МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Середа Юрій Іванович

Програмний засіб реалізації криптоалгоритму AES для захисту конфіденційної інформації / Software of cryptoalgorithm AES implementation for confidential information protecting

спеціальність: 6.050102 - Комп'ютерна інженерія
освітньо-професійна програма - Комп'ютерні системи та мережі

Випускна кваліфікаційна робота

Виконав: студент групи КСМ-41/1
Середа Юрій Іванович
_____

Науковий керівник:
к.т.н. Дубчак Л.О.
_____

Випускну кваліфікаційну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
О. М. Березький

ТЕРНОПІЛЬ - 2019

РЕЗЮМЕ

Робота виконана на 76 сторінках, містить 11 рисунків, 14 таблиць, 9 додатків, 2 графічних матеріалів.

Метою дипломного проекту є розробка програмної реалізації шифрування симетричного крипто алгоритму на мові VHDL. Основним результатом даної роботи є розроблена VHDL-модель ядра процесора шифрування згідно з алгоритмом AES.

Розроблене ядро системи може бути використане в будь-якій організації (навчальні заклади, підприємства з виготовлення обладнання, підприємства з виготовлення систем захисту інформації, тощо), де є потреба у створенні процесорів симетричного блокового шифрування для захисту інформації від неавторизованого втручання, в тому числі з комп'ютерних мереж.

Ключові слова: ЗАХИСТ ІНФОРМАЦІЇ, СИМЕТРИЧНИЙ КРИПТОАЛГОРИТМ, AES, VHDL, ACTIVE-HDL.

RESUME

Work is executed on 76 pages, contains 11 figures, 14 tables, 9 additions, from them 2 graphic material.

The purpose of the workt is development of hardware representation of enciphering of symmetric cryptoalgorithm. By a basic this job performance VHDL of model kernel of processor of enciphering is worked out according to the algorithm of AES.

The worked out hardcore can be used in any organization (educational establishments, enterprises from making of equipment, enterprise from making of the systems and others like that), where a requirement is in creation of processors of the symmetric sectional enciphering for the unauthorized interference, including from computernetworks.

Keywords: information security, symmetric cryptoalgorithm, AES, VHDL, ACTIVE-HDL.

ЗМІСТ

Вступ…….……..9
1 Криптографічний аналіз та оцінка алгоритмів шифрування…….10
1.1 Симетричні криптосистеми……..…….…….....10
1.2 Особливості симетричного алгоритму шифрування AES ……16
1.3 Формування вимог і постановка задачі……...22
2 Проектування операції шифрування криптоалгоритму AES…….…...24
2.1 Розробка вхідного та вихідного блоків FIFO…….24
2.2 Розробка компоненти керування даними ……..28
2.3 Розробка компоненти шифрування AES Cipher …..……....….....31
2.4 Функціональна симуляція проекту…….34
3 Симуляція та верифікація проекту ……...……..38
3.1 Обґрунтування вибору елементної бази ……38
3.2 Реалізація проекту у вибраній елементній базі ……42
3.3 Дослідження характеристик шифрування криптоалгоритму AES…..47
4 Техніко-економічне обґрунтування розробки…….....49
4.1 Розрахунок витрат на розробку програмного забезпечення …….....49
4.2 Визначення експлуатаційних витрат …….……55
4.3 Розрахунок ціни споживання програмного продукту……..58
Висновки…….……...60
Список використаних джерел…….…….....62
Додаток А VHDL-код системи шифрування алгоритму AES …...…..……...65
Додаток Б Світлокопія тез……73
Додаток В Довідка про впровадження……76

ВСТУП

З розвитком і постійним розповсюдженням комп'ютерних телекомунікацій в різних сферах людської діяльності, все більш гостро постає питання безпечного обміну даними через незахищені канали передачі. Одним із ефективних засобів захисту інформації є криптографічні перетворення.

У 2001 році, після кількох років відкритого обговорення, американський Національний інститут стандартів і технологій затвердив новий розширений стандарт шифрування AES (Advanced Encryption Standard), що прийшов на зміну DES (Data Encryption Standard), який більше 20-ти років був дефакто загальносвітовим стандартом шифрування [1].

Мета даного дипломного проекту полягає у розробці та дослідженні програмної реалізації операцій шифрування симетричного криптоалгоритму AES мовою опису апаратних ресурсів VHDL.

Для реалізації цієї мети необхідно вирішити такі завдання:

– аналіз сучасних симетричних криптосистеми захисту інформації;

– вирішення практичної задачі шифрування інформації на основі алгоритму AES;

– розробка параметризованої реалізації математичного апарату AES мовою VHDL;

– дослідження архітектури „Квадрат" на основі засобів керування характеристиками алгоритму AES, що закладені в систему;

– дослідження продуктивності синтезованих процесорів шифрування за алгоритмом AES.

1 КРИПТОГРАФІЧНИЙ АНАЛІЗ ТА ОЦІНКА АЛГОРИТМІВ ШИФРУВАННЯ

1.1 Симетричні криптосистеми

Постулатом для симетричних криптосистем є таємність ключа. Симетричні криптосхеми в даний час прийнято підрозділяти на блокові та потокові [1].

Блокові криптосистеми розбивають текст повідомлення (файлу, документа і т.д.) на окремі блоки і потім здійснюють перетворення цих блоків з використанням ключа.

У потокових криптосистемах на основі ключа системи виробляється якась послідовність, так звана вихідна гамма, яка потім накладається на текст повідомлення. Таким чином, перетворення тексту здійснюється начебто потоком у міру вироблення гамми. Як правило, ці криптосистеми використовуються для потреб військових, шифрування в засобах зв'язку і т.д.

Блокові шифри оперують з блоками відкритого тексту. Вони повинні задовольняти наступні вимоги:

- достатня криптостійкість;
- простота процедур шифрування та дешифрування;
- прийнятна надійність.

Криптостійкістю вважається час, необхідний для розкриття шифру при використанні найкращого методу криптоаналізу. Надійність - це частина інформації, яка дешифрується за допомогою якогось криптоаналітичного алгоритму.

Саме перетворення шифру повинне використовувати наступні принципи (по Д. Шеннону) [1]:

- розсіювання (diffusion), тобто зміна будь-якого знаку відкритого тексту чи ключа впливає на велику кількість знаків шифротексту;

– перемішування (confusion) – використання перетворень, які перешкоджають отриманню статистичних залежностей між шифротекстом та відкритим текстом.

Практично всі сучасні блокові шифри є композиційними, тобто складаються з композиції простих перетворень. Саме по собі перетворення може і не забезпечувати потрібних властивостей, але їх ланцюжок дозволяє одержати необхідний результат. Наприклад, стандарт DES складається з 16 циклів. В іноземній літературі такі шифри часто називають пошаровими (layered). Якщо ж використовується одне і те ж перетворення, то такий композиційний шифр називають ітераційним шифром [2].

Об'єктивно порівняти переваги алгоритмів шифрування досить складно. Претенденти на роль AES мають багато загального, але є і важливі відмінності, хоча і не існує прийнятного методу, що дозволяє надійно оцінити, які з цих відмінностей впливають на безпеку зашифрованих даних.

Щоб вирішити цю задачу, фахівці із шифрування розробили методику для дослідження алгоритмів шифрування. Один з підходів передбачав аналіз версій кожного алгоритму зі скороченим числом раундів. Оскільки у всіх п'ятьох кандидатах передбачене виконання серії окремих раундів, криптографи можуть вивчати спрощені версії кожного алгоритму, зменшуючи число виконуваних раундів.

Наприклад, при шифруванні 128-розрядним ключем Rijndael виконує 10 раундів. Криптографи проаналізували рівень захисту Rijndael і виявили недоліки при виконанні семи чи меншого числа раундів. Аналогічним чином були перевірені й інші кандидати на роль AES. У результаті було виявлено, що Rijndael стає досить стійким до атак вже починаючи з восьмого раунду і виконує після цього ще два раунди шифрування. Фахівці NIST прийшли до висновку, що дане рішення має адекватний запас захисту, хоча інші кандидати мають навіть більший запас міцності.

Для оцінки продуктивності в NIST провели тестування програмних і

апаратних реалізацій алгоритмів, а також реалізацій для смарт-карт (названих у NISI версіями з ресурсними обмеженнями - restricted space). При тестуванні програмного забезпечення розглядалися 32-розрядні реалізації на C, Java і для смарт-карт на базі ARM, а крім того, реалізації на 64- і 8-розрядних процесорах і процесорах для обробки цифрових сигналів [3].

На рисунку 1.1 наводяться порівняльні оцінки продуктивності програмного забезпечення всіх кандидатів на роль AES. При аналізі версій алгоритмів, призначених для смарт-карт, і інших реалізацій з ресурсними обмеженнями NIST використовував два середовища: смарт-карти з 8-розрядним процесором і процесором Motorola 6805, які мають оперативну пам'ять ємністю всього 120 байт.

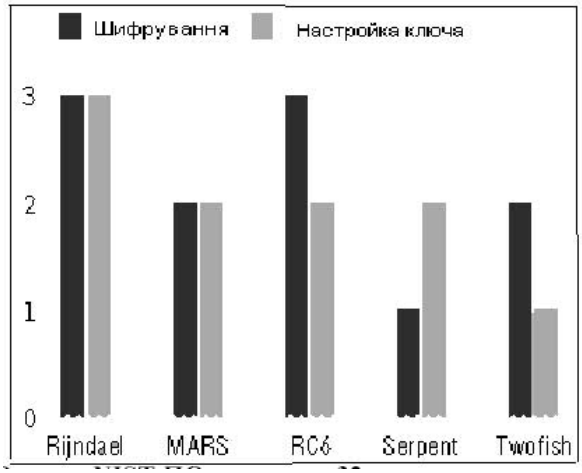

Рисунок 1.1 — Оцінка NIST ПЗ на основі 32-розрядних реалізацій

Фахівці NIST також аналізували характеристики архітектури, здатні впливати на продуктивність (рисунок 1.2). Тут, зокрема, ними оцінювалися відносні витрати на шифрування і дешифрування, а також можливість рівнобіжної реалізації алгоритму, що дозволяє домогтися більш високої швидкості обробки. Ще одна характеристика, що бралася до уваги, стосувалася витрат на зміну ключів - більшість алгоритмів шифрування передбачає деяку попередню обробку ключа перш, ніж вони можуть почати шифрування і дешифрування. Ця затримка має важливе значення для пристроїв, що виконують шифрування для безлічі різних з'єднань, таких, як захищений сервер Web чи шлюз IPSec [4].

MARS виявився найскладнішим з усіх представлених кандидатів. У той час як інші алгоритми у всіх раундах використовують ту саму функцію, у MARS застосовуються чотири різні функції. Як показало тестування варіантів зі скороченим числом раундів, це забезпечує даному алгоритму дуже високий запас міцності. Однак його складність змусила засумніватися в цих оцінках, а деякі з фахівців, що брали участь у тестуванні, порахували, що MARS вимагає більш ретельного аналізу, ніж той, котрий можна зробити у відведений для експертизи час.

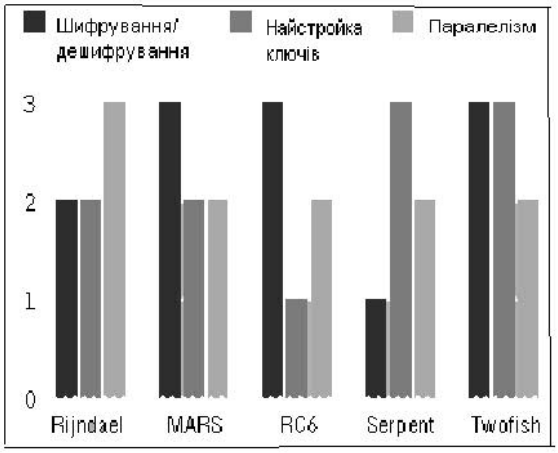

Рисунок 1.2 — Оцінка NIST особливостей архітектури AES

У MARS використані множення, змінне чергування і великі таблиці даних. Усе це значно ускладнює його захист від атак на реалізацію, при тому, що модифікація MARS з метою посилення захисту від таких атак серйозно знижує його продуктивність.

За продуктивність MARS не одержав оцінок вище середніх. У цілому продуктивність його програмної реалізації знаходиться на середньому рівні, хоча результати значно варіюються в залежності від застосовуваних процесорів і компіляторів. Оцінки апаратних реалізацій виявилися нижче середнього, поза залежністю від довжини ключа. MARS не дуже добре підходить для реалізацій у смарт-картах, оскільки вимагає оперативну пам'ять великої ємності. У кінцевому рахунку цей алгоритм виявився відкинутий через оцінку по

продуктивності і винятково високої складності.

RC6 - це простий алгоритм з адекватним запасом міцності. Він базується на RC5, розробленим раніше в RSA Security, застосування якого не виявило якихось серйозних проблем. Як і в MARS, у RC6 використовуються множення і змінне чергування, у силу чого RC6 важко захистити від атак на реалізацію, хоча і не настільки складно, як MARS. 14

Крім того, RC6 працює досить швидко. У деяких випадках, зокрема, у програмних реалізаціях на 32-розрядних процесорах, він випереджає Rijndael, але апаратні реалізації мають лише середню продуктивність. RC6 потрібно багато оперативної пам'яті, у силу чого він не дуже добре підходить для середовищ з ресурсними обмеженнями. RC6 не став переможцем через низьку продуктивність при апаратній реалізації.

Serpent схожий на Rijndael, але замість виконання невеликого числа більш складних раундів Serpent виконує більше простих раундів. Завдяки своїй простій, надійній архітектурі Serpent повторює деякі характеристики DES і, у цілому, спирається на добре відомі операції. Через цю простоту і популярність, оцінити надійність Serpent виявилося набагато простіше, і після вивчення версії зі скороченим числом раундів з'ясувалося, що він має високий запас міцності. Serpent відноситься до тих алгоритмів, які найпростіше захистити від атак на реалізацію.

На жаль, програмні реалізації Serpent виявилися найповільнішими серед фіналістів. З іншого боку, у деяких випадках тестери змогли організувати конвеєр апаратних реалізацій, що показав дуже високу продуктивність. Збільшення розміру ключа не впливало на швидкість роботи. Через низькі вимоги до пам'яті Serpent добре підходить для застосування в смарт-картах.

Хоча Serpent пропонував краще сполучення простоти і запасу міцності, чим Rijndael, він уступив останньому через низьку продуктивність програмних реалізацій [5].

Twofish використовує кардинально новий підхід, при якому половина

ключа використовується для зміни роботи самого алгоритму шифрування, і в цьому підалгоритмі як власний ключ шифрування застосовується інша половина вихідного ключа. Ця особливість приводить до поділу ключа, що, на думку деяких аналітиків, може зробити алгоритм хитким до атак, організованих за принципом "розділяй і пануй". При подібній атаці хакер може спробувати визначити, який ключ був обраний у підалгоритмі, і відразу ж одержати половину значення ключа. Однак при аналізі тестерам не вдалося провести ні одну з подібних атак.

Вивчення варіантів Twofish зі скороченим числом раундів показало, що він має високий запас міцності. Однак, як і у випадку з MARS, його незвичайна структура викликала сумніви. Деякі тестери відзначали, що через складність Twofish проаналізувати його детально у відведені для цього терміни виявилося дуже складно.

Twofish вразливий до атак на реалізацію, але його можна модифікувати таким чином, щоб він був здатним ефективно протистояти деяким атакам. У цілому Twofish показав середню продуктивність. Продуктивність програмних реалізацій виявилася нижче середньої, а час попередньої обробки ключа найбільшим. Продуктивність апаратних реалізацій була середньою. Завдяки обмеженим вимогам до пам'яті цей алгоритм підходить для реалізації на смарт-картах. NIST не вибрав Twofish через його порівняно низьку продуктивність і складність алгоритму.

Rijndael - швидкий і компактний алгоритм із простою математичною структурою, завдяки чому він виявився простим для аналізу при оцінці рівня захисту, і ніяких претензій фахівці NIST при цьому не висловили. З іншого боку, через цю простоту хакерам буде потрібно вивчити більш обмежений математичний апарат - і якщо десь у Rijndael є сховані проблеми, то рано чи пізно їх хто-небудь знайде. Атаки на версію зі скороченим числом раундів показали, що Rijndael не має такого запасу міцності, як інші кандидати, а збільшення числа раундів сповільнює його роботу.

Крім того, Rijndael продемонстрував гарну стійкість до атак на реалізацію, при яких хакер намагається декодувати зашифроване повідомлення, аналізуючи зовнішні прояви алгоритму, у тому числі рівень енергоспоживання і час виконання. У NIST були перевірені уразливість усіх кандидатів до таких атак і їх здатність протистояти їм, звичайно за рахунок спеціального кодування, для вирівнювання рівня енергоспоживання. Rijndael можна легко захистити від таких атак, оскільки він спирається в основному на булеві операції.

Загальна продуктивність програмних реалізацій Rijndael виявилася найкращою. Він прекрасно пройшов усі тести зі смарт-картами й в апаратних реалізаціях. Алгоритму в значній мірі властивий внутрішній паралелізм, що дозволяє забезпечити ефективне використання процесорних ресурсів [6]. Збільшення довжини ключа трохи сповільнює його роботу, оскільки при обробці ключів більшої довжини алгоритм передбачає виконання додаткових раундів шифрування.

1.2 Особливості симетричного алгоритму шифрування AES

Для того, щоб алгоритм AES став гідною заміною DES, його архітектура повинна задовольняти декільком критеріям: високий ступінь захисту, проста структура і висока продуктивність.

Очевидно, що найвищим пріоритетом для алгоритму AES є захист. Уже на рівні внутрішньої архітектури він повинен мати надійність, достатню для того, щоб протистояти майбутнім спробам його злому.

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

Наступною вимогою, запропонованою до AES, є висока продуктивність.

Широке поширення алгоритму на ринку потребує прийнятної продуктивності при роботі на самих різних платформах від смарт-карт до великих серверів. Відмінна продуктивність алгоритму вимагає високу швидкість роботи при шифруванні і дешифруванні, а також при реалізації графіка ключа.

У 1998 році NIST оголосив конкурс на створення алгоритму, що задовольняє висунутим Інститутом вимогам. Він опублікував усі несекретні дані про тестування

кандидатів на роль AES і зажадав від авторів алгоритмів повідомити про базові принципи побудови, які використовуються у них константи, таблиці S-box. На відміну від ситуації з DES, NIST при виборі AES не став опиратися на секретні і, як наслідок, заборонені до публікації дані про дослідження алгоритмів-кандидатів [7].

Щоб бути затвердженим як стандарт, алгоритм повинен:

– реалізувати шифрування приватним ключем;

– являти собою блоковий шифр;

– працювати з 128-розрядними блоками даних і ключами трьох розмірів (128, 192 і 256 розрядів).

Додатково кандидатам рекомендувалося [8]:

– 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 усі зацікавлені сторони могли самостійно провести незалежний криптоаналіз алгоритму і переконатися, що в ньому не закладено ніяких недокументованих можливостей.

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

Перед проведенням першого туру конкурсу в NIST надійшло 21

пропозиція, з яких 15 задовольняли висунутим критеріям. Потім були проведені дослідження цих рішень, у тому числі зв'язані з дешифруванням і перевіркою продуктивності, і отриманню експертних оцінок фахівців із криптографії. У серпні 1999 року NIST оголосив п'ять фіналістів, короткі відгуки про які приводяться в таблиці 1.1.

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

Таблиця 1.1—- Переможці першого етапу конкурсу NIST

Алгоритм	Фірма, яка запропонувала	Основна перевага	Основний недолік	Країна	Швидкодія
MARS	Корпорація IBM	Високий рівень захисту	Складна реалізація	USA	8 Мбайт/с
RC6	Компанія RSA Security	Дуже простий	Недостатній рівень захисту	USA	12 Мбайт/с
Rijndael	Джоан Дімен і Вінсент Ріджмен	Проста архітектура, гарні загальні характеристики	Незадовільний графік	BE	7 Мбайт/с
Serpent	Росс Андерсон, Елі Біхам і Ларс Кнудсен	Високий рівень захисту	Складна архітектура, невисока продуктивність	UK, NO	2 Мбайт/с
Twofish	Брюс Шнайер і інші співробітники компанії Counterpane Internet Security	Прийнятна продуктивність і рівень захисту	Складна архітектура	USA	11 Мбайт/с

Процес вибору алгоритму для AES характеризувався відкритістю і прозорістю. За словами Едварда Робака, глави відділу NIST по захисту інформації і голови комітету AES, процес вибору стандарту шифрування докорінно змінився в порівнянні з 1975 роком, коли уряд США оголосив конкурс алгоритмів для DES.

«Рівень знань широкого кола фахівців в області шифрування зараз зовсім інший, — підкреслив Робак. — Алгоритми-претенденти повинні бути у вільному доступі, щоб будь-хто міг зробити спробу їх злому. Якщо алгоритм вистоїть, то користувачі будуть йому довіряти» [8].

«Ми сподіваємося, що AES буде жити довго і стане могутнім захистом для електронної комерції в новому сторіччі, — продовжував Робак. — Коли думаєш про трильйони доларів, що AES зможе захистити, то усвідомлюєш важливість того, що зараз відбувається. Тому ми намагаємося залучити до нього якнайбільше народу» [8].

На першій сесії обговорювалися питання, що стосуються програмованих матриць (FPGA — field programmable gate array). Під час дискусії учасники прийшли до висновку, що деякі з алгоритмів AES, програмна реалізація яких не є досконалістю, виявляються набагато ефективніше в реалізації FPGA. Прикладом тому може служити алгоритм Serpent, що прекрасно реалізується на апаратному рівні.

Під час другої сесії проводилася оцінка реалізації алгоритмів на різних платформах, у тому числі PA-RISC, IA-64, Alpha, високорівневих смарт-картах і сигнальних процесорах. Однак учасники не прийшли до єдиного висновку і стало ясно, що жоден з алгоритмів не підходить для всіх платформ відразу. У цілому, очевидно, досить добре на різних платформах працює алгоритм Rijndael.

III сесія була присвячена оглядам і порівнянню продуктивності претендентів на стандарт, у тому числі їх реалізацій на мовах Ассемблер, C і Java.

Останній матеріал, що обговорювався на цій сесії і представлений

Андреасом Стернбенцем з австрійського Інституту прикладної обробки інформації і комунікацій, стосувався продуктивності Java. Стернбенц прийшов до висновку, що для Java найшвидшим став алгоритм RC6, а Rijndael показує прекрасні результати при роботі з 128-розрядними ключами і набагато гірше поводиться при використанні ключів більшої довжини. Очевидно, хоча Rijndael і має чудовий графік ключа, він використовує додаткові цикли для ключів більшої довжини.

Підсумок третьої сесії такий: продуктивність залежить від реалізації, і, хоча є деякі стійкі результати, ситуація постійно міняється.

Варто зазначити, що за інших рівних умов збільшення числа раундів в алгоритмі може значно збільшити його рівень захисту. Наприклад, алгоритм Serpent, як було відзначено, має найвижчий рівень захисту серед фіналістів, використовує 32 раунди — більше, ніж кожний з його конкурентів.

Після IV сесії стало зрозуміло, що до жодного з алгоритмів-кандидатів не було спроби дешифрувати код, використовуючи всі потенційно можливі для нього числа раундів. Однак під час тестування при шифруванні скороченим числом раундів деякі алгоритми показали потенційні вади [9].

Неформальна додаткова сесія виявилася дуже корисною й інформативною. Навіть просте перерахування представлених алгоритмів доводить, що при виборі AES необхідно враховувати безліч факторів: порівняльне дослідження продуктивності фіналістів AES, що використовують FPGA; емпіричний взаємозв'язок функцій різних раундів; криптоаналітика — уроки для AES; IP Security і ключі шифрування; залежні від даних перестановки і псевдовипадковість (ідеалізованого) RC6; важкі для аналізу шифри; реалізація п'яти фіналістів AES на процесорі ARM; продуктивність фіналістів на мультимедійному VLIW-процесорі TriMedia.

Виступи на V сесії продемонстрували результати тестування алгоритму Rijndael зі скороченим числом раундів, тим самим підтвердивши той факт, що Rijndael може виявитися недостатньо захищеним, якщо його розробники не збільшать число раундів.

Однак загальний висновок сесії полягав у тому, що не було проведено достатньої кількості тестів для криптографічних алгоритмів, що, як передбачається, повинні протистояти атакам у найближчому десятилітті.

NIST міг прийняти рішення інтегрувати в остаточний стандарт усі п'ять алгоритмів-претендентів. Проводивши шосту сесію Майлс Сміт, що раніше працював в NIST, помітив, що остаточний варіант для стандарту AES приходиться вибирати з 31 можливої комбінації (вибрати всі п'ять, чотири з п'яти, три з п'яти і т.д.). Основне питання саме і полягало у тому, який варіант вийде переможцем.

Дон Джонсон з компанії Certicom говорив про необхідність стійкості — здатності AES витримувати могутні атаки в майбутньому. Джонсон упевнений, що, хоча і несправедливо протиставляти сучасну криптографічну технологію майбутнім невідомим атакам, битву необхідно виграти в будь-якому випадку. Він згадав квантові обчислення як приклад тих могутніх засобів злому, що майбутні хакери можуть мати у своєму арсеналі.

Джонсон також засумнівався чи зможе стандарт AES, якщо він буде створений на основі якого-небудь одного алгоритму, працювати на різних платформах — RISC, смарт-карти, FPGA і т.д. Отже, що на даному етапі набагато більше питань, ніж відповідей.

Матеріали сьомої сесії говорять про те, що отримані результати так і не змогли допомогти вибрати найбільше гідного кандидата на стандарт. Більше того, результати тестування можна було інтерпретувати будь-яким чином.

Сама жарка дискусія розгорілася з питання про те, чи повинен стандарт на AES містити в собі відразу кілька алгоритмів. Аргумент на користь такого рішення полягав у наступному: якщо перший алгоритм виявиться невдалим, можна буде переключитися на запасний. Однак учасники конференції з великим скепсисом поставилися до цього твердження: зрештою, оскільки всі алгоритми-кандидати пропонують приблизно один і той рівень захисту, у тому випадку, якщо перший буде зламаний, не встоять і інші. Більше того, AES, який передбачає використання декількох алгоритмів, збільшить труднощі в

реалізації. Однак, оскільки питання про інтелектуальну власність здобуває все більшого значення, було визнано, що використання додаткового алгоритму буде дуже ефективним [10].

1.3 Формування вимог і постановка задачі

На практиці часто необхідно здійснювати передачу інформації між віддаленими абонентами. Щоб зберегти зміст повідомлення в таємниці, така інформація шифрується. Використання криптосистем для захисту інформації від

несанкціонованого доступу базоване на виконанні наступних умов [7]:

- може бути вироблена таємна інформація, яка називається таємним ключем і яка має порівняно невеликий об’єм;
- таємний ключ може розподілятись між двома абонентами по захищеному каналу таким чином, щоб він не став відомим стороннім людям;
- таємним ключем володіють лише законні користувачі криптосистеми, які його не розголошують;
- забезпечується цілісність обчислювача, тобто засіб шифрування захищений від зміни чи модифікування;
- забезпечується цілісність процесу шифрування, тобто в процесі шифрування дані змінюються у відповідності алгоритму перетворення і не можуть бути змінені цілеспрямовано за допомогою яких-небудь зовнішніх впливів ні на одному із кроків перетворення.

Порушення хоча б однієї з перших чотирьох умов створює умови для проведення тривіальної атаки на криптосистему з метою отримання доступу до змісту зашифрованої інформації. Порушення ж останньої умови також може використовуватись для нападу на шифр, проте у цьому випадку необхідні більш складні методи атаки, трудомісткість яких залежить від конкретних алгоритмів

перетворення та їх конкретних програмної, апаратної чи програмно-апаратної реалізації.

Усі описані вище умови задовольняє шифр AES, який є симетричним блоковим шифром. Як стверджує Національне Бюро Стандартів США, алгоритм AES має такі властивості [11]:

– простота в розумінні;

– висока ступінь складності, яка робить його розкриття дорожчим отриманого при цьому прибутку;

– метод захисту базується на ключах і не залежить від “таємності” алгоритму;

– економічний в реалізації та ефективний у швидкодії.

Важливою характеристикою цього алгоритму є його гнучкість при реалізації та використанні в різноманітних застосуваннях обробки даних.

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

2 ПРОЕКТУВАННЯ ОПЕРАЦІЇ ШИФРУВАННЯ КРИПТОАЛГОРИТМУ AES

2.1 Розробка вхідного та вихідного блоків FIFO

Для розробки структурної схеми проекту – вузла реалізації операції шифрування на основі криптоалгоритму AES, було обрано мову VHDL (veryhigh hardware description language), яка є мовою опису апаратних ресурсів і однією із найпотужніших мов, що використовуються у сучасних засобах проектування цифрових схем (системи автоматизованого проектування, САПР) [12].

В загальному випадку проектований вузол має складатися із декількох базових блоків, які будуть виконувати свою частину: вхідний блок FIFO; блок керування даними; блок шифрування AES Cipher; вихідний блок FIFO. VHDL код наведено в додатку А.

Вхідний блок FIFO буферизує запис операцій з testbench і перетворює дані шириною від 64 біт до 128 біт, що потрібні для подальшого модуля шифрування. Даний блок містить контекст та інціалізуючі вектори. Вихідний блок FIFO буферизує результати шифрування, які надходять від блоку шифрування, і перетворює 128 бітний потік даних на 64 бітний для внутрішнього використання, зокрема, для testbench.

Вхідний блок FIFO здійснює три циклічних буфери з 8-ма розташуваннями. Число розташувань конфігурується в залежності від швидкості вибірки testbench’а і механізму шифрування. Двоє з буферів відображають 64 розряди, а при використанні третього ще 16 розрядів. Всі буфери разом використовують операцію зчитування і записують вказівник, щоб гарантувати їхню синхронізацію.

Дані, які будуть шифруватися записуються в FIFO 64-розрядну транзакцію. Тому для шифрування AES 128-розрядні дані приходять в блок FIFO. Варто зазначити, що інформація про ініціалізуючий вектор та текст

мають бути також завантажені в цей блок. Після використання режиму шифрування всі поля повинні бути скинуті і мати значення 0.

При кожному записі на testbench вказівник запису збільшується на одиницю. Для того, щоб захистити FIFO від переповнення, вказівник запису порівнюється з вказівником зчитування.

Зовнішній вигляд компоненти вхідного блоку «Input FIFO» зображено на рисунку 2.1.

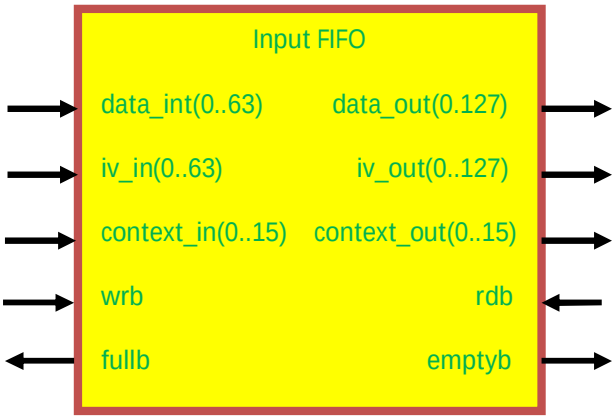

Рисунок 2.1 — Компонента «Input FIFO»

Дана компонента має наступні входи та виходи:

- data_in(0..63) — вхідні дані, що подаються на модуль Cipher для шифрування. Одна частина 128-розрядного блоку AES задіяна в кожному циклі;
- iv_in(0..63) — забезпечує ввід ініціалізуючого вектора на Cipher в режимі шифрування CBC;
- context_in(0..15) — забезпечує Cipher модуль знанням яким чином потрібно обробляти асоційовані дані. Шифрування context_in полягає в наступному: біт 0: «1»=Start of Packet. «0»=середина або кінець пакету даних; біт «1»: «1»=шифрування, «0»=дешифрування; біти 2:3: вказують режим шифрування («01»=ECB режим, «10»=CBC режим). Всі інші значення

ігноруються. Біти з [4:15] вказують індекс використаного ключа;

- wrb — вказує дані на вході data_input, iv_in, context_in, що запишуться у вхідному блоці input FIFO;
- fullb — вказує статус входу FIFO. «0» вказує, що вхід FIFO повний, а «1» — вхід input FIFO має місце для наступної транзакції;
- key_address(0..4) — вихід, що забезпечує адресний простір ключів наступного необхідного ключа;
- read_mem — вихід, що забезпечує читання для асоційованої ключової пам'яті;
- key_in(0..127) — вхід, де подається ключ для шифрування для кожного раунду;
- data_outputt(0..63) — результати шифрування даних з інформацією контексту;
- rdb — вказує, що testbench готовий прийняти нові дані з виходу FIFO;
- emptyb — вказує статус виходу FIFO: «0» вказує, що він пустий (тому немає необхідності продовжувати запит), і «1» вказує, що вихід FIFO має по крайній мірі ще один біт 128-розрядної транзакції перед входженням в emptyb;
- clock — дозволяє синхронізувати сигнал на всіх часових елементах проекту;
- resetb — здійснює перевантаження синхронізації всіх часових елементів в проекті.

Функціональну структуру даного проекту зображено в додатку Б.

Розглянемо вихідний блок FIFO. Він здійснює один циклічний буфер з 8. Число розташувань конфігурується в залежності від швидкості вибірки testbench і циклу шифрування. Буфер є 64-бітним. Зашифровані дані представлені в FIFO як 128-бітне слово. На кожній транзакції, яка ініційована блоком керування, два розташування буде заповнено. Перше розташування відповідає бітам від 0 до 63 блоку зашифрованих даних і другому розташуванню відповідають біти ві 64 до 127.

При кожному записі від блоку керування вказівник запису збільшується на дві позиції і внутрішній вміст лічильника збільшується. Для захисту FIFO від переповнення потрібно вказівник запису порівнювати з вирахуваним показником. Якщо вказівник запису дорівнює вирахуваному показнику, то контент лічильника вказує, що FIFO має останні 6 одиниць даних, на виході FIFO буде сигнал попередження для блоку керування. Блок керування не повинен вписувати нові дані в FIFO поки fullb сигнал не буде знято.

На рисунку 2.2 зображено блок Output FIFO.

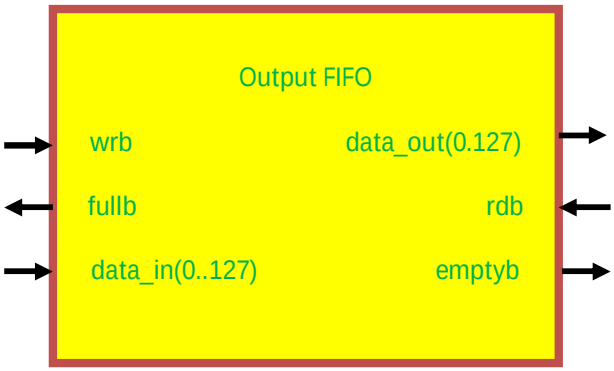

Рисунок 2.2 — Компонента Output FIFO

Testbench контролює зчитування з Output FIFO. При кожному зчитуванні з тестера прочитується один буфер. Таким чином, для компоненти AES дві транзакції повинні зчитуватися з самого початку повністю 128-бітного блоку AES з Output FIFO. Вказівник зчитування збільшується на одиницю і порівнюється з вказівником запису, якщо новий вказівник зчитування знаходиться в межах одного розташування з вказівником запису.

2.2 Розробка компоненти керування даними

Компонента керування даними здійснює керування 27 станами послідовності операцій шифру (рисунок 2.3). Блок керування даними здійснює

керування всіма операціями шифрування, включаючи зчитування даних з вхідного блоку FIFO, які мають бути зашифровані, і запис шифротексту на вихідний блок FIFO. Цей блок реалізує стани схеми, що дозволяє регулювати роботу двигуна шифрування для кожного такту процесу шифрування.

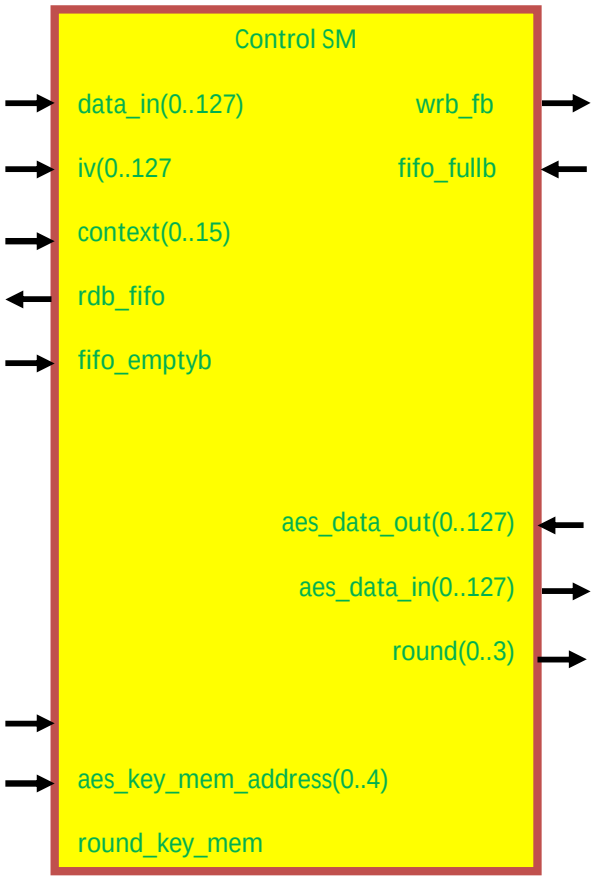

Рисунок 2.3 — Компонента керування даними

На рисунку 2.4 зображено діаграму станів для блоку керування даними. Вона включає наступні операції:

- ініціалізуючий стан блоку є IDLE і так залишається поки на порт FIFO_EMPTYB не надійде сигнал «1» з вхідного блоку FIFO, машинні стани переходять в Get_Context стани;
- в стані Get_Context машина зчитує біти context для визначення, який просес буде запущено. Якщо біти 2 і 3 контекстного вводу є рівними «01» стан машини змінюється на ECB_ENCRYPT_1 стан. Якщо біти 2 і 3 є рівними «10»,

тоди стан машини змінюється на CBC_ENCRYPT_1 стан;

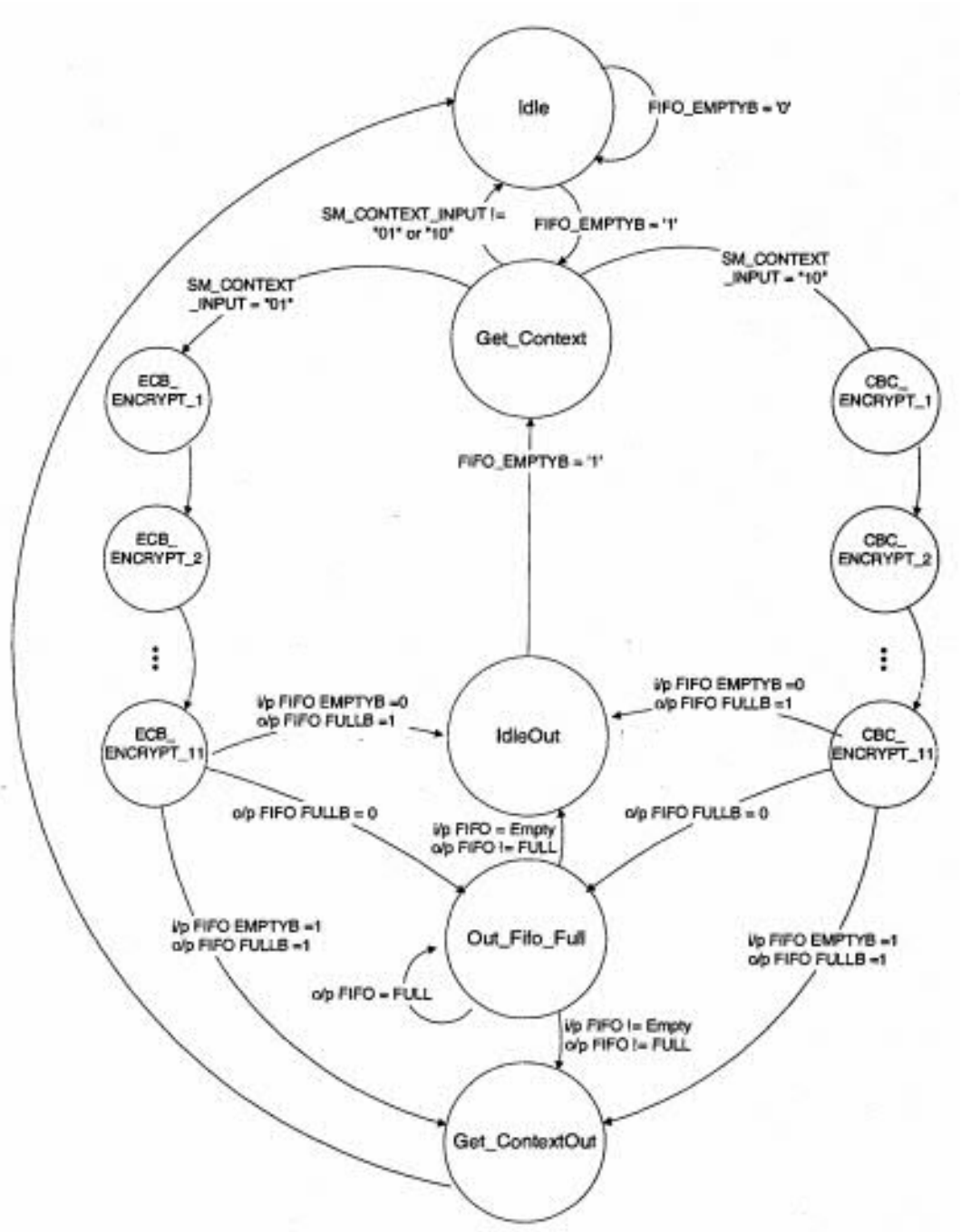

Рисунок 2.4 — Діаграма станів блоку керування даними

- в ECB_ENCRYPT_1 стані блок направляє дані для шифрування блоком AES Cipher. На наступних часових циклах стани блоку встановлені від 2 до 10;
- в ECB_ENCRYPT_1 стані дані повністю шифруються, пройшовши всі

раунди. В цьому стані контролер перевіряє стан прапорця FIFO_EMPTYB на вході Input FIFO та FIFO_FULLB на компоненті Output FIFO. Якщо прапорець FIFO_FULLB встановлено «0», тоді стан блоку змінюється на стан Out_FIFO_Full. Таким чином, якщо обидва прапорці FIFO_EMPTYB і FIFO_FULLB встановлено в «1», тоді стан блоку змінюється на Get_ContextOut стан, оскільки з'явилися нові дані для шифрування і вихід FIFO є пустим. Якщо прапорець FIFO_EMPTYB встановлено «0» і FIFO_FULLB є «1», тоді стан блоку змінюється на IDLEOUT стан протягом того часу, поки немає даних для шифрування;

- функціонування CBC_ENCRYPT_1 станом через CBC_ENCRYPT_11 стан блоку подібно до стану ECB_ENCRYPT_X стану. Першопочаткова різниця в тому, що CBC_ENCRYPT_1 вхідні дані будуть шифруватися шляхом операції xor з ініціалізуючим вектором IV, якщо це є перший блок пакету. Якщо ні, тоді вхідні дані шифруються шляхом операції xor з попереднім результатом шифрування;
- в Get_ContextOut стані блок повертає наступні дані: ініціалізуючий вектор і контекстну інформацію, що буде зашифрована наступним блоком даних;
- в Out_FIFO_Full блок перевіряє налаштування прапорця FIFO_FULLB, який визначає доступний простір в Output FIFO. Коли простір стає доступним, зашифровані дані записуються в FIFO і стан блоку змінюється на IDLEOUT або Get_ContextOut стан, який залежить від того чи є дані для шифрування на вході FIFO;
- в стані Get_ContextOut блок керування записує зашифровані дані в Output FIFO і зчитує дані, ініціалізуючий вектор і контекст для наступного блоку шифрування даних.

2.3 Розробка компоненти шифрування AES Cipher

Блок AES Cipher реалізує два підходи шифрування даних за допомогою симетричного криптоалгоритму AES. Перша реалізація базується на використанні S-Box. Спрощена схема роботи блоку AES Cipher за цією реалізацією наведена в додатку В.

AES — це симетричний блоковий шифр, що оперує блоками даних розміром 128 і довжиною ключа 128, 192 чи 256 біт - назва стандарту відповідно "AES-128", "AES-192", і "AES-256".

Введемо наступні позначення:

Nb - число 32-бітних слів що містяться у вхідному блоці, Nb = 4;

Nk - число 32-бітних слів, що містяться в ключі шифрування, Nk = 4,6,8;

Vr - число раундів шифрування, як функція від Nb і Nk , Vr = 10,12,14.

Вхідні (input), проміжні (state) і вихідні (output) результати перетворень, Які виконуються у рамках криптоалгоритму, називаються станами (State). Стан можна представити у вигляді матриці 4 x Nb, елементами якої є байти (чотири рядки по Nb байт) у порядку $S_{00},S_{10},S_{20},S_{30},S_{01},S_{ll},S_{21},S_{;31},$. Рисунок 2.5 демонструє таке представлення, що носить назву архітектури «Квадрат» [15].

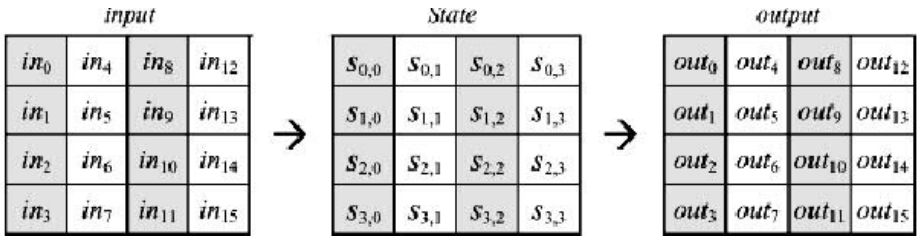

Рисунок 2.5 — Приклад представлення блоку у вигляді матриці 4x Nb

На старті процесів шифрування і дешифрування масив вхідних даних $in_{0,}$ $in_{1,}$ $in_{2,}$ ....in_{15} перетвориться в масив State за правилом s[r,c] = in[r + 4c], де 0<r<4 і 0<r<4 і 0<=c<Nb. Наприкінці дії алгоритму виконується зворотне

перетворення out[r+4c]=s[r,c], де 0<r<4 і 0<c<Nb - вихідні дані виходять з байтів стану в тому ж порядку.

Чотири байти в кожнім стовпці стану являють собою 32-бітне слово, якщо м - номер рядка в стані, то одночасно він є індексом кожного байта в цьому слові. Отже стан можна представити як одномірний масив 32-бітних слів $w_O,...,w_{NB}$. , де номер стовпця стану с є індекс у цьому масиві.

Ключ шифрування також як і масив State представляється у виді прямокутного масиву з чотирма рядками. Число стовпців цього масиву дорівнює Nk.

Для алгоритму AES число раундів Nr визначається на старті в залежності від значення Nk (рисунок 2.6).

	Nk	*Nb*	*Nr*
AES – 128	4	4	10
AES – 192	6	4	12
AES - 256	8	4	14

Рисунок 2.6 — Залежність значення Nr від Nkn Nb

Розглянемо функції шифрування та дешифрування алгоритму AES. Введемо наступні позначення:

- SubBytes — заміна байтів- побайтова нелінійна підстановка в State-блоках (S-Box) з використанням фіксованої таблиці замін розмірністю 8x256 (affain map table);

- ShiftRows() — зрушення рядків - циклічне зрушення рядків масиву State на різну кількість байт;

- MixColumns — перемішування стовпців - множення стовпців стану, розглянутих як багаточлени над GF(2^8);

- AddRoundKey — додавання з раундовим ключем - порозрядне XOR вмісту State з поточним фрагментом розгорнутого ключа.

Після заповнення масиву State елементами вхідних даних до нього

застосовується перетворення AddRoundKey далі, у залежності від величини Nk , масив State піддається раундовій трансформації 10, 12 чи 14 разів, причому фінальний раунд є трохи скороченим - у ньому відсутнє перетворення MixColumns(). Вихідними даними описаної послідовності операцій є шифротекст - результат дії функції шифрування AES.

У специфікації алгоритму AES пропонуються два види реалізацій функції дешифрування відмінних один від одного послідовністю зворотних перетворень функції шифрування і послідовністю планування ключів.

Введемо наступні позначення:

InvSubBytes - зворотна SubBytes() заміна байтів- побайтова нелінійна підстановка в State-блоках з використанням фіксованої таблиці замін розмірністю 8x256 (inverse affain map);

InvShiftRows - зворотне зрушення рядків ShiftRows()- циклічне зрушення рядків масиву State на різну кількість байт;

InvMixColumns - відновлення значень стовпців - множення стовпців стану, розглянутих як багаточлени над GF(2^8).

Якщо замість SubBytes( ), ShiftRows( ), MixColumns( ) і AddRoundKey( ) у зворотній послідовності виконати інверсні їм перетворення, можна побудувати функцію оберненого дешифрування. При цьому порядок використання раундових ключів є оберненим стосовно того, що використовується при шифруванні.

Шифрування AES Cipher, що використовує T-box підхід, подано в додатку Г.

Зовнішній вигляд компоненти AES Cipher зображено на рисунку 2.7.

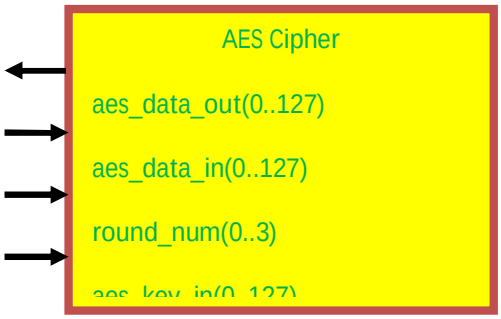

Рисунок 2.7 — Компонента AES Cipher

Дана компонента має наступні виходи:

- на виході «aes_data_out_round0» обчислює операцію xor для заданого ключа і вхідного тексту (для 0 раунду);
- на виході «aes_data_out_final» обчислює операцію xor для заданого ключа і результат SubBytes (для 10 раунду);
- на виході «aes_data_out_mid» обчислює операцію xor для заданого ключа і результат операції MixColumns (для раундів з 1 по 9);
- на виході «aes_data_out_last» забезпечується зареєстрована версія aes_data_out_final.

2.4 Функціональна симуляція проекту

Використовуючи вбудований у пакет Active HDL засіб Test Bench проведено функціональну верифікацію проекту – робота симетричного блокового алгоритму шифрування AES.

Test Bench (випробувальний стенд) призначений для перевірки правильності роботи проектованого пристрою (компоненти). Тестована компонента перевіряється за допомогою стимулів (вхідним сигналам та шинам даних присвоюється необхідне для роботи пристрою значення) і при цьому контролюється реакція даної компоненти шляхом нагляду та дослідження (зондування) вихідних сигналів.

Саме пакет Active HDL дозволяє в деякій мірі автоматизувати розробку випробувального стенду.

Для того, щоб використати згадану автоматизацію необхідно виконати наступну послідовність дій:

- в меню Tools вибрати команду „Generate Test Bench…”;

- необхідно вказати до якого інтерфейсного та архітектурного описів буде генеруватися даний випробувальний стенд та вибрати тип випробувального стенду (Single Process або Wawes Based);
- можна добавити тестові вектори із зовнішнього файлу (вказати шлях та ім’я даного файлу);
- вибрати ім’я інтерфейсного опису даного випробувального стенду та ім’я папки, де буде зберігатися даний випробувальний стенд.

Саме автоматично згенерований випробувальний стенд складається з наступних частин:

- Add your library and packages declaration – декларація бібліотек та пакетів, необхідних для даного проекту;
- Component declaration of the tested unit – декларація компонент саме тої компоненти, що тестується;
- Stimulus signals - signals mapped to the input and inout ports of tested entity – визначення сигналів, до яких можна подавати стимули (сигнали тестованої компоненти);
- Observed signals - signals mapped to the output ports of tested entity – сигнали, за якими необхідно наглядати (вихідні сигнали компоненти);
- Add your code – частина випробувального стенду, куди можна ввести додатковий код (звичайно, якщо в цьому є необхідність);
- Unit Under Test port map – визначення портів тестованої компоненти;
- Add your stimulus – місце для введення стимулів на порти компоненти, що досліджується.

В додатку Д зображено з’єднання testbench з модулем “aes Cipher ”. Testbench виконує три основні операції:

- операція вхідного інтерфейсу;
- операція вихідного інтерфейсу;
- операція інтерфейсу занесення ключа в пам’ять.

Текст випробувального стенду до компоненти “aes Cipher ” наведено у

додатку Е.

Для успішного дослідження тестованої компоненти підключено наступні бібліотеки (вбудовані у пакет Active HDL):

- ieee.std_logic_textio.all;
- std.textio.all;

Підключення саме цих бібліотек дозволить користуватися функціями читання та запису у файл, а саме використано для запису отриманих поточних результатів роботи (отриманих на вихідних сигналах та шинах) у текстовий файл під час симуляції проекту.

Також додатково введено змінну „clock”, що має тип даних „time” і рівна 10 ns. Змінна введена для того, щоб користувачу спростити процедуру вчасного визначення подачі стимулів на порти тестованої компоненти.

Як було згадано раніше, мова VHDL дозволяє описувати процеси, що виконуються паралельно (у часі). Така особливість була використана при написанні випробувального стенду і якості подачі стимулів на входи досліджуваної компоненти.

Були розписані наступні процеси:

- clk_gen – визначений для подачі стимулів на сигнал „CLK” – тактової частоти;
- rst_pr – для визначення сигналу „Reset”;
- mode_pr – для визначення команд на вхідну шину керування роботою компоненти;
- data – для подачі даних на шину даних досліджуваної компоненти;
- додатковий процес (без імені), що використовується для запису отриманих результатів на виходах досліджуваної компоненти у файл "out.txt".

Із часових діаграм, отриманих у пакеті Active HDL при симуляції роботи компоненти, було обрано час завантаження і частина часу роботи (генерування ПВЧ) компоненти.

3 СИМУЛЯЦІЯ ТА ВЕРИФІКАЦІЯ ПРОЕКТУ

3.1 Обґрунтування вибору елементної бази

Вибір елементної бази для реалізації проекту суттєво впливає на основні характеристики проектованого пристрою, тому етап визначення елементної бази для імплементації проекту є важливим.

Серед можливих видів елементної бази програмовані логічні інтегральні схеми (ПЛІС) займають чільне місце, оскільки дозволяють отримувати ефективну реалізацію проекту з точки зору продуктивності, апаратної сумісності, споживання електроенергії та ін.

Одним із потужних виробників сучасних ПЛІС є фірма Altera. Сьогодні ПЛІС високого ступеня інтеграції разом із оптимізованими ядрами інтелектуальної власності (intellectual property core – IP Core) дозволяють виробникам електронних систем реалізувати ту ж саму функціональність на одному кристалі мікросхеми, що раніше потребувала використання друкованої плати з десятками інтегральних схем меншого ступеня інтеграції. Від проектів звичайної нескладної логіки і до цілих систем, реалізованих на одному кристалі, використання ПЛІС гарантує серйозні переваги такої апаратури на ринку.

Фірма Altera пропонує широкий діапазон сімейств сучасних ПЛІС для забезпечення потреб розв’язання актуальних задач народного господарства у вигляді апаратних засобів: Stratix™, APEX™ II, APEX, HardCopy™, Mercury™, ACEX®, MAX®, FLEX®, а також Excalibur™ .

ПЛІС фірми Altera забезпечують максимальну швидкодію сучасних високо-інтегрованих систем, побудованих на мікросхемах. Продуктивність такої системи сягає до 450 МГц, частота синхронізації сягає до 1.25ГГц, а затримка між виводами є меншою за 3.5 ns.

ПЛІС сімейства Stratix є одними із найновіших пристроїв, і являють

собою ідеальний вибір для реалізації комплексних високопродуктивних систем високого ступеня інтеграції. Ці ПЛІС базуються на новій високопродуктивній архітектурі, що дозволяє спростити синтез складної системи, зменшити загальний час, затрачений на проектування, та забезпечує серйозні ринкові переваги апаратним засобам.

Нова архітектура ПЛІС сімейства Stratix в поєднанні з програмними засобами автоматизованого проектування Altera Quartus® II дозволяє реалізувати методологію модульного проектування, доступну оптимізацію за продктивністю проектних модулів та гнучку інтеграцію системних функцій в проектованому пристрої. Такі ПЛІС є оптимізовані для задоволення вимог щодо продуктивності, об'ємів пам'яті та зменшення часу проектування та дозволяють будувати системи з високим рівним смуги пропускання корисних сигналів.

ПЛІС сімейства APEX II також включають високопродуктивні програмовані пристрої з широкою смугою пропускання. Такі ПЛІС можуть бути використані для побудови широкого діапазону апаратних засобів та можуть підтримувати цілу низку різноманітних протоколів: RapidIO™, POS-PHY L4, Flexbus, PCI-X, UTOPIA Level IV, CSIX . ПЛІС сімейства APEX II забезпечують підтримку для об'єднання та спільного використання інтерфейсів високошвидкісної пам'яті, а також стандарти високошвидкісних портів, включаючи: True-LVDS™, Flexible-LVDS™, LVPECL, PCML, HyperTransport™.

ПЛІС сімейства Excalibur включають ядра програмного забезпечення та високошвидкісні пристрої , що базуються на технології ARM. Вони інтегрують складні процесори і дозволяють розробнику легко здійснювати повно інтегровані рішення при проектуванні системи на програмованій інтегральній схемі.

ПЛІС сімейства Mercury вміщають засоби передачі даних з підтримкою швидкості до 1.25 Гбіт за секунду на один канал передачі даних.

ПЛІС сімейства HardCopy забезпечують економічну ефективність, малий ризик та менший час проектування в порівнянні до мікросхем, що виготовляються на замовлення (ASIC). Внаслідок комбінації специфічного проектування в напівпровіднику та автоматизованого процесу конверсії, Altera дозволяє розробникам легко переносити проекти з програмованих інтегральних схем на низько вартісні імплементації на замовлення. Використання конкурентноздатних ПЛІС сімейства HardCopy проектні групи можуть отримати переваги більшої швидкості виконання проектних робіт в порівнянні з ASIC.

ПЛІС фірми Altera виготовленні за сучасними технологіями CMOS процесів. Останні ПЛІС, що виконанні за технологією мідної металізації усіх шарів мікросхеми, дозволяють досягнути рівня технологічного процесу 0.13 мікрон, забезпечуючи високопродуктивні рішення. Цей процес підвищує продуктивність та зменшує економічні затрати на процес виробництва, дозволяючи виробляти високоефективні ПЛІС. Крім того, такий підхід дозволяє суттєво зменшити споживану потужність при тому ж рівні інтеграції [13].

ПЛІС фірми Altera доступні у широкому діапазоні економних рішень упаковки в корпуси, включаючи тонкий пластиковий квадратний корпус (thin quad flat pack TQFP), інноваційні та ефективні за площею корпуси з кульковими контактами розміром 1.0 мм (FineLine BGA) та 0.8 мм (Ultra FineLine BGA).

Щоб створювати проекти на основі ПЛІС САПР фірми Altera Quartus II забезпечує зручне графічне середовище, здатне підтримувати проекти типу “система на програмованій інтегральній схемі” (SOPC). Цей програмний засіб дозволяє отримати високопродуктивні проектні рішення протягом мінімального часу затрат на проектування. САПР Quartus II підтримує створення проектів у таких типах сімейств ПЛІС: Stratix, APEX II, APEX, Mercury, ARM-based Excalibur, FLEX 10K, ACEX 1K, FLEX 6000, MAX 7000B, MAX 7000A, та MAX 3000A. Для збільшення продуктивності проектувальних робіт у САПР фірми Altera Quartus II наявні великі бібліотеки мега-функцій та

протестованих блоків IP Core, оптимізованих для ефективної реалізації проекту на певній архітектурі ПЛІС.

ПЛІС сімейства ACEX базуються на використанні спеціальних програмованих блоків LUT і дозволяють отримати високий рівень продуктивності проекту за умови невеликої вартості та апаратних затрат. Пристрої такого типу є ідеальними для використання в засобах телекомунікацій: кабельні модеми, DSL модеми, економічні розгалужувачі та роутери. ПЛІС сімейства ACEX мають ті ж самі переваги по економічній ефективності, що й ASIC мікросхеми, а також гнучкість перепрограмування та невеликий час проектувальних робіт. Вони також дозволяють здійснювати реконфігурацію ПЛІС без потреби розмонтування схеми (in-circuit reconfigurability – ICR), що дозволяє розробнику швидко та економічно здійснювати поновлення нових версій проекту. Таким чином проекти можуть бути спроектовані, перевірені, виготовленні та поновлені з мінімальними труднощами та часовими затратами.

ПЛІС сімейства ACEX 1K , що базуються на використанні новітніх, економних 2.5 В SRAM процесах, можуть мати від 576 до 4992 логічних елементів. Працюючи від напруги живлення 2.5 В такі пристрої є повністю сумісними з 64-бітним інтерфейсом PCI 66 МГц. Затрати на розробку ASIC, включаючи затрати інженерних ресурсів та засобів розробки є великими. Традиційний процес проектування ASIC вимагає підвищених інженерних ресурсів для проведення верифікації на програмному та фізичному рівні. Ці обидві задачі потребують інтенсивних затрат часу та ресурсів. ПЛІС сімейства ACEX 1K забезпечують зниження потреб до інженерних ресурсів та пришвидшують сам процес проектування.

У таблиці 3.1 наведено основні параметри деяких типів мікросхем сімейства ACEX 1K . В таблиці вибрано такі параметри: кількість типових логічних елементів; максимальна кількість логічних елементів у системі; кількість логічних елементів; кількість вбудованих логічних блоків; загальна

кількість біт RAM (Random Access Memory); максимальна кількість вхідних/вихідних виводів.

Логічний елемент ПЛІС сімейства ACEX 1K може працювати в таких режимах роботи:

- нормальний;
- арифметичний;
- лічильник прямого/зворотного рахунку;
- лічильник з очищенням.

Таблиця 3.1 – Основні параметри мікросхем сімейства ACEX 1K

ПЛІС	Кількість типових логічних елементів	Максимальна кількість логічних елементів у системі	Кількість логічних елементів	Кількість вбудованих логічних блоків	Загальна кількість біт RAM	Максимальна кількість вхідних/вихідних виводів
EP1K10	10 000	56 000	576	3	12 288	136
EP1K30	30 000	119 000	1 728	6	24 576	171
EP1K50	50 000	199 000	2 880	10	40 960	249
EP1K100	100 000	257 000	4 992	12	49 152	333

3.2 Реалізація проекту у вибраній елементній базі

Застосування в інженерній практиці методів автоматизації проектування дозволяє перейти від традиційного макетування проектованої апаратури до її моделювання за допомогою ЕОМ. Більше того, за допомогою ЕОМ можна здійснити цикл наскрізного проектування , що включає:

- синтез структури та принципової схеми пристрою;
- аналіз його характеристик в різноманітних режимах роботи із врахуванням розкиду параметрів компонентів та наявності дестабілізуючих факторів, а також параметричну оптимізацію;
- синтез топології, що включає розміщення елементів на платі чи кристалі та розводку міжз`єднань;
- верифікацію топології;
- виготовлення конструкторської документації.

Різноманітність конструктивно-технологічних методів виготовлення ВІС обумовлена прагненням не тільки поліпшити їхні техніко-економічні показники, але і досягти загальну мету: мінімізувати тривалість процесу проектування та забезпечити проектування ВІС високої складності; а також підвищити якість проектування ВІС (в основному безпомилковість).

Тривалість процесу проектування значно скорочується при використанні обчислювальної техніки, бібліотек компонентів, стандартизації програм і т.п. А це, у свою чергу, скорочує терміни проектування і виготовлення апаратних засобів сучасних комп’ютерних систем.

Для реалізації проекту у заданій елементній базі можуть бути використані різноманітні програмні продукти імплементації VHDL-опису у мікросхеми: MAX+PLUS II, QUARTUS II, Sinplify, Leonardo та інші.

QUARTUS II являє собою повно-інтегрований, архітектурно-незалежний пакет програмних засобів для проектування логічних схем на усіх сімействах

ПЛІС фірми Altera. У САПР QUARTUS II проект являє собою повну множину власне проектних файлів, файлів присвоєння, файлів симуляції, системних налаштувань, та ієрархічної інформації про проект.

QUARTUS II підтримує сумісність з САПР Active HDL – шляхом переносу проекту через описи апаратних засобів на мові VHDL. Тому для початку роботи у QUARTUS II слід перенести усі файли ресурсів проекту з розширенням *.VHD в окремий каталог для проведення імплементації у вибраній ПЛІС сімейства ACEX 1K. Такими файлами опису проекту є наступні:

- Input_FIFO.vhd;
- Output_FIFO.vhd;
- Control_SM.vhd;
- AES_Cipher.

Процес імплементації проекту у системі QUARTUS II виконується компілятором, що виконує такі етапи:

- побудова бази даних проекту за допомогою підсистеми Database Builder;
- проведення логічного синтезу проекту за допомогою підсистеми Logic Synthesizer;
- проведення розміщення проекту в полі компонувального простору мікросхеми за допомогою підсистеми Fitter;
- створення програмних файлів для програмування ПЛІС за допомогою підсистеми Assembler;
- проведення часового аналізу проекту за допомогою підсистеми Timing Analyser.

Компілятор є основним ядром системи QUARTUS II, що забезпечує потужний процес проектування, який користувач може налаштувати таким чином, щоби отримати найефективнішу реалізацію імплементації проекту у напівпровіднику. Автоматична локалізація помилок та розширена документація дозволяють суттєво спростити процес проектування.

Компілятор системи QUARTUS II вміщає низку модулів? що здійснюють перевірку проекту на помилки, синтез логічної структури, розміщення в полі компонувального простору та генерування вихідних файлів для проведення симуляції, часового аналізу та програмування кінцевого пристрою. Спочатку компілятор витягає інформацію про ієрархічні зв'язки між проектними файлами та перевіряє проект на грубі помилки. Після цього він створює організаційну структур проекту і об'єднує усі проектні файли у базу даних, що може бути ефективно опрацьована.

Користувач може налаштувати компілятор для ефективного здійснення усіх процесів, щоб підвищити швидкість проектування та оптимізувати використання ресурсів пристрою. Результати імплементації можуть переглядатися під час та після компіляції у вигляді графічних чи текстових файлів.

За допомогою системи QUARTUS II було проведено процедуру розміщення елементів у заданому елементному базисі. На рисунку 3.1 зображено результати розміщення проекту в полі компонувального простору мікросхеми EP1K10FC256-1 сімейства ACEX 1K.

Аналіз результатів показує, що проект реалізовано майже рівномірно у полі компонувального простору мікросхеми. Задіяні блоки розміщені в сусідніх сегментах мікросхеми, що дозволить зменшити максимальну довжину шляху проходження сигналів. Це в свою чергу дозволить підвищити продуктивність проектованого пристрою.

У таблиці 3.2 наведено узагальнені результати використання ресурсів мікросхеми при реалізації проекту. Аналіз результатів таблиці 3.2 показує, що використання логічних елементів є надзвичайно низьким, а використання виводів мікросхеми є майже повним.

Таблиця 3.2 – Узагальнені результати використання ресурсів мікросхеми

Параметр	Значення
Пристрій, на якому здійснювалася компіляція	EP1K30QC208-1
Загальна кількість логічних елементів	134 / 1728 (7 %)
Загальна кількість використаних виводів	139 / 147 (94 %)
Загальна кількість використаних біт	0 / 24 576 (0 %)

У таблиці 3.3 наведено узагальнені результати використання ресурсів внутрішньої комутації мікросхеми при реалізації проекту.

Таблиця 3.3 – Результати використання ресурсів внутрішньої комутації мікросхеми

Стрічка в ПЛІС	Використані внутрішні з’єднання	Використані внутрішні з’єднання правого сегменту	Використані внутрішні з’єднання лівого сегменту
A	5 / 144 (3 %)	4 / 72 (5 %)	11 / 72 (15 %)
B	8 / 144 (5 %)	2 / 72 (2 %)	3 / 72 (4 %)
C	7 / 144 (4 %)	10 / 72 (13 %)	5 / 72 (6 %)
D	6 / 144 (4 %)	8 / 72 (11 %)	9 / 72 (12 %)
E	6 / 144 (4 %)	0 / 72 (0 %)	9 / 72 (12 %)
F	3 / 144 (2 %)	13 / 72 (18 %)	7 / 72 (9 %)
Разом	35 / 864 (4 %)	37 / 432 (8 %)	44 / 432 (10 %)

Аналіз таблиці 3.3 показує, що лінії внутрішньої комутації використані у невеликому обсязі, що говорить про можливість використання вільних ресурсів мікросхеми для розміщення інших проектних модулів.

3.3 Дослідження характеристик шифрування симетричного криптоалгоритму AES

Для оцінки продуктивності процесора шифрування згідно з алгоритмом DES необхідно враховувати як тактову частоту його роботи, так і кількість тактів імпульсів синхронізації, які необхідно затратити на обробку одного блоку даних. Крім того, для переводу виміру характеристики продуктивності у загально прийнятий вигляд біт/с. Для цього необхідно враховувати розмір блоку даних, який обробляється. Тому для оцінки продуктивності процесора шифрування згідно з алгоритмом AES буде проводитися згідно з формулою:

$$P = n*f / k \qquad (3.1)$$

де n – розмір блоку даних алгоритму шифрування AES;

f – тактова частота, на якій працює процесор, розміщений у ПЛІС;

k – кількість тактів синхронізації, необхідних для прийому одного блоку даних.

Зауважимо, що оскільки зашифрування і розшифрування даних проходить за однакову кількість тактів, то продуктивність зашифрування і розшифрування буде однаковою. Для алгоритму шифрування AES n = 128. Процесор дозволяє завантажувати дані кожен такт синхронізації, тому k = 1. Підставивши отримані значення тактових частот після синтезу процесора у ПЛІС у вираз (3.1), отримаємо характеристики продуктивності роботи

процесорів, реалізованих на різних кристалах (таблиця 3.4).

Таблиця 3.4 – Продуктивність синтезованих процесорів шифрування за алгоритмом AES

ПЛІС	Тактова частота, МГц	Продуктивність шифрування, Мбіт/с
Xilinx Virtex XC2V250-4	120	7680
Xilinx Spartan 2e XC2S200E-7	98	6272

Отримані значення продуктивності розроблених процесорів свідчать, що ці процесори можна використовувати як складові систем захисту інформації виконаної як на ПЛІС, так і на основі замовлених інтегральних схем. Продуктивність процесорів є достатньою для обробки потоку даних, інтенсивність якого є до 7 Гбіт/с. Такі значення продуктивності є достатніми для обробки даних, які циркулюють у комп'ютерних мережах із швидкостями передачі даних від 1 до 5 Гбіт/с.

Кількість використаних логічних комірок у кожному кристалі дозволяє розмістити додаткові вузли для створення багатофункціонального процесора шифрування. Зокрема, розроблений процесор можна доповнити вузлами комутування даних з його входу і виходу, що дозволить створити орієнтований на декілька режимів і операцій процесор шифрування.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ

В цьому розділі бакалаврської роботи (БР) проводиться економічне обґрунтування доцільності розробки програмного забезпечення. Зокрема, здійснюється розрахунок витрат на розробку програмного забезпечення, експлуатаційних витрат, ціни споживання програмного забезпечення. В заключній частині визначаються показники економічної ефективності нового програмного продукту, обґрунтовуються відповідні висновки.

4.1 Розрахунок витрат на розробку програмного забезпечення

Витрати на розробку і впровадження програмних засобів (К) включають:

$$K = K_1, + K_2 \qquad (4.1)$$

де K_1 - витрати на розробку програмних засобів, грн;

K_2 - витрати на відлагодження і дослідну експлуатацію програми рішення задачі на комп’ютері, грн.

Витрати на розробку програмних засобів включають:

- витрати на оплату праці розробників ($B_{ОП}$);
- витрати на відрахування у спеціальні державні фонди ($Вф$);
- витрати на покупні вироби ($Пв$);
- витрати на придбання спецобладнання для проведення експериментальних робіт ($Об$);
- накладні витрати ($Н$);
- інші витрати ($Ів$).

4.1.1 Розрахунок витрат на оплату праці

Витрати на оплату праці включають заробітну плату (ЗП) всіх категорій працівників, безпосередньо зайнятих на всіх етапах проектування. Розмір ЗП обчислюється на основі трудомісткості відповідних робіт у людино-днях та середньої ЗП відповідних категорій працівників.

У розробці програмного забезпечення задіяні наступні спеціалісти - розробники, а саме – керівник проекту, студент-дипломник, консультант техніко-економічного розділу.

Таблиця 4.1 - Вихідні дані для розрахунку витрат на оплату праці

Посада виконавців	Місячний оклад, грн
Керівник ВКР, доцент	5286
Консультант техніко-економічного розділу, доцент	6086
Студент	1330

Витрати на оплату праці розробників проекту визначаються за формулою:

$$B_{ОП} = \sum_{i=1}^{N} \sum_{j=1}^{M} n_{ij} \cdot t_{ij} \cdot C_{ij} \qquad (4.2)$$

де n_{ij} – чисельність розробників i-ої спеціальності j-го тарифного розряду, осіб;

t_{ij} – затрачений час на розробку проекту співробітником i-ої спеціальності j-го тарифного розряду, год;

C_{ij} – годинна ставка працівника i-ої спеціальності j-го тарифного розряду, грн.

Середньо годинна ставка працівника може бути розрахована за формулою:

$$C_{ij} = \frac{C_{ij}^{0}(1+h)}{РЧ_{i}}, \quad (4.3)$$

де C_{ij} – основна місячна заробітна плата розробника *i*-ої спеціальності *j*-го тарифного розряду, грн;

h – коефіцієнт, що визначає розмір додаткової заробітної плати (при умові наявності доплат);

$РЧ_i$ - місячний фонд робочого часу працівника *i*-ої спеціальності *j*-го тарифного розряду, год (приймаємо 168 год).

Результати розрахунку записують до таблиці 4.2.

Таблиця 4.2 - Розрахунок витрат на оплату праці

Посада виконавців	Час розробки, год	Погодинна заробітна плата, грн/год	Витрати на розробку, грн
Керівник ВКР, доцент	16	56,6	905,6
Консультант техніко-економічного розділу, доцент	2	76	152
Студент	100	7,9	790
Разом	1847,6		

4.1.2 Відрахування на соціальні заходи

Величну відрахувань у спеціальні державні фонди визначають у відсотковому співвідношенні від суми основної та додаткової заробітних плат. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 20,5 % від суми заробітної плати:

$$B_{\Phi} = \frac{20,5}{100} \cdot 1847,6 = 378,76 \text{ грн.} \quad (4.4)$$

4.1.3 Розрахунок витрат на матеріали та комплектуючі

У таблиці 4.3 наведений перелік купованих виробів і розраховані витрати на них.

Таблиця 4.3 – Розрахунок витрат на матеріали та комплектуючі

Найменування купованих виробів	Одиниця виміру	Ціна, грн	Кількість куповани х виробів	Сума, грн	Транспортні витрати (10% від суми)	Загальна сума, грн
Папір (формат А4)	уп	90,0	1	90,00	9,0	99,0
Ручка кулькова	шт	20,0	1	20,00	2,00	22,0
Олівець простий	шт	4,0	1	4,00	0,4	4,40
Зошит, 18 арк	шт	6,0	1	6,00	0,6	6,60
Тонер для принтера	уп	50	1	50	5,0	55,0
Разом						187

4.1.4 Витрати на використання комп’ютерної техніки

Витрати на використання комп’ютерної техніки включають витрати на амортизацію комп’ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп’ютером. За даними обчислювального центру ТНЕУ для комп’ютера типу IBM PC/ATX вартість години роботи становить 5,2 грн. Середній щоденний час роботи на комп’ютері – 2 години. Розрахунок витрат на використання комп’ютерної техніки приведений в таблиці 4.4.

Таблиця 4.4- Розрахунок витрат на використання комп’ютерної техніки

Назва етапів робіт, при виконанні яких використовується комп’ютер	Час використання комп’ютера, год.	Витрати на використання комп’ютера, грн.
Проведення досліджень та оформлення їх результатів	60	312
Оформлення техніко-економічного розділу	8	41,6
Оформлення ВКР	12	62,4
Разом	80	416

4.1.5 Накладні витрати

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими відсотками до витрат на оплату праці.

Середньостатистичний відсоток накладних витрат приймемо 150% від заробітної плати:

$$H = 1{,}5 \cdot 1847{,}6 = 2771{,}4 \ \ (грн.) \tag{4.5}$$

4.1.6 Інші витрати

Інші витрати є витратами, які не враховані в попередніх статтях. Вони становлять 10% від заробітної плати:

$$I = 1847{,}6 \cdot 0{,}1 = 184{,}76 \ \ (грн) \tag{4.6}$$

Витрати на розробку програмного забезпечення складають:

$$K_1 = B_{ОП} + B_Ф + B_{ПВ} + H + I \tag{4.7}$$

$$K_1 = 1847{,}6 + 378{,}76 + 187 + 2771{,}4 + 184{,}76 = 5369{,}86 \ (грн) \tag{4.8}$$

Витрати на відлагодження і дослідну експлуатацію програмного продукту визначаємо за формулою:

$$K_2 = S_{м.г.} \cdot t_{від} \tag{4.9}$$

де $S_{м.г.}$ - вартість однієї машино-години роботи ПК, грн/год.;

$t_{від}$ - комп'ютерний час, витрачений на відлагодження і дослідну експлуатацію створеного програмного продукту, год.

Загальна кількість днів роботи на комп'ютері дорівнює 40 днів. Середній щоденний час роботи на комп'ютері – 2 години. Вартість години роботи комп'ютера дорівнює 5,2 грн. Тому:

$$K_2 = 5{,}2 \cdot 80 = 416 \text{ грн.} \quad (4.10)$$

На основі отриманих даних складаємо кошторис витрат на розробку програмного забезпечення (таблиця 2.5).

Таблиця 4.5 – Кошторис витрат на розробку програмного забезпечення

Найменування витрат	Сума витрат, грн
Витрати на оплату праці	1847,6
Відрахування у спеціальні державні фонди	378,76
Витрати на куповані вироби	187
Накладні витрати	2771,4
Інші витрати	184,76
Витрати на відлагодження і дослідну експлуатацію програмного продукту	416
Разом	5785,52

4.2 Визначення експлуатаційних витрат

Для оцінки економічної ефективності розроблюваного програмного продукту слід порівняти його з аналогом, тобто існуючим програмним забезпеченням ідентичного функціонального призначення.

Експлуатаційні одноразові витрати по програмному забезпеченню і аналогу включають вартість підготовки даних і вартість роботи комп'ютера (за час дії програми):

$$E_{\text{П}} = E_{1\text{П}} + E_{2\text{П}} \quad (4.11)$$

де $E_{\text{п}}$ - одноразові експлуатаційні витрати на ПЗ (аналог), грн.;

$E_{1\text{п}}$ - вартість підготовки даних для експлуатації ПЗ (аналогу), грн;

$E_{2\text{п}}$ - вартість роботи комп'ютера для розробки програмного забезпечення (аналогу), грн.

Річні експлуатаційні витрати $B_{\text{еп}}$ визначаються за формулою:

$$B_{E\text{П}} = E_{\text{П}} * N_{\text{П}} \tag{4.12}$$

де $N_{\text{п}}$ - періодичність експлуатації ПЗ (аналогу), раз/рік.

Вартість підготовки даних для роботи на комп'ютері визначається за формулою:

$$E_{1\text{П}} = \sum_{l=1}^{n} n_i t_i c_i \tag{4.13}$$

де i - категорії працівників, які приймають участь у підготовці даних (i=1,2,...,n);

n_i - кількість працівників i-ої категорії, осіб;

t_i - трудомісткість роботи співробітників i-ої категорії по підготовці даних, год.;

c_i - середнього годинна ставка працівника i-ої категорії з врахуванням додаткової заробітної плати, що знаходиться із співвідношення:

$$c_i = \frac{c_i^0 (1+b)}{m} \tag{4.14}$$

де c_i^0 - основна місячна заробітна плата працівника і-ої категорії, грн;

b - коефіцієнт, який враховує додаткову заробітну плату;

m - кількість робочих годин у місяці, год.

Для роботи з даними як для поточного програмного забезпечення так і аналогу потрібен один працівник, основна місячна заробітна плата якого складає: с° = 1330 *грн*. Тоді:

$$c_1 = \frac{1330(1+0{,}57)}{22*8} = 11{,}86\ грн/год \qquad (4.15)$$

Трудомісткість підготовки даних для програмного забезпечення складає 1 год., для аналога 1,5 год.

Таблиця 4.6 – Розрахунок витрат на підготовку даних та реалізацію програмного забезпечення на комп'ютері

Час роботи співробітників, год	Середньогодинна заробітна плата, грн/год	Витрати, грн
Проектне рішення		
1	11,86	11,86
Аналог		
1,5	11,86	17,79

Витрати на експлуатацію комп'ютера визначається за формулою:

$$E_{2П} = t * S_{МГ} \qquad (4.16)$$

де *t* - витрати машинного часу для реалізації програмного продукту, год;

$S_{МГ}$ - вартість однієї години роботи комп'ютера, грн/год.

$$E_{2п} = 1 \cdot 5{,}2 = 5{,}2\ (грн);\ E_{2а} = 1{,}5 \cdot 5{,}2 = 7{,}8\ (грн) \qquad (4.17)$$

$$E_п = 11{,}86 + 5{,}2 = 17{,}06\ (грн.;\ E_а = 17{,}79 + 7{,}8 = 25{,}59 (грн) \qquad (4.18)$$

$$B_{en} = 17{,}06 \cdot 252 = 4299{,}12\ (грн);\ B_{ea} = 25{,}59 \cdot 252 = 6448{,}68\ (грн) \qquad (4.19)$$

4.3 Розрахунок ціни споживання програмного продукту

Ціна споживання - це витрати на придбання і експлуатацію програмного продукту за весь строк його служби:

$$Ц_{С(П)} = Ц_{П} + B_{(E)NPV} \qquad (4.20)$$

де $Ц_{п}$ - ціна придбання програмного продукту, грн.

$$Ц_{П} = К(1 + \frac{П_{Р}}{100}) + К_{0} + К_{к} \qquad (4.21)$$

де $К$ - кошторисна вартість;

$П_{р}$ - рентабельність;

$К_{о}$ - витрати на прив'язку та освоєння програного забезпечення на конкретному об'єкті, грн;

$К_{к}$ - витрати на доукомплектування технічних засобів на об'єкті, грн.

$$Ц_{Д} = 5785{,}52 \cdot (1 + 0{,}3) = 7521{,}2\ (грн.). \qquad (4.22)$$

Вартість витрат на експлуатацію програмного забезпечення (за весь час його експлуатації), грн:

$$B_{enpv} = \sum_{t=0}^{T} \frac{B_{eП}}{(1+R)^{t}} \qquad (4.23)$$

де $B_{еп}$ - річні експлуатаційні витрати, грн;

T - термін служби програмного забезпечення, років;

R - річна ставка проценту банку.

$$B_{е\pi рv} = \sum_{t=1}^{5} \frac{4299{,}12}{(1+0{,}08)^t} = 17200{,}15 \ (грн) \quad (4.24)$$

$$B_{е\pi рv} = \sum_{t=1}^{5} \frac{6448{,}68}{(1+0{,}08)^t} = 25800{,}2 \ (грн) \quad (4.25)$$

Тоді ціна споживання програмного забезпечення дорівнюватиме:

$$Ц_{сп} = 7521{,}2+17200{,}15=24721{,}35 \ (грн) \quad (4.26)$$

Аналогічно визначається ціна споживання для аналогу:

$$Ц_{са} = 6500{,}0+25800{,}2=32300{,}2 \ (грн) \quad (4.27)$$

4.4 Визначення показників економічної ефективності

Економічний ефект в сфері розробки програмного продукту:

$$E_{ПР} = Ц_{П} - Ц_{А} \quad (4.28)$$

$$E_{ПР} = 7521{,}2 - 6500{,}0 = 1021{,}2 \ (грн) \quad (4.29)$$

Річний економічний ефект в сфері експлуатації:

$$E_{КС} = B_{ЕА} - B_{ЕП} \quad (4.30)$$

$$E_{кс}=6448,68 - 4299,12=2149,56\ (грн) \quad (4.31)$$

Додатковий економічний ефект у сфері експлуатації:

$$\Delta E_{екс} = \sum_{t=1}^{T} E_{екс}(1+R)^{T-t} \quad (4.32)$$

$$\Delta E_{екс} = \sum_{t=1}^{5} 2149,56 * (1+0,08)^{5-t} = 12595,71 (грн) \quad (4.33)$$

Сумарний ефект складає:

$$E = E_{пр} + \Delta E_{екс} = 1021,2 + 12595,71 = 13616,91\ (грн) \quad (4.34)$$

В даному розділі проведено розрахунок витрат на розробку програмного забезпечення. Здійснено порівняння з існуючим аналогом, і цим показано, що вказане програмне забезпечення має переваги в порівнянні з аналогами, зокрема: надійність, простота використання, зручність. Згідно проведеного економічного обґрунтування зазначене програмного забезпечення є конкурентноздатним. Крім того, отримано економічний ефект у розмірі 13616 грн. і тому розробка і впровадження цього програмного забезпечення є економічно доцільними.

ВИСНОВКИ

В даній бакалаврській роботі розроблено VHDL-модель шифрування згідно з алгоритмом AES. Ця модель призначена для синтезу на програмовані логічні та замовлені інтегральні схеми і її можна використати для побудови спеціалізованих комп'ютерних систем захисту інформації від несанкціонованого доступу. Основною перевагою розробленої моделі є її незалежність від цільової платформи синтезу, високі показники продуктивності.

У бакалаврській роботі проведено:

– аналіз стану реалізації алгоритму шифрування AES на основі програмованих і спеціалізованих процесорах. Вказано, що перспективним напрямком є створення спеціалізованих процесорів симетричного блокового шифрування;

– розроблення архітектуру конвеєрного процесора шифрування згідно з алгоритмом AES. Зокрема розроблено інтерфейс і структурну схему процесора, його функціональних вузлів;

– функціональне тестування процесора з використанням інтегрованого середовища Active-HDL 5.1. Функціональна перевірка роботи процесора показала правильність його роботи як на зашифровування, так і на розшифровування інформації;

– моделювання та реалізацію ядра конвеєрного процесора симетричного блокового шифрування згідно з алгоритмом AES на VHDL. Розроблено VHDL-модель ядра процесора. Синтез ядра процесора на програмовані логічні інтегральні схеми фірми Xilinx Virtex2 і Spartan2e та подальше дослідження характеристики продуктивності процесорів дозволили встановити, що розроблені процесори дозволяють обробляти потоки даних інтенсивністю до 7

Гбіт/с у реальному часі.

Результати бакалаврської роботи апробовано на інтернет-конференції (додаток Б) та мають практичне використання (додаток В).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Березький О., Березька К., Батько Ю., Мельник Г. Синтез альтернативних рішень при структурному проектуванні систем автоматизованої мікроскопії. Вісник Національного університету «Львівська політехніка». Комп'ютерні науки та інформаційні технології. 2009. Вип. 638. С. 64 - 72.

2. Veta Mitko, JPW Pluim, PJ van Diest, M. Viergever. Breast cancer histopathology image analysis: A review. Biomedical Engineering, IEEE Transactions. 2014. №61(5). P. 1400-1411.

3. Chen Jia-Mei, Qu Ai-Ping. New breast cancer prognostic factors identified by computer-aided image analysis of HE stained histopathology images. Nature Scientific Reports. 2015. № 5. P. 33-38.

4. Zhang Y.J. An Overview of Image and Video Segmentation in the Last 40 Years, in Advances in Image and Video Segmentation. IRM Press. 2006. P.115.

5. Sharma M., Chouhan V. Objective Evaluation Parameters of Image Segmentation Algorithms. International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958. Volume 2. Issue 2. 2012. P.84-87.

6. Christensen H. I., Phillips P. J. Empirical evaluation methods in computer vision. World Scientific Publishing Company. July 2002. 45 pp.

7. Philipp-Foliguet S., Guigues L. Multi-scale criteria for the evaluation of image segmentation algorithms. Journal of multimedia. Vol. 3. No. 5. 2008. P.42-56.

8. Chinnadurai V., Chandrashekhar G. Improvised levelset method for segmentation and grading of brain tumors in dynamic contrast susceptibility and apparent diffusion coefficient magnetic resonance images. International journal of engineering science and technology. Vol.2(5). 2010. P.1461-1472.

9. Zhang Y.J. Image segmentation evaluation in this century. Encyclopedia of Information Science and Technology. 2nd ed.. IGI Global. 2009. P.1812-1817.

10. Padamavati S., Subashini P., Sumi A. Empirical Evaluation of suitable segmentation algorithm for IR Images. IJCSI. Vol. 7. Issue 4. No.2. July2010.

11. Березский О. Н., Березская Е.Н. Количественная оценка качества сегментации изображений на основе метрик. Управляющие системы и машины. 2015. №6. С.59-65.

12. Березький О. М., Батько Ю.М., Мельник Г.М. Методи сегментації біомедичних зображень. Вісник Хмельницького національного університету. Технічні науки. 2010. №1. С.189- 197.

13. Ross T.J. Fuzzy Logic with Engineering Applications. McGraw-HillInc. USA. 1995. 600 p.

14. Дубчак Л.О. Нечітка система захисту інформації в телемедицині. Системи обробки інформації. 2015. № 8 (133). С.97 – 101.

15. Yinpeng J., Fayadb L., Laine A. Contrast Enhancement by Multi-scale Adaptive Histogram Equalization. Proceedings of SPIE. Vol. 4478. 2001. P. 206-213

16. Бережная М.А. Методы проектирования нечетких устройств принятия решений на основе программируемых логических интегральных микросхемах. Технология приборостроения. 2009. №2. С. 16–23.

17. Михайленко В.С., Никольский В.В. Использование нечеткой адаптивной системы управления для компьютерного мониторинга сетью котельных установок. URL: http://aaecs.org/mihailenko-vs-nikolskii-vv-ispolzovanie-nechetkoi-adaptivnoi-sistemi-upravleniya-dlya-kompyuternogo-monitoringa-setyu-kotelnih-ustanovok.html

18. Ozyer T., Alhajj R., Barker K. Intrusion detection by integrating boosting genetic fuzzy classifier and data mining criteria for rule pre-screening. Journal of Network and Computer Applications. 2007. №30. P.99-113.

19. Гнатчук Є.Г. Інформаційна технологія подання та опрацювання знань на основі нечіткої логіки в експертних системах діагностування комп'ютерних засобів: автореф. дис. на здобуття наук. ступеня канд. техн. наук: спец. 05.13.06 «Інформаційні технології». Львів. 2008. 20 с.

20. Ротштейн О.П., Ларюшкін Є.П., Мітюшкін Ю.І. Soft Computing в біотехнології: багатофакторний аналіз і діагностика. Вінниця. УНІВЕРСУМ-Вінниця. 2008. 144 с.

21. Мороз О.В., Свентух А.О. Економічна ідентифікація параметрів стійкості та ризикованості функціонування господарських систем. Вінниця. УНІВЕРСУМ-Вінниця. 2008. 168 с.

22. Панкевич О.Д., Штовба С.Д. Діагностування тріщин будівельних конструкцій за допомогою нечітких баз знань. Вінниця: УНІВЕРСУМ-Вінниця. 2005. 108с.

23. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. М.: Телеком. 2006. 382 с.

24. Abadeh M., Habibi J., Lucas C. Intrusion Detection Using a Fuzzy Genetics-Based Learning Algorithm. Journal of Network and Computer Applications. 2007. №30. P.414-428.

25. Методичні рекомендації до виконання дипломного проекту з освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050102 «Комп’ютерна інженерія» фахового спрямування «Комп’ютерні системи та мережі» / О.М. Березький, Л.О.Дубчак, Р.Б. Трембач, Г.М. Мельник, Ю.М. Батько, С.В. Івасьєв / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2016. 65 с.