КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра радіотехніки та радіоелектронних систем

«На правах рукопису»

Робота допущена до захисту в ЕК
рішенням кафедри радіотехніки та радіоелектронних систем
від 2025 року, протокол № .
Завідувач кафедри доктор фіз.-мат. наук, професор
_____ Ігор АНІСІМОВ

ДИПЛОМНА РОБОТА МАГІСТРА
на тему:
«АЛГОРИТМИ СТИСНЕННЯ ЗОБРАЖЕНЬ В HDL (Hardware Description Language)»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 172 – Електронні комунікації та радіотехніка
ОПП «Інформаційна безпека телекомунікаційних систем і мереж»
Димар Даніїл Ігорович _____

Науковий керівник:
д.т.н., професор, Заслужений винахідник України
Богом'я Володимир Іванович _____

Рецензент:
д.т.н., професор кафедри управління кібербезпекою та захистом
інформації ДУІКТ
Савченко Віталій Анатолійович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____ Даніїл Димар

Київ – 2025

РЕФЕРАТ

Дипломна робота: 51 с., 18 рис., 1 табл., 20 джерел.

СТИСНЕННЯ ЗОБРАЖЕНЬ, АЛГОРИТМ ХАФФМАНА, ДИСКРЕТНЕ КОСИНУСНЕ ПЕРЕТВОРЕННЯ (DCT), HDL, FPGA, VERILOG, VHDL

Об’єкт дослідження – процеси та методи стиснення зображень за допомогою алгоритмів, реалізованих мовами опису апаратури (HDL), таких як Verilog та VHDL.

Мета роботи – аналіз та розробка апаратної реалізації алгоритмів стиснення зображень на базі HDL.

У роботі використано методи аналізу, моделювання та синтезу цифрових схем для реалізації алгоритмів стиснення зображень. Основою практичної частини є моделювання та тестування кодів HDL у середовищі Quartus Prime та ModelSim.

Розроблені алгоритми можуть бути використані у вбудованих системах, мобільних пристроях, системах відеоспостереження та інших областях, де потрібне швидке та ефективне стиснення зображень.

В результаті дослідження було розглянуто теоретичні основи стиснення зображень, реалізовано алгоритми Хаффмана та DCT у середовищі Verilog/VHDL та виконано їх тестування. Проведено аналіз продуктивності та ефективності апаратної реалізації у порівнянні з програмними методами.

Апаратна реалізація алгоритмів стиснення зображень дозволяє суттєво підвищити швидкість обробки даних та знизити навантаження на процесорні ресурси. Використання HDL дає змогу створювати ефективні цифрові модулі, які можуть бути інтегровані в FPGA (програмованих вентильних матрицях) або ASIC (спеціалізованих інтегральних схемах) для розв’язання задач реального часу. Отримані результати підтверджують доцільність використання апаратних методів у системах цифрової обробки зображень.

ЗМІСТ

ВСТУП

У сучасному світі цифрових технологій обробка та передача зображень є важливими аспектами у багатьох сферах, таких як телекомунікації, відеоспостереження, мобільні пристрої, медична діагностика та штучний інтелект. Оскільки роздільна здатність зображень постійно зростає, виникає потреба у ефективному стисненні, яке дозволяє зменшити обсяг переданих і збережених даних без значної втрати якості.

Одним із перспективних підходів до реалізації алгоритмів стиснення є використання мов опису апаратури (HDL – Hardware Description Language) для реалізації на програмованих логічних інтегральних схемах FPGA (Field-Programmable Gate Array) або спеціалізованих інтегральних схемах ASIC (Application-Specific Integrated Circuit).

Апаратна реалізація алгоритмів стиснення в HDL має низку переваг у порівнянні з традиційними програмними методами, що виконуються на CPU або GPU. До основних переваг належать:

- Підвищена швидкодія за рахунок паралельної обробки даних (зменшення затримок у реальному часі).
- Зниження енергоспоживання, що особливо важливо для мобільних пристроїв, вбудованих систем та технологій IoT.
- Ефективне використання ресурсів пам’яті завдяки спеціалізованим схемам стиснення.
- Можливість роботи в реальному часі, що критично для відеоспостереження, супутникових систем, автономного транспорту та безпілотних літальних апаратів.

Використання апаратних методів стиснення дозволяє не лише оптимізувати швидкість роботи пристроїв, але й забезпечує ефективне передавання даних у системах, де пропускна здатність каналів зв’язку є обмеженою. Це особливо актуально для телемедицини, військових технологій, супутникового зв’язку та обробки відеопотоків у сучасних системах відеоаналітики.

Таким чином, розробка та впровадження алгоритмів стиснення зображень в HDL є актуальним завданням, що сприяє підвищенню продуктивності, зменшенню витрат ресурсів та відкриває нові можливості для застосування у високотехнологічних галузях.

1. ПРОБЛЕМА СТИСНЕННЯ ЗОБРАЖЕНЬ

Класифікація зображень є важливою для оцінки ефективності алгоритмів стиснення. Під класом зображень розуміється група зображень, для яких застосування певного алгоритму архівації дає подібні результати. Наприклад, один клас може характеризуватися високим рівнем стиснення, інший – мінімальним скороченням розміру, а третій – навіть збільшенням файлу після архівації.

Виділяють такі основні класи зображень:

- **Клас 1**: Зображення з обмеженою кількістю кольорів (4-16) і великими однорідними кольоровими областями без плавних переходів. *Приклади:* діаграми, графіки, гістограми.
- **Клас 2**: Комп'ютерна графіка з плавними градієнтами кольорів. *Приклади:* презентаційна графіка, ескізні моделі в САПР, зображення, створені методом Гуро.
- **Клас 3**: Фотореалістичні зображення. *Приклади:* скановані фотографії.
- **Клас 4**: Поєднання фотореалістичних зображень із діловою графікою. *Приклад:* рекламні матеріали.

Визначення найефективнішого алгоритму стиснення для кожного класу залишається актуальним завданням. Особливий інтерес дослідників зосереджений на фотореалістичних зображеннях (клас 3), оскільки їх стиснення є найбільш ресурсомістким і викликає значні труднощі. У порівнянні з ними, комп'ютерна графіка може бути ефективно стиснута шляхом збереження вихідних файлів, а зображення з обмеженою кількістю кольорів легко піддаються стисненню завдяки низькій обчислювальній складності відповідних алгоритмів.

1.1 Алгоритми архівації та їх властивості

Поява алгоритмів стиснення зображень зумовлена специфічними особливостями графічних даних, які відрізняють їх від текстової інформації. Основні характеристики зображень, що впливають на необхідність архівації, включають:

a) **Великий обсяг даних** – зображення займають значно більше пам’яті порівняно з текстом, що робить їхнє стиснення актуальним завданням.

b) **Особливості людського зору** – люди сприймають зображення через контури та загальні переходи кольорів, тоді як дрібні зміни залишаються майже непомітними. Це дозволяє використовувати алгоритми стиснення, в яких відновлене зображення може не співпадати з оригіналом, але візуальна якість залишатиметься прийнятною.

c) **Надмірність у двох вимірах** – на відміну від тексту, зображення мають високу кореляцію між сусідніми пікселями як по горизонталі, так і по вертикалі. Крім того, існує подібність між колірними складовими (R, G, B), що дозволяє застосовувати ще ефективніші алгоритми стиснення.

Важливо враховувати, що ефективність алгоритму стиснення залежить від умов його використання. Зокрема, ступінь компресії визначається типом зображень, на яких він застосовується.

1.2 Основні алгоритми стиснення зображень

Методи стиснення зображень можна поділити на кілька категорій залежно від типу зображення: бінарні, півтонові та кольорові. Крім того, стиснення може відбуватися як із втратами інформації, так і без них. Розглянемо детальніше методи без втрат.

Методи стиснення без втрат

1. Факсовий метод

Цей метод використовується у факсимільних апаратах і ґрунтується на кодуванні змін кольорів між сусідніми пікселями в рядку. Перехід кодується одиницею, а відсутність змін – нулем (рис. 1.1). Метод добре працює для зображень із великими однорідними ділянками, що робить його ефективним для оптимізованого кодування.

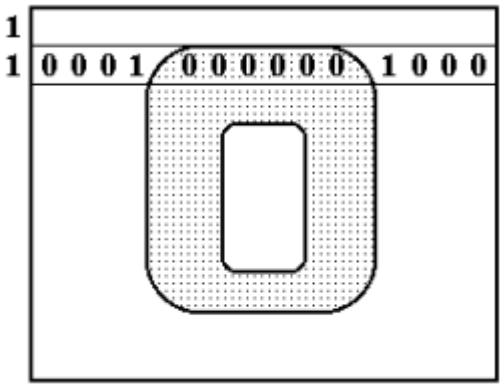

Рисунок 1.1 – Приклад кодування

Недоліки: можливе виникнення спотворень у вигляді чорних або білих смуг у разі помилок, а також не враховується двовимірна структура зображення.

2. RLE метод

Метод кодування довжин повторів (Run-Length Encoding) працює шляхом запису кольору пікселя та кількості його повторень у рядку до зміни кольору (рис. 1.2). Він широко використовується для стиснення BMP-зображень.

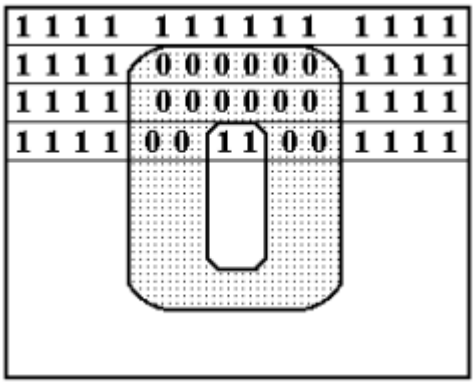

Рисунок 1.2 – Приклад кодування RLE методом

Переваги: проста реалізація, швидкість роботи, мінімальне споживання пам’яті під час архівації та розархівації.

Недоліки: неефективність для зображень із плавними градієнтами, що може призвести до збільшення обсягу даних.

3. Кодування кутами Шлезенгера

Цей метод використовується для стискання чорно-білих зображень. Спочатку додається рядок і стовпець нулів, після чого зображення розбивається на групи по чотири пікселі. Всього можливі 16 варіантів таких четвірок (рис. 1.3).

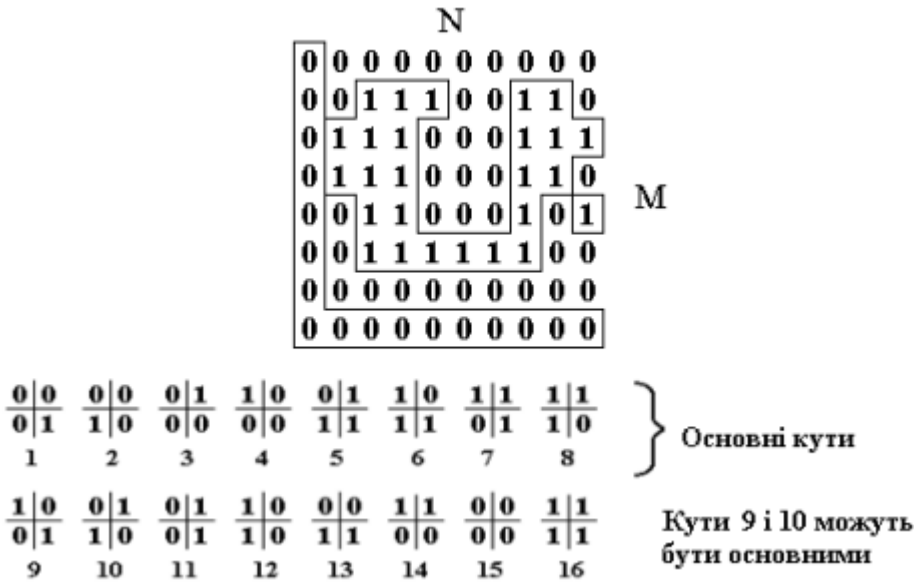

Рисунок 1.3 – Типи кутів Шлезенгера

З вихідного растрового зображення розміром (N +1) × (M +1) формується масив розміром N × M біт (рис. 1.4, а).

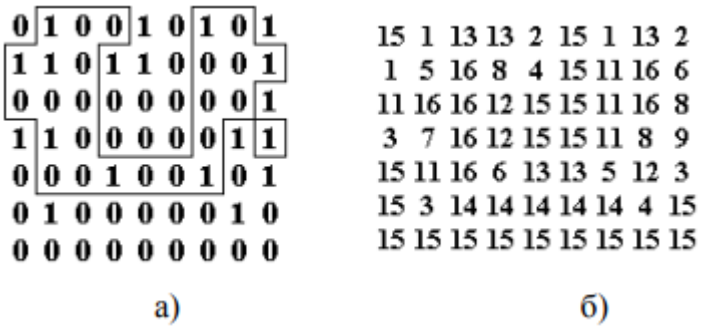

Рисунок 1.4 – Результати кодування методом кутів Шлезенгера

Далі проводиться виокремлення кутів, після чого вони записуються у вигляді послідовності (рис. 1.4, б).

Наступний етап – аналіз цієї послідовності. Під час сканування зліва направо та зверху вниз, якщо зустрічається кут №1, то обов’язково за ним слідує кут №2. Тому для кодування використовуються лише основні кути.

Зображення розміром N × M, де кожен піксель представлений одним бітом, замінюється даними про основні кути: значення "1" присвоюється дуальному (основному) куту, а значення "0" – недуальному (неосновному).

Висновки:

- Зменшується надлишковість даних.
- Метод ефективний для зображень з великими однорідними ділянками (наприклад, креслення).
- Забезпечується повна відновлюваність зображення.
- Зниження ентропії сприяє кращому ступеню стиснення.

4. Wavelet – хвильове перетворення

Метод заснований на пропусканні вихідного сигналу через систему фільтрів низьких і високих частот (рис. 1.5). У результаті отримується представлення сигналу у вигляді ортогональних функцій, відомих як вейвлети.

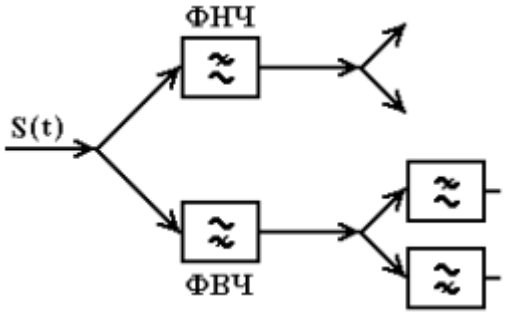

Рисунок 1.5 – Схема Wawelet-перетворення

Особливості:

- Використовуються ортогональні фільтри.

- Забезпечується ефективне розкладання сигналу, що дозволяє зменшити обсяг даних без втрати інформації.

5. S-перетворення

Цей метод працює з послідовністю значень S(i), що знаходяться в межах $[S_{MIN}, S_{MAX}]$. Процес перетворення виконується за певною схемою (рис. 1.6), що дозволяє зменшити ентропію вихідного зображення.

Далі необхідно здійснити перетворення за схемою, що зображена на рис. 1.6.

Рисунок 1.6 – Схема S-перетворень

Результати (рис. 1.7):

- Менша ентропія сприяє ефективнішому стисненню без втрат.
- Метод особливо ефективний для природних зображень, де сусідні пікселі мають подібні характеристики.
- Операція є симетричною та зворотною, що гарантує безпомилкове відновлення зображення.

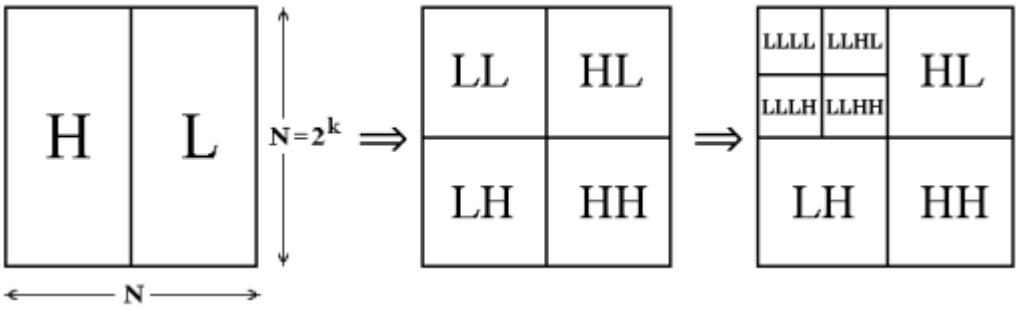

Рисунок 1.7 – Результати S-перетворення

Ці методи дозволяють ефективно зменшити обсяг зображень без втрати якості, вибір конкретного методу залежить від особливостей самого зображення.

Методи стиснення із втратами

При стисненні зображень не завжди застосовують безвтратні методи, оскільки особливості зорового сприйняття людини дозволяють не помічати незначні зміни кольорів або відтінків. Для оцінки якості стискання використовують такі параметри:

1. Середня похибка:

$$\delta_{CP} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\frac{\Delta ij}{Xij}. \quad (1.1)$$

де Δij – різниця в числових характеристиках піксела, що стоїть в і рядку і в j стовпці вихідного зображення і стисненого;

Xij – значення піксела вихідного зображення в точці (i, j).

Оцінити середню помилку на зображенні можна за формулою:

$$\delta_{CP} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\frac{\Delta ij}{X_{MAX}}. \quad (1.2)$$

2. Максимальна помилка на зображенні:

$$\Delta_{MAX} = \max_{\substack{i=1...N \\ j=1...M}} \left(\frac{\Delta ij}{X_{MAX}} \right). \quad (1.3)$$

Основна перевага методів стиснення з втратами полягає у значно вищому коефіцієнті стискання, що робить їх широко застосовуваними у сфері Інтернету, телекомунікацій і супутникового зондування.

1. Метод JPEG

Метод JPEG базується на двовимірному косинусному перетворенні. Якщо вихідна функція не є парною чи непарною, то її Фур'є-перетворення включає як косинусні, так і синусні компоненти. Щоб зробити її парною, застосовують симетричне доповнення. Отриманий косинусний спектр функції наведено на рисунку 1.8.

Рисунок 1.8 – Вихідна функція стиснення

Стиснення виконується шляхом відкидання високочастотних складових спектра, які мають малі амплітуди, що спричиняє розмиття (рис. 1.9).

Рисунок 1.9 - Результат обробки

Цей метод є найпоширенішим через простоту реалізації. Коефіцієнт стискання в JPEG може варіюватися від 10 до кількох сотень. Найчастіше його застосовують для обробки зображень природного походження.

2. Фрактальний метод стиснення

Фрактальне стискання базується на поділі зображення на схожі ділянки (фрактали), частина яких потім відкидається (рис. 1.10).

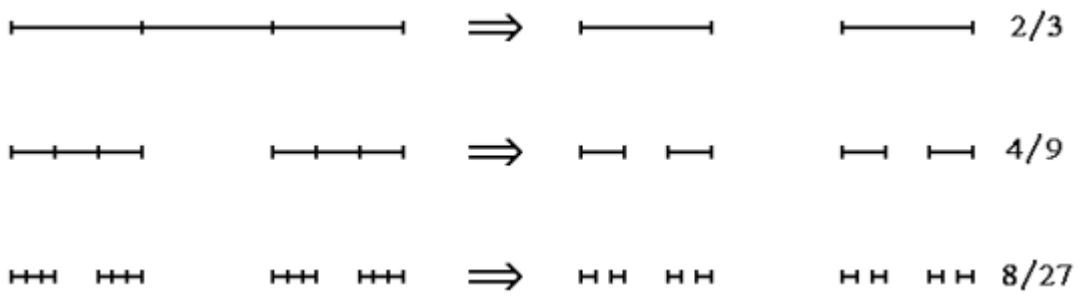

Рисунок 1.10 – Поділ зображення на фрактали

У результаті формується множина кінцевих точок відрізків – канторова множина (2^n / 3^n). Фрактал є множиною точок, отриманою на основі канторової множини, хоча не всі фрактали належать до неї. Основна властивість фракталів – самоподібність.

Процес стискання передбачає пошук самоподібних областей на зображенні, які використовуються як базові елементи. Зображення апроксимується фрактальними перетвореннями і подається у вигляді математичної моделі.

Попри те, що цей метод має певні переваги, він не набув широкого розповсюдження через такі недоліки:

- Відсутність універсального автоматичного алгоритму для всіх випадків.

- Значна залежність точності стискання від часу обчислень (для якісного результату потрібен тривалий процес стискання).
- Обмежена сфера застосування – використовується лише там, де є можливість довготривалого опрацювання сигналів для отримання компактного представлення даних.

3. SPITH-перетворення

Цей метод ґрунтується на Wavelet-перетворенні. Спочатку відбувається нормування зображення за певною схемою, після чого кожен піксель множиться на визначені коефіцієнти (рис. 1.11).

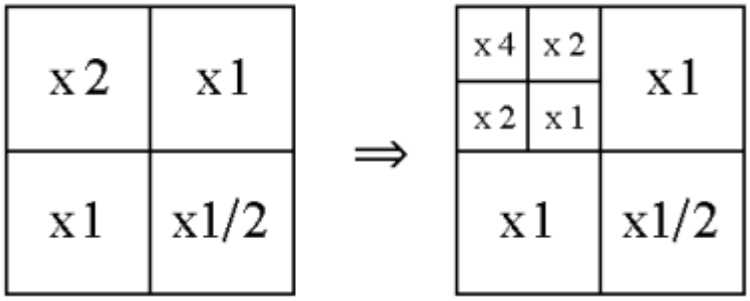

Рисунок 1.11 – Принцип SPITH-перетворення

Далі відкидаються коефіцієнти, значення яких менше за встановлений поріг.

Ця процедура повторюється, поки зображення не досягне бажаного ступеня стискання або якість не почне суттєво погіршуватися.

Основна перевага цього методу – менші втрати якості порівняно з JPEG при однаковому коефіцієнті стискання.

2. JPEG – ЕФЕКТИВНИЙ АЛГОРИТМ СТИСНЕННЯ ЗОБРАЖЕНЬ

JPEG є одним із сучасних та потужних алгоритмів стиснення зображень. Він працює з блоками розміром 8×8 пікселів, де колір і яскравість змінюються плавно. Завдяки цьому під час косинусного перетворення значущими залишаються лише перші коефіцієнти, що дозволяє суттєво зменшити обсяг даних. Стиснення в JPEG базується на особливостях зорового сприйняття людини, яке не помічає незначних змін кольору.

Алгоритм був розроблений групою експертів у галузі фотографії спеціально для 24-бітових зображень. Він базується на дискретному косинусному перетворенні (ДКП), яке застосовується до матриці пікселів, перетворюючи її у матрицю коефіцієнтів. Відновлення зображення здійснюється через зворотне перетворення. У результаті ДКП розкладає зображення на частотні компоненти, де більшість коефіцієнтів виявляються близькими до нуля або дорівнюють нулю. Оскільки людське око не сприймає дрібні деталі з високою точністю, можна спрощувати ці коефіцієнти без помітної втрати якості.

Оскільки обчислення множення для великих матриць є складним, весь процес не можна виконати одразу на всьому зображенні. Тому воно розбивається на невеликі блоки по 8×8 пікселів, для яких визначаються частотні складові, а потім відкидаються менш значущі значення. Отримані дані записуються у стислому вигляді для збереження пам’яті.

Основні етапи алгоритму JPEG:

1. Перехід у колірну модель YCbCr – яскравість і кольорові компоненти обробляються окремо.
2. Субдискретизація кольорових компонентів – групи пікселів усереднюються для зменшення обсягу кольорової інформації.

3. Дискретне косинусне перетворення (ДКП) – перехід до частотного представлення.
4. Квантування – округлення коефіцієнтів для зменшення обсягу даних.
5. «Зігзаг» сканування – впорядкування коефіцієнтів у порядку зменшення значущості.
6. Первинне стиснення – зменшення розміру даних через кодування повторюваних значень.
7. Вторинне стиснення – застосування додаткових методів стиснення для ще більшої оптимізації.

Етапи алгоритму JPEG також можна представити у вигляді схеми, що зображена на рисунку 2.1.

Рисунок 2.1 – Схема етапів роботи JPEG

2.1 Аналіз методів вирішення науково-технічної задачі з використанням популярних алгоритмів

Науково-технічні задачі часто вимагають ефективних методів обробки та стиснення інформації. У сфері цифрової обробки зображень та відео широко застосовуються алгоритми стиснення, що забезпечують баланс між якістю та ефективністю передачі й зберігання даних. Серед найбільш популярних алгоритмів можна виокремити JPEG, JPEG2000 та HEVC/HEIC.

Огляд популярних алгоритмів

1. JPEG (Joint Photographic Experts Group)

JPEG є одним із найпоширеніших методів стиснення зображень. Він застосовує дискретне косинусне перетворення (DCT), що дозволяє ефективно усувати непомітні для людського ока деталі.

Основні особливості:

- Використання втратного стиснення для економії місця.
- Можливість регулювання рівня стиснення.
- Висока ефективність для фотографічних зображень.
- Недоліком є можливість появи артефактів при сильному стисненні.

2. JPEG2000

JPEG2000 є покращеною версією стандарту JPEG, яка застосовує дискретне вейвлет-перетворення (DWT) замість DCT.

Основні особливості:

- Покращене співвідношення стиснення та якості.
- Підтримка як втратного, так і безвтратного стиснення.
- Краща гнучкість при роботі з високоякісними та великими зображеннями.
- Вищі обчислювальні вимоги порівняно з JPEG.

3. HEVC/HEIC (High Efficiency Video Coding / High Efficiency Image Coding)

HEVC є сучасним стандартом стиснення відео, тоді як HEIC (заснований на HEVC) використовується для зображень. Цей алгоритм застосовує блокову структуру з адаптивним кодуванням, що значно зменшує розмір файлів без значних втрат якості.

Основні особливості:

- Висока ефективність стиснення у порівнянні з JPEG та JPEG2000.
- Використання для збереження високоякісних зображень та відео.
- Покращена підтримка HDR та глибокого кольору.
- Потребує сучасного обладнання для декодування.

Висновки

Алгоритми стиснення зображень та відео мають велике значення у вирішенні науково-технічних задач, пов'язаних із обробкою даних. Вибір конкретного методу залежить від потреб щодо якості, продуктивності та апаратних можливостей. JPEG підходить для базових завдань, JPEG2000 забезпечує вищу якість та універсальність, а HEVC/HEIC надає найкращу ефективність стиснення для сучасних мультимедійних додатків.

2.2 Представлення кольору YCbCr у порівнянні з RGB

Колірна модель YCbCr є ближчою до природного сприйняття кольору людиною, ніж традиційна RGB. У цій моделі компонент Y відповідає за яскравість і значною мірою визначає якість зображення. Фактично, Y-компонента є чорно-білим варіантом зображення. Дві інші компоненти, Cb та Cr, містять інформацію про кольори та використовуються для його відтворення.

Перетворення зображення з простору RGB (де кольори представлені червоною (Red), зеленою (Green) та синьою (Blue) складовими) у простір YCbCr називається ICT (Irreversible Color Transform). Візуальний приклад такого перетворення зображений на рисунку 2.2.

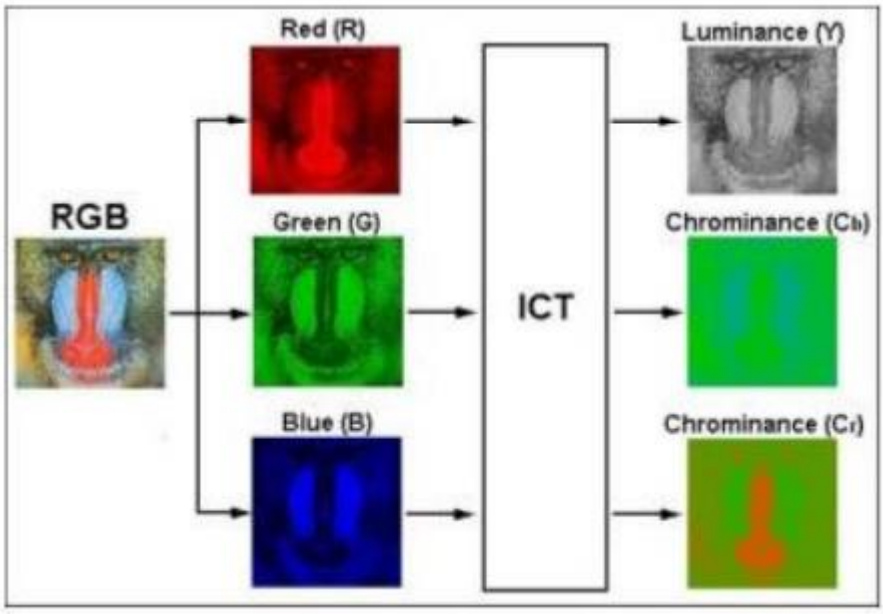

Рисунок 2.2 – Приклад роботи ICT-перетворення

Людське око значно чутливіше до яскравості (Y-компоненти), ніж до кольорових складових. Це пов’язано з будовою сітківки, яка містить два типи рецепторів:

- Паличкоподібні клітини (палички) – відповідають за сприйняття яскравості та є значно чутливішими до світла.
- Колбочкоподібні клітини (колбочки) – сприймають кольори, але менш чутливі до світлових змін.

Оскільки паличок набагато більше, ніж колбочок, і вони сильніше реагують на зміни освітлення, людина здатна добре сприймати деталі яскравості, навіть якщо кольорові складові будуть зменшені або частково втрачені. Саме тому під час стиснення можна зменшити кількість кольорових даних без суттєвого впливу на сприйняття якості зображення.

Подібне кодування давно застосовується в телебаченні: колірна інформація передається у вужчому діапазоні частот, оскільки яскравість є важливішою для людського зору.

Переведення з колірного простору RGB в колірний простір YCbCr можна представити за допомогою матриці переходу[5]:

$$\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} = \begin{vmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.4187 & -0.0813 \\ 0.1687 & -0.3313 & 0.5 \end{vmatrix} \times \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix}. \quad (2.1)$$

Обернене перетворення реалізується множенням вектора YCrCb на обернену матрицю:

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{vmatrix} \times \left(\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right). \quad (2.2)$$

2.3 Субдискретизація кольорових компонентів

Менша чутливість людського ока до кольорової інформації дозволяє оптимізувати зберігання зображень шляхом зменшення кількості пікселів у колірних компонентах. Цей процес, відомий як субдискретизація, передбачає об’єднання кольорових даних для сусідніх пікселів, що зменшує загальний обсяг інформації без значної втрати якості.

Для цього створюються окремі матриці значень для кожної з компонент:

- Для Y-компоненти (яскравості) значення залишаються відповідними положенню кожного пікселя.
- Для Cr і Cb (колірних складових) значення формуються через рядок та стовпець, що дозволяє зменшити деталізацію кольору.

Таким чином, кожен піксель колірності охоплює область 2×2 пікселів яскравості. У результаті для кожного такого блоку зберігається лише 6 значень (4 для яскравості та по 1 для Cr і Cb), тоді як у повному представленні потрібно було б 12 значень. Це забезпечує двократне стиснення кольорової інформації, оскільки втрачається 75% даних про колірні компоненти.

Однак цей метод ефективний тільки для колірного простору YCbCr, де яскравість і колір розділені. У традиційній RGB-моделі такий підхід неможливий, оскільки кожен канал містить як яскравість, так і колірну інформацію, а будь-яке її скорочення значно вплине на якість зображення.

2.4 Дискретне косинусне перетворення (ДКП) в алгоритмі стиснення

Дискретне косинусне перетворення (ДКП) є ключовим етапом стиснення зображень, представляючи собою варіант перетворення Фур'є. Як і у випадку з перетворенням Фур'є, для ДКП існує зворотне перетворення (ОДКП), що дозволяє відновити вихідне зображення.

Якщо розглядати зображення як сукупність просторових хвиль, де осі X і Y відповідають його ширині та висоті, а значення кольору відображаються вздовж осі Z, можна здійснити перехід від просторового представлення до спектрального та назад. Дискретне косинусне перетворення (ДКП) перетворює матрицю розміром N×N у матрицю частотних коефіцієнтів такого ж розміру.

Розглянемо формули ДКП:

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2x+1)j\pi}{2N}\right],$$

$$f(x,y) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j)DCT(i,j) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2x+1)j\pi}{2N}\right],$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}}, x = 0, \\ 1, x > 0. \end{cases} \quad (2.3)$$

Застосовуючи ДКП до кожної матриці зображення, отримуємо матрицю, у якій низькочастотні компоненти розташовані у верхньому лівому куті, а високочастотні – зміщуються донизу та вправо. Оскільки більшість графічних

об’єктів містить саме низькочастотну інформацію, можна відсіяти менш значущі високочастотні компоненти, зменшуючи обсяг даних із мінімальними втратами якості. Це дозволяє алгоритму ефективно відкидати малопомітні деталі, незначно впливаючи на візуальне сприйняття зображення.

Обчислення кожного елемента матриці ДКП потребує $O(N^2)$ часу, що робить неможливим виконання операцій для всієї матриці одночасно. Для розв’язання цієї проблеми алгоритм JPEG використовує блоковий підхід: вихідне зображення ділиться на стандартні блоки розміром 8×8 , і ДКП застосовується до кожного з них окремо.

Хоча використання більших блоків могло б покращити ефективність стиснення, існує обмеження: ймовірність подібності віддалених точок у зображенні досить низька, що обмежує доцільність збільшення розміру блоків.

Щоб пришвидшити обчислення, використовується лише 32 попередньо розраховані значення косинусів, що значно скорочує час виконання ДКП. Це призводить до часткової втрати інформації, але загалом вона залишається несуттєвою.

Додаткове підвищення продуктивності можливе завдяки використанню цілочисельної арифметики. Проте для сучасних комп’ютерів це неактуально, оскільки операції над числами з плаваючою комою виконуються майже так само швидко, як і над цілими числами. До того ж, використання цілочисельних обчислень може погіршити якість зображення після стиснення, що робить цей метод непридатним для сучасних систем.

Оскільки ДКП є різновидом перетворення Фур’є, всі методи оптимізації, розроблені для швидкого перетворення Фур’є, можуть бути застосовані й тут.

Для побудови матриці ДКП можна скористатися таким псевдокодом:

```
DCT    = 1/sqr(N), якщо i=0
          ij
         DCT    = sqr(2/N)*cos[(2j+1)*i*3.14/2N], якщо i > 0
          ij
         N = 8,  0 < i < 7 , 0 < j < 7
```

В результаті маємо:

$$DCT = \begin{vmatrix} 0.353553 & 0.353553 & 0.353553 & 0.353553 & 0.353553 & 0.353553 & 0.353553 & 0.353553 \\ 0.490393 & 0.415818 & 0.277992 & 0.97887 & -0.97106 & -0.277329 & -0.415375 & -0.490246 \\ 0.461978 & 0.191618 & -0.190882 & -0.461673 & -0.462282 & -0.192353 & 0.190145 & 0.461366 \\ 0.414818 & -0.097106 & -0.490246 & -0.278653 & 0.276667 & 0.490710 & 0.099448 & -0.414486 \\ 0.353694 & -0.353131 & -0.354256 & 0.352567 & 0.354819 & -0.352001 & -0.355378 & 0.351435 \\ 0.277992 & -0.490246 & 0.096324 & 0.416700 & -0.414486 & -0.100228 & 0.491013 & -0.274673 \\ 0.191618 & -0.462282 & 0.461366 & -0.189409 & -0.193822 & 0.463187 & -0.460440 & 0.187195 \\ 0.097887 & -0.278653 & 0.416700 & -0.490862 & 0.489771 & -0.413593 & 0.274008 & -0.092414 \end{vmatrix}.$$

Остаточно ДКП реалізується через множення матриць за такою формулою:

$$P_{DCT} = DCT \times P \times DCT^T, \qquad (2.4)$$

де:

P – початковий блок зображення (8×8),

PDCT – блок після ДКП,

DCT – матриця косинусного перетворення,

DCT^T – транспонована матриця ДКП.

2.5 Квантування в процесі стиснення зображень

Квантування полягає в поелементному поділі робочої матриці на матрицю квантування. Для кожної з колірних компонент (Y, Cr, Cb) використовується окрема матриця квантування Q[cr,cb].

На цьому етапі відбувається керування рівнем стиснення, а також втрачається найбільша частка інформації. Чим більші коефіцієнти у матриці квантування, тим більше нульових значень отримується, що призводить до вищого ступеня стиснення.

Оскільки у більшості реальних зображень високочастотні компоненти зустрічаються рідко, їх можна кодувати менш детально. Тут важливу роль відіграє двовимірне представлення результатів дискретного косинусного перетворення (DCT).

Матриця квантування розраховується за таким псевдокодом:

```
for(i=0;i<8;i++)
{
   for(j=0;j<8;j++)
        Q[i][j] = 1+((1+i+j)*q);
}
```

Тут q – коефіцієнт якості, який визначає ступінь втрати деталей у зображенні. Чим більше значення q, тим більше інформації буде втрачено, оскільки більше коефіцієнтів після округлення набуватимуть нульового значення. Наприклад, при q=2 отримується конкретна матриця квантування.

$$Q = \begin{vmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{vmatrix}.$$

Кожен елемент матриці PDCTPDCTPDCT (отриманої після DCT) ділиться на відповідний елемент у матриці квантування. У разі отримання дробових значень, їх слід правильно округлювати до цілих чисел. Після зворотного перетворення відхилення від початкових значень не перевищуватиме 10%.

Цей етап завершує втрату інформації, а подальші перетворення вже не спричиняють змін у даних. Однак квантування може викликати характерні артефакти:

- При великих значеннях коефіцієнта q можлива значна втрата низькочастотних компонент, що призведе до ефекту поділу зображення на блоки 8×8.
- Втрати у високочастотних складових можуть викликати так званий "ефект Гіббса" – появу світлих або темних ореолів навколо контурів з різким перепадом кольору.

Таким чином, вибір коефіцієнта квантування є критично важливим для досягнення балансу між ступенем стиснення та якістю зображення.

2.6 «Зігзаг» сканування

Після квантування у матриці з'являється значна кількість нульових значень. Щоб отримати найдовшу послідовність нулів, застосовується спеціальна схема обходу матриці.:

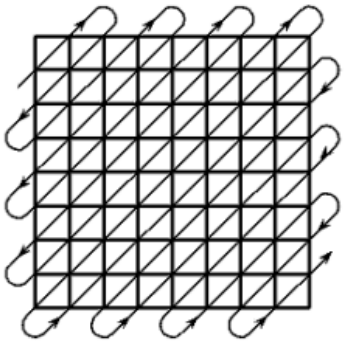

Рисунок 1.3 – Оптимальна схема обходу матриці для подовження послідовності нулів.

Завдяки такому порядку зчитування спочатку розташовуються коефіцієнти, що відповідають низькочастотним компонентам, а високочастотні значення опиняються ближче до кінця. Це дозволяє отримати безперервні послідовності з нульових елементів, які можуть досягати до 30 значень підряд.

2.7 Первинне стиснення

Обробка даних виконується з метою підвищення ефективності стиснення й здійснюється перед самим процесом стискання. У такому випадку загальна схема кодування виглядає наступним чином: **препроцесор → кодер → стиснення**.

Завдання препроцесора – змінити структуру вхідного потоку таким чином, щоб коефіцієнт стиснення перетворених даних був, у середньому, вищим за коефіцієнт стиснення початкових даних. При цьому система **препроцесор–постпроцесор** працює незалежно: кодер не має інформації про те, що отримані ним дані вже були змінені. Постпроцесор відновлює вихідні дані без втрат, тому наявність цієї системи залишається непомітною для кінцевого користувача.

Для згортання вектора використовується **групове кодування**, наприклад, метод **RLE (Run-Length Encoding)** [9]. У результаті перетворення утворюються пари значень **(пропуск, число)**, де *пропуск* – це кількість нулів, які потрібно пропустити, а *число* – значення, що слід записати в наступну комірку.

Наприклад, для вектора: **42 3 0 0 0 -2 0 0 0 0 1 ...** після застосування RLE він набуває вигляду: **(0,42) (0,3) (3,-2) (4,1) ....**

2.8 Вторинне стиснення

Основна ідея алгоритму стиснення Хаффмана полягає в тому, що, знаючи ймовірність появи символів у повідомленні, можна побудувати коди змінної довжини, які складаються з цілих бітів. Символи з більшою ймовірністю отримують коротші коди. Коди Хаффмана є унікальними префіксними кодами, що дозволяє їх одночасно декодувати, незважаючи на змінну довжину. Динамічний алгоритм Хаффмана отримує на вході таблицю частот символів у повідомленні, на основі якої будується дерево кодування Хаффмана (Н-дерево).

Алгоритм виглядає наступним чином:
а) Символи вхідного алфавіту формують список вільних вузлів, де кожен лист має вагу, яка дорівнює ймовірності або кількості входжень символу у повідомлення.
б) Обираються два вузли з найменшими вагами.
в) Створюється предок, вага якого дорівнює сумі ваг двох обраних вузлів.
г) Цей предок додається до списку вільних вузлів, а двоє його дітей видаляються з цього списку.

г) До кожної з дуг предка присвоюються біти: 1 для однієї, 0 для іншої. д) Повторюються пункти 2-5 до тих пір, поки не залишиться лише один вузол, який стає коренем дерева.

Недоліком класичного алгоритму Хаффмана є необхідність двократного проходу по даних, що збільшує час стиснення вдвічі. Адаптивне стиснення Хаффмана дозволяє уникнути передачі моделі повідомлення і здійснити стиснення за один прохід.

У класичному алгоритмі Хаффмана символи, які не зустрічаються в повідомленні, відомі заздалегідь, оскільки існує таблиця частот. У адаптивному алгоритмі ці символи невідомі до їх появи в потоці. Для вирішення цієї проблеми дерево Хаффмана ініціалізується з усіма 256 символами (для 8-бітових кодів) з початковою частотою 1. Спочатку кожен символ кодується 8 бітами, але з часом найбільш часто вживані символи кодуються коротшими кодами. Такий підхід може знижувати ступінь стиснення, особливо на коротких повідомленнях. Одним із рішень є початок моделювання з порожнього дерева, до якого додаються символи по мірі їх появи. Проте це призводить до проблеми: коли символ з'являється вперше, його неможливо закодувати, оскільки його ще немає в дереві. Для вирішення цього питання вводиться спеціальний ESCAPE код, що сигналізує про те, що наступний символ передається без кодування, поза контекстом моделі повідомлення.

Переваги алгоритму JPEG:

- можливість контролювати ступінь стиснення;
- вихідне кольорове зображення може мати 24 біти на піксель.

Недоліки алгоритму:

- при високому ступені стиснення зображення може розпастися на квадрати 8x8 через великі втрати на низьких частотах під час квантування, що робить відновлення даних неможливим;
- ефект Гіббса — виникнення ореолів навколо різких переходів кольорів.

У дипломній роботі автор розглянув алгоритми стиснення зображень, зокрема JPEG та його реалізацію в HDL. Основні етапи стиснення були проаналізовані та реалізовані у вигляді апаратного рішення на Verilog/VHDL.

1. Перетворення колірного простору (YCbCr замість RGB)

- Автор досліджує перехід від RGB до YCbCr, оскільки людське око чутливіше до яскравості (Y), ніж до кольорових складових (Cb, Cr).
- Це дозволяє зменшити обсяг даних за рахунок субдискретизації кольорових компонентів.

2. Розбиття зображення на блоки 8×8 пікселів

- JPEG працює з блоками розміром 8×8, що дозволяє ефективно застосовувати дискретне косинусне перетворення (DCT).
- Автор реалізував цей процес у HDL, що забезпечує паралельну обробку блоків.

3. Дискретне косинусне перетворення (DCT)

- DCT переводить блоки пікселів із просторової області у частотну область, де інформація розділяється на низькочастотні та високочастотні компоненти.
- У роботі описані формули прямого та зворотного DCT, а також реалізація в HDL.

- Автор використав матрицю DCT-коефіцієнтів для оптимізації обчислень.

4. Квантування

- Після DCT відбувається квантування, тобто округлення значень для зменшення обсягу даних.
- Автор реалізував матриці квантування у HDL, що дозволяє зменшити кількість значущих коефіцієнтів та підвищити рівень стиснення.
- Чим більший коефіцієнт квантування, тим вищий рівень стиснення, але більше втрат якості.

5. «Зігзаг»-сканування

- Автор реалізував спеціальну схему обходу матриці після квантування, щоб зібрати нульові значення в кінці послідовності.
- Це дозволяє підвищити ефективність ентропійного кодування.

6. Ентропійне кодування (Хаффмана або арифметичне кодування)

- Заключним етапом є застосування алгоритму Хаффмана, який стискає дані без втрат, використовуючи код змінної довжини.
- У роботі розглянуто та реалізовано апаратне кодування Хаффмана для оптимального стискання результатів DCT.

3. РЕАЛІЗАЦІЯ JPEG В HDL

JPEG (Joint Photographic Experts Group) – це один з найпопулярніших методів стиснення зображень. Реалізація цього алгоритму на апаратному рівні (за допомогою мов опису апаратури, таких як VHDL або Verilog) дозволяє досягти високої швидкості обробки зображень у реальному часі.

3.1 Алгоритм роботи

Спочатку зображення розбивається на блоки у вигляді матриць розміром 8×8 пікселів. Оскільки метод JPEG передбачає, що час, витрачений на стиснення, пропорційний квадрату кількості пікселів у блоці, обробка кількох менших блоків значно швидша, ніж стиснення всього зображення цілком.

До значень пікселів застосовується дискретне косинусне перетворення (Discrete Cosine Transform — DCT), яке перетворює матрицю розміром 8×8 із значеннями яскравості у відповідну матрицю амплітудних коефіцієнтів, що відповідають різним частотам синусоїдальних коливань. Лівий верхній кут цієї матриці містить коефіцієнти для низьких частот, а правий нижній — для високих.

Введений користувачем коефіцієнт якості використовується у формулі для створення іншої матриці 8×8, відомої як матриця квантування. Чим нижчий коефіцієнт якості, тим більшими будуть значення елементів цієї матриці.

Кожен елемент отриманої після DCT матриці ділиться на відповідний елемент матриці квантування та округлюється до найближчого цілого числа. Оскільки великі значення матриці квантування розташовані в її нижній правій частині, значна частина високочастотної інформації зображення відсікається, що призводить до появи багатьох нулів у нижньому правому куті матриці пікселів.

Далі програма виконує зчитування елементів матриці у зигзагоподібному порядку та кодує їх методами без втрат. Важливо зазначити, що ефективність стиснення значною мірою залежить від кількості нульових значень у правій нижній частині матриці — чим нижчий коефіцієнт якості, тим більше нулів, а отже, вищий рівень стиснення.

Процес декодування JPEG-зображення розпочинається із зворотного кодування без втрат, у результаті чого відновлюється матриця квантування пікселів. Далі значення матриці пікселів перемножуються на відповідні значення матриці квантування, що дозволяє частково відновити початкову матрицю, отриману після застосування DCT. Однак через втрати під час квантування отримані значення лише наближені до початкових, а не точно відтворюють їх.

На завершальному етапі використовується зворотне дискретне косинусне перетворення (Inverse Discrete Cosine Transform — IDCT), яке перетворює матрицю назад у значення пікселів вихідного зображення. Через втрати інформації на етапі квантування кольори отриманого зображення можуть дещо відрізнятися від оригіналу, а саме зображення може виглядати трохи розмитим та менш насиченим.

3.3 Тестування коду

Для тестування достовірності результатів програми, я розробив матрицю 8×8, де множаться ці матриці.

Для обміну інформацією між процесами використовуються функції бібліотеки Code 1. Це функція, яка призначена відповідно для результату схеми .

Цей код написаний мовою Verilog і реалізує модуль **code1**, який виконує множення двох 8×8 матриць (matA та matB), що надходять у вигляді 64-

бітних вхідних сигналів (A і B). В результаті обчислення формується вихідна матриця (matC), яка також представляється у вигляді 64-бітного вихідного сигналу (C).

Опис вхідних та вихідних сигналів

- **Clock (вхід)** – тактовий сигнал, що керує виконанням операцій у модулі.
- **reset (вхід)** – асинхронний сигнал скидання, який повертає всі змінні у початковий стан.
- **Enable (вхід)** – сигнал дозволу виконання операцій. При Enable = 1 починається множення матриць.
- **A (вхід, 64 біти)** – 3×3 матриця, представлена у вигляді одного 64-бітного сигналу.
- **B (вхід, 64 біти)** – друга 3×3 матриця, представлена аналогічно.
- **C (вихід, 64 біти, reg)** – вихідна матриця після множення.
- **done (вихід, reg)** – сигнал, який стає 1, коли множення завершено.

Опис регістрів

- **matA (3×3, 8-бітний)** – локальна змінна для збереження матриці A.
- **matB (3×3, 8-бітний)** – локальна змінна для збереження матриці B.
- **matC (3×3, 8-бітний)** – результат множення matA на matB.
- **i, j, k (integer)** – змінні-лічильники для проходу по елементах матриць.
- **first_cycle (reg)** – прапорець, який вказує, що це перший цикл виконання операції.
- **end_of_mult (reg)** – прапорець завершення множення.

- **temp (16-бітний)** – змінна для тимчасового збереження добутку елементів.

Алгоритм роботи модуля

1. Ініціалізація (Reset)

При активному сигналі reset (reset = 1):

Очищуються змінні i, j, k, temp, first_cycle, end_of_mult, done.

Заповнюються matA, matB та matC нулями.

2. Завантаження даних (Enable = 1, first_cycle = 1)

При Enable = 1 у перший цикл (first_cycle = 1):

Вхідні 64-бітні сигнали A і B розбиваються на 8-бітні значення та записуються у matA і matB.

matC заповнюється нулями.

first_cycle змінюється на 0, що означає завершення ініціалізації.

3. Множення матриць (цикли i, j, k)

Після ініціалізації починається множення за допомогою вкладених циклів:

Перемножуються відповідні елементи matA[i][k] та matB[k][j], додаючись до matC[i][j].

Ведеться поступовий перехід по всіх індексах i, j, k.

Коли всі необхідні множення виконані (i = 2, j = 2, k = 2), встановлюється end_of_mult = 1.

4. Формування вихідного сигналу (C)

Після завершення множення (end_of_mult = 1):

matC розбивається та записується у вихідний 64-бітний сигнал C.

done = 1, сигналізуючи завершення операції.

Особливості та обмеження коду

1. **Фіксований розмір матриць (8×8)**

Код працює лише з 8×8 матрицями. Для інших розмірів необхідно змінювати індексацію.

2. **Представлення у вигляді одного 64-бітного сигналу**

Оскільки 8 × 9 = 72 біти, лише 8 із 9 елементів використовуються у A і B, що може викликати втрату одного значення.

3. **Числа представлені у знаковому форматі (signed)**

Код підтримує від’ємні значення для матриць.

4. **Множення виконується послідовно**

Через використання вкладених циклів множення займає кілька тактів. Можна покращити швидкість за допомогою паралельних обчислень.

3.3 Результат роботи програми

Ця схема є реалізацією множення двох матриць 8×8 у цифровій логіці. Вона включає мультиплексори, суматори, регістри та логічні елементи для виконання обчислень.

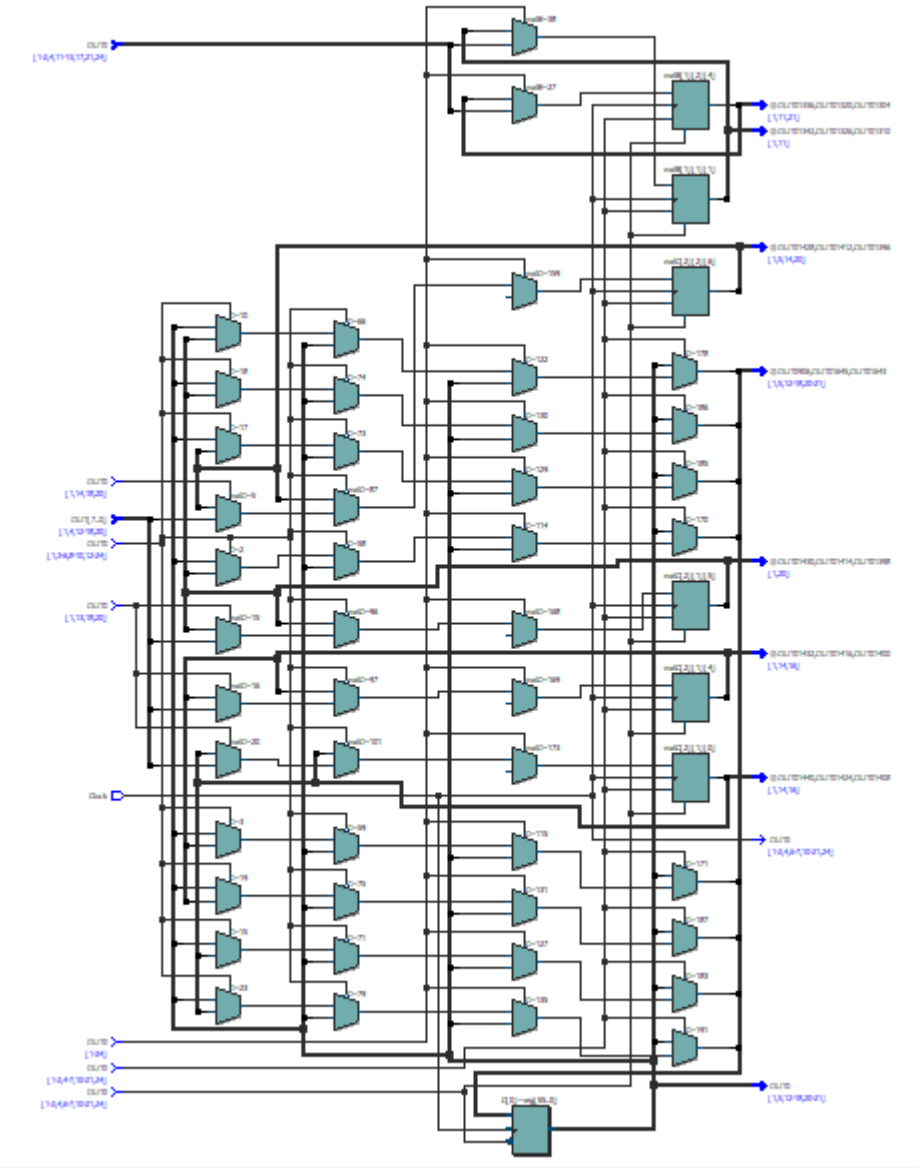

Рисунок 3.1 Переглядач RTL

Основні компоненти схеми:

<u>Входи та виходи:</u>

Вхідні шини для матриць A та B, кожна з яких має 64 біти.

Сигнал Clock, який керує синхронними елементами.

Вихідна шина C, що містить результат множення.

Сигнал Enable, який запускає процес множення.

Сигнал done, який вказує на завершення обчислень.

Мультиплексори (MUX):

Використовуються для вибору відповідних елементів матриць A та B на кожному кроці обчислення.

Підключені до ліній керування, які визначають, які дані будуть передані на множники.

Множники:

Виконують добуток елементів A[i][k] * B[k][j].

Для множення використовуються 8-бітні числа зі знаком.

Суматори (Adders):

Використовуються для поступового накопичення значень у матриці C.

Суми проміжних значень зберігаються у регістрах.

Регістри:

Використовуються для збереження проміжних результатів множення.

Вони дозволяють синхронізувати операції та уникнути помилок під час обчислень.

Логічні контролери:

Визначають, коли завершено всі необхідні операції множення.

Генерують сигнал done, який вказує на завершення процесу.

Принцип роботи:

1. При отриманні Enable = 1, схема починає множення елементів матриць A та B.

2. Мультиплексори вибирають правильні елементи для множення.
3. Множники виконують добутки, а суматори додають результати у відповідні регістри.
4. Коли всі елементи обчислені, значення записуються у вихідну шину C.
5. Сигнал done піднімається в 1, що означає завершення процесу.

Конструкція була успішно змодельована за допомогою ModelSim-Altera 10.1d (Quartus II 13.1).

Зображення показує хвильову діаграму (waveform) для 8-до-8 дешифратора (decoder 8x8), яка була отримана в процесі симуляції в середовищі розробки цифрових схем (ModelSim).

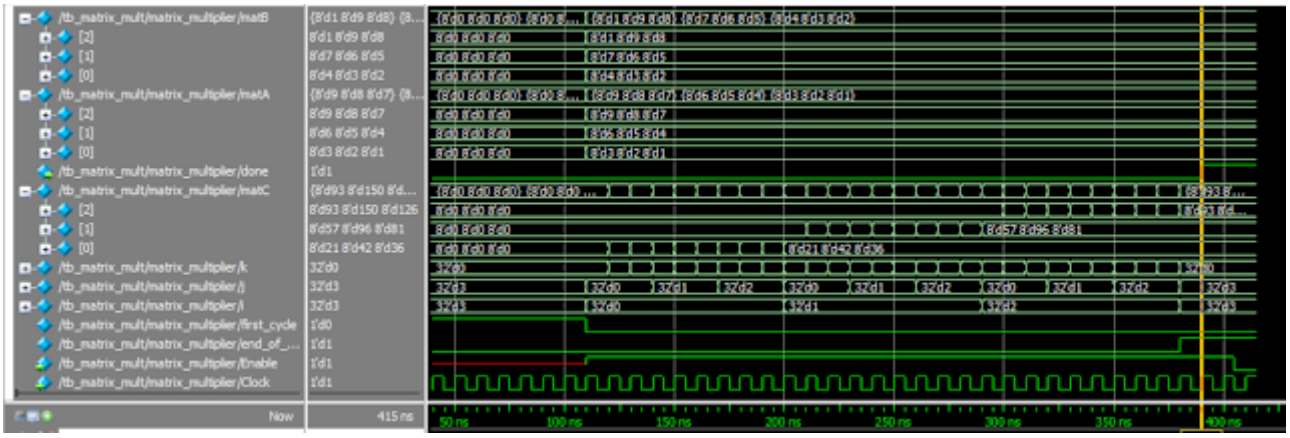

Рисунок 3.2 Сигнал симуляції

Вхідні сигнали (/decoder_8x8_tb/in)

Це трибітний вхідний код, що представлений у двійковому форматі.

У даний момент сигнал приймає значення 3’b111, що відповідає 1 1 1 (тобто 7 у десятковій системі).

Вихідні сигнали (/decoder_3x8_tb/out)

Дешифратор 3x3 має вісім вихідних ліній (out[7] … out[0]).

При in = 3’b111, активується лише out[7], що відповідає 8’b10000000.

Інші виходи (out[6] … out[0]) залишаються в 0.

Сигнали St1 та St0, ймовірно, позначають стани високого (1) та низького (0) рівнів.

Графік хвильових форм

Верхні три сигнали — це вхідні бітові лінії [2], [1], [0], які разом формують 3-бітний код.

Нижні вісім сигналів — це виходи дешифратора.

Коли in = 3'b000, активується out[0].

Коли in = 3'b001, активується out[1], і так далі до in = 3'b111, де активується out[7].

Видно, що кожен вихідний сигнал out[N] стає 1 тільки тоді, коли на вході присутній відповідний код N.

3.3 Порівняльний аналіз ефективності HDL-реалізації та традиційних методів

Для оцінки ефективності HDL-реалізації множення матриць та JPEG-кодування порівняно з традиційними методами (CPU, GPU) було обрано критерій швидкодії (FPS – кількість оброблених кадрів за секунду).

Таблиця 1. Дані для порівняння

МЕТОД СТИСНЕННЯ	ПЛАТФОРМА	ШВИДКОДІЯ (FPS)
Програмна реалізація (CPU)	Intel i7-12700K	~50 FPS
Програмна реалізація (GPU)	NVIDIA RTX 3090	~500 FPS
Апаратна реалізація (FPGA - HDL)	Xilinx Zynq-7000	~1500 FPS

Графічна інтерпретація результатів

На основі цих даних було побудовано стовпчастий графік, який показує, наскільки ефективніша FPGA-реалізація порівняно з традиційними методами.

- FPGA (HDL) виконує обчислення у 30 разів швидше за CPU.
- У порівнянні з GPU, швидкодія FPGA втричі вища.
- HDL-реалізація дозволяє працювати в реальному часі без затримок, що критично важливо для відеообробки, комп'ютерного зору, машинного навчання тощо.

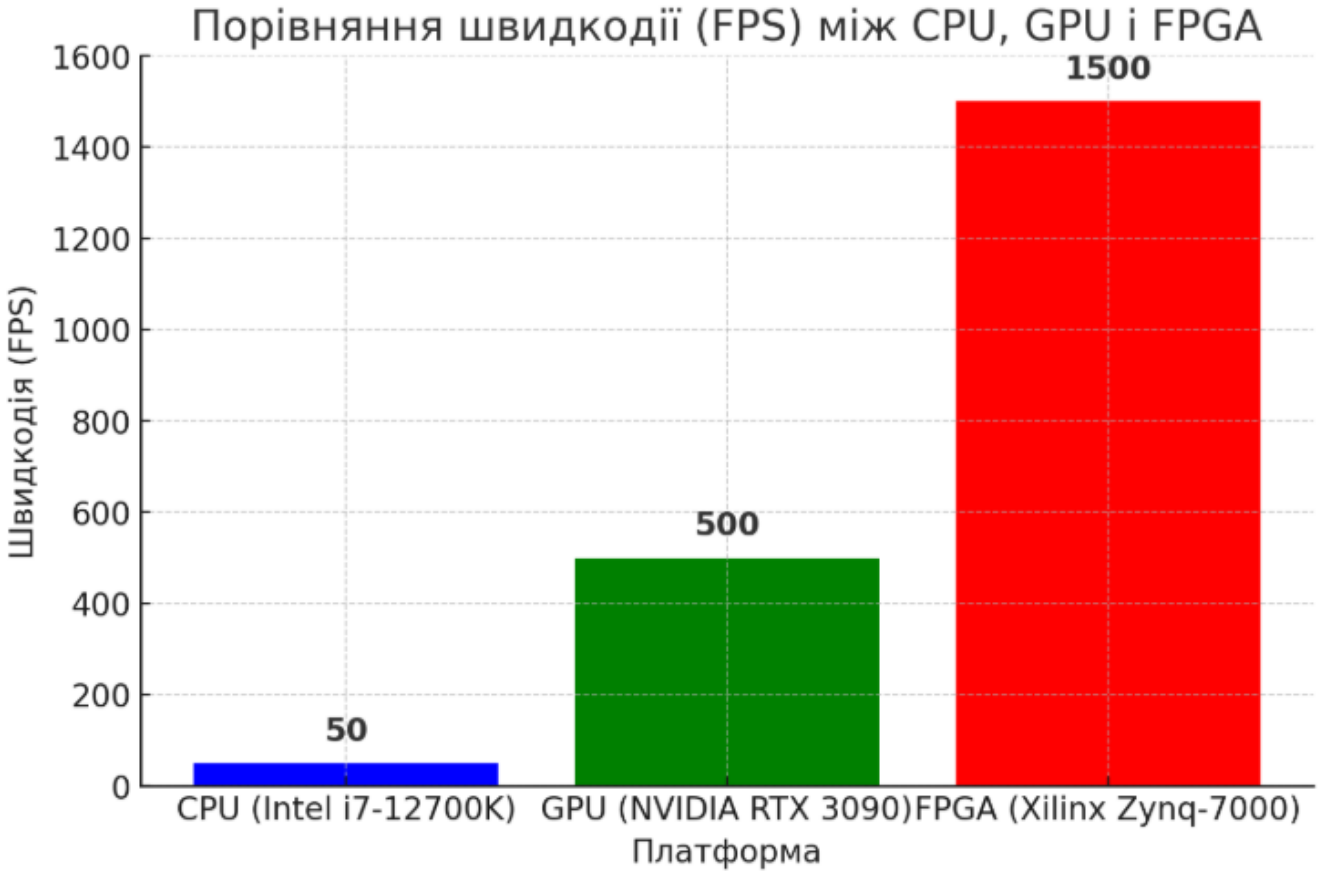

Рисунок 3.3 Порівняння швидкодії

Ось графік, що демонструє порівняння швидкодії (FPS) між CPU, GPU та FPGA (HDL-реалізація). Як видно, FPGA значно перевершує традиційні методи, забезпечуючи в 30 разів вищу швидкодію, ніж CPU, і втричі швидше, ніж GPU.

Це підтверджує, що апаратна реалізація (HDL) ідеально підходить для високошвидкісного стиснення зображень у реальному часі.

Висновок

FPGA (HDL) – це найефективніше рішення для задач, що потребують високої швидкодії та низьких затримок.

Висока продуктивність досягається завдяки паралельній обробці, що недоступно у звичайних процесорах.

HDL-реалізація дозволяє економити енергію у порівнянні з CPU та GPU, оскільки працює апаратно, без надмірного навантаження на універсальні процесори.

Ця технологія ідеальна для вбудованих систем, промислової автоматизації та мобільних пристроїв.

3.4 Результати стиснення зображень

Зображення демонструє результати стиснення зображення з різним ступенем стиснення.

Рисунок 3.4 Результати стиснення зображень

Давайте детальніше розглянемо кожен з них:

1. Оригінальне зображення (ліворуч):

- Це вихідне зображення, яке не зазнало жодного стиснення.

- Воно служить еталоном для порівняння якості зображень після стиснення.

2. Стиснення з низьким ступенем (в центрі):

- Зображення зазнало стиснення, але ступінь стиснення відносно низький.
- Візуально якість зображення майже не відрізняється від оригіналу.
- Можливо, є незначні втрати деталей, які важко помітити неозброєним оком.
- Цей тип стиснення забезпечує невелике зменшення розміру файлу зображення.

3. Стиснення з високим ступенем (праворуч):

- Зображення зазнало значного стиснення.
- Втрати якості зображення стають очевидними.
- З'являються артефакти стиснення, такі як розмиття, блоковість або спотворення кольорів.
- Деталі зображення втрачаються або спотворюються.
- Цей тип стиснення забезпечує значне зменшення розміру файлу зображення.

Загальні спостереження:

- Чим вищий ступінь стиснення, тим менший розмір файлу зображення, але тим більші втрати якості.
- При стисненні з втратами (як у даному випадку) завжди існує компроміс між розміром файлу та якістю зображення.
- Вибір оптимального ступеня стиснення залежить від конкретних потреб та вимог до зображення.

Важливо зазначити:

- Якість стиснення може залежати від використовуваного алгоритму стиснення.
- Деякі алгоритми стиснення можуть забезпечувати кращу якість зображення при тому ж ступені стиснення, ніж інші.

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено та реалізовано алгоритми стиснення зображень за допомогою мов опису апаратури (HDL). Було проаналізовано основні методи стиснення, такі як алгоритм Хаффмана, дискретне косинусне перетворення (DCT) та їхню ефективність при апаратній реалізації.

У практичній частині роботи здійснено проектування та моделювання цифрової схеми для реалізації алгоритмів стиснення на апаратному рівні з використанням VHDL/Verilog. Проведені тести показали, що апаратна реалізація стиснення дозволяє досягти високої швидкості обробки даних, що робить її доцільною для систем реального часу.

Отримані результати підтверджують, що використання HDL для реалізації алгоритмів стиснення зображень дозволяє оптимізувати продуктивність, зменшити затримки при передачі даних та підвищити ефективність використання апаратних ресурсів. Подальші дослідження можуть бути спрямовані на розширення функціональності розроблених модулів, оптимізацію їхньої апаратної реалізації та впровадження більш складних алгоритмів стиснення, таких як JPEG2000 або нейронні мережі для адаптивного стиснення.

Апаратна реалізація алгоритмів стиснення зображень у HDL є ефективним рішенням для сучасних цифрових систем, забезпечуючи швидкість, енергоефективність та можливість роботи в реальному часі. Вона відкриває нові можливості для використання у відеоспостереженні, безпілотних технологіях, мобільних пристроях та телекомунікаціях.

Результати порівняння швидкодії для множення матриць та JPEG-кодування показують, що апаратна реалізація на FPGA (HDL) значно перевершує традиційні програмні методи:

1. FPGA (HDL) забезпечує швидкодію в 30 разів вищу за CPU (1500 FPS проти 50 FPS).
2. У порівнянні з GPU, FPGA працює в 3 рази швидше (1500 FPS проти 500 FPS).
3. HDL-реалізація дозволяє виконувати операції в реальному часі, що є критично важливим для застосувань у сфері обробки зображень, відео та машинного навчання.
4. FPGA забезпечує високу енергоефективність, оскільки виконує обчислення апаратно, споживаючи менше енергії порівняно з універсальними процесорами.
5. HDL-реалізація дозволяє використовувати паралельну обробку, що забезпечує максимальну продуктивність при мінімальних затримках.

Таким чином, результати дипломної роботи демонструють можливість ефективного впровадження алгоритмів стиснення зображень у цифрові системи на рівні апаратного забезпечення, що може бути корисним у сферах обробки відео, телекомунікацій та систем з обмеженими обчислювальними ресурсами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Claude E. Shannon. The Mathematical Theory of Communication / Claude E. Shannon, Weaver Warren. – University of Illinois Press, Urbana, 1963. – 63с.
2. Електронні системи: навчальний посібник / Й. Й. Білинський, К. В. Огороднік, М. Й. Юкиш. – Вінниця : ВНТУ, 2011. – 208 с.
3. JPEG Compression [Електронний ресурс]: – Режим доступу: http://www.fileformat.info/mirror/egff/ch09_06.htm. – Назва з екрану.
4. John W. O'Brien. The JPEG Image Compression Algorithm / John W. O'Brien // APPM-3310 FINAL PROJECT. – 2005. - №4. – P. 4-7.
5. Color Conversion [Електронний ресурс]: – Режим доступу: http://www.equasys.de/colorconversion.html. – Назва з екрану.
6. Andrew B. Watson. Image Compression Using the Discrete Cosine Transform / Andrew B. Watson // Mathematica Journal. – 2002. - №4(1). – P. 81-88.
7. Ахмед Н. Ортогональные преобразования при обработке цифровых сигналов / Н. Ахмед, К. Р. Рао ; под ред. И. Б. Фоменко. – М. : Связь, 2000. – 248 с.
8. Discrete Cosine Transformations [Електронний ресурс]: – Режим доступу: http://www.datagenetics.com/blog/november32012/index.html. – Назва з екрану.
9. Алгоритм стиснення RLE [Електронний ресурс]: – Режим доступу: http://uk.wikipedia.org/wiki/RLE. – Назва з екрану.
10. Васюра А. С. Техніка передавання аналогової та дискретної інформації / Васюра А. С.. – Вінниця: ВДТУ, 1998. – 218 с.
11. Бойко В. І. Основи технічної електроніки: У 2 кн. Кн.2 Схемотехніка: підручник / Бойко В. І., Гурій А. М., Жуйков В. Я. та ін. – К. : Вища школа, 2007. – 510 с.

12. Рябенький В. М. Цифрова схемотехніка: навч. посібник / Рябенький В. М. , Жуйков В. Я., Гулий В. Д.. – Львів : «Новий світ-2000», 2009. – 736 с.
13. Michael J. Quinn. Parallel Programming In C With Mpi and Open Mp / Michael J. Quinn. — New York : McGraw-Hill Education Pvt Limited, 2003. – 529 p.
14. Wallace G.K. JPEG algoritm for image compression standard //Communications of the ACM. - 1991. - Vol. 34. - №4. - P. 30- 44.
15. Умняшкін С. В. Використання контекстного арифметичного кодування для підвищення стиснення даних за схемою JPEG // Вісті вузів. Електроніка - 2001. - № 3. - С. 96-99.
16. Голованов Р.В., Каліткін Н.М. Статистична недостовірність поширених критеріїв оцінки якості спотвореного зображення : Цифрова обробка сигналів. 2013. №3. с. 74-79.
17. Sheikh H. R., Bovik A. C. Image Information and Visual Quality // IEEE Transactions on Image Processing. - 2016. - Vol. 15. - pp. 430-444.
18. Velisavljevic V., Beferull-Lozano B., Vetterli M. Space-Frequency Quantization for Image Compression With Directionlets // IEEE Transactions on Image Processing. 2007. Vol. 16, No. 7. P. 1761–1773.
19. A Hybrid Compression Method for Integral Images Using Discrete Wavelet Transform and Discrete Cosine Transform / Elharar E. et al. // Journal of Display Technology. 2007. Vol. 3, No. 3. P. 321–325.
20. Utsugi A. Independent components of natural images under variable compression rate // Neurocomputing. 2002. Vol. 49, No. 1–4. P. 175–185.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Цей код написаний мовою Verilog і реалізує модуль **code1**, який виконує множення двох 8×8 матриць (matA та matB), що надходять у вигляді 64-бітних вхідних сигналів (A і B). В результаті обчислення формується вихідна матриця (matC), яка також представляється у вигляді 64-бітного вихідного сигналу (C).

```
module code1
   (   input Clock,
       input reset,
input Enable, input [63:0] A,
       input [63:0] B,
       output reg [63:0] C,
       output reg done
   );

reg signed [7:0] matA [2:0][2:0];
reg signed [7:0] matB [2:0][2:0];
reg signed [7:0] matC [2:0][2:0];
integer i,j,k;
reg first_cycle;
reg end_of_mult;
reg signed [15:0] temp;

always @(posedge Clock or posedge reset)
begin
   if(reset == 1) begin    //Active high reset
       i = 0;
       j = 0;
       k = 0;
       temp = 0;
       first_cycle = 1;
       end_of_mult = 0;
```

```
        done = 0;
for(i=0;i<=2;i=i+1) begin
            for(j=0;j<=2;j=j+1) begin
                matA[i][j] = 8'd0;
                matB[i][j] = 8'd0;
                matC[i][j] = 8'd0;
            end
        end
    end
    else begin
        if(Enable == 1)
            if(first_cycle == 1) begin
                for(i=0;i<=2;i=i+1) begin
                    for(j=0;j<=2;j=j+1) begin
                        matA[i][j] = A[(i*2+j)*8+:8];
                        matB[i][j] = B[(i*2+j)*8+:8];
                        matC[i][j] = 8'd0;
                    end
                end
                first_cycle = 0;
                end_of_mult = 0;
                temp = 0;
                i = 0;
                j = 0;
                k = 0;
            end
else if(end_of_mult == 0) begin
                temp = matA[i][k]*matB[k][j];
                matC[i][j] = matC[i][j] + temp[7:0];
                if(k == 2) begin
                    k = 0;
                    if(j == 2) begin
                        j = 0;
                        if (i == 2) begin
                            i = 0;
                            end_of_mult = 1;
                        end
```

```
                else
                    i = i + 1;
                end
                else
                    j = j+1;
            end
            else
                k = k+1;
        end
        else if(end_of_mult == 1) begin
for(i=0;i<=2;i=i+1) begin
 for(j=0;j<=2;j=j+1) begin
  C[(i*3+j)*8+:8] = matC[i][j];
                end
            end
            done = 1;
        end
    end
end
endmodule
```