МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи
з дисципліни «Мова опису апаратури Verilog»
на тему «Інтегрована навчальна система верифікації даних на базі FPGA»
для студентів спеціальності 123 (F7) «Комп’ютерна інженерія»
за освітньою програмою «Комп’ютерна інженерія»
всіх форм навчання

2025

Методичні вказівки до лабораторної роботи з дисципліни «Мова опису апаратури Verilog» на тему «Інтегрована навчальна система верифікації даних на базі FPGA» для студентів спеціальності 123 (F7) «Комп’ютерна інженерія» за освітньою програмою «Комп’ютерна інженерія» всіх форм навчання / Укл.: С.С. Грушко, І.Я. Зеленьова – Запоріжжя: НУ «Запорізька політехніка», 2025. – 17 с.

Укладач: С.С. Грушко, к.т.н., доцент
І.Я. Зеленьова, к.т.н., доцент

Рецензент: Г.Г. Киричек, к.т.н., доцент

Відповідальний
за випуск: С.С. Грушко, к.т.н., доцент

Затверджено:
на засіданні кафедри
«Комп’ютерні системи та мережі»
Протокол № 2
від 29.08.2025 р.

Рекомендовано до видання
НМК факультету КНТ
Протокол № 3
від 25.09.2025 р.

ЗМІСТ

ВСТУП

Дані методичні вказівки розроблено на основі результатів дипломного проєкту бакалавра Климакова Олексія Євгеновича «Розробка FPGA системи верифікації оцінок» (2025 р.) та призначено для впровадження у навчальний процес підготовки фахівців за спеціальністю F7 (123) «Комп'ютерна інженерія».

Лабораторна робота знайомить студентів з практичними аспектами проєктування цифрових систем на програмованих логічних інтегральних схемах (FPGA) з використанням мови опису апаратури Verilog. Основна увага приділяється розробці системи верифікації даних, яка забезпечує цілісність та достовірність інформації за допомогою криптографічних хеш-функцій.

У процесі виконання роботи студенти набувають практичних навичок:

– проєктування цифрових систем мовою Verilog;

– реалізації криптографічних алгоритмів на апаратному рівні;

– роботи з середовищем розробки Active-HDL;

– аналізу та оптимізації використання ресурсів FPGA;

– тестування та верифікації розроблених модулів.

Матеріал методичних вказівок структуровано таким чином, щоб забезпечити поступовий перехід від теоретичних основ до практичної реалізації системи. Наведені приклади коду та детальні пояснення дозволяють студентам самостійно виконати всі етапи розробки та отримати працездатну систему верифікації даних.

Методичні вказівки можуть бути використані студентами денної та заочної форм навчання, а також магістрантами при виконанні курсових та дипломних проєктів за тематикою комп'ютерної інженерії та захисту інформації.

1 ЛАБОРАТОРНА РОБОТА
ІНТЕГРОВАНА НАВЧАЛЬНА СИСТЕМА ВЕРИФІКАЦІЇ ДАНИХ НА БАЗІ FPGA

Мета роботи: розробка FPGA-системи для верифікації оцінок, придатної для використання в освітніх цифрових системах.

1.1 Короткі теоретичні відомості

1.1.1 Призначення та сфера застосування системи

Система верифікації оцінок призначена для забезпечення надійності та достовірності даних про навчальні досягнення студентів в умовах електронного документообігу. Перехід від паперових журналів до цифрових систем вимагає надійних механізмів захисту інформації про оцінювання. У цій лабораторній роботі реалізовано FPGA-систему, яка забезпечує автоматичну верифікацію оцінок за допомогою криптографічних хеш-функцій.

Основні функції системи:

– формування цифрового відбитка (хешу) для кожної оцінки при її внесенні до системи;

– перевірка цілісності даних через порівняння збережених та обчислених хешів;

– виявлення несанкціонованих змін в історії оцінювання;

– забезпечення неспростовності записів про академічні досягнення.

1.1.2 Базові вимоги до системи верифікації

Для надійної роботи система верифікації оцінок має відповідати п’ятьом ключовим вимогам: цілісність даних, конфіденційність, неспростовність, доступність та надійність.

Цілісність даних. Після внесення оцінки будь-які її зміни мають виявлятися автоматично. Це досягається шляхом обчислення хеш-значення під час створення запису. У разі спроби модифікації оцінки повторне обчислення хешу дасть інше значення, що сигналізує про порушення цілісності.

Конфіденційність. Інформація про оцінки студентів є персональними даними. Система забезпечує контрольований доступ до даних, дозволяючи перегляд та верифікацію лише авторизованим користувачам.

Неспростовність. Жодна зі сторін не може заперечувати факт створення або зміни запису про оцінку. Хеш-значення слугує доказом автентичності даних у момент їх створення.

Доступність та надійність. Система має забезпечувати безперебійну роботу та швидкий доступ до даних. FPGA-реалізація гарантує стабільну продуктивність навіть при високому навантаженні.

1.1.3 Основи криптографічних хеш-функцій

Хеш-функція - це математичний алгоритм, який перетворює вхідні дані довільного розміру на вихідний рядок фіксованої довжини (хеш-значення або дайджест). Цей рядок є своєрідним "цифровим відбитком" даних.

Властивості криптографічних хеш-функцій:

– детермінованість - однакові вхідні дані завжди дають однакове хеш-значення. Це дозволяє відтворити хеш для перевірки цілісності;

– стійкість до пошуку прообразу - за хеш-значенням практично неможливо відновити початкові дані. Навіть знаючи хеш оцінки, зловмисник не зможе визначити її значення;

– стійкість до колізій - надзвичайно складно знайти два різних набори даних з однаковим хешем. Для 32-бітного хешу ймовірність колізії становить приблизно 1 до 4 мільярдів;

– лавинний ефект - навіть мінімальна зміна вхідних даних (наприклад, зміна оцінки з 4 на 5) призводить до кардинальної зміни хеш-значення.

Приклад застосування в системі.

Вхідні дані: ID студента (12345) + Предмет (101) + Оцінка (85)

Хеш-значення: 0xA3F5B2C8 (32-бітне число у шістнадцятковому форматі)

При зміні оцінки на 86 хеш зміниться, наприклад, на 0x7D8E1F4A

Поширені сімейства хеш-функцій:

– SHA (Secure Hash Algorithm): SHA-1, SHA-2, SHA-3;

– BLAKE: BLAKE2, BLAKE3;

– MD (Message Digest): MD4, MD5.

У цій роботі використовується алгоритм Jenkins hash function [1] через його простоту реалізації на FPGA та достатню стійкість для навчальних цілей.

1.1.4 Алгоритм Jenkins hash function

Алгоритм Jenkins hash function, розроблений Бобом Дженкінсом, є швидким методом створення хешу для даних довільної довжини. Алгоритм (рис.1.1.) обробляє вхідні дані побайтово та формує 32-бітне хеш-значення через послідовність арифметичних і побітових операцій.

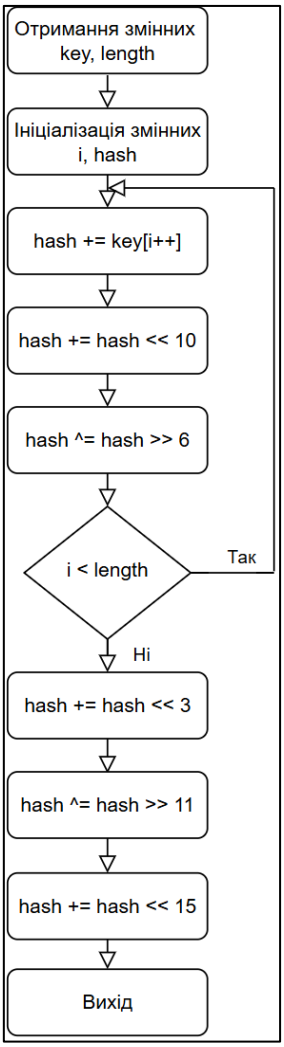

Рисунок 1.1 – Робота алгоритму Jenkins hash function

Етап 1. Ініціалізація.

```
hash = 0
```

Початкове значення хешу встановлюється в нуль.

Етап 2. Обробка кожного байта вхідних даних.

Для кожного байта input_data виконуються три операції:

```
hash = hash + byte              // Додавання байта
hash = hash + (hash << 10)      // Зсув вліво на 10 бітів та додавання
hash = hash ^ (hash >> 6)       // XOR зі зсувом вправо на 6 бітів
```

Приклад обробки байта:

Поточний hash = 0x00001234

Байт даних = 0x5A

Після додавання: hash = 0x0000128E

Після зсуву та додавання: hash = 0x0004A68E

Після XOR: hash = 0x0004B956

Етап 3. Фінальне перемішування.

Після обробки всіх байтів виконуються три завершальні операції:

```
hash = hash + (hash << 3      // Зсув вліво на 3 біти
hash = hash ^ (hash >> 11)    // XOR зі зсувом вправо на 11 бітів
hash = hash + (hash << 15)    // Зсув вліво на 15 бітів
```

Ці операції забезпечують рівномірний розподіл бітів у результаті та підвищують стійкість до колізій.

Етап 4. Формування результату.

Результатом є 32-бітне значення хешу, яке використовується для верифікації даних.

Переваги алгоритму для FPGA-реалізації:

– використовує прості операції (додавання, зсуви, XOR), які ефективно реалізуються апаратно;

– не потребує великих обсягів пам’яті для проміжних обчислень;

– забезпечує детермінований час виконання;

– достатня стійкість для навчальних систем верифікації.

1.2 Хід роботи

1.2.1 Створення дизайну проєкту

На початку роботи створюємо новий проєкт у середовищі Active-HDL [2]. Для цього запускаємо Active-HDL та в головному меню обираємо File > New > Workspace. Це відкриває майстер створення проєкту, де задаємо робочу директорію і назву робочої області (рис. 1.2).

Цей процес призводить до створення базової структури проєкту. Модуль верхнього рівня, який запускається на FPGA, відповідатиме за обробку даних, тоді

як хост-програма забезпечуватиме взаємодію з зовнішніми системами. На цьому етапі конфігурація інтерфейсів не передбачена.

Далі обираємо пункт Create an Empty Design (рис. 1.3) та мову опису апаратури Verilog (рис. 1.4). В наступному вікні задаємо назву проєкту (рис. 1.5). Наприкінці створення переконуємось, що нема помилок, та натискаємо Finish (рис. 1.6).

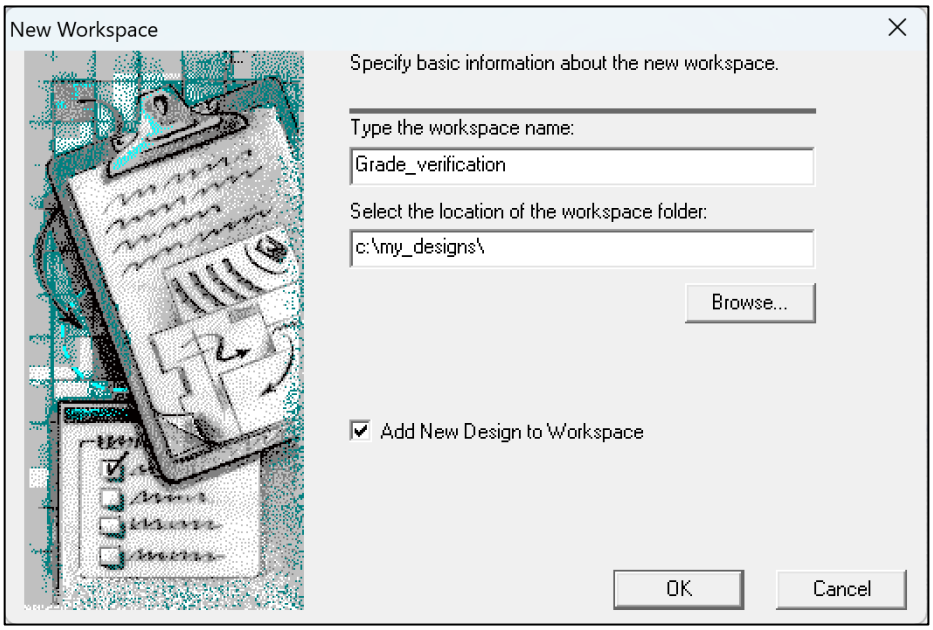

Рисунок 1.2 – Створення нової робочої області в Active-HDL

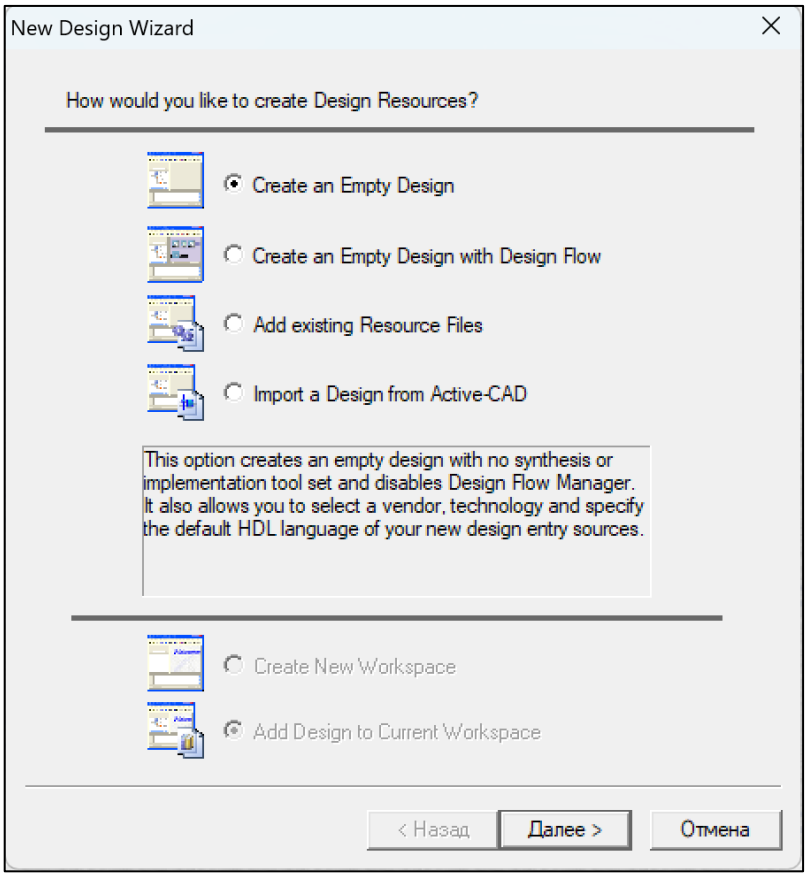

Рисунок 1.3 – Вибір дизайну проєкту

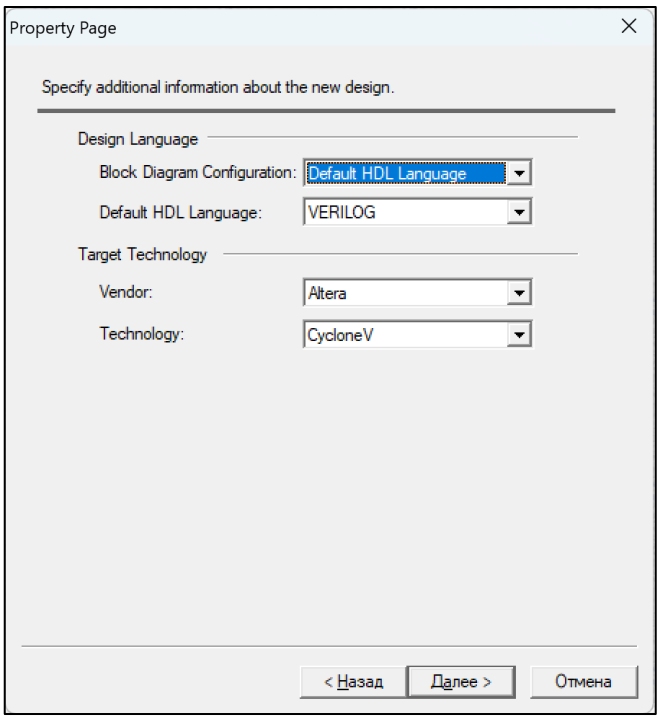

Рисунок 1.4 – Вибір мови опису апаратури та платформи

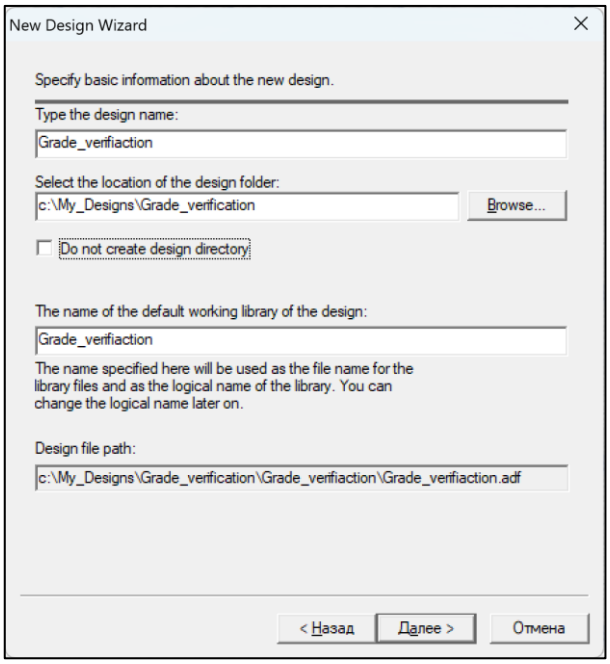

Рисунок 1.5 – Створення проєкту

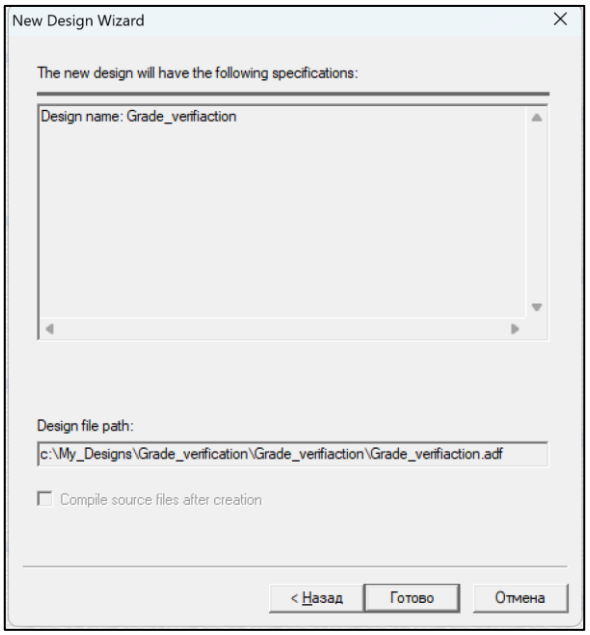

Рисунок 1.6 – Вікно підтвердження створення

Для додання файлу натискаємо на Add New File (рис 1.7). У наступному вікні обираємо тип файлу та задаємо його назву (рис. 1.8).

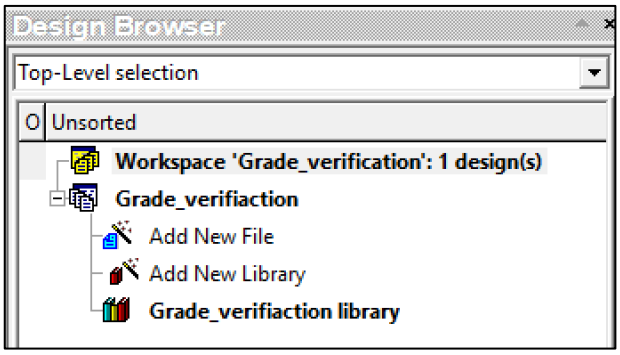

Рисунок 1.7 – Додання нового файлу до проєкту

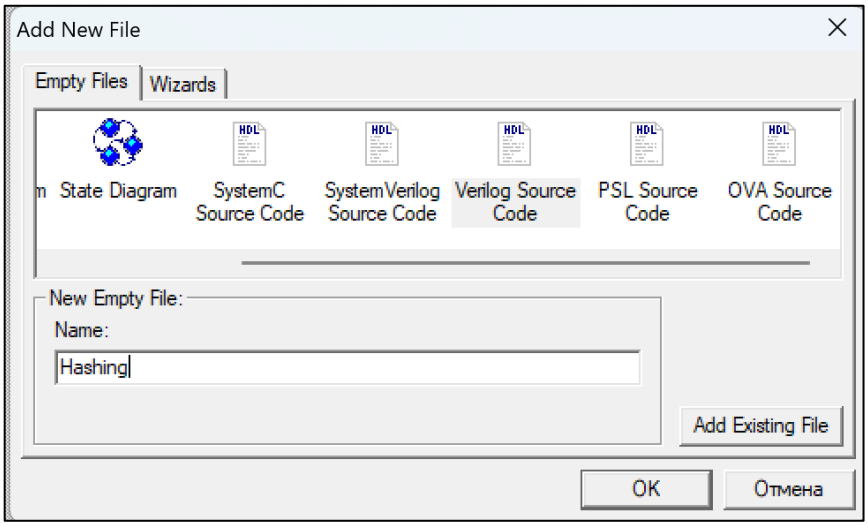

Рисунок 1.8 – Вибір типу файлу та задання його назви

Тепер можна переходити до реалізації системи.

1.2.2 Реалізація складників системи

Реалізація всіх складників системи верифікації оцінок зосереджена в єдиному Verilog-модулі, описаному у файлі hashing.v [3]. Цей модуль об’єднує функціональність прийому даних, хешування та верифікації, що дозволяє оптимізувати використання ресурсів FPGA і спростити взаємодію між компонентами. Модуль працює в двох режимах: хешування (створення хешу для внесення оцінки) і верифікація (перевірка цілісності оцінки шляхом порівняння хешів).

Складник прийому даних відповідає за отримання вхідних даних, їх попередню обробку та підготовку до подальшого хешування або верифікації. У файлі hashing.v цей складник реалізовано через інтерфейс модуля, який визначає вхідні та вихідні сигнали. Код інтерфейсу наведено в лістингу 1.1.

Лістинг 1.1 – Прийом даних

```
module hashing (
    input wire clk,
    input wire reset,
    input wire [31:0] data_in,
    input wire [31:0] length_in,
    input wire valid_in,
    input wire mode,
    input wire [31:0] hash_ref,
    output reg [31:0] hash_out,
    output reg valid_out,
    output reg hash_ok
);
```

Цей код визначає інтерфейс модуля, який виконує функцію прийому даних. Вхідні сигнали включають:

– clk — тактовий сигнал для синхронізації роботи модуля;

– reset — сигнал скидання для ініціалізації регістрів у нульовий стан;

– data_in — 32-бітний вхід для передачі структурованих даних (наприклад, ідентифікатор студента, оцінка, метадані);

– length_in — 32-бітний вхід, що вказує на кількість байтів у вхідних даних;

– valid_in — сигнал, який позначає, що вхідні дані готові до обробки;

– mode — сигнал, що визначає режим роботи (0 для хешування, 1 для верифікації);

– hash_ref — 32-бітне еталонне значення хешу, яке використовується під час верифікації.

Вихідні сигнали включають:

– hash_out — 32-бітне значення обчисленого хешу;

– valid_out — сигнал, що вказує на завершення обробки;

– hash_ok — сигнал, який позначає успішність верифікації (1, якщо хеші збігаються, 0, якщо ні).

Складник прийому даних забезпечує надійне отримання структурованих даних. Дані надходять у форматі 32-бітних слів, що містять інформацію про оцінку, таку як ідентифікатор студента, предмета, викладача та значення оцінки. Сигнал valid_in активується, коли дані готові, а сигнал mode визначає, чи потрібно передати дані на хешування, чи на верифікацію для перевірки існуючого запису. Сигнал reset дозволяє скинути всі внутрішні регістри модуля до початкового стану.

Складник хешування відповідає за створення криптографічного хеш-значення для вхідних даних. Його основна функція полягає в обробці даних, отриманих від складника прийому, і генерації 32-бітного хешу. У файлі hashing.v цей складник реалізовано в основному блоці логіки, який виконує обчислення хешу. Код хешування наведено в лістингу 1.2.

Лістинг 1.2 – Хешування даних

```
reg [31:0] hash;
reg [2:0] j;
reg [7:0] current_byte;

always @(*) begin
    if (reset) begin
        hash = 32'b0;
        hash_out = 32'b0;
        j = 0;
        valid_out = 1'b0;
    end else begin
        valid_out = 1'b0;
        hash_ok = 0;
```

Кінець лістингу 1.2

```
        if (valid_in) begin
            hash = 32'b0;
            hash_out = 32'b0;
            j = 0;
            while (j < length_in) begin
                case (j)
                    3'b000: current_byte[7:0] = data_in[7:0];
                    3'b001: current_byte[7:0] = data_in[15:8];
                    3'b010: current_byte[7:0] = data_in[23:16];
                    3'b011: current_byte[7:0] = data_in[31:24];
                    default: current_byte[7:0] = 8'b0;
                endcase

                hash = hash + current_byte[7:0];
                hash = hash + (hash << 10);
                hash = hash ^ (hash >> 6);
                j = j + 1;
            end

            hash = hash + (hash << 3);
            hash = hash ^ (hash >> 11);
            hash = hash + (hash << 15);

            hash_out = hash;
            valid_out = 1'b1;
        end
    end
end
```

Цей код реалізує алгоритм Jenkins hash function у режимі хешування. Складник хешування активується, коли сигнал valid_in дорівнює 1 і сигнал mode дорівнює 0, що вказує на необхідність створення хешу для нових даних. Вхідні дані з data_in розбиваються на байти за допомогою конструкції case, яка вибирає відповідний

байт із 32-бітного слова залежно від значення лічильника j. Наприклад, при j = 0 вибирається перший байт (data_in[7:0]), при j = 1 — другий (data_in[15:8]) тощо. Лічильник j обмежений значенням length_in, яке визначає кількість байтів для обробки.

Для кожного байта виконуються три основні операції алгоритму Jenkins: додавання байта до поточного значення хешу (hash = hash + current_byte), зсув вліво на 10 бітів (hash = hash + (hash << 10)) і побітове виключне АБО зі зсувом вправо на 6 бітів (hash = hash ^ (hash >> 6)). Ці операції повторюються для всіх байтів, забезпечуючи змішування даних і створення унікального хешу. Після обробки всіх байтів виконується фіналізація: зсув вліво на 3 біти (hash = hash + (hash << 3)), побітове виключне АБО зі зсувом вправо на 11 бітів (hash = hash ^ (hash >> 11)) і зсув вліво на 15 бітів (hash = hash + (hash << 15)). Результат записується в регістр hash_out, а сигнал valid_out встановлюється в 1, вказуючи на завершення обчислень.

Складник верифікації відповідає за перевірку цілісності оцінок шляхом порівняння обчисленого хеш-значення з еталонним хешем. У файлі hashing.v функція верифікації інтегрована в основний модуль і реалізується через умовний блок, який активується, коли сигнал mode дорівнює 1. Код верифікації наведено в лістингу 1.3.

Лістинг 1.3 – Верифікація даних

```
if (mode) begin
    if (hash_ref == hash_out) begin
        hash_ok = 1;
        $display("Hashing compare success, hash = %h", hash_out);
end else begin
        hash_ok = 0;

$display("Hashing compare fail, hash = %h, hash ref = %h",
hash_out, hash_ref);
    End
```

Кінець лістингу 1.3

```
end else begin
    $display("Hashing success, hash = %h", hash_out);
end
```

Цей код виконує верифікацію, порівнюючи обчислений хеш (hash_out) із еталонним значенням (hash_ref). Якщо сигнали valid_in і mode дорівнюють 1, складник хешування спочатку обчислює хеш для вхідних даних (data_in) за алгоритмом Jenkins, як описано вище. Після цього обчислений хеш порівнюється з hash_ref. Якщо вони збігаються, сигнал hash_ok встановлюється в 1, і система виводить повідомлення про успішну верифікацію через $display. У разі невідповідності hash_ok залишається 0, а повідомлення вказує на помилку, включаючи значення обох хешів для діагностики.

Складник верифікації використовує ту саму логіку обчислення хешу, що й складник хешування, що гарантує узгодженість результатів. Порівняння хешів виконується миттєво після обчислення, що забезпечує швидку перевірку навіть при великому обсязі запитів. Сигнал hash_ok дозволяє користувачу отримати підтвердження валідності оцінки через інтерфейс системи.

1.3 Зміст звіту

1. Мета роботи.

2. Лістинг модуля верифікації оцінок з коментарями та часові діаграми його роботи, що підтверджують працездатність системи.

3. Висновки до роботи.

4. Відповіді на контрольні питання.

1.4 Контрольні питання

1. Назвіть ключові вимоги до системи верифікації оцінок.

2. Наведіть основні властивості криптографічних хеш-функцій.

3. Опишіть принцип роботи алгоритму Jenkins hash function.

4. Які переваги використання FPGA для реалізації криптографічних алгоритмів порівняно з CPU чи GPU?

5. Назвіть основні компоненти архітектури сучасних FPGA-платформ.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Jenkins B. A hash function for hash Table lookup. 2013. URL: https://www.burtleburtle.net/bob/hash/doobs.html

2. Getting Started with Active-HDL: Application Notes / Aldec. 2025. URL: https://www.aldec.com/en/support/resources/documentation/articles/1054

3. Климаков О.Є. Розробка FPGA системи верифікації оцінок: бакалаврська робота, спеціальність 123 «Комп’ютерна інженерія» / О.Є. Климаков — Запоріжжя: НУ «Запорізька політехніка», 2025. — 68 с.