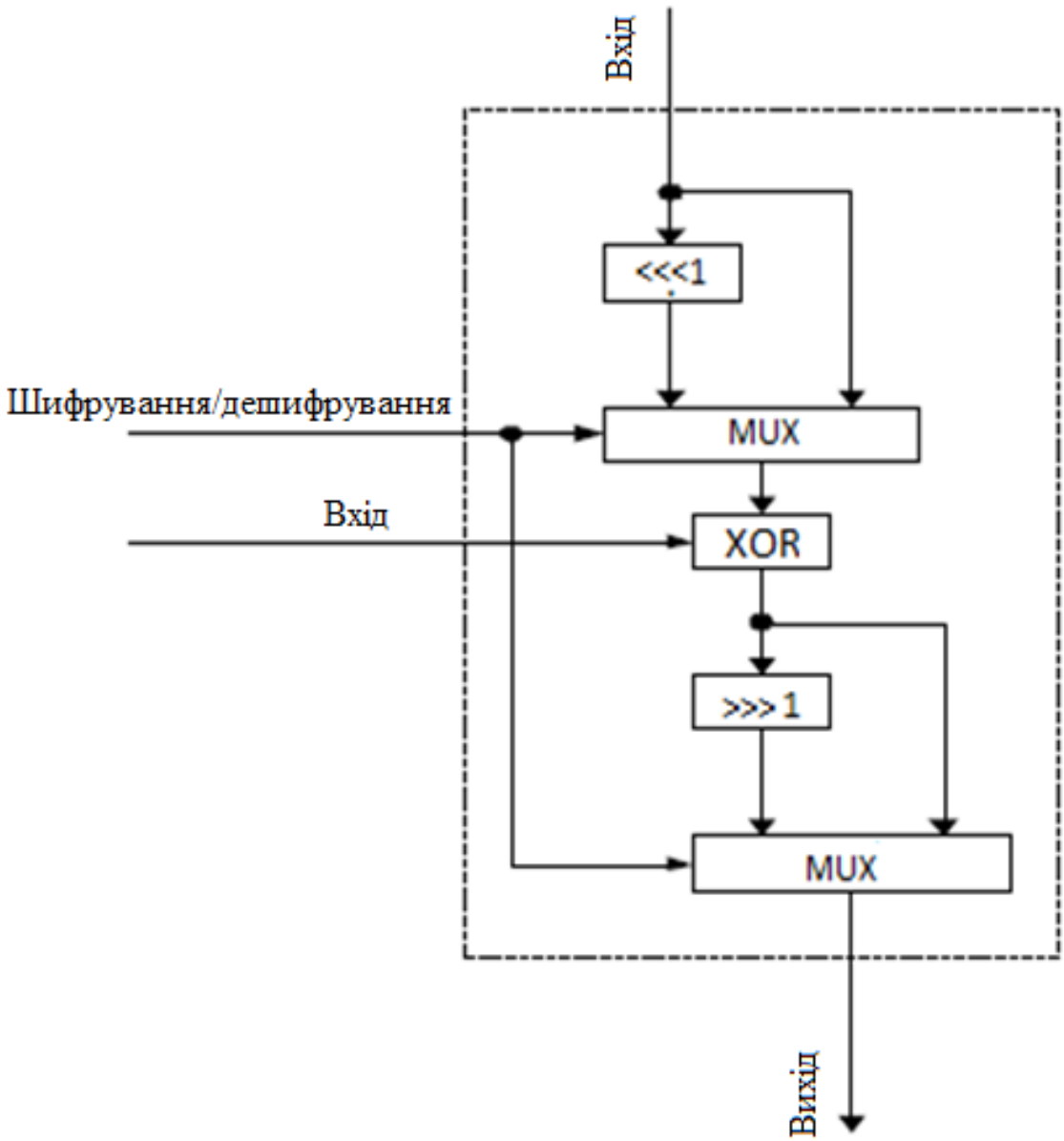

Рисунок 2.6 — Структура розробленого блоку для операції вибору

Модуль регістра вводу складається з регістру даних вхідної інформації, що проходить етап відбілювання. Блок даних містить інформацію розміром 128 біт. Вихідний сигнал записує дані в регістр (рисунок 2.7). Вхідне відбілювання виконується асинхронно. Тільки на виході модуля є чотири блоки 32-бітних слів, що відповідають блокам з 4 слів початкових даних, над якими проведено операцію відбілювання.

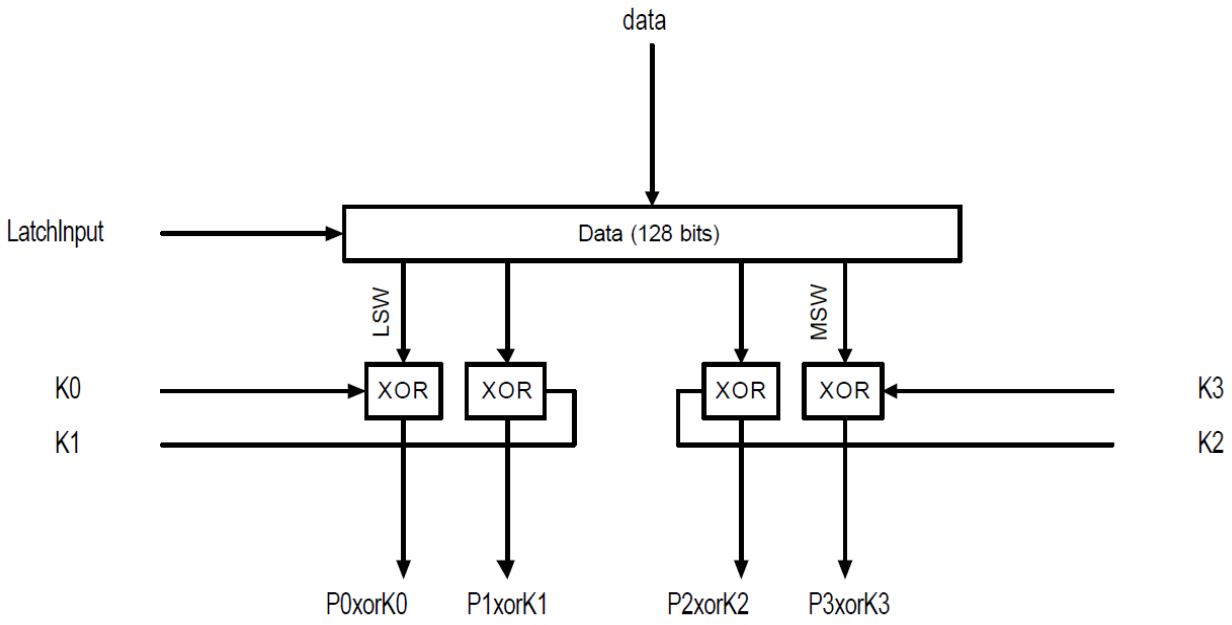

Рисунок 2.7 — Структурна схема модуля регістра вводу

Цей результат є достовірний, якщо 4 підключі встановлено правильно на вході. Підключі K_0, K_1, K_2 і K_3 використовуються для шифрування та K_4, K_5, K_6 і K_7 — для дешифрування.

Модуль регістра виводу об’єднує етап вихідного відбілювання і запису в регістр. Чотири вхідні слова додаються за операцією XOR з 4 підключами для отримання шифротексту (рисунок 2.8).

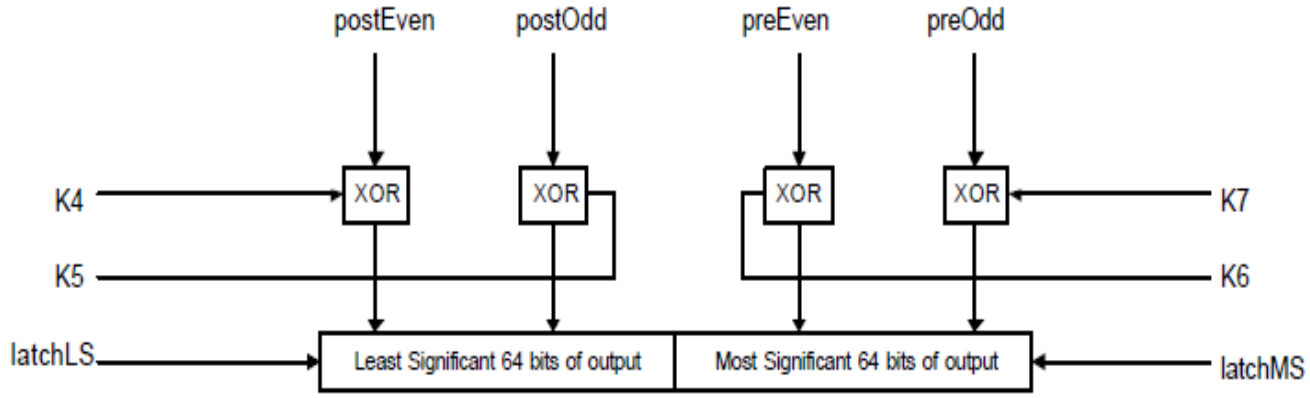

Рисунок 2.8 — Структурна схема модуля регістра виводу

Піключі мають бути встановлені як K_4, K_5, K_6 і K_7 — для шифрування та K_0, K_1, K_2 і K_3 — дешифрування. Оскільки, структура даного криптоалгоритму передбачає, що тільки два ключі будуть доступними в будь-який момент часу. На виході будуть два сигнали latchLS і latchMS.

Модуль регістра ключів використовується для зберігання ключів довжиною 128-біт і виводить дві змінні (S_0 і S_1), використовуючи RS-матрицю. Ключ записується в регістр, якщо перемичка 'Latchkey' є відкритою.

Дві S-змінні обчислюються в два етапи, використовуючи багатократно RS-матрицю. Протягом часового циклу, коли записується ключ, мультиплексор встановлює на вході RS-матрицю в два останні значущі слова ключа і результат записується в регістр S_0. В інший період часу на вхід подається два перших значущих слова і на виході S_1 отримуємо значення S_1.

Протягом проектування було проведено мінімізацію апаратної реалізації шляхом використання модифікованої структури шифру, тобто, щоб для всіх обчислень бралася одна і таж g функція. В додатку А показано діаграму модифікованої f-функції, яка повинна задовольняти наступні вимоги: ввід та інверсія вводу мультиплексуються для того, щоб вивести два вводи, які можуть бути представлені як входи для g-функції. Окрім того, щоб обчислити результат за перетворенням PHT, необхідно додати два регістри для зберігання значень першого входу або попередній доти, поки обчислюється другий вхід. Таким чином, ці два регістри працюють як конвеєрні регістри. Інші два мультиплексори використовуються для того, щоб можна було вибирати чи обчислювати ключ чи шифрувати або дешифрувати дані.

Впровадження модифікації f-функції разом з іншими елементами проекту призвело до завершальної архітектури, що наведена в додатку Б. Мультиплексори використовують для того, щоб можна було вибрати дані з модуля, який містить відкритий текст, що використовується на початку циклу шифрування і дешифрування, а також дані, отримані після кожного раунду шифрування та дешифрування.

Регістри при виведенні модифікованої *f*-функції також є конвеєрними. Вони зберігають значення підключів K (K_{2r+8} *і* K_{2r+9}, *де* r — число раундів від 0 до 15), що використовується в кожному циклі шифрування та дешифрування.

Після кожного раунду дані записуються в 4 регістра. Цей крок виконує необхідну підкачку даних після кожного раунду. На останньому етапі пристрій повинен підтримувати реалізацію управління системою (додаток В).

Шифром керує відносно складний пристрій. В той час, як в стані очікування контролер чекає запиту користувача і розповсюджує свою доступність, поширюючи сигнал. Сигнал ‘loadKey’ переводить контролер у стан, в якому він фіксує ключову інформацію довжиною 128 біт, що надходить із вхідного порта. Сигнал запуску встановлює контролер в режим шифрування або дешифрування. Варто зазначити, що контролер управляє операціями шифрування та дешифрування.

Перший крок — обчислити операцію відбілювання 4 вхідних даних підключа. Після того ми будемо мати обчислену кожну пару і два верхні регістри зліва фіксують результат вхідного модуля регістра (додаток Б). Після обчислення другої пари ключової інформації відбувається фіксація двох інших регістрів справа.

Після цього контролер проходить через 16 раундів шифрування або дешифрування. Ці раунди складаються з чотирьох станів. Парні і непарні ключі для раунду обчислені і записані в регістри. Тоді використовується *f*-функція для того, щоб можна було отримати деякі вихідні дані, що додаються до вмісту регістрів. У кінці раунду результат записується у регістр. Виходи раундів мають підкачку пам’яті між лівою і правою сторонами схеми.

Після завершення 16 раундів чотири блоки вихідних підключів, що пройшли операцію відбілювання, обчислені і використовуються як два для того, щоб можна було провести операцію відбілювання виводу. Як тільки контролер переходить у стан очікування, регістр зашифрованого тексту записується на вихід процесу.

Варто зазначити, що підключі повинні бути згенеровані на вході *f*-функції розміром від 0 до 39 біт. Виконання цього в контролері було б неприпустимим.

Контролер використовує лічильник, який рахує яким чином мають бути згенеровані ключі (0,1,2,3,8,9,10 … 39,4,5,6,7 для шифрування). У звичайному режимі він відключений, а включається при необхідності.

Отже, шифр Twofish містить відносно велику кількість елементів, які потрібно апаратно реалізувати, щоб отримати коректний результат. Специфікація шифру має багато нюансів, які для некваліфікованого криптографа можуть скласти проблему. Складність системи утруднює тестування кожного компонента. Після перегляду всього VHDL коду, була розглянута реалізація на мові програмування C++ для того, щоб перевірити тестові таблиці відповідей.

3 РОЗРОБКА ТА ДОСЛІДЖЕННЯ ПРОЕКТОВАНОГО ПРИСТРОЮ

3.1 Розробка компоненти шифрування даних криптоалгоритму Twofish

В попередньому розділі даного дипломного проекту були розглянуті основні засади побудови структурних схем симетричного шифру Twofish. Варто зазначити, що всі підключі обчислюються за межами кристалу і потім записуються в пам’ять мікросхеми.

Враховуючи той факт, що метою даного дипломного проектування є апаратна реалізація операцій шифрування та дешифрування криптоалгоритму Twofish засобами мови VHDL, то було розроблено інтерфейсний опис даної компоненти (додаток Г).

Розглянемо зовнішній інтерфейс апаратної реалізації алгоритму Twofish (рисунок 3.1). Метою створення даного інтерфейсу є його загальне призначення, щоб дозволило здійснювати просте керування в мікропроцесорні системі і дозволило виконувати шифрування з максимальною швидкістю.

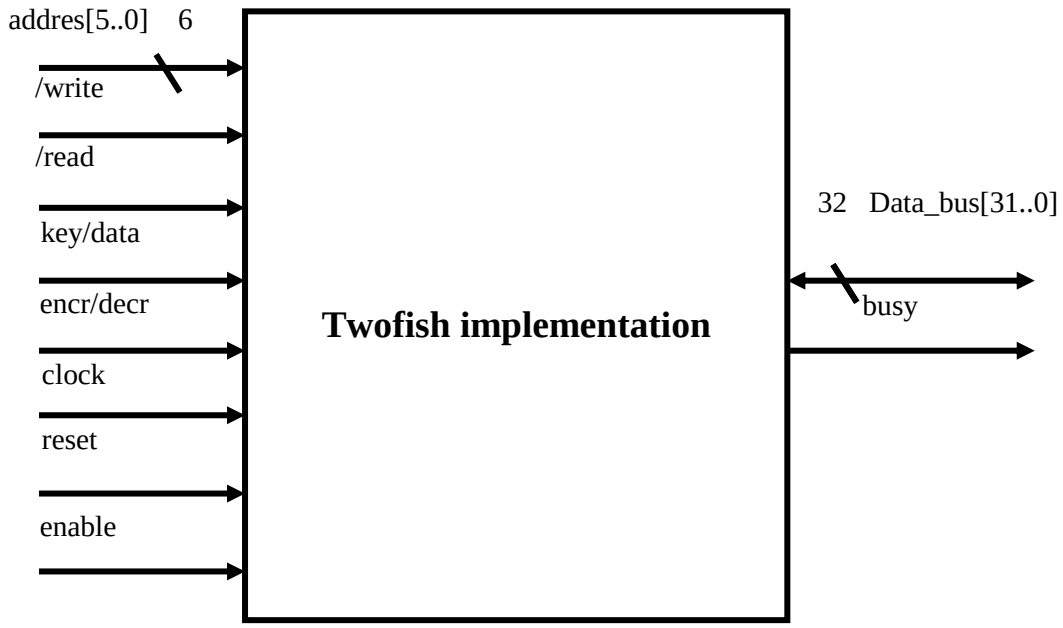

Рисунок 3.1 — Зовнішній інтерфейс шифру Twofish

Дана компонента має наступні входи та виходи:

- address[5..0] — 6-розрядна адресна шина, яка дозволяє отримати доступ до всіх регістрів у межах визначеної схеми. Варто зазначити, що 5-й розрядний адрес [5] є старший значущий адрес;

- /write — вхідний сигнал для запису операцій. Найвищі піки даного сигналу вказують на те, що відбувається запис інформації в регістри;

- /read — вхідний сигнал для зчитування операцій. Найвищі піки даного сигналу вказують на те, що дані зчитуються через комп'ютер;

- key/data — вхідний сигнал, який дозволяє вибрати або дані або ключі для запису в регістри. У найвищому стані відбувається вибір ключів;

- encr/decr — з цим вхідним сигналом хост-система дозволяє вибрати виконання однієї з операцій: шифрування та дешифрування. У найвищому стані відбувається операція шифрування;

- clock — дозволяє синхронізувати сигнал на всіх часових елементах проекту для одного раунду;

- reset – здійснює перевантаження синхронізації всіх часових елементів в проекті;

- enable — цей вхідний сигнал дозволяє у високому стані записувати або зчитувати інформацію з будь-якого регістру в межах пристрою. Цей сигнал може бути важливим для системи мікропроцесора. Якщо немає необхідності, він може постійно знаходитися у стані 1;

- busy — цей вихідний сигнал генерується пристроєм. Високий стан сигналу вказує на те, що відбувається операція шифрування. Зміна стану сигналу з високого у низький — вказує на те, що операція шифрування завершилася;

- data_bus[31..0] — двонаправлена шина даних довжиною 32 біти. Самим значущим бітом даної шини є 31-й біт.

Весь вхідний зв'язок з пристроєм шифрування формується через регістри, які є доступними для використання адресного простору. На рисунку 3.2 можна побачити адресний простір регістрів, поділений на декілька сегментів. Команди пересилання даних призначені для завантаження чисел з оперативної пам'яті в

регістри даних, запису даних із чисельних регістрів в оперативну пам'ять, копіювання даних з одного чисельного регістра в інший.

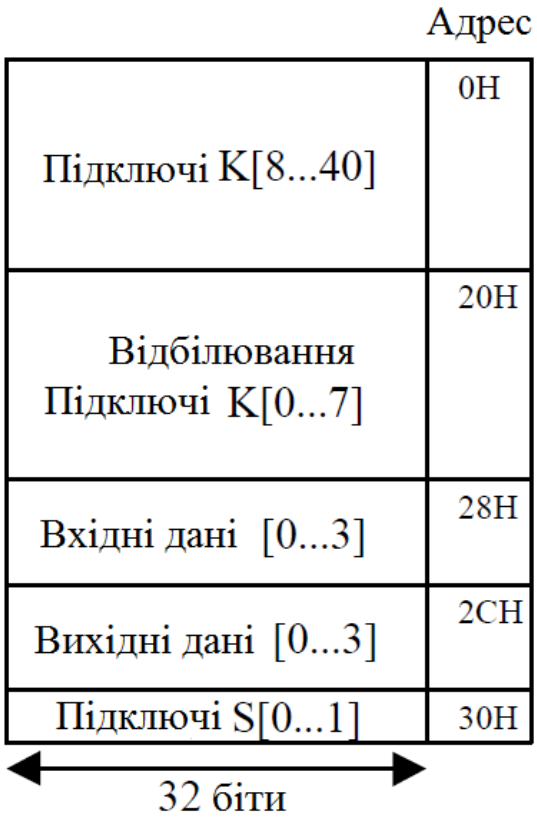

Рисунок 3.2 — Адресний простір використання пам’яті для зберігання внутрішніх ключів

Для кожного сегмента надається лише початковий адрес. Всі регістри в межах одного сегмента розташовані послідовно. Наприклад, *K*[8] має адрес 0*H*, *K*[10] – 2*H* і т.д. Регістри даних поділені на дві групи: вхідні і вихідні. Проте, доступ до них можна було тримати за тією ж адресою і вони відрізнялися б лише назвами вхідних сигналів /write або /read, але їх розділено таким чином, що три останні значущі біти даних адреси співпадають з трьома останніми значущими бітами адреси відповідного ключа. Це дозволяє спростити використання операції відбілювання ключів.

Отже, підсумовуючи роботу даної компоненти, можна зазначити, що:

- дані і ключі будуть читатися і записуватися через двонаправлену шину довжиною 32 біти;

- для потоку даних використовується pipeline структура;

- всі підключі обчислюються завчасно і зберігаються на зовнішній пам’яті і доступні для кожного раунду обчислення.

Адресну шину потрібно для початку встановити в одну з адрес у межах вхідного адресного простору регістра (рисунок 3.3). Після того, як адреса виявилася стійкою, в шину даних встановлюються відповідні дані або значення ключа до зростання фронту сигналу на вході /write, який вказує на фіксування даних у даному регістрі.

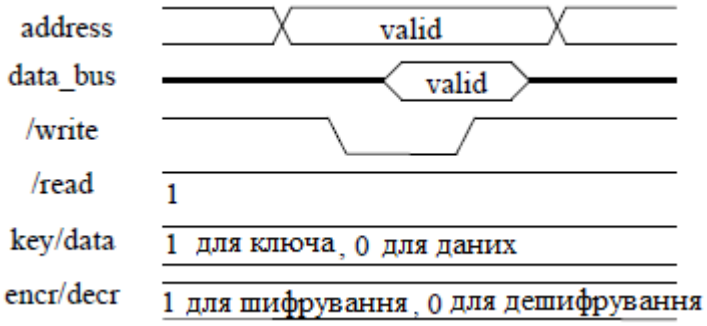

Рисунок 3.3 — Діаграма запису даних та ключа

Ключ встановлюється у відповідний стан, в залежності від того, що необхідно записати в регістр — дані чи ключ. Фактично, цей сигнал використовується для того, щоб переключити адресну шину внутрішньої пам’яті між зовнішньою адресною шиною і внутрішнім лічильником, що використовується під час шифрування. Сигнал encr/decr має бути встановлено завчасно в необхідний стан, що залежить від наміру проводити операцію шифрування чи дешифрування.

Як тільки дані і ключі подані до регістрів, їх можна переписати, проте не можна прочитати. В цьому випадку дані, що записуються до регістра за адресою 2BH, подають сигнал на пристрій, що можна виконувати операцію шифрування за алгоритмом Twofish.

Операція зчитування даних є схожі до операцію запису (рисунок 3.4). Замість /write сигнал /read використовується для такту зчитування. Звичайно, в той

же час пристрій шифрує дані, що надходять до data_bus. Таким чином хост-система виступає в ролі шини.

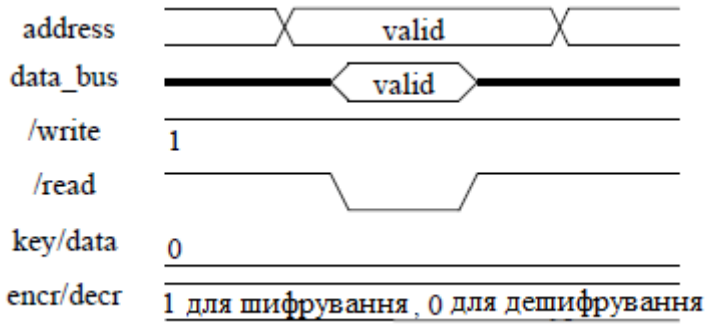

Рисунок 3.4 — Діаграма зчитування даних пристроєм

Дані, що надійшли до data_bus зчитуються, після чого вхід регістру /read встановлюється в 0.

Використання ключової інформації пристроєм відбувається лише після того, як відбулася активація сигналу. Ця послідовність є дуже простою:

1) дані подаються на вхід регістрів і зчитуються, після чого зберігаються протягому одного раунду. Відповідно на вході встановлюється сигнал 1;

2) виконується операція шифрування;

3) записуються зашифровані дані в вихідні регістри і на вході встановлюється сигнал 0.

Існує два важливих рішення, які необхідно враховувати:

1) після того як сигнал на вході переходить в стан 1 можна подавати наступні дані;

2) коли сигнал переходить в 0 стан, тоді можна зчитувати зашифровані дані.

Варто зазначити, що зашифровані дані потрібно зчитувати одразу, оскільки при надходженні наступних даних для шифрування, попередні можуть бути втрачені.

Структура pipeline, що використовується в даному інтерфейсі здійснює запис і зчитування даних можливим протягом обробки операції шифрування.

Послідовність операції організована наступним чином: запис даних; запис наступних даних; якщо пристрій зайнятий, то він переходить в стан очікування і сигнал встановлено в 0 стан; зчитувати зашифровані дані і так все спочатку.

3.2 Послідовність синтезу операції шифрування алгоритму Twofish

Чільне місце серед широкої гами засобів проектування з використанням мови VHDL займає пакет Active-VHDL. Active-VHDL є інтегрованим засобом розробки VHDL проектів. Ядром системи є VHDL симулятор з підтримкою стандарту мови. Наявність вбудованих допоміжних засобів таких як редактор тексту з функцією синтаксичного аналізу, засобу побудови та відлагодження керуючих автоматів, бібліотеки широковживаних конструкцій мови та систем підказок дозволяють до мінімуму звести затрати розробника на допоміжні операції і, тим самим, скоротити тривалість розробки. В поєднанні з зовнішніми програмними засобами відомих фірм Synplicity, Xilinx, Actel, Altera, Lucent пакет Active-VHDL дозволяє розробляти повністю завершені пристрої.

На етапі логічного синтезу виконується синтез принципової схеми пристрою (т.зв. netlist) з використанням бібліотечних елементів заданого кристалу за заданим вихідним VHDL описом прострою. Для логічного синтезу процесора використано програму Synplify Pro ver. 7.0 фірми Synplicity, яка займає лідируюче положення на ринку засобів логічного синтезу для ПЛІС. Створений даною програмою netlist є вихідним файлом програмних інструментів розташування і трасування (Place and Route tools). Такою програмою для кристалів серії Virtex фірми Xilinx є програмний засоб з пакету Xilinx WebPack 5.1 фірми Xilinx, – Design Manager.

Кінцевим результатом виклику засобів розташування і трасування є файл бітового потоку (bitstream file), який завантажується до програмно-керованого цим бітовим потоком FPGA-пристрою. Даний пакет спроможний генерувати VHDL-

модель пристрою після синтезу, отриману на кінцевому етапі проектування, тим самим дозволяючи провести остаточну часову симуляцію на фізичному рівні спроектованого пристрою. Модель грунтується на примітивних моделях, які підтримуються фірмовими VHDL-бібліотеками виробників FPGA/CPLD.

Отрманий netlist екпортувався в Xilinx для застосування апаратної реалізації. Для пристрою було вибрано елементну базу сімейства XC4000XL. В більшості було обрано характеристики, описані нижче.

В FPGA Express:

- тактова частота — 50 МГц;
- клас швидкості — 09;
- швидкість перегляду — повільна;
- глобальний буфер — автоматичний, але у випадку використання сигналів clock та reset;
- ієрархія — відсутня, крім випадку зберігання підключів.

В Design Manager:

- акуратна незв’язна логіка — on;
- логічне розміщення елементів дозволяє зменшити логічний рівень — on;
- генерування функцій з 5-ма входами — on;
- стратегія CLB пакету — повний пристрій;
- Pack CLB — структура;
- Pack I/O реєструється/закривається в IOBs — off.

Після проведення аналізу реалізації всіх компонентів шифру Twofish, що проведна в Xilinx FPGA, здійснено перевірку їх синтезу: *q*-перестановки, MDS-матриці, PHT перетворення, *f*-функція, шифрування та дешифрування даних.

Основна мета тестування компонентів пристрою полягала в тому, щоб знати скільки пам’яті потрібна і яка швикодія. Всі вищезгадані компонети реалізують внутрішні функції, тому важко визначити затримку кожної, оскільки їх входи не з’єднані із зовнішніми контактами. Затримка, що представлена зовнішніми контактами є значно більшою ніж затримка серед CLBs. Щоб уникнути цього ефекту, всі схеми були протестовані в конфігурації, що показана в додатку Д.

Було прийнято рішення використовувати два рівня регістрів — між входами і виходами, оскільки перший рівень (reg1 і reg4) міг бути розташований в елементній базі IOB, тоді в цьому випадку шлях між CLBs і регістрами буде довгий. Затримка розповсюдження вимірювалася як часовий інтервал між моментами, коли з'являється стабільне значення на виходах регістру reg2 і час, коли дане значення з'являється на входах регістру ння reg3. У випадку послідовних з'єднань бралися до уваги час встановлення регістру reg3 і розповсюдження затримки через reg2.

В пристроях Xilinx час установки дорівнює 0 нс, якщо регістр розташований в той же самий CLB як продовження комбінаційної логіки. В іншому випадку не перевищує 2 нс. Затримка, що викликана регістром reg2, складає приблизно 3-4 нс. У випадку комбінації проекту ці додаткові затримки можуть бути прийняті до уваги тільки для одного раунду функції.

3.3 Дослідження часових характеристик пристрою

Для проведення часової симуляції в середовищі необхідно вказати тип симулятора: ActiveVHDL. Для кінцевого проведення часової симуляції мікропроцесора призначені файли time_sim.vhd та time_sim.sdf. Дані файли завантажуються в середовища Active-VHDL, де файл time_sim.vhd є VHDL описом пристрою після синтезу та компілюється і симулюється як звичний VHDL компонент проекту. Результатом симуляції є коректна робота синтезованого пристрою на кристалах Xilinx.

Реалізація *q*-перестановок оправдала надії, оскільки потребує 16 логічних комірок. Час роботи складає 9.6 нс (рисунок 3.5). Послідовно в проекті затримка розповсюдження сигналу складає 13.3 нс. Окрім того, потрібно розглянути час установки для регістру reg3. Мінімальний безпечний період складає 13.8 нс.

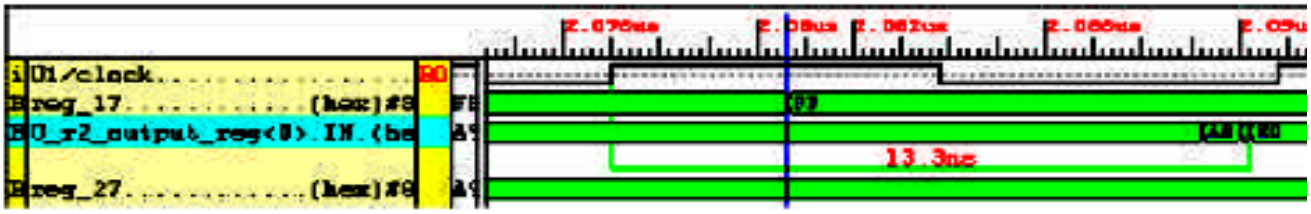

Рисунок 3.5 — Часова симуляція *q*-перестановок

Розглянемо реалізацію MDS-матриці (рисунок 3.6). Компілятор Xilinx показав, що вона не є набагато швидшою, ніж виконання *q*-перестановки, викликаючи затримку 9.5 нс (13.5 нс з регістру reg2). Копмілятор реалізував тільки 48 логічних комірок (прогнозувалося 52), проте це призвело до багаторівневої структури. Ймовірно, що єдиний спосіб реалізувати пристрій полягав би в тому, що потрібно використовувати компоненти бібліотеки.

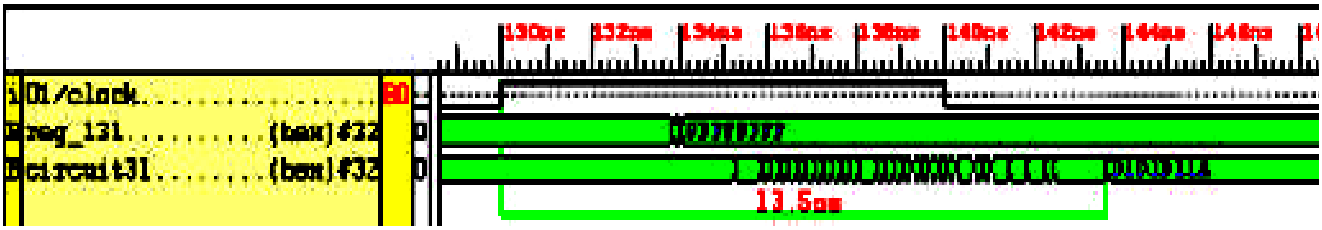

Рисунок 3.6 — Часова симуляція MDS-матриці

Перетворення PHT було реалізовано з використанням суматорів з переносом Xilinx CLBs і потребувало 34 елементи. Прогнозувалася довга затрика, яка необхідна для розповсюдження сигналу через весь суматор, проте Xilinx FPGA є достатньо швидким і затримка склала 6.8 нс (10.8 нс — беручи до уваги затримку розповсюдження сигналу через reg2). Мінімальний часовий період звіту склав 14.7 нс (рисунок 3.7).

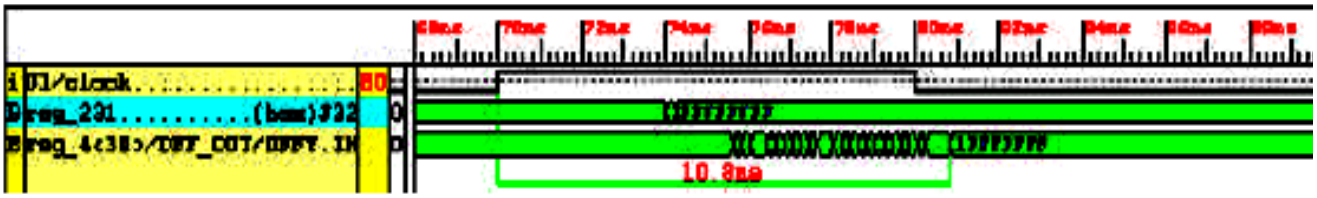

Рисунок 3.7 — Часова симуляція PHT-перетворення

Апаратна реалізація криптоалгориму Twofish складається з повної функції. Резалізовано комбінаційну і послідовну версію схеми з варіантом оптимізації. Найменшим пристроєм, який реалізує повну функцію шифрування є XC4028XL. Для завершення проекту був використаний пристрій HQ208.

Як і очікувалося, комбінаційна версія реалізація є швидшою нж послідовна версія. Пропускна здатність послідовної реалізаціє є досить низькою. Згідно результатам, отриманими для індивідуальних компонентів Twofish, ми очікували тактову частоту рівну 50 МГц. Фактично, тактова частота не перевищувала рівень 36 МГц. Це нижче на 50% від пропускної здатності комбінаційної схеми. Це може бути результатом поганої роботи, оскільки проект займає приблизно 90% доступного простору в пристрої XC4028.

Хоча послідовна реалізаціє є повільнішою, проте її можна розглядати у виборі шифру ECB або з використанням операцій чергувань, оскільки для цих методів обчислення можуть бути виконані для багатьох блоків даних одночасно. Очевидно, що рішення потребує реалізації додаткових буферів FIFO замість регістрів на вході і виході одного раунду схеми і одного буфера FIFO з лівої половини даних. Реалізація цих буферів була б дорогою, проте ми можемо використати той факт, що Xilinx CLB можна сконфігурувати як 16 x 1 двонаправлений порт RAM. Тому, кожен буде потребувати, щоб буфер FIFO довжиною 32 біти використовував тільки 32 логічні комірки. Використовуючи pipeline метод для нашого послідовного проекту (сім рівнів регістрів), дав би пропускну здатність 263.9 Мбіт/с. Один буфер FIFO може зберігати до 16 блоків даних, тому пропускна здатність може бути збільшена. Ця операція не потребує ніяких додаткових апаратних засобів.

Зауважимо, що, оскільки шифрування і дешифрування даних проходить за однакову кількість тактів, то продуктивність цих операцій буде однаковою.

Отже, синтез, розміщення і трасування розробленої VHDL моделі процесора шифрування за алгоритмом Twofish показало, що ці моделі є придатними для реалізації у ПЛІС. Разом з тим, завдяки переносимості VHDL, можна синтезувати

розроблену модель процесора із використанням бібліотеки замовлених інтегральних схем.

4 ОХОРОНА ПРАЦІ

Поняття “охорона праці” визначено статтею 1 Закону України “Про охорону праці”. Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів і засобів, спрямованих на збереження здоров’я і працездатності людини в процесі праці [9].

Основною ціллю охорони праці є створення на кожному робочому місці безпечних умов праці, експлуатації обладнання, зменшення або повна нейтралізація дії шкідливих і небезпечних виробничих факторів на організм людини і зниження виробничого травматизму та професійних захворювань.

4.1 Аналіз санітарно-гігієнічних умов праці

У даному розділі розглядаються питання по охороні праці при розробці апаратної реалізації операції шифруваня симетричного криптоалгоритму Twofish.

Даний програмний комплекс розроблявся на базі інформаційно-аналітичного центру. У відділі постійно працює до дев’яти чоловік. Види виконуючих робіт – вивчення матеріалів у відповідності з розвитком техніки, розробка, налагодження і запуск програм на комп’ютерах відділу. Для цих цілей у відділі використовується 9 персональних ЕОМ класу Celeron 1700, а також лазерний принтер формату А4 марки Canon LBP-810. На персональних комп’ютерах (ПК) встановлені монітори марки SAMSUNG SyncMaster 755 DFX (розширення 1024x768).

План інформаційно-аналітичного відділу фірми “Polyservice” подано на рисунку 4.1.

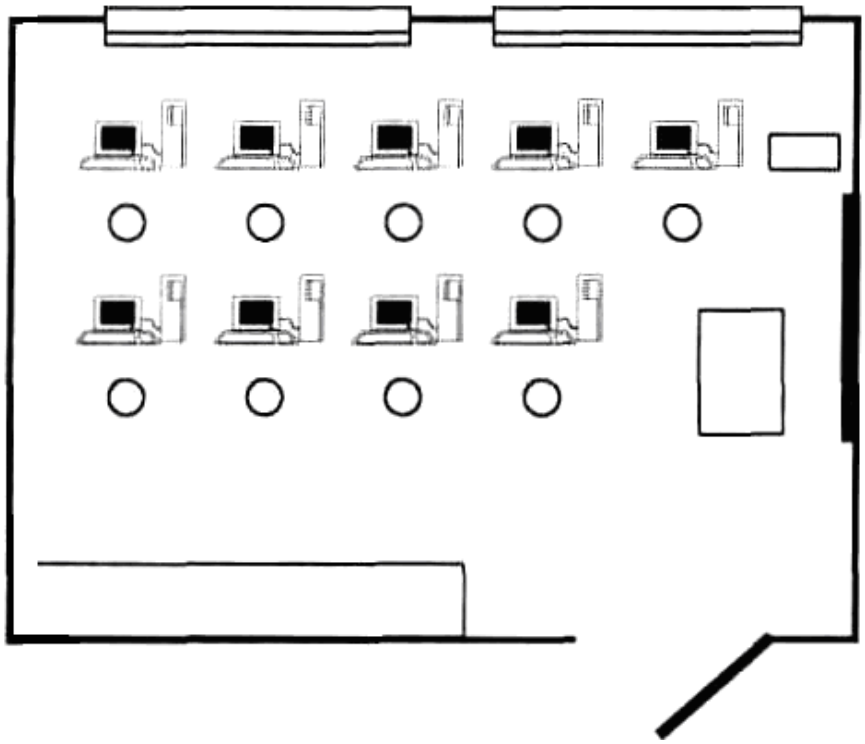

Рисунок 4.1 – План інформаційно-аналітичного центру

Розміри відділу описані в таблиці 4.1.

Таблиця 4.1 − Розміри відділу

Позначення	Визначення	Значення
I	Довжина	9 м
d	Ширина	7 м
h	Висота	4 м
S_0	Площа	63 м^2
V_0	Об’єм	252 м^3

Згідно СН-245-71, на одного працюючого об’єм приміщення повинен складати не менше 20 м^3, площа – не менше 6 м^2. Число працюючих у приміщенні Np=9. Таким чином, на одне робоче місце приходиться площа S=63/9=7 (м^2) і об’єм V = 252/9=28 (м^3). Ці значення відповідають вимогам.

Крім того, повинні дотримуватися норми приміщення, подані у таблиці 4.2.

Таблиця 4.2 – Норми приміщення

Параметр	Значення
ширина основних проходів	≥1200 мм
ширина допоміжних проходів	≥700 мм
відстань між двома столами, якщо між ними є стілець	≥1300 мм

У розглянутому приміщенні відділу: відстань між двома столами становить 1500 мм, відстань між двома столами в ряді − 1500 мм, а між рядами − 2000 мм.

Отже, норми виконуються.

У технічних умовах роботи ЕОМ вказуються робочі діапазони параметрів мікроклімату:

− температура повітря 5 − 45 °С,

− відносна вологість повітря 40 − 90 %.

Однак, вимоги точного регулювання параметрів повітряного середовища приміщення значно звужують ці діапазони.

З метою забезпечення комфортних умов для персоналу, а також максимальної безвідмовності функціонування техніки, встановлюють вимоги до повітряного середовища приміщень [10]. Так, у відділі повинні бути:

− температура повітря 18 – 22 °С,

− відносна вологість повітря 50 – 60 %,

− атмосферний тиск 1013 – 1013,5 гПа.

При зниженні тиску погіршується відвід тепла від елементів ЕОМ, знижуються ізоляційні властивості повітря.

Як було показано вище, показники об’єму і площі приміщення на одного працюючого відповідають нормативним значенням.

Роботи, що проводяться в інформаційно-аналітичному відділі фірми “Polyservice” відносяться до легких фізичних робіт групи 1а, відповідно до ГОСТ 12.1.005-88, оскільки вони проходять сидячи і не вимагають фізичного навантаження, здійснюються при нормальних метеорологічних умовах і не

викликають забруднення одягу і рук. Витрати енергії не перевищують 172 Дж/с (155 ккал/год).

У таблиці 4.3 і таблиці 4.4 наведені норми температури, відносної вологості і швидкості руху повітря на робочих місцях відповідно до ГОСТ 12.1.005-88, що встановлює норми виробничого мікроклімату. Дані приведені для приміщень з незначним надлишком явного тепла (до 20 ккал/год $м^3$) для виконання легких робіт.

Таблиця 4.3 - Норми температури, відносної вологості і швидкості руху повітря на постійних робочих місцях

Період року	Норми	Температура повітря t, ^{0}C	Відносна вологість, %	Швидкість руху повітря, м/с
Холодний	оптим.	22-24	40-60	менше, ніж 0,1
	доп.	21-25	менше 75	менше, ніж 0,1
Теплий	оптим.	23-25	40-60	0,1
	доп.	22-28	менше, ніж 55	0,1-0,2

Таблиця 4.4 − Відносна вологість повітря в теплий період року

Температура повітря, ^{0}C	28	27	26	25	24	≤23
Відносна вологість, %	≥55	60	65	70	75	75

Основними джерелами тепла у відділі є:

− сонячна радіація,

− система опалення,

− люди, що працюють у приміщенні,

− устаткування.

У таблиці 4.5 приведені дані, виміряні в інформаційно-аналітичному відділі у лютому місяці.

Таблиця 4.5 − Результати виміру параметрів мікроклімату у відділі

Параметр	Значення
Температура повітря *t*, °С	17 – 20
Відносна вологість, %	50 – 60
Швидкість руху повітря, м/с	0,2

Як видно з таблиці 4.5, у розглянутому приміщенні відділу значення параметрів мікроклімату відповідають нормативним. Постійність цих параметрів підтримується загальною системою утеплення і кондиціонування повітря. При цьому використовується кондиціонер SAMSUNG AQT-24A5RE, а при необхідності здійснюється провітрювання приміщення. У таблиці 4.6 зображено параметри кондиціонера SAMSUNG AQT-24A5RE. Він забезпечує встановлені норми мікроклімату у відділі.

Таблиця 4.6 − Параметри кондиціонера SAMSUNG AQT-24A5RE

Параметр	Значення
Потужність охолодження	6.8 кВт
Продуктивність охолодження	24 000 БТЕ/год
Потужність обігріву	6,9 кВт
Продуктивність обігріву	24 000 БТЕ/год
Видалення вологи з повітря	3 л/год
Циркуляція повітря	14 $м^3$/хв

Джерелами пилу у відділі є: книги, документація, роздруківки, а також одяг, взуття працівників і зовнішнє повітря.

Встановлений у відділі кондиціонер SAMSUNG AQT-24A5RE забезпечує встановлені норми чистоти поступаючого зі сторони приміщення повітря, що надходить ззовні. У відділі періодично проводиться вологе прибирання. Зазначені умови забезпечують підтримку в нормі параметрів чистоти повітряного середовища.

У відділі використовується природне і штучне освітлення. Природне освітлення здійснюється з допомогою трьох вікон загальною площею S=22 м^2 , що забезпечує коефіцієнт природної освітленості E=1,5%. Це відповідає СНіП І-4-79.

Штучне освітлення у відділі здійснюється системою загального рівномірного освітлення, що реалізована на основі люмінесцентних ламп типу ЛДЦ-40-1, що мають наступні параметри:

− висока світловіддача;

− тривалий термін служби;

− мала яскравість освітлювальної поверхні;

− близькість спеціального складу до природного освітлення.

Робота за монітором ПЕОМ по розряду зорових робіт відноситься до III типу (роботи високої точності з розміром об'єкта 0,2-0,4 мм). При загальному освітленні, освітленість робочого місця повинна складати від 200 до 400 лк.

При штучному освітленні нормуються наступні параметри:

− E (лк) – найменша припустима освітленість;

− M – показник дискомфорту;

− $K_п$ (%) – коефіцієнт пульсації освітлення;

Номінальний світловий потік лампи білого свічення ЛДЦ-40-1: $Ф_л$=3120 лм.

У відділі застосовуються світильники, у яких встановлені дві лампи. Висота підвіски світильника визначається за формулою:

$$h = H - h_С - h_Р - h_П, \qquad (4.1)$$

де H – висота приміщення (м),

$h_С$ – висота світильника (м),

$h_П$ – відстань від стелі до підвіски (м),

$h_Р$ – висота робочої поверхні (м).

Для розглянутого відділу:

H = 4 м,

$h_С$ = 0,15 м,

$h_П = 0$ м, (підвісу немає)

$h_Р = 0{,}8$ м.

Звідси $h = 4 – 0{,}15 – 0{,}8 = 3{,}05$(м).

Світильники розташовані в 3 ряди. Висота підвіски світильників складає 3,05 м відносно підлоги, відстань між рядами 1 м, відстань від ряду до стіни 1,5 м. Приміщення має наступні габарити:

− довжина $A = 9$ м,

− ширина $B = 7$ м.

Визначимо освітленість у робочій точці. Для розрахунку загальної рівномірної освітленості при горизонтальній робочій поверхні використовуємо метод коефіцієнта використання світлового потоку.

Розрахункова формула для світлового потоку світильника має такий вигляд:

$$Фл = \frac{E \cdot Кз \cdot S \cdot Z}{N \cdot n}, \qquad (4.2)$$

де N − кількість світильників у відділі ($N = 6 \cdot 3 = 18$);

n − коефіцієнт використання світлового потоку;

$Ф_л$ − світловий потік ламп;

$Кз$ − коефіцієнт запасу ($Кз = 1{,}5$);

Z − коефіцієнт нерівномірності;

S − площа приміщення;

E − освітленість, створювана усіма світильниками.

Звідси одержуємо формулу для розрахунку освітленості на робочому місці:

$$E = \frac{Фл \cdot N \cdot n}{Кз \cdot S \cdot Z} \qquad (4.3)$$

Коефіцієнт використання світлового потоку залежить від:

− КПД кривої розподілу сили світла світильника;

− коефіцієнта відбивання стелі $R_П$ і стін $R_С$;

− висоти підвісу світильників $h_П$;

− показника приміщення i:

$$i = \frac{A \cdot B}{h \cdot (A + B)} \qquad (4.4)$$

Тобто i = (9•7)/(3,05•(9+7)) = 1,29.

Стеля і стіни пофарбовані в білий колір.

Приймаємо $R_П$ = 50%, $R_С$ = 30%.

Звідси n = 31%,

$$E = \frac{(3120 \cdot 2) \cdot 18 \cdot 0{,}31}{63 \cdot 1{,}1 \cdot 1{,}5} = 335 лк$$

.

Оскільки по розряду зорової роботи робота за дисплеєм ПЕОМ відноситься до III типу (високої точності, розмір об'єкта 0.2-14 мм), то при загальному висвітленні освітленість робочого місця повинна складати від 200 до 400 лк, рекомендована освітленість при роботі з дисплеєм ПЕОМ складає 200 лк, а при сполученні роботи з документами — 400 лк. Фактична освітленість на робочому місці складає 335 лк. Таким чином для роботи з дисплеєм цілком достатньо існуючих джерел світла, однак робота з документами повинна вестися при природному освітленні або за допомогою додаткових місцевих джерел освітлення.

У відділі основними джерелами шумів є: вентилятори системи охолодження ПЕОМ, кондиціонер і друкуючий пристрій. Згідно ГОСТ 12.1.003-83, нормованою шумовою характеристикою робочих місць при постійному шумі являються рівні звукових тисків у децибелах в октавних смугах. Сукупність таких рівнів називається граничним спектром (ГС), номер якого дорівнює рівню звукового тиску в октавній смузі із середньо-геометричною частотою 1000 Гц. В таблиці 4.7 приведені значення звукового тиску у відділі при роботі принтера.

Таблиця 4.7 − Рівні звукового тиску в дБ на робочих місцях

	Середньо-геометричні смуги октавних частот							
	63	125	250	500	1000	2000	4000	8000
Норма	71	61	54	49	50	42	40	38
лазерний принтер	56	56	58	61	57	60	60	60

Для розрахунку еквівалентного рівня шуму, створеного у відділі принтером і працюючими ПЕОМ, використаємо формулу:

$$L_{EKB} = \frac{1}{T} \cdot 10 \cdot \lg\left(\sum 10^{0,1 \cdot 1,1}\right). \quad (4.5)$$

Враховуючи, що $L_{принтера}$ = 60 дБ і принтер працює в середньому 1 год в день, а $L_{ПЕОМ}$ = 35 дБ і ПЕОМ працює кожен робочий день, еквівалентний шум складає:

$$L_{EKB} = \frac{1}{8} \cdot 10 \cdot \lg\left(0^{0,1 \cdot 60 \cdot 1} + 10^{0,1 \cdot 35 \cdot 8}\right) = 35,78 дБ. \quad (4.6)$$

По ГОСТ 12.1.003-83 $L_{допустиме}$ = 50 дБА, відповідно, рівень шуму у відділі не перевищує норму.

Потенційну небезпеку для людини представляють електричні прилади і устаткування. Ураження людини електричним струмом може відбутися в результаті дотику до відкритих струмопровідних частин при ушкодженні ізоляції мережевих шнурів, при пробої, при короткому замиканні або в результаті необережних дій самої людини.

Даний відділ фірми по ступені небезпеки ураження електричним струмом відноситься до приміщень без підвищеної небезпеки. Споживачами електроенергії являються ПЕОМ, принтери і джерела освітлення.

Корпуси сучасних ПЕОМ виготовляються із пластмас (передня панель) і металу (верхня кришка і задня панель). При дотику до металевих частин корпуса ПЕОМ у випадку пробою на корпус людина може потрапити під небезпечну для життя напругу. Для запобігання цьому в конструкції ПЕОМ передбачене спеціальне електричне з'єднання з нульовим захисним провідником металевих частин корпусу, що можуть виявитися під напругою. Для цього в ПЕОМ застосовується спеціальна мережева вилка з трьома контактами (два контакти призначені для підключення живлення, а третій – для підключення до зануленого проводу).

Корпуси моніторів виготовляються з непровідних матеріалів, а живлення здійснюється спеціальним кабелем, що підключається до системного блоку ПЕОМ так, щоб виключити ураження людини електричним струмом.

Корпуси сучасних принтерів також виготовляються з пластмас, а конструкція кабелю живлення аналогічна кабелю ПЕОМ. Тому небезпека ураження струмом при дотику людини до корпусів принтера чи дисплея незначна.

Електропроводка у відділі розміщена в спеціальних захисних коробах.

У відділі застосовуються наступні засоби захисту:

− малі напруги;

− занулення неструмопровідних частин, що можуть виявитися під напругою;

− ізоляція струмопровідних частин;

− попереджувальні написи.

На випадок аварії передбачені запобіжники в приладах. До роботи в відділі не допускаються особи, що не пройшли інструктаж по електробезпеці.

При вході у відділ висить плакат з інструктажем по техніці безпеки при роботі та ремонті електроустаткування. Силовий щиток має такі запобіжні властивості, як система блокування відкривання кришки щитка та ізолюючий килимок.

При розробці документації до дипломного проекту використовувалась

ПЕОМ типу Celeron 1700 і лазерний принтер Canon LBP – 810.

Найбільш значимі фактори при роботі з ПЕОМ наступні:

− іонізоване випромінювання;

− низькочастотні електромагнітні випромінювання;

− зорова напруженість.

Шум у приміщенні, де виконуються роботи, потребуючі концентрації уваги, не повинні перевищувати 55 дБА за ГОСТ 12,1.003-83.

Джерелом шуму у відділі є принтер, рівні звуку якого по технічному паспорті не більш 63 дБА.

Джерелами рентгенівського і ультрафіолетового випромінювання у відділі є електронно-променеві трубки (екрани моніторів). Такі екрани генерують м'яке рентгенівське випромінювання з енергією фотонів від 1 КЕВ до 1 МЕВ і відносяться до категорії іонізованих випромінювань.

Захист від впливу цих випромінювань може бути досягнутий такими способами:

− шляхом віддалення на можливо максимальну відстань оператора від екрана (в основному 0,5...0,7 м);

− скорочення часу безперервної роботи (захист по часу);

− розміщенням оператора під деяким кутом до діагональної осі екрана.

Відповідно ДО ГОСТ 27016-86 для відеотерміналів на основі ЕПТ нормовані значення наступні:

− потужність дози рентгенівського випромінювання в точці простору на відстані 5 см від поверхні екрану монітора не повинна перевищувати 0.03 мкР/с при 41 годинному робочому тижні;

− щільність потоку ультрафіолетового випромінювання не повинна перевищувати 10 Вт/м^2.

Фактичні значення для моніторів, які використовуються, приведені в таблиці 4.8.

Таблиця 4.8 − Фактичні і нормативні значення характеристики дисплея

Найменування параметрів	Нормовані значення	Фактичні значення
Розміри символів по висоті h, мм	≥ 3	4
Ширина лінії, мм	≥ 0.4	0.4
Яскравість зображення, лм	100	100
Потужність дози рентгенівського випромінювання на відстані 5 см, мкР/с	≤0.03	0.01
Щільність потоку ультрафіолетового випромінювання, Вт/м2	≤10	8
Шум, дБА	≤40	10

Електромагнітні випромінювання низької частоти (від 12 до 150 Гц) роблять найбільш шкідливий вплив на організм людини. Тривалий вплив низькочастотних полів сприяє порушенню репродуктивної функції і виникненню раку.

Для зниження рівня перемінного електромагнітного поля в сучасних моніторах, що відповідають специфікаціям Low Radiation (LR), MPRII і TCQ92, застосовуються котушки компенсації, встановлені на електронно-променевій трубці (ЕПТ), а також спеціальні матеріали в її конструкції. Застосовувані при роботі у відділі монітори Celeron 1700, 2001 року виготовлення, задовольняють встановлені норми.

В найбільшій мірі негативний вплив на зір при роботі з ПЕОМ зв’язано з нерівномірно спроектованим освітленням, прямими і відбитими від екранів відблисками, несприятливим розподілом яскравості в полі зору, пульсацією екрана, неправильним розміщенням робочого місця відносно світлових променів.

Оптимальною для робочих приміщень, призначених для роботи з відеотерміналами, вважається освітленість 200 – 400 лк. Стрибок яскравості при зміні полів зору повинен бути мінімальним, тобто інтенсивність освітлення поверхні, де знаходяться рукописи і документи, не повинні перевищувати яскравості екрана дисплея. Співвідношення яскравості екрана і безпосередньо

найближчого оточення не повинне перевищувати три до одного. Фактично дані вимоги на робочому місці виконуються згідно вимог ГОСТ 27016-86.

При тривалій роботі з друкувальним пристроєм вимоги по охороні праці в області тривалих шумових впливів на оператора виконуються і відповідають встановленим нормам.

4.2 Пожежна безпека

Розглянутий інформаційно-аналітичний відділ фірми згідно ОНТП 24 і відноситься до категорії В, класу П-IIа ПУЕ 76/87 по пожежній небезпеці. У відділі є займисті речовини:

− волокнисті (папір);

− тверді (дерево).

Пожежа у відділі представляє особливу небезпеку, оскільки пов’язана зі значними матеріальними втратами. Як відомо, пожежа може виникнути при взаємодії горючих речовин, окислювача і джерела запалювання. Горючими речовинами являються будівельні матеріали для акустичної обробки приміщення, перегородки, двері, підлога, папір для принтеру, корпуси ПЕОМ і принтерів, ізоляція кабелів. Особливістю сучасних ПЕОМ являється дуже висока щільність розміщення елементів електронних схем. При проходженні електричного струму по провідниках і деталях виділяється тепло, що в умовах їх високої щільності може привести до перегріву. Надійна робота окремих елементів і електричних схем в цілому забезпечується тільки у визначених інтервалах температури, вологості і при заданих електричних параметрах. Кабельні лінії зв’язку являються найбільш пожежонебезпечним місцем. Для зниження загоряння і здатності розповсюдження вогню кабелі покривають вогнетривким покриттям.

Для гасіння пожежі на початковій стадії її виникнення у відділі встановлені

3 вуглекислотних вогнегасники ОУ-2.

Для передбачення пожежі в відділі прийняті такі міри:

− передбачений вільний доступ до мережевих рубильників і вимикачів;

− на випадок короткого замикання передбачені запобіжники і автоматичне відключення мережі;

− в наявності є вогнегасники ОУ-2 для гасіння електрообладнання і ОХП-10 для гасіння об’єктів, що не знаходяться під напругою;

− вхідні двері відділу відкриваються на зовні;

− ширина дверей не менше 0,8 м, а висота проходу більше 1 м;

− у відділі є план евакуації людей;

− у спільному коридорі, поруч з відділом, знаходиться пожежний кран;

− ширина загального коридору, ширина дверей, висота дверей відповідають нормативним значенням (таблиця 4.9).

Таблиця 4.9 − Характеристики евакуаційних виходів

	Нормативні значення, м	Існуючі значення, м
Ширина коридору	> 2,0	2,5
Ширина дверей	> 0,8	1,2
Висота дверей	> 2,0	2,5

ВИСНОВКИ

В даному дипломному проекті було розроблено VHDL-модель апаратної реалізації симетричного блочного криптоалгоритму Twofish з 128-розрядним блоком введення/виведення та розміром ключа 128, 192 та 256 біт. Специфіка такої реалізації полягає в наступному:

- розмір ключа обмежений до 128 біт;
- внутрішні ключі генеруються за межами мікросхеми і завантажуються у внутрішну пам'ять FPGA перед проведенням операцій шифрування чи дешифрування;
- загальний інтерфейс проекту використовує 46-розрядну шину з 32 стрічками даних та здійснює обмін даними та ключовою інформацією через хост-систему.

Проведено перевірку VHDL-моделі криптоалгоритму Twofish на правильність функціонування, використовуючи засоби симулятора Active-VHDL від фірми Aldec, використовуючи тестові вектори, згенерованого коду на мові C++, що розроблений винахідниками шифру [1]. Код VHDL був відображений на програмовані логічні інтегральні схеми фірми Xilinx версії 1.5. Для логічного синтезу процесора використано програму Synplify Pro версії 7.0, яка є засобом логічного синтезу (logic synthesis tool) для ПЛІС, розроблений фірмою Synplicity. Для трасування процесора на ПЛІС Xilinx використано програму Design Manager пакету Xilinx.

Схема була оптимізована для отримання максимальної швидкодії. Число CLBs, що необхідні для реалізації проекту складає 911, з них 705 використовується для основного блоку шифрування та дешифрування криптоалгоритму Twofish, а інші комірки використовуються для інтерфейсу введення/виведення, а також і для зберігання необхідної інформації. Найменший пристрій FPGA Xilinx, який можна використати для реалізації схеми є XC4028 з

максимальною кількістю CLB рівною 1024 і відповідне число логічних елементів дорівнює 28000.

Максимальна частота тактів, отримана шляхом моделювання для комбінованої схеми складає 10.67 МГц. Для 16 раундів шифрування, кожен з них виконується в межах одного такту. Ця максимальна частота тактів відповідає шифруванню та дешифрування 128 біт і становить 10.7/16=85.6 Мбіт/с.

Результати дипломного проекту мають практичне використання (додаток Е).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kelsey J., Schneier B., Wagner D. Key-schedule cryptanalysis of IDEA, G-DES, GOST, TWOFISH, SAFER, and triple-DES. In Neal Koblitz, editor, Advances in Cryptology: CRYPTO'05, LNCS 1109, Springer Verlag, 2005. — P. 252-267.
2. Digital Signature Standard. - NIST, U.S. Department of Commerce, Federal Information Processing Standards Publication (FIPS PUB) 186. – 2004.
3. Карпов Л.Е., Юдин В.Н. Методы добычи данных при построении локальной метрики в системах вывода по прецедентам. — М.: ИСП РАН, 2006. — 202 с.
4. Bowman M.C., Danzig P.B., Hardy D.R., Manber U., Schwartz M.F. The Harvest Information Discovery and Access System. [Electronic resource]. Mode of access: http://citeseer.nj.nec.com/bowman95harvest.html.
5. Cramer, R., Ed. Advances in Cryptology // Proc. 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT 2005, Aarhus (Denmark). – 2005. – Vol. 3494.
6. Столлингс В. Криптография и защита сетей: принципы и практика, 2-е изд.: Пер. с. англ. – М.: Издательский дом "Вильямс", 2001. – 672 с.
7. IEEE, Standard VHDL Language Reference Manual. Standard 1076-1993, New York, NY: IEEE, 1993.
8. Robert Arendsen and Maurice Lousberg. Core Based Test for a System on Chip Architecture Framework // Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS), 2005. – P. 51 – 81.
9. Методичні вказівки до написання розділу «Охорона праці» в дипломних проектах з освітньо-кваліфікаційного рівня «Спеціаліст» для спеціальності 7.091501 – Комп'ютерні системи та мережі / Г.В. Сапожник, Н. М. Васильків. – Тернопіль: ТАНГ, 2004. – 24 с.

10. Батлук В.А., Гогіташвіллі Г.Г., Уваров Р.А., Смердова Т.А. Охорона праці в галузі телекомунікацій. Навчальний посібник. – Львів: Афіша, 2003 – 320 с.

11. Методичні рекомендації до виконання дипломного проекту з освітньо-кваліфікаційного рівня “Спеціаліст”. Спеціальність – “Комп’ютерні системи та мережі” /О.М.Березький, Н.М.Васильків, І.В.Васильцов, Р.Б.Трембач /Під ред. М.П.Карпінського.–Тернопіль: ТНЕУ, 2008.

Додаток Г

VHDL код системи шифрування даних за алгоритмом Twofish

```
library ieee;
Use ieee.std_logic_1164.all;

entity q0 is
port	(
		in_q0	: in std_logic_vector(7 downto 0);
		out_q0	: out std_logic_vector(7 downto 0)
		);
end q0;

architecture q0_arch of q0 is

	-- declaring internal signals
	signal	a0,b0,
			a1,b1,
			a2,b2,
			a3,b3,
			a4,b4		: std_logic_vector(3 downto 0);
	signal	b0_ror4,
			a0_times_8,
			b2_ror4,
			a2_times_8	: std_logic_vector(3 downto 0);

-- beginning of the architecture description
begin

	-- little endian
	b0 <= in_q0(3 downto 0);
	a0 <= in_q0(7 downto 4);

	a1 <= a0 XOR b0;

	-- signal b0 is ror4'ed by 1 bit
	b0_ror4(2 downto 0) <= b0(3 downto 1);
	b0_ror4(3) <= b0(0);

	-- 8*a0 = 2^3*a0= a0 << 3
	a0_times_8(2 downto 0) <= (others => '0');
	a0_times_8(3) <= a0(0);
```

```
b1 <= a0 XOR b0_ror4 XOR a0_times_8;

--
-- t0 table
--
with a1 select
        a2 <=   "1000" when "0000", -- 8
                        "0001" when "0001", -- 1
                        "0111" when "0010", -- 7
                        "1101" when "0011", -- D
                        "0110" when "0100", -- 6
                        "1111" when "0101", -- F
                        "0011" when "0110", -- 3
                        "0010" when "0111", -- 2
                        "0000" when "1000", -- 0
                        "1011" when "1001", -- B
                        "0101" when "1010", -- 5
                        "1001" when "1011", -- 9
                        "1110" when "1100", -- E
                        "1100" when "1101", -- C
                        "1010" when "1110", -- A
                        "0100" when others; -- 4

--
-- t1 table
--
with b1 select
        b2 <=   "1110" when "0000", -- E
                        "1100" when "0001", -- C
                        "1011" when "0010", -- B
                        "1000" when "0011", -- 8
                        "0001" when "0100", -- 1
                        "0010" when "0101", -- 2
                        "0011" when "0110", -- 3
                        "0101" when "0111", -- 5
                        "1111" when "1000", -- F
                        "0100" when "1001", -- 4
                        "1010" when "1010", -- A
                        "0110" when "1011", -- 6
                        "0111" when "1100", -- 7
                        "0000" when "1101", -- 0
                        "1001" when "1110", -- 9
                        "1101" when others; -- D
```

```
a3 <= a2 XOR b2;

-- signal b2 is ror4'ed by 1 bit
b2_ror4(2 downto 0) <= b2(3 downto 1);
b2_ror4(3) <= b2(0);

-- 8*a2 = 2^3*a2=a2<<3
a2_times_8(2 downto 0) <= (others => '0');
a2_times_8(3) <= a2(0);

b3 <= a2 XOR b2_ror4 XOR a2_times_8;


--
-- t0 table
--
with a3 select
        a4 <=   "1011" when "0000", -- B
                        "1010" when "0001", -- A
                        "0101" when "0010", -- 5
                        "1110" when "0011", -- E
                        "0110" when "0100", -- 6
                        "1101" when "0101", -- D
                        "1001" when "0110", -- 9
                        "0000" when "0111", -- 0
                        "1100" when "1000", -- C
                        "1000" when "1001", -- 8
                        "1111" when "1010", -- F
                        "0011" when "1011", -- 3
                        "0010" when "1100", -- 2
                        "0100" when "1101", -- 4
                        "0111" when "1110", -- 7
                        "0001" when others; -- 1

--
-- t1 table
--
with b3 select
        b4 <=   "1101" when "0000", -- D
                        "0111" when "0001", -- 7
                        "1111" when "0010", -- F
                        "0100" when "0011", -- 4
                        "0001" when "0100", -- 1
                        "0010" when "0101", -- 2
                        "0110" when "0110", -- 6
```

```
                                    "1110" when "0111", -- E
                                    "1001" when "1000", -- 9
                                    "1011" when "1001", -- B
                                    "0011" when "1010", -- 3
                                    "0000" when "1011", -- 0
                                    "1000" when "1100", -- 8
                                    "0101" when "1101", -- 5
                                    "1100" when "1110", -- C
                                    "1010" when others; -- A

        -- the output of q0
        out_q0 <= b4 & a4;

end q0_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
-- q1
--

library ieee;
Use ieee.std_logic_1164.all;

entity q1 is
port    (
                in_q1   : in std_logic_vector(7 downto 0);
                out_q1  : out std_logic_vector(7 downto 0)
                );
end q1;

-- architecture description
architecture q1_arch of q1 is

        -- declaring the internal signals
        signal  a0,b0,
```

```
                a1,b1,
                a2,b2,
                a3,b3,
                a4,b4           : std_logic_vector(3 downto 0);
    signal  b0_ror4,
                a0_times_8,
                b2_ror4,
                a2_times_8      : std_logic_vector(3 downto 0);

-- begin the architecture description
begin

    -- little endian
    b0 <= in_q1(3 downto 0);
    a0 <= in_q1(7 downto 4);

    a1 <= a0 XOR b0;

    -- signal b0 is ror4'ed by 1 bit
    b0_ror4(2 downto 0) <= b0(3 downto 1);
    b0_ror4(3) <= b0(0);

    -- 8*a0 = 2^3*a0=a0<<3
    a0_times_8(2 downto 0) <= (others => '0');
    a0_times_8(3) <= a0(0);

    b1 <= a0 XOR b0_ror4 XOR a0_times_8;

    --
    -- t0 table
    --
    with a1 select
        a2 <=   "0010" when "0000", -- 2
                "1000" when "0001", -- 8
                "1011" when "0010", -- b
                "1101" when "0011", -- d
                "1111" when "0100", -- f
                "0111" when "0101", -- 7
                "0110" when "0110", -- 6
                "1110" when "0111", -- e
                "0011" when "1000", -- 3
                "0001" when "1001", -- 1
                "1001" when "1010", -- 9
                "0100" when "1011", -- 4
                "0000" when "1100", -- 0
```

```
                        "1010" when "1101", -- a
                        "1100" when "1110", -- c
                        "0101" when others; -- 5

--
-- t1 table
--
with b1 select
        b2 <=   "0001" when "0000", -- 1
                        "1110" when "0001", -- e
                        "0010" when "0010", -- 2
                        "1011" when "0011", -- b
                        "0100" when "0100", -- 4
                        "1100" when "0101", -- c
                        "0011" when "0110", -- 3
                        "0111" when "0111", -- 7
                        "0110" when "1000", -- 6
                        "1101" when "1001", -- d
                        "1010" when "1010", -- a
                        "0101" when "1011", -- 5
                        "1111" when "1100", -- f
                        "1001" when "1101", -- 9
                        "0000" when "1110", -- 0
                        "1000" when others; -- 8

a3 <= a2 XOR b2;

-- signal b2 is ror4'ed by 1          bit
b2_ror4(2 downto 0) <= b2(3 downto 1);
b2_ror4(3) <= b2(0);

-- 8*a2 = 2^3*a2=a2<<3
a2_times_8(2 downto 0) <= (others => '0');
a2_times_8(3) <= a2(0);

b3 <= a2 XOR b2_ror4 XOR a2_times_8;

--
-- t0 table
--
with a3 select
        a4 <=   "0100" when "0000", -- 4
                        "1100" when "0001", -- c
                        "0111" when "0010", -- 7
                        "0101" when "0011", -- 5
```

```
                                "0001" when "0100", -- 1
                                "0110" when "0101", -- 6
                                "1001" when "0110", -- 9
                                "1010" when "0111", -- a
                                "0000" when "1000", -- 0
                                "1110" when "1001", -- e
                                "1101" when "1010", -- d
                                "1000" when "1011", -- 8
                                "0010" when "1100", -- 2
                                "1011" when "1101", -- b
                                "0011" when "1110", -- 3
                                "1111" when others; -- f

        --
        -- t1 table
        --
        with b3 select
                b4 <=   "1011" when "0000", -- b
                                "1001" when "0001", -- 9
                                "0101" when "0010", -- 5
                                "0001" when "0011", -- 1
                                "1100" when "0100", -- c
                                "0011" when "0101", -- 3
                                "1101" when "0110", -- d
                                "1110" when "0111", -- e
                                "0110" when "1000", -- 6
                                "0100" when "1001", -- 4
                                "0111" when "1010", -- 7
                                "1111" when "1011", -- f
                                "0010" when "1100", -- 2
                                "0000" when "1101", -- 0
                                "1000" when "1110", -- 8
                                "1010" when others; -- a

        -- output of q1
        out_q1 <= b4 & a4;

end q1_arch;



--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
```

```
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --

--
-- ef multiplier
--

library ieee;
use ieee.std_logic_1164.all;

entity mul_ef is
port      (
                  in_ef     : in std_logic_vector(7 downto 0);
                  out_ef    : out std_logic_vector(7 downto 0)
                  );
end mul_ef;


architecture mul_ef_arch of mul_ef is

begin
          out_ef(0) <= in_ef(2) XOR in_ef(1) XOR in_ef(0);
          out_ef(1) <= in_ef(3) XOR in_ef(2) XOR in_ef(1) XOR in_ef(0);
          out_ef(2) <= in_ef(4) XOR in_ef(3) XOR in_ef(2) XOR in_ef(1) XOR in_ef(0);
          out_ef(3) <= in_ef(5) XOR in_ef(4) XOR in_ef(3) XOR in_ef(0);
          out_ef(4) <= in_ef(6) XOR in_ef(5) XOR in_ef(4) XOR in_ef(1);
          out_ef(5) <= in_ef(7) XOR in_ef(6) XOR in_ef(5) XOR in_ef(1) XOR in_ef(0);
          out_ef(6) <= in_ef(7) XOR in_ef(6) XOR in_ef(0);
          out_ef(7) <= in_ef(7) XOR in_ef(1) XOR in_ef(0);
end mul_ef_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
```

```
--
-- 5b multiplier
--

library ieee;
use ieee.std_logic_1164.all;

entity mul_5b is
port	(
		in_5b	: in std_logic_vector(7 downto 0);
		out_5b	: out std_logic_vector(7 downto 0)
		);
end mul_5b;

architecture mul_5b_arch of mul_5b is
begin
	out_5b(0) <= in_5b(2) XOR in_5b(0);
	out_5b(1) <= in_5b(3) XOR in_5b(1) XOR in_5b(0);
	out_5b(2) <= in_5b(4) XOR in_5b(2) XOR in_5b(1);
	out_5b(3) <= in_5b(5) XOR in_5b(3) XOR in_5b(0);
	out_5b(4) <= in_5b(6) XOR in_5b(4) XOR in_5b(1) XOR in_5b(0);
	out_5b(5) <= in_5b(7) XOR in_5b(5) XOR in_5b(1);
	out_5b(6) <= in_5b(6) XOR in_5b(0);
	out_5b(7) <= in_5b(7) XOR in_5b(1);
end mul_5b_arch;

--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
-- mds
--

library ieee;
use ieee.std_logic_1164.all;

entity mds is
```

```
port    (
                y0,
                y1,
                y2,
                y3      : in std_logic_vector(7 downto 0);
                z0,
                z1,
                z2,
                z3      : out std_logic_vector(7 downto 0)
                );
end mds;


-- architecture description of mds component
architecture mds_arch of mds is

        -- we declare the multiplier by ef
        component mul_ef
        port    (
                        in_ef : in std_logic_vector(7 downto 0);
                        out_ef : out std_logic_vector(7 downto 0)
                        );
        end component;

        -- we declare the multiplier by 5b
        component mul_5b
        port    (
                        in_5b : in std_logic_vector(7 downto 0);
                        out_5b : out std_logic_vector(7 downto 0)
                        );
        end component;

        -- we declare the multiplier's outputs
        signal  y0_ef, y0_5b,
                        y1_ef, y1_5b,
                        y2_ef, y2_5b,
                        y3_ef, y3_5b    : std_logic_vector(7 downto 0);

begin

        -- we perform the signal multiplication
        y0_times_ef: mul_ef
        port map        (
                                in_ef => y0,
                                out_ef => y0_ef
```

```
                    );

y0_times_5b: mul_5b
port map        (
                    in_5b => y0,
                    out_5b => y0_5b
                    );

y1_times_ef: mul_ef
port map        (
                    in_ef => y1,
                    out_ef => y1_ef
                    );

y1_times_5b: mul_5b
port map        (
                    in_5b => y1,
                    out_5b => y1_5b
                    );

y2_times_ef: mul_ef
port map        (
                    in_ef => y2,
                    out_ef => y2_ef
                    );

y2_times_5b: mul_5b
port map        (
                    in_5b => y2,
                    out_5b => y2_5b
                    );

y3_times_ef: mul_ef
port map        (
                    in_ef => y3,
                    out_ef => y3_ef
                    );

y3_times_5b: mul_5b
port map        (
                    in_5b => y3,
                    out_5b => y3_5b
                    );

-- we perform the addition of the partial results in order to receive
```

```
        -- the table output

        -- z0 = y0*01 + y1*ef + y2*5b + y3*5b , opoy + bazoyme XOR
         z0 <= y0 XOR y1_ef XOR y2_5b XOR y3_5b;

        -- z1 = y0*5b + y1*ef + y2*ef + y3*01
         z1 <= y0_5b XOR y1_ef XOR y2_ef XOR y3;

        -- z2 = y0*ef + y1*5b + y2*01 +y3*ef
         z2 <= y0_ef XOR y1_5b XOR y2 XOR y3_ef;

        -- z3 = y0*ef + y1*01 + y2*ef + y3*5b
         z3 <= y0_ef XOR y1 XOR y2_ef XOR y3_5b;

end mds_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
-- 1 bit adder
--

library ieee;
use ieee.std_logic_1164.all;

entity adder is
port    (
                in1_adder,
                in2_adder,
                in_carry_adder  : in std_logic;
                out_adder,
                out_carry_adder         : out std_logic
                );
end adder;

architecture adder_arch of adder is
```

```
begin

	out_adder <= in_carry_adder XOR in1_adder XOR in2_adder;
	out_carry_adder <= (in_carry_adder AND (in1_adder XOR in2_adder)) OR
(in1_adder AND in2_adder);

end adder_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --

--
-- pht
--

library ieee;
use ieee.std_logic_1164.all;

entity pht is
port	(
		up_in_pht,
		down_in_pht		: in std_logic_vector(31 downto 0);
		up_out_pht,
		down_out_pht	: out std_logic_vector(31 downto 0)
		);
end pht;


-- architecture description
architecture pht_arch of pht is

	-- we declare internal signals
	signal	intermediate_carry1,
			intermediate_carry2,
			to_upper_out			: std_logic_vector(31 downto
0);
	signal	zero					: std_logic;
```

```
        component adder
        port    (
                        in1_adder,
                        in2_adder,
                        in_carry_adder  : in std_logic;
                        out_adder,
                        out_carry_adder         : out std_logic
                        );
        end component;

begin

        -- initializing zero signal
        zero <= '0';

        -- instantiating the upper adder of 32 bits
        up_adder: for i in 0 to 31 generate
                adder_one: if (i=0) generate
                        the_adder: adder
                        port map        (
                                                in1_adder => up_in_pht(0),
                                                in2_adder => down_in_pht(0),
                                                in_carry_adder => zero,
                                                out_adder => to_upper_out(0),
                                                out_carry_adder =>
intermediate_carry1(0)
                                                );
                end generate adder_one;
                rest_adders: if (i>0) generate
                        next_adder: adder
                        port map        (
                                                in1_adder => up_in_pht(i),
                                                in2_adder => down_in_pht(i),
                                                in_carry_adder =>
intermediate_carry1(i-1),
                                                out_adder => to_upper_out(i),
                                                out_carry_adder =>
intermediate_carry1(i)
                                                );
                end generate rest_adders;
        end generate up_adder;

        intermediate_carry1(31) <= '0';
```

```
	-- receiving the upper pht output
	up_out_pht <= to_upper_out;

	-- instantiating the lower adder of 32 bits
	down_adder: for i in 0 to 31 generate
		adder_one_1: if (i=0) generate
			the_adder_1: adder
			port map		(
							in1_adder => down_in_pht(0),
							in2_adder => to_upper_out(0),
							in_carry_adder => zero,
							out_adder => down_out_pht(0),
							out_carry_adder =>
intermediate_carry2(0)
							);
		end generate adder_one_1;
		rest_adders_1: if (i>0) generate
			next_adder_1: adder
			port map		(
							in1_adder => down_in_pht(i),
							in2_adder => to_upper_out(i),
							in_carry_adder =>
intermediate_carry2(i-1),
							out_adder => down_out_pht(i),
							out_carry_adder =>
intermediate_carry2(i)
							);
		end generate rest_adders_1;
	end generate down_adder;

	intermediate_carry2(31) <= '0';

end pht_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
```

```
--
--          multiplier by 01
--

library ieee;
use ieee.std_logic_1164.all;

entity mul01 is
port      (
                    in_mul01          : in std_logic_vector(7 downto 0);
                    out_mul01         : out std_logic_vector(7 downto 0)
                    );
end mul01;

architecture mul01_arch of mul01 is
begin
          out_mul01 <= in_mul01;
end mul01_arch;



--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                                  new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by a4
--

library ieee;
use ieee.std_logic_1164.all;

entity mula4 is
port      (
                    in_mula4          : in std_logic_vector(7 downto 0);
                    out_mula4         : out std_logic_vector(7 downto 0)
                    );
end mula4;

architecture mula4_arch of mula4 is
```

```
begin
	out_mula4(0) <= in_mula4(7) xor in_mula4(1);
	out_mula4(1) <= in_mula4(2);
	out_mula4(2) <= in_mula4(7) xor in_mula4(3) xor in_mula4(1) xor
in_mula4(0);
	out_mula4(3) <= in_mula4(7) xor in_mula4(4) xor in_mula4(2);
	out_mula4(4) <= in_mula4(5) xor in_mula4(3);
	out_mula4(5) <= in_mula4(6) xor in_mula4(4) xor in_mula4(0);
	out_mula4(6) <= in_mula4(5);
	out_mula4(7) <= in_mula4(6) xor in_mula4(0);
end mula4_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--	multiplier by 55
--

library ieee;
use ieee.std_logic_1164.all;

entity mul55 is
port	(
		in_mul55	: in std_logic_vector(7 downto 0);
		out_mul55	: out std_logic_vector(7 downto 0)
		);
end mul55;

architecture mul55_arch of mul55 is
begin
	out_mul55(0) <= in_mul55(7) xor in_mul55(6) xor in_mul55(2) xor
in_mul55(0);
	out_mul55(1) <= in_mul55(7) xor in_mul55(3) xor in_mul55(1);
	out_mul55(2) <= in_mul55(7) xor in_mul55(6) xor in_mul55(4) xor
in_mul55(0);
```

```
	out_mul55(3) <= in_mul55(6) xor in_mul55(5) xor in_mul55(2) xor
in_mul55(1);
	out_mul55(4) <= in_mul55(7) xor in_mul55(6) xor in_mul55(3) xor
in_mul55(2) xor in_mul55(0);
	out_mul55(5) <= in_mul55(7) xor in_mul55(4) xor in_mul55(3) xor
in_mul55(1);
	out_mul55(6) <= in_mul55(7) xor in_mul55(6) xor in_mul55(5) xor
in_mul55(4) xor in_mul55(0);
	out_mul55(7) <= in_mul55(7) xor in_mul55(6) xor in_mul55(5) xor
in_mul55(1);
end mul55_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--	multiplier by 87
--

library ieee;
use ieee.std_logic_1164.all;

entity mul87 is
port	(
		in_mul87	: in std_logic_vector(7 downto 0);
		out_mul87	: out std_logic_vector(7 downto 0)
		);
end mul87;

architecture mul87_arch of mul87 is
begin
	out_mul87(0) <= in_mul87(7) xor in_mul87(5) xor in_mul87(3) xor
in_mul87(1) xor in_mul87(0);
	out_mul87(1) <= in_mul87(6) xor in_mul87(4) xor in_mul87(2) xor
in_mul87(1) xor in_mul87(0);
	out_mul87(2) <= in_mul87(2) xor in_mul87(0);
	out_mul87(3) <= in_mul87(7) xor in_mul87(5);
```

```
	out_mul87(4) <= in_mul87(6);
	out_mul87(5) <= in_mul87(7);
	out_mul87(6) <= in_mul87(7) xor in_mul87(5) xor in_mul87(3) xor
in_mul87(1);
	out_mul87(7) <= in_mul87(6) xor in_mul87(4) xor in_mul87(2) xor
in_mul87(0);
end mul87_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--	multiplier by 5a
--

library ieee;
use ieee.std_logic_1164.all;

entity mul5a is
port	(
		in_mul5a	: in std_logic_vector(7 downto 0);
		out_mul5a	: out std_logic_vector(7 downto 0)
		);
end mul5a;

architecture mul5a_arch of mul5a is
begin
	out_mul5a(0) <= in_mul5a(7) xor in_mul5a(5) xor in_mul5a(2);
	out_mul5a(1) <= in_mul5a(6) xor in_mul5a(3) xor in_mul5a(0);
	out_mul5a(2) <= in_mul5a(5) xor in_mul5a(4) xor in_mul5a(2) xor
in_mul5a(1);
	out_mul5a(3) <= in_mul5a(7) xor in_mul5a(6) xor in_mul5a(3) xor
in_mul5a(0);
	out_mul5a(4) <= in_mul5a(7) xor in_mul5a(4) xor in_mul5a(1) xor
in_mul5a(0);
	out_mul5a(5) <= in_mul5a(5) xor in_mul5a(2) xor in_mul5a(1);
```

```
	out_mul5a(6) <= in_mul5a(7) xor in_mul5a(6) xor in_mul5a(5) xor in_mul5a(3)
xor in_mul5a(0);
	out_mul5a(7) <= in_mul5a(7) xor in_mul5a(6) xor in_mul5a(4) xor
in_mul5a(1);
end mul5a_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --

--
--	multiplier by 58
--

library ieee;
use ieee.std_logic_1164.all;

entity mul58 is
port	(
		in_mul58	: in std_logic_vector(7 downto 0);
		out_mul58	: out std_logic_vector(7 downto 0)
		);
end mul58;

architecture mul58_arch of mul58 is
begin
	out_mul58(0) <= in_mul58(5) xor in_mul58(2);
	out_mul58(1) <= in_mul58(6) xor in_mul58(3);
	out_mul58(2) <= in_mul58(7) xor in_mul58(5) xor in_mul58(4) xor
in_mul58(2);
	out_mul58(3) <= in_mul58(6) xor in_mul58(3) xor in_mul58(2) xor
in_mul58(0);
	out_mul58(4) <= in_mul58(7) xor in_mul58(4) xor in_mul58(3) xor
in_mul58(1) xor in_mul58(0);
	out_mul58(5) <= in_mul58(5) xor in_mul58(4) xor in_mul58(2) xor
in_mul58(1);
	out_mul58(6) <= in_mul58(6) xor in_mul58(3) xor in_mul58(0);
	out_mul58(7) <= in_mul58(7) xor in_mul58(4) xor in_mul58(1);
```

```
end mul58_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by db
--

library ieee;
use ieee.std_logic_1164.all;

entity muldb is
port      (
                    in_muldb          : in std_logic_vector(7 downto 0);
                    out_muldb         : out std_logic_vector(7 downto 0)
                    );
end muldb;

architecture muldb_arch of muldb is
begin
          out_muldb(0) <= in_muldb(7) xor in_muldb(6) xor in_muldb(3) xor
in_muldb(2) xor in_muldb(1) xor in_muldb(0);
          out_muldb(1) <= in_muldb(7) xor in_muldb(4) xor in_muldb(3) xor
in_muldb(2) xor in_muldb(1) xor in_muldb(0);
          out_muldb(2) <= in_muldb(7) xor in_muldb(6) xor in_muldb(5) xor
in_muldb(4);
          out_muldb(3) <= in_muldb(5) xor in_muldb(3) xor in_muldb(2) xor
in_muldb(1) xor in_muldb(0);
          out_muldb(4) <= in_muldb(6) xor in_muldb(4) xor in_muldb(3) xor
in_muldb(2) xor in_muldb(1) xor in_muldb(0);
          out_muldb(5) <= in_muldb(7) xor in_muldb(5) xor in_muldb(4) xor
in_muldb(3) xor in_muldb(2) xor in_muldb(1);
          out_muldb(6) <= in_muldb(7) xor in_muldb(5) xor in_muldb(4) xor
in_muldb(1) xor in_muldb(0);
          out_muldb(7) <= in_muldb(6) xor in_muldb(5) xor in_muldb(2) xor
in_muldb(1) xor in_muldb(0);
```

```
end muldb_arch;


--
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by 9e
--

library ieee;
use ieee.std_logic_1164.all;

entity mul9e is
port      (
                    in_mul9e          : in std_logic_vector(7 downto 0);
                    out_mul9e         : out std_logic_vector(7 downto 0)
                    );
end mul9e;

architecture mul9e_arch of mul9e is
begin
          out_mul9e(0) <= in_mul9e(6) xor in_mul9e(4) xor in_mul9e(3) xor
in_mul9e(1);
          out_mul9e(1) <= in_mul9e(7) xor in_mul9e(5) xor in_mul9e(4) xor in_mul9e(2)
xor in_mul9e(0);
          out_mul9e(2) <= in_mul9e(5) xor in_mul9e(4) xor in_mul9e(0);
          out_mul9e(3) <= in_mul9e(5) xor in_mul9e(4) xor in_mul9e(3) xor
in_mul9e(0);
          out_mul9e(4) <= in_mul9e(6) xor in_mul9e(5) xor in_mul9e(4) xor in_mul9e(1)
xor in_mul9e(0);
          out_mul9e(5) <= in_mul9e(7) xor in_mul9e(6) xor in_mul9e(5) xor in_mul9e(2)
xor in_mul9e(1);
          out_mul9e(6) <= in_mul9e(7) xor in_mul9e(4) xor in_mul9e(2) xor
in_mul9e(1);
          out_mul9e(7) <= in_mul9e(5) xor in_mul9e(3) xor in_mul9e(2) xor
in_mul9e(0);
end mul9e_arch;
```

```
--
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
--                                                          new component
--
--
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --

--
--          multiplier by 56
--

library ieee;
use ieee.std_logic_1164.all;

entity mul56 is
port      (
                    in_mul56          : in std_logic_vector(7 downto 0);
                    out_mul56         : out std_logic_vector(7 downto 0)
                    );
end mul56;

architecture mul56_arch of mul56 is
begin
          out_mul56(0) <= in_mul56(6) xor in_mul56(2);
          out_mul56(1) <= in_mul56(7) xor in_mul56(3) xor in_mul56(0);
          out_mul56(2) <= in_mul56(6) xor in_mul56(4) xor in_mul56(2) xor
in_mul56(1) xor in_mul56(0);
          out_mul56(3) <= in_mul56(7) xor in_mul56(6) xor in_mul56(5) xor
in_mul56(3) xor in_mul56(1);
          out_mul56(4) <= in_mul56(7) xor in_mul56(6) xor in_mul56(4) xor
in_mul56(2) xor in_mul56(0);
          out_mul56(5) <= in_mul56(7) xor in_mul56(5) xor in_mul56(3) xor
in_mul56(1);
          out_mul56(6) <= in_mul56(4) xor in_mul56(0);
          out_mul56(7) <= in_mul56(5) xor in_mul56(1);
end mul56_arch;
```

```
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by 82
--

library ieee;
use ieee.std_logic_1164.all;

entity mul82 is
port      (
                    in_mul82          : in std_logic_vector(7 downto 0);
                    out_mul82         : out std_logic_vector(7 downto 0)
                    );
end mul82;

architecture mul82_arch of mul82 is
begin
          out_mul82(0) <= in_mul82(7) xor in_mul82(6) xor in_mul82(5) xor
in_mul82(3) xor in_mul82(1);
          out_mul82(1) <= in_mul82(7) xor in_mul82(6) xor in_mul82(4) xor
in_mul82(2) xor in_mul82(0);
          out_mul82(2) <= in_mul82(6);
          out_mul82(3) <= in_mul82(6) xor in_mul82(5) xor in_mul82(3) xor
in_mul82(1);
          out_mul82(4) <= in_mul82(7) xor in_mul82(6) xor in_mul82(4) xor
in_mul82(2);
          out_mul82(5) <= in_mul82(7) xor in_mul82(5) xor in_mul82(3);
          out_mul82(6) <= in_mul82(7) xor in_mul82(5) xor in_mul82(4) xor
in_mul82(3) xor in_mul82(1);
          out_mul82(7) <= in_mul82(6) xor in_mul82(5) xor in_mul82(4) xor
in_mul82(2) xor in_mul82(0);
end mul82_arch;
```

```
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by f3
--

library ieee;
use ieee.std_logic_1164.all;

entity mulf3 is
port      (
                    in_mulf3          : in std_logic_vector(7 downto 0);
                    out_mulf3         : out std_logic_vector(7 downto 0)
                    );
end mulf3;

architecture mulf3_arch of mulf3 is
begin
          out_mulf3(0) <= in_mulf3(7) xor in_mulf3(6) xor in_mulf3(2) xor in_mulf3(1)
xor in_mulf3(0);
          out_mulf3(1) <= in_mulf3(7) xor in_mulf3(3) xor in_mulf3(2) xor in_mulf3(1)
xor in_mulf3(0);
          out_mulf3(2) <= in_mulf3(7) xor in_mulf3(6) xor in_mulf3(4) xor in_mulf3(3);
          out_mulf3(3) <= in_mulf3(6) xor in_mulf3(5) xor in_mulf3(4) xor in_mulf3(2)
xor in_mulf3(1);
          out_mulf3(4) <= in_mulf3(7) xor in_mulf3(6) xor in_mulf3(5) xor in_mulf3(3)
xor in_mulf3(2) xor in_mulf3(0);
          out_mulf3(5) <= in_mulf3(7) xor in_mulf3(6) xor in_mulf3(4) xor in_mulf3(3)
xor in_mulf3(1) xor in_mulf3(0);
          out_mulf3(6) <= in_mulf3(6) xor in_mulf3(5) xor in_mulf3(4) xor in_mulf3(0);
          out_mulf3(7) <= in_mulf3(7) xor in_mulf3(6) xor in_mulf3(5) xor in_mulf3(1)
xor in_mulf3(0);
end mulf3_arch;
```

```
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --

--
--          multiplier by 1e
--


library ieee;
use ieee.std_logic_1164.all;

entity mul1e is
port      (
                    in_mul1e          : in std_logic_vector(7 downto 0);
                    out_mul1e         : out std_logic_vector(7 downto 0)
                    );
end mul1e;

architecture mul1e_arch of mul1e is
begin
          out_mul1e(0) <= in_mul1e(5) xor in_mul1e(4);
          out_mul1e(1) <= in_mul1e(6) xor in_mul1e(5) xor in_mul1e(0);
          out_mul1e(2) <= in_mul1e(7) xor in_mul1e(6) xor in_mul1e(5) xor in_mul1e(4)
xor in_mul1e(1) xor in_mul1e(0);
          out_mul1e(3) <= in_mul1e(7) xor in_mul1e(6) xor in_mul1e(4) xor in_mul1e(2)
xor in_mul1e(1) xor in_mul1e(0);
          out_mul1e(4) <= in_mul1e(7) xor in_mul1e(5) xor in_mul1e(3) xor in_mul1e(2)
xor in_mul1e(1) xor in_mul1e(0);
          out_mul1e(5) <= in_mul1e(6) xor in_mul1e(4) xor in_mul1e(3) xor in_mul1e(2)
xor in_mul1e(1);
          out_mul1e(6) <= in_mul1e(7) xor in_mul1e(3) xor in_mul1e(2);
          out_mul1e(7) <= in_mul1e(4) xor in_mul1e(3);
end mul1e_arch;

--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
```

```
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by c6
--

library ieee;
use ieee.std_logic_1164.all;

entity mulc6 is
port      (
                    in_mulc6          : in std_logic_vector(7 downto 0);
                    out_mulc6         : out std_logic_vector(7 downto 0)
                    );
end mulc6;

architecture mulc6_arch of mulc6 is
begin
          out_mulc6(0) <= in_mulc6(6) xor in_mulc6(5) xor in_mulc6(4) xor in_mulc6(3)
xor in_mulc6(2) xor in_mulc6(1);
          out_mulc6(1) <= in_mulc6(7) xor in_mulc6(6) xor in_mulc6(5) xor in_mulc6(4)
xor in_mulc6(3) xor in_mulc6(2) xor in_mulc6(0);
          out_mulc6(2) <= in_mulc6(7) xor in_mulc6(2) xor in_mulc6(0);
          out_mulc6(3) <= in_mulc6(6) xor in_mulc6(5) xor in_mulc6(4) xor
in_mulc6(2);
          out_mulc6(4) <= in_mulc6(7) xor in_mulc6(6) xor in_mulc6(5) xor
in_mulc6(3);
          out_mulc6(5) <= in_mulc6(7) xor in_mulc6(6) xor in_mulc6(4);
          out_mulc6(6) <= in_mulc6(7) xor in_mulc6(6) xor in_mulc6(4) xor in_mulc6(3)
xor in_mulc6(2) xor in_mulc6(1) xor in_mulc6(0);
          out_mulc6(7) <= in_mulc6(7) xor in_mulc6(5) xor in_mulc6(4) xor in_mulc6(3)
xor in_mulc6(2) xor in_mulc6(1) xor in_mulc6(0);
end mulc6_arch;



--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                              new component
--
```

```
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by 68
--

library ieee;
use ieee.std_logic_1164.all;

entity mul68 is
port      (
                    in_mul68          : in std_logic_vector(7 downto 0);
                    out_mul68         : out std_logic_vector(7 downto 0)
                    );
end mul68;


architecture mul68_arch of mul68 is
begin
          out_mul68(0) <= in_mul68(7) xor in_mul68(6) xor in_mul68(4) xor
in_mul68(3) xor in_mul68(2);
          out_mul68(1) <= in_mul68(7) xor in_mul68(5) xor in_mul68(4) xor
in_mul68(3);
          out_mul68(2) <= in_mul68(7) xor in_mul68(5) xor in_mul68(3) xor
in_mul68(2);
          out_mul68(3) <= in_mul68(7) xor in_mul68(2) xor in_mul68(0);
          out_mul68(4) <= in_mul68(3) xor in_mul68(1);
          out_mul68(5) <= in_mul68(4) xor in_mul68(2) xor in_mul68(0);
          out_mul68(6) <= in_mul68(7) xor in_mul68(6) xor in_mul68(5) xor
in_mul68(4) xor in_mul68(2) xor in_mul68(1) xor in_mul68(0);
          out_mul68(7) <= in_mul68(7) xor in_mul68(6) xor in_mul68(5) xor
in_mul68(3) xor in_mul68(2) xor in_mul68(1);
end mul68_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
```

```
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by e5
--

library ieee;
use ieee.std_logic_1164.all;

entity mule5 is
port      (
                  in_mule5          : in std_logic_vector(7 downto 0);
                  out_mule5         : out std_logic_vector(7 downto 0)
                  );
end mule5;


architecture mule5_arch of mule5 is
begin
          out_mule5(0) <= in_mule5(6) xor in_mule5(4) xor in_mule5(2) xor in_mule5(1)
xor in_mule5(0);
          out_mule5(1) <= in_mule5(7) xor in_mule5(5) xor in_mule5(3) xor in_mule5(2)
xor in_mule5(1);
          out_mule5(2) <= in_mule5(3) xor in_mule5(1) xor in_mule5(0);
          out_mule5(3) <= in_mule5(6);
          out_mule5(4) <= in_mule5(7);
          out_mule5(5) <= in_mule5(0);
          out_mule5(6) <= in_mule5(6) xor in_mule5(4) xor in_mule5(2) xor
in_mule5(0);
          out_mule5(7) <= in_mule5(7) xor in_mule5(5) xor in_mule5(3) xor in_mule5(1)
xor in_mule5(0);
end mule5_arch;



--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
```

```
--
--          multiplier by 02
--

library ieee;
use ieee.std_logic_1164.all;

entity mul02 is
port      (
                    in_mul02          : in std_logic_vector(7 downto 0);
                    out_mul02         : out std_logic_vector(7 downto 0)
                    );
end mul02;


architecture mul02_arch of mul02 is
begin
          out_mul02(0) <= in_mul02(7);
          out_mul02(1) <= in_mul02(0);
          out_mul02(2) <= in_mul02(7) xor in_mul02(1);
          out_mul02(3) <= in_mul02(7) xor in_mul02(2);
          out_mul02(4) <= in_mul02(3);
          out_mul02(5) <= in_mul02(4);
          out_mul02(6) <= in_mul02(7) xor in_mul02(5);
          out_mul02(7) <= in_mul02(6);
end mul02_arch;



--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                                  new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--          multiplier by a1
--

library ieee;
use ieee.std_logic_1164.all;
```

```
entity mula1 is
port	(
			in_mula1		: in std_logic_vector(7 downto 0);
			out_mula1		: out std_logic_vector(7 downto 0)
			);
end mula1;

architecture mula1_arch of mula1 is
begin
	out_mula1(0) <= in_mula1(7) xor in_mula1(6) xor in_mula1(1) xor
in_mula1(0);
	out_mula1(1) <= in_mula1(7) xor in_mula1(2) xor in_mula1(1);
	out_mula1(2) <= in_mula1(7) xor in_mula1(6) xor in_mula1(3) xor in_mula1(2)
xor in_mula1(1);
	out_mula1(3) <= in_mula1(6) xor in_mula1(4) xor in_mula1(3) xor in_mula1(2)
xor in_mula1(1);
	out_mula1(4) <= in_mula1(7) xor in_mula1(5) xor in_mula1(4) xor in_mula1(3)
xor in_mula1(2);
	out_mula1(5) <= in_mula1(6) xor in_mula1(5) xor in_mula1(4) xor in_mula1(3)
xor in_mula1(0);
	out_mula1(6) <= in_mula1(5) xor in_mula1(4);
	out_mula1(7) <= in_mula1(6) xor in_mula1(5) xor in_mula1(0);
end mula1_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--	multiplier by fc
--

library ieee;
use ieee.std_logic_1164.all;

entity mulfc is
port	(
```

```
                in_mulfc        : in std_logic_vector(7 downto 0);
                out_mulfc       : out std_logic_vector(7 downto 0)
                );
end mulfc;


architecture mulfc_arch of mulfc is
begin
        out_mulfc(0) <= in_mulfc(7) xor in_mulfc(5) xor in_mulfc(2) xor in_mulfc(1);
        out_mulfc(1) <= in_mulfc(6) xor in_mulfc(3) xor in_mulfc(2);
        out_mulfc(2) <= in_mulfc(5) xor in_mulfc(4) xor in_mulfc(3) xor in_mulfc(2)
xor in_mulfc(1) xor in_mulfc(0);
        out_mulfc(3) <= in_mulfc(7) xor in_mulfc(6) xor in_mulfc(4) xor in_mulfc(3)
xor in_mulfc(0);
        out_mulfc(4) <= in_mulfc(7) xor in_mulfc(5) xor in_mulfc(4) xor in_mulfc(1)
xor in_mulfc(0);
        out_mulfc(5) <= in_mulfc(6) xor in_mulfc(5) xor in_mulfc(2) xor in_mulfc(1)
xor in_mulfc(0);
        out_mulfc(6) <= in_mulfc(6) xor in_mulfc(5) xor in_mulfc(3) xor in_mulfc(0);
        out_mulfc(7) <= in_mulfc(7) xor in_mulfc(6) xor in_mulfc(4) xor in_mulfc(1)
xor in_mulfc(0);
end mulfc_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--      multiplier by c1
--

library ieee;
use ieee.std_logic_1164.all;

entity mulc1 is
port    (
                in_mulc1        : in std_logic_vector(7 downto 0);
                out_mulc1       : out std_logic_vector(7 downto 0)
```

```
                    );
end mulc1;


architecture mulc1_arch of mulc1 is
begin
        out_mulc1(0) <= in_mulc1(7) xor in_mulc1(5) xor in_mulc1(4) xor in_mulc1(3)
xor in_mulc1(2) xor in_mulc1(1) xor in_mulc1(0);
        out_mulc1(1) <= in_mulc1(6) xor in_mulc1(5) xor in_mulc1(4) xor in_mulc1(3)
xor in_mulc1(2) xor in_mulc1(1);
        out_mulc1(2) <= in_mulc1(6) xor in_mulc1(1);
        out_mulc1(3) <= in_mulc1(5) xor in_mulc1(4) xor in_mulc1(3) xor
in_mulc1(1);
        out_mulc1(4) <= in_mulc1(6) xor in_mulc1(5) xor in_mulc1(4) xor
in_mulc1(2);
        out_mulc1(5) <= in_mulc1(7) xor in_mulc1(6) xor in_mulc1(5) xor
in_mulc1(3);
        out_mulc1(6) <= in_mulc1(6) xor in_mulc1(5) xor in_mulc1(3) xor in_mulc1(2)
xor in_mulc1(1) xor in_mulc1(0);
        out_mulc1(7) <= in_mulc1(7) xor in_mulc1(6) xor in_mulc1(4) xor in_mulc1(3)
xor in_mulc1(2) xor in_mulc1(1) xor in_mulc1(0);
end mulc1_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--      multiplier by 47
--

library ieee;
use ieee.std_logic_1164.all;

entity mul47 is
port    (
                in_mul47        : in std_logic_vector(7 downto 0);
                out_mul47       : out std_logic_vector(7 downto 0)
```

```
                );
end mul47;

architecture mul47_arch of mul47 is
begin
        out_mul47(0) <= in_mul47(4) xor in_mul47(2) xor in_mul47(0);
        out_mul47(1) <= in_mul47(5) xor in_mul47(3) xor in_mul47(1) xor
in_mul47(0);
        out_mul47(2) <= in_mul47(6) xor in_mul47(1) xor in_mul47(0);
        out_mul47(3) <= in_mul47(7) xor in_mul47(4) xor in_mul47(1);
        out_mul47(4) <= in_mul47(5) xor in_mul47(2);
        out_mul47(5) <= in_mul47(6) xor in_mul47(3);
        out_mul47(6) <= in_mul47(7) xor in_mul47(2) xor in_mul47(0);
        out_mul47(7) <= in_mul47(3) xor in_mul47(1);
end mul47_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--      multiplier by ae
--

library ieee;
use ieee.std_logic_1164.all;

entity mulae is
port    (
                in_mulae        : in std_logic_vector(7 downto 0);
                out_mulae       : out std_logic_vector(7 downto 0)
                );
end mulae;

architecture mulae_arch of mulae is
begin
        out_mulae(0) <= in_mulae(7) xor in_mulae(5) xor in_mulae(1);
        out_mulae(1) <= in_mulae(6) xor in_mulae(2) xor in_mulae(0);
```

```
	out_mulae(2) <= in_mulae(5) xor in_mulae(3) xor in_mulae(0);
	out_mulae(3) <= in_mulae(7) xor in_mulae(6) xor in_mulae(5) xor in_mulae(4)
xor in_mulae(0);
	out_mulae(4) <= in_mulae(7) xor in_mulae(6) xor in_mulae(5) xor in_mulae(1);
	out_mulae(5) <= in_mulae(7) xor in_mulae(6) xor in_mulae(2) xor in_mulae(0);
	out_mulae(6) <= in_mulae(5) xor in_mulae(3);
	out_mulae(7) <= in_mulae(6) xor in_mulae(4) xor in_mulae(0);
end mulae_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                              new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--	multiplier by 3d
--

library ieee;
use ieee.std_logic_1164.all;

entity mul3d is
port	(
		in_mul3d	: in std_logic_vector(7 downto 0);
		out_mul3d	: out std_logic_vector(7 downto 0)
		);
end mul3d;

architecture mul3d_arch of mul3d is
begin
	out_mul3d(0) <= in_mul3d(4) xor in_mul3d(3) xor in_mul3d(0);
	out_mul3d(1) <= in_mul3d(5) xor in_mul3d(4) xor in_mul3d(1);
	out_mul3d(2) <= in_mul3d(6) xor in_mul3d(5) xor in_mul3d(4) xor
in_mul3d(3) xor in_mul3d(2) xor in_mul3d(0);
	out_mul3d(3) <= in_mul3d(7) xor in_mul3d(6) xor in_mul3d(5) xor
in_mul3d(1) xor in_mul3d(0);
	out_mul3d(4) <= in_mul3d(7) xor in_mul3d(6) xor in_mul3d(2) xor
in_mul3d(1) xor in_mul3d(0);
```

```
	out_mul3d(5) <= in_mul3d(7) xor in_mul3d(3) xor in_mul3d(2) xor
in_mul3d(1) xor in_mul3d(0);
	out_mul3d(6) <= in_mul3d(2) xor in_mul3d(1);
	out_mul3d(7) <= in_mul3d(3) xor in_mul3d(2);
end mul3d_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++ --

--
--	multiplier by 19
--

library ieee;
use ieee.std_logic_1164.all;

entity mul19 is
port	(
		in_mul19	: in std_logic_vector(7 downto 0);
		out_mul19	: out std_logic_vector(7 downto 0)
		);
end mul19;

architecture mul19_arch of mul19 is
begin
	out_mul19(0) <= in_mul19(7) xor in_mul19(6) xor in_mul19(5) xor
in_mul19(4) xor in_mul19(0);
	out_mul19(1) <= in_mul19(7) xor in_mul19(6) xor in_mul19(5) xor
in_mul19(1);
	out_mul19(2) <= in_mul19(5) xor in_mul19(4) xor in_mul19(2);
	out_mul19(3) <= in_mul19(7) xor in_mul19(4) xor in_mul19(3) xor
in_mul19(0);
	out_mul19(4) <= in_mul19(5) xor in_mul19(4) xor in_mul19(1) xor
in_mul19(0);
	out_mul19(5) <= in_mul19(6) xor in_mul19(5) xor in_mul19(2) xor
in_mul19(1);
```

```
	out_mul19(6) <= in_mul19(5) xor in_mul19(4) xor in_mul19(3) xor
in_mul19(2);
	out_mul19(7) <= in_mul19(6) xor in_mul19(5) xor in_mul19(4) xor
in_mul19(3);
end mul19_arch;


--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --
--
--                                                          new component
--
--
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ --

--
--	multiplier by 03
--

library ieee;
use ieee.std_logic_1164.all;

entity mul03 is
port	(
		in_mul03	: in std_logic_vector(7 downto 0);
		out_mul03	: out std_logic_vector(7 downto 0)
		);
end mul03;

architecture mul03_arch of mul03 is
begin
	out_mul03(0) <= in_mul03(7) xor in_mul03(0);
	out_mul03(1) <= in_mul03(1) xor in_mul03(0);
	out_mul03(2) <= in_mul03(7) xor in_mul03(2) xor in_mul03(1);
	out_mul03(3) <= in_mul03(7) xor in_mul03(3) xor in_mul03(2);
	out_mul03(4) <= in_mul03(4) xor in_mul03(3);
	out_mul03(5) <= in_mul03(5) xor in_mul03(4);
	out_mul03(6) <= in_mul03(7) xor in_mul03(6) xor in_mul03(5);
	out_mul03(7) <= in_mul03(7) xor in_mul03(6);
end mul03_arch;
```

Додаток Е

Завідувачу кафедри
комп’ютерної інженерії
к.т.н., доц. О.М. Березькому

ДОВІДКА ПРО ВИКОРИСТАННЯ

Виконаний студентом групи КСМзс-51 факультету комп’ютерних інформаційних технологій Тернопільського національного економічного університету Жалко О.М. дипломний проект та тему „Апаратна реалізація операції шифрування криптоалгоритму Twofish” відповідає профілю підприємства, має практичну значимість і планується для використання на фірмі “Polyservice” ПП Булата М.М. А саме, можливість захисту конфіденційної інформації.

Директор підприємства (організації) _____

Прізвище, ініціали підпис

М.П.