Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Метод побудови апаратної архітектури для система комп'ютерного зору на основі FPGA»

КвРКІП.170231.17.02.01 ПЗ

Виконала: студентка 2 курсу, група КІ2м–21–1 _____ Атаманюк О.В.
Підпис Ініціали, прізвище

Керівник доктор техн. наук, професор _____ Лисенко С.М.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
_____ _____ 2023 р.

Хмельницький, 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень магістр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма освітньо-наукова програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"01" 09 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Атаманюк Ользі Вадимівні

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод побудови апаратної архітектури для система комп'ютерного зору на основі FPGA

Керівник проекту (роботи) Лисенко С.М., д.т.н.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 09.01.2023 р. № 1

2. Строк подання студентом проекту (роботи) на кафедру 01.05.2023 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Алгоритми та методи та засоби реалізації комп'ютерного зору

FPGA як основа для розроблення апаратних прискорювачів

Метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA

Програмно-апаратний засіб для систем комп'ютерного зору на основі FPGA

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи магістра

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІС		

7. Дата видачі завдання «_06_»_____09_____2022р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	05.09.2022	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	05.10.2022	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	05.11.2022	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	05.12.2022	виконано
5	Робота над науковою статтею	05.01.2023	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2022	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	05.04.2023	виконано
8	Оформлення пояснювальної записки згідно вимог	15.04.2023	виконано
9	Попередній захист ДРМ	18.04.2023	виконано
10	Захист ДРМ на засіданні ЕК	До 10.05.2023	

Студент _____ О.В. Атаманюк
Підпис Ініціали, прізвище

Керівник роботи _____ С.М. Лисенко
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема кваліфікаційної роботи: «Метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA».

Автор роботи: Атаманюк Ольга Вадимівна.

Керівник роботи: Лисенко Сергій Миколайович.

Пояснювальна записка: 81с., 29 рис., 7 табл., 1 дод., 81 джерел.

Графічна частина: 12 презентаційних слайдів.

FPGA, АПАРАТНА АРХІТЕКТУРА, СИСТЕМИ КОМП’ЮТЕРНОГО ЗОРУ, АЛГОРИТМИ ОБРОБКИ ЗОБРАЖЕНЬ ПРОГРАМОВАНІ ЛОГІЧНІ ПРИСТРОЇ, ПЛАТФОРМА ДЛЯ ОБРОБКИ ВІЗУАЛЬНОЇ ІНФОРМАЦІЇ, НИЗЬКОРІВНЕВЕ ПРОГРАМУВАННЯ, ПЛАТФОРМА ДЛЯ ДОСЛІДЖЕННЯ АЛГОРИТМІВ ЗОРУ, ВИСОКОЯКІСНЕ ВІДЕО ТА ЗОБРАЖЕННЯ З ВИКОРИСТАННЯМ FPGA.

Об’єктом дослідження є комп’ютерних зір.

Предметом дослідження є модель, метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA.

Метою кваліфікаційної роботи магістра є підвищення швидкодії систем комп’ютерного зору.

Для розв’язання поставлених задач використовувалися методи побудови апаратної архітектури для систем комп’ютерного зору.

Наукова новизна отриманих результатів:

− удосконалено метод побудови апаратної архітектури для систем комп’ютерного зору, який на відміну від відомих використовує FPGA та грунтується на алгоритмі SGBM, і який забезпечує високу швидкодію систем комп’ютерного зору;

− набули подальшого розвитку програмно-технічні засоби комп’ютерного зору на основі FPGA.

На основі проведених досліджень розроблено апаратно-програмні засоби комп’ютерного зору на основі FPGA.

Практична значимість отриманих результатів полягає у підвищенні швидкодії систем комп’ютерного зору на основі FPGA.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 5

ВСТУП 9

1 АЛГОРИТМИ ТА МЕТОДИ ТА ЗАСОБИ РЕАЛІЗАЦІЇ КОМП'ЮТЕРНОГО ЗОРУ 13

1.1 Класифікація зображень 13

1.2 Виявлення об'єктів 15

1.3 Сегментація зображення 17

1.4 Апаратна реалізація 19

1.5 Графічні процесори (GPUs) 20

1.6 Програмовані вентильні матриці як засоби побудови апаратної архітектури для систем комп'ютерного зору 22

1.7 Дослідження спеціальних апаратних прискорювачів 27

1.8 Висновки 28

2 FPGA ЯК ОСНОВА ДЛЯ РОЗРОБЛЕННЯ АПАРАТНИХ ПРИСКОРЮВАЧІВ 31

2.1 Дослідження елементної бази побудови апаратних прискорювачів 31

2.2 Застосування HDLS 34

2.3 Застосування мови побудови апаратного забезпечення 35

2.4 Синтез високого рівня (HLS – High Level Synthesis) 35

2.4.1 Мова програмування на основі HLS 37

2.4.2 SystemC 38

2.4.3 Доменно-спеціальна мова HLS 39

2.5 FPGA для експертів 41

2.6 Компоненти побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA 42
2.7 Висновки 50
3 МЕТОД ПОБУДОВИ АПАРАТНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМ КОМП’ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA 52
3.1 Напівглобальне зіставлення блоків 53
3.2 Архітектура 59
3.3 Оцінка підходу 62
3.3.1 Точність 62
3.3.2 Продуктивність, яка не залежить від платформи 63
3.4 Висновок 66
4 ПРОГРАМНО-АПАРАТНИЙ ЗАСІБ ДЛЯ СИСТЕМ КОМП’ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA 68
4.1 Архітектура FPGA 68
4.2 Проектування логічних блоків 72
4.2.1 Ядро Гауса 73
4.2.2 Вхід FIFO 74
4.2.3 Додавання та множення 76
4.2.4 Контейнер 78
4.2.5 Система PCIe 81
4.3 Висновки 86
ВИСНОВКИ 87
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 90
ДОДАТОК А ЛІСТИНГ МЕТОДУ ПОБУДОВИ АПАРАТНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМ КОМП’ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA 98

ДОДАТОК Б КОПІЯ ПУБЛІКАЦІЇ 103

ДОДАТОК В ПРЕЗЕНТАЦІЯ 115

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ANN (англ. Artifical Neural Network) – штучна нейронна мережа;

API (англ. Application Programming Interface) – програмний інтерфейс програми;

ASIC (англ. Application-Specific Integrated) – інтегральна схема спеціального призначення;

BNN (англ. Binary Neural Network) – бінарно-нейронна мережа;

CLB (англ. Circuit Configurable Logic Block) – конфігурований логічний блок схеми;

CPU (англ. Central Processing Unit) – центральний процесор;

DAC (англ. digital to analog converter) – цифро-аналоговий перетворювач;

DDR (англ. Double Data Rate) – один з типів оперативної пам'яті, які використовуються в комп'ютерах;

DMA (англ. Direct Memory Access) – прямий доступ до пам'яті;

DSE (англ. Design Space Exploration) – видалення небажаних проектних точок на основі параметрів, що цікавлять;

DSL (англ. Domain Specific Language) – предметно-орієнтована мова програмування;

DSP (англ. Digital Signal Processor) – цифрова обробка сигналів;

FF (англ. Flip-Flop) – схеми, які мають два стабільні стани, які можуть зберігати інформацію про стан;

FIFO (англ. first in, first out) – перший прийшов перший вийшов, загальний принцип накопичення та обробки завдань;

FIR-фільтр (англ. finite impulse response) – один з видів лінійних цифрових фільтрів, характерною особливістю якого є обмеженість за часом його імпульсної характеристики;

FPGA (англ. Field Programmable Gate Array) – програмовані вентильні матриці;

GAN (англ. Generative adversarial networks) – генеративна змагальна мережа;

GCC (англ. GNU Compiler Collection,) – набір компіляторів для різних мов програмування;

GPU (англ. Graphics Processing Unit) – графічний процесор

GPGPU (англ. General-Purpose Computing on Graphics Processing Units) – обчислення загального призначення на графічних процесорах;

HBM (англ. high bandwidth memory) – пам'ять з високою пропускною здатністю;

HCL (англ. Hardware Construction Language) – мова побудови апаратури;

HDL (англ. Hardware Description Language) – мова опису апаратури;

HLS (англ. High-level Synthesis) – високорівневий синтез;

HPC (англ. High Performance Computing) – високопродуктивні обчислення використовують суперкомп’ютери та комп’ютерні кластери для вирішення складних обчислювальних задач;

INP (англ. In-Network Processing) – техніка, яка використовується в системах бази даних датчиків, за допомогою якої записані дані обробляються самими вузлами датчиків;

IP (англ. Intellectual Property) – інтелектуальна власність;

JSON (англ. JavaScript Object Notation) – текстовий формат обміну даними між комп'ютерами;

LSTM (англ. Long short-term memory) – довга короткочасна пам'ять;

LUT (англ. LookupTable) – таблиця пошуку;

MFLOP – загальна міра швидкості комп’ютерів, які використовуються для виконання обчислень із плаваючою комою;

NCCL (англ. NVIDIA Collective Communications Library) – бібліотека колективних комунікацій NVIDIA;

NOC (англ. Network on Chip) – методологія проектування багатоядерних інтегральних схем (багаточипових модулів) для створення систем на кристалі;

OpenCL (англ. Open Computing Language) – фреймворк для створення комп'ютерних програм;

OpenMP (англ. Open Multi-Processing) –набір директив компілятора, бібліотечних процедур та змінних середовища;

PCIE (англ. Peripheral Component Interconnect Express) – комп'ютерна шина, що використовує програмну модель шини PCI і високопродуктивний фізичний протокол, заснований на послідовній передачі даних;

PCM (англ. pulse code modulation) – імпульсно-кодова модуляція;

RAID (англ. Redundant Array of Independent Disks) – технологія віртуалізації даних, яка об'єднує кілька дисків в логічний елемент для надійності збереження;

RAM (англ. Random Access Memory) – пам'ять з довільним доступом;

RNN (англ. Recurrent Neural Network) – вид нейронних мереж, де зв'язки між елементами утворюють спрямовану послідовність;

RTL (англ. Register-Transfer Level) – рівень передачі регістрів;

SGBM (англ. Semi-GlobalBlockMatching) – алгоритм комп’ютерного бачення для оцінки щільної карти невідповідності з випрямленої пари стереозображень;

SIMD (англ. Single Instruction, Multiple Data) – одиночний потік команд, множинний потік даних;

SNN (англ. Spiking Neural Network) – штучні нейронні мережі, які більш точно імітують природні нейронні мережі;

SOC (англ. System-on-Chip) – дизайн електронної схеми, яка вміщує функціональні складові цілого пристрою на одній мікросхем;

SQL (англ. Structured Query Language) – мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних;

SSD (англ. Single Shot MultiBox Detector) – одноразовий детектор MultiBox;

SVM (англ. Support Vector Machine) – метод аналізу даних для класифікації та регресійного аналізу за допомогою моделей з керованим навчанням з

пов'язаними алгоритмами навчання, які називаються опорно-векторними машинами;

VHDL (англ. Very high speed integrated circuits) – мова опису апаратури інтегральних схем;

ПК – персональний комп'ютер;

ЦП – центральний процесор

ЦСР – цифровий сигнальний процесор;

ШПФ – швидке перетворення Фур’є

ШІ – штучний інтелект;

ВСТУП

Цифровий комп'ютерний зір є актуальним з точки зору розвитку технологій та їх використання в різних сферах життя.

Цифровий комп'ютерний зір - це технологія, що дозволяє комп'ютерам сприймати та розуміти зображення, які раніше були доступні лише для людей. Ця технологія базується на штучному інтелекті та машинному навчанні і дозволяє комп'ютерам аналізувати, класифікувати та розпізнавати зображення.

Цифровий комп'ютерний зір застосовується в багатьох сферах, таких як медицина, автомобільна промисловість, реклама, безпека, відеоспостереження та багато інших. Наприклад, в медицині цифровий комп'ютерний зір використовується для аналізу зображень рентгенівських знімків та МРТ, що дозволяє лікарям точніше діагностувати хвороби та розробляти ефективні плани лікування.

У автомобільній промисловості цифровий комп'ютерний зір використовується для розпізнавання та класифікації зображень, що дозволяє автомобілям більш ефективно реагувати на дорожні ситуації та забезпечити безпеку на дорогах.

Крім того, цифровий комп'ютерний зір є важливим елементом в розробці віртуальної та доповненої реальності, що дає можливість розширювати можливості візуальної взаємодії між людиною та комп'ютером.

Цифровий зір може бути реалізований за допомогою FPGA (Field Programmable Gate Array), який є програмованим логічним пристроєм, який дозволяє створювати кастомні логічні схеми та оброблювати великі обсяги даних швидко та ефективно.

Для реалізації цифрового зору за допомогою FPGA зазвичай використовуються спеціалізовані платформи розробки, такі як Vivado від Xilinx або Quartus від Intel. Для розробки програми необхідно мати розуміння алгоритмів комп'ютерного зору, машинного навчання та програмування FPGA.

Реалізація цифрового зору на FPGA дозволяє отримати високу швидкість обробки даних та можливість розробляти кастомні рішення для різних завдань комп'ютерного зору. Однак, розробка таких систем вимагає високої кваліфікації в галузі програмування FPGA та знання алгоритмів обробки зображень та машинного навчання.

Програмовані вентильні матриці (FPGA) — це нове доповнення до світу прискорення центрів обробки даних. Хоча базова технологія існує десятиліттями, її застосування в центрах обробки даних поступово починає набирати обертів. Однак існує безліч проблем, які перешкоджають широкому застосуванню FPGA в центрах обробки даних.

Ланцюжки інструментів із закритим вихідним кодом призводять до блокування постачальника та нестабільних потоків інструментів. Порівняно з стандартними рішеннями, що використовують центральні та графічні процесори, FPGA є дорожчими та потребують більше часу для розробки..

Тим не менш, FPGA також пропонують можливість розробки швидших прискорювачів з меншою енергетичною оболонкою для ефективного програмного забезпечення.

FPGA завжди справлялася з такими завданнями, як обробка з низькою затримкою (наприклад, високочастотна торгівля), завдання логіки зчеплення або потокова передача з високою пропускною здатністю.

Раніше вони не конкурували з процесорами загального призначення, а саме ЦП і графічні процесори в просторі центру обробки даних для прискорення додатків. FPGA більше конкурувала з інтегральною схемою спеціального застосування (ASIC). Якщо потрібний обсяг був відносно невеликим, наприклад, в автомобільній промисловості, FPGA була гарною альтернативою перед тим, як обертати кремній, що призводить до високих початкових витрат.

Відповідно, можливим є використовувати різні методи. Сьогодні найпоширенішим підходом є моделювання розробки та перегляд окремих сигналів у потрібний час або використання методів, де консольний вихід генерується

залежно від стану прискорювача. Деякі методи, як-от нечітке тестування, з’явилися протягом багатьох років, щоб допомогти помітити помилки.

Актуальність роботи полягає в методі побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA, що може забезпечити високу продуктивність та енергоефективність, які є важливими для багатьох застосувань у таких галузях, як медицина, автомобільна промисловість, безпека та багато інших напрямів.

Метою кваліфікаційної роботи магістра є підвищення швидкодії систем комп’ютерного зору.

Поставлена мета досягається за допомогою розв’язання таких задач:

– дослідити методи забезпечення стійкості корпоративної комп’ютерної мережі;

– дослідити методи побудови апаратної архітектури для систем комп’ютерного зору;

– проаналізувати сучасні програмно-технічні засоби для систем комп’ютерного зору;

– дослідити апаратні компоненти для функціонування програмно-технічних засобів комп’ютерного зору на основі FPGA;

– розробити метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA;

– реалізувати метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA.

Об’єктом дослідження є комп’ютерних зір.

Предметом дослідження є метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA.

Наукова новизна отриманих результатів:

1) Удосконалено метод побудови апаратної архітектури для систем комп’ютерного зору, який на відміну від відомих використовує FPGA та

ґрунтується на алгоритмі SGBM, і який забезпечує високу швидкодію систем комп’ютерного зору.

2) Набули подальшого розвитку програмно-технічні засоби комп’ютерного зору на основі FPGA.

Практична цінність отриманих результатів. В результаті виконаного наукового дослідження буде розроблено апаратно-програмні засоби комп’ютерного зору на основі FPGA.

Для розв’язання поставлених задач використовуються основні положення вивчення основ теорії комп'ютерного зору та FPGA-технологій, аналіз різних методів та архітектур для реалізації систем комп'ютерного зору на основі FPGA з метою вибору оптимального підходу, розробка алгоритмів та програмного забезпечення для побудови апаратної архітектури систем комп'ютерного зору на основі FPGA, розробка апаратної архітектури системи комп'ютерного зору на основі FPGA з урахуванням потреб користувача та вимог до продуктивності та енергоефективності.

За темою кваліфікаційної роботи магістра опублікована одна стаття у фаховому науковому виданні в фаховому журналі «Вісник Хмельницького національного університету» №2 за 2023 рік.

1 АЛГОРИТМИ ТА МЕТОДИ ТА ЗАСОБИ РЕАЛІЗАЦІЇ КОМП'ЮТЕРНОГО ЗОРУ

1.1 Класифікація зображень

Класифікація образів є своєрідною біологічно первинною здатністю системи зорового сприйняття людини. Це відіграє вирішальну роль у сфері комп'ютерного зору, метою якого є автоматична класифікація зображень у заздалегідь визначені класи. Протягом десятиліть дослідники прокладали шлях у розробці передових методів для підвищення точності класифікації. Традиційно моделі класифікації можуть добре працювати лише на невеликих наборах даних, таких як CIFAR-10 і MHIST. Великий стрибок у розвитку класифікації зображень стався, коли зображення великого розміру сучасна історія нейронної мережі після тривалого періоду.

Модель CNN складається з кількох шарів згортки, за якими слідують функції активації та рівні об'єднання, а також кілька повністю пов'язаних рівнів перед прогнозуванням. Він переходить у глибоку структуру, щоб полегшити механізми фільтрації, виконуючи згортки в багатомасштабних картах ознак, що призводить до дуже абстрактних і дискримінаційних ознак [1].

AlexNet має 8 шарів згортки, 3 шари об'єднання та 3 повністю зв'язані шари із загальною кількістю 60 мільйонів параметрів. Крім того, розширення даних і вилучення даних широко використовуються сьогодні як ефективні стратегії навчання. Тому AlexNet відомий як основа сучасного глибокий CNN [2].

Хоча глибші мережі пропонують кращу точність, просте збільшення кількості шарів не може безперервно покращувати точність через зникнення/вибухання градієнтної інформації під час навчання мережі ResNet, який робить ще один великий прогрес глибокої мережевої структури, пропонує використовувати скорочене з'єднання між залишковими блоками, щоб повністю використовувати інформацію з попередніх рівнів і зберегти градієнти під час

зворотного поширення. Слідуючи ідеї ResNet, DenseNet об’єднує і, отже, повторно використовує функції з усіх попередніх шарів. DenseNet має велику перевагу в точності класифікації на ImageNet.

Базуючись на цих роботах у літературі, з’єднання різних мережевих рівнів продемонструвало багатообіцяюче покращення у представленнях навчання більш глибоких мереж. Використовуючи ResNet або DenseNet як основну магістральну структуру, дослідники зосереджуються на покращенні функціональності блоків нейронної мережі [3].

Більшість глибоких мереж полегшують класифікацію, щоб бути більш точними. Однак у багатьох реальних програмах класифікації, таких як робототехніка, автономне водіння, смартфони тощо, завдання класифікації сильно обмежене доступними обчислювальними ресурсами. Таким чином, проблема полягає в пошуку оптимальної точності за умови обмеженого обчислювального бюджету (тобто пам’яті та/або MFLOP).

Крім загальноприйнятих проблема класифікації зображень із тисячами класів і складних сцен, класифікація за кількома мітками (наприклад, атрибуція обличчя) і дрібнозернисту класифікацію (наприклад, класифікацію Stanford Dogs) представляють великий інтерес у сфері комп’ютерного зору.

Крім того, великий успіх глибокого навчання в області зображень стимулювало різноманітні методи вивчення надійних представлень функцій для класифікації відео, де семантичний вміст, такий як людські дії або складні події [39] автоматично класифікуються.

Ранні роботи часто розглядають відеокліп як набір кадрів. Класифікація відео реалізується шляхом агрегування функцій CNN на рівні кадру шляхом усереднення або кодування. Нарешті, стандартні класифікатори, такі як SVM використовується для розпізнавання. На відміну від каркасно-рівневого методу класифікації, існує низка інших підходів із застосуванням наскрізних моделей CNN для вивчення прихованих просторово-часових моделей у відео [4].

Зокрема, просторовий CNN використовується для моделювання інформації про зовнішній вигляд із кадрів RGB, тоді як часовий CNN використовується для вивчення інформації про рух із щільного оптичного потоку між сусідніми кадрами. Оскільки двопотоковий підхід зображує лише рухи в межах короткого часового вікна та не враховує часовий порядок різних кадрів, кілька моделей рекурентного з'єднання для послідовних даних, у тому числі рекурентні нейронні мережі (RNN) і моделі довготривалої короткочасної пам'яті (LSTM), використовуються для моделювання часової динаміки для відео [5].

1.2 Виявлення об'єктів

Виявлення об'єктів, яке полягає у визначенні та локалізації екземплярів об'єктів або з великої кількості попередньо визначених категорій у природних зображеннях, або для даного конкретного об'єкта (наприклад, обличчя Дональда Трампа, спотворена область на зображенні тощо), є ще одне важливе та складне завдання комп'ютерного зору. Виявлення об'єктів і класифікація зображень стикаються зі схожими технічними проблемами: обидва вони мають працювати з великою кількістю об'єктів, що змінюються. Однак виявлення об'єкта складніше, ніж класифікація зображення, оскільки воно має визначити точну локалізацію об'єкта інтересу [6].

Історично більшість дослідницьких зусиль було зосереджено на виявленні одної категорії даних об'єктів, наприклад пішохід і обличчя шляхом розробки набору відповідних функцій. У цих роботах об'єкти виявляються за допомогою набору попередньо визначених шаблонів об'єктів, які відповідають кожному розташуванню на зображенні чи піраміді об'єктів. Стандартні класифікатори, такі як SVM і Adaboost часто використовуються для цієї мети.

Щоб побудувати надійну систему виявлення об'єктів загального призначення, дослідницьке співтовариство почало розробляти великомасштабні багатокласові набори даних в останні роки. Pascal-VOC 2007 з 20 класами та MS-

COCO із 80 категоріями об’єктів є два знакових набори даних виявлення об’єктів. У цих двох наборах даних результати виявлення оцінюються за двома можливими показниками:

1) середня точність (AP) шляхом підрахунку правильно виявлених обмежувальних рамок, для яких коефіцієнт перекриття перевищує 0,5;

2) середня середня точність (mAP) шляхом усереднення Пов’язані значення APs різними порогами коефіцієнта перекриття [7].

Останнім часом глибоке навчання значно просунуло сферу виявлення об’єктів.

R-CNN був першим двоетапним методом серед найраніших загальних методів виявлення об’єктів на основі CNN. Він застосовує AlexNet для вилучення вектора ознак фіксованої довжини з кожної пропозиції регіону зі зміненим розміром, який є кандидатом на об’єкт, згенерованим алгоритмом вибіркового пошуку. Після цього кожен регіон класифікується за набором лінійних SVM для певної категорії.

Метод демонструє значне покращення mAP порівняно з традиційним сучасним детектором DPM. Однак він неефективний через його багатоступінчатий складний конвеєр і резервування та вилучення функцій CNN із численних пропозицій регіону [8].

Двоступенева архітектура R-CNN здатна запропонувати чудову точність. Однак, ефективність у реальному часі потрібна для виявлення об’єктів багатьма реальними додатками. У зв’язку з цим часто надають перевагу простим одноступеневим архітектурам. YOLO — це перший одноетапний метод, який перетворює задачу виявлення як проблему регресії [9].

Одноразовий детектор MultiBox (SSD) дотримується подібної одноетапної стратегії. Він перевершує YOLO за точністю завдяки двом основним удосконаленням. По-перше, SSD витягує важливі функції з багатомасштабних карт функцій CNN. По-друге, він застосовує ряд обмежувальних рамок за замовчуванням, дотримуючись концепції прив’язки. YOLOv2 поглинає переваги

SSD і Faster R-CNN, представляючи якірний механізм. Нова модель YOLOv2 покращує точність виявлення та значно скорочує час висновку порівняно з YOLO на Pascal-VOC 2007. Однак її точність усе ще гірша, ніж у двоетапних методів для загальних завдань виявлення (наприклад, MS-COCO).

В останній реалізації YOLOv3, кілька опорних блоків призначаються на трьох різних масштабованих картах функцій, таким чином створюючи набагато більше пропозицій, ніж YOLO та YOLOv2. Таким чином, невеликі об’єкти можна виявити точно. Це найсучасніша система виявлення об’єктів у режимі реального часу програми [10].

Окрім одноступеневої та двоступеневої архітектури, існує кілька інших прожекторів для об’єкта виявлення. Наприклад, взаємозв’язок різних об’єктів розглядається шляхом проектування модуля відношення об’єктів у Ref [11]. Generative Adversarial Network (GAN) використовується для створення надроздільної здатності невеликих об’єктів або функції [12], щоб допомогти з виявленням дрібних предметів. На відміну від значного прогресу у виявленні об’єктів, зосередженого на нерухомих зображеннях, виявлення об’єктів відео приділялося менше уваги. загалом, Виявлення об’єктів для відео реалізовано шляхом об’єднання результатів виявлення об’єктів на поточному кадрі та відстеження об’єктів із попередніх кадрів.

1.3 Сегментація зображення

Сегментація зображення розглядається як класифікація на рівні пікселів, метою якої є поділ зображення на значущі області шляхом класифікації кожного пікселя в певну сутність. У традиційній сегментації зображень ідея неконтрольованого злиття та поділу локальних регіонів була широко досліджена на основі кластеризації, оптимізуючи глобальні критерії або взаємодії користувача.

Розквіт технологій глибокого навчання сприяв широкомасштабній керованій класифікації, яка переходить від класифікації об’єктів на рівні зображення до

локалізації об’єктів на рівні ящиків і далі до піксельної сегментації об’єктів.

Таким чином, сучасна сегментація зображень є об’єктно-орієнтованою і може бути розділена на дві тонкі гілки:

1) семантична сегментація, яка призначає кожен піксель зображення класу семантичного об’єкта;
2) сегментація екземплярів, яка передбачає різні мітки для різних екземплярів об’єктів як подальше покращення до семантичної сегментації [13].

Ранні методи сегментації, засновані на глибокому навчанні, зазвичай застосовують CNN як дескриптор функції для кожного пікселя, який описується його оточуючим фрагментом. Цей фреймворк на основі CNN є проблематичним щодо ефективності та недостатньо точним для вилучення надлишкових функцій.

FCN показує покращення точності пікселів порівняно з традиційними методами сегментації. Однак базова структура FCN не в змозі охопити велику кількість функцій і не враховує просторову узгодженість між пікселями, що перешкоджає її застосуванню до певних проблем і сценаріїв. У будь-якому випадку, успіх архітектури FCN робить її популярною, і її активно слідують багато подальших робіт із сегментації [14].

Загалом, використання моделей класифікації без повністю пов’язаних шарів як магістральної мережі для створення карт об’єктів із низькою роздільною здатністю називається кодувальником, тоді як симетричне відображення від зображення з низькою роздільною здатністю до результату попіксельної класифікації називається декодером.

З добре відомою магістральною мережею як кодером альтернативні роботи сегментації на основі CNN зазвичай є варіантом реалізації декодера.

Сегментація зображення є складною проблемою, яка потребує обох хороших піксельна точність, яка покладається на дрібнозернисті локальні особливості та точність класифікації, для якої глобальний контекст зображення є вирішальним для вирішення локальних неоднозначностей. Однак стратегія об’єднання в класичні

архітектури CNN є недоліком, оскільки втрачається детальна інформація під час виконання кроків об’єднання [15].

Кілька запропонованих методів, таких як RefineNet і PSPNet намагаються уникнути або відновити втрату зниженої дискретизації в кодері, поєднавши функції низького та високого рівнів. RefineNet розробляє модуль декодера, використовуючи залишкові з’єднання як короткого, так і далекого радіусу дії для захоплення багатої контекстної інформації. У PSPNet пропонується модуль пірамідного об’єднання для агрегування різної регіональної контекстної інформації.

Вищезазначені наскрізні архітектури в основному зосереджені на семантичній сегментації. Для порівняння, більшість робіт із сегментації екземплярів дотримуються принципу, що сегментація передує розпізнаванню.

Хоча попіксельна сегментація зображення швидко прогресує з надзвичайною точністю, вона все ще далека від практичного використання, наприклад семантичної сегментації відео, через високу складність щільного прогнозування. Таким чином, розробка високоефективної системи сегментації зображень одна з найбільших проблем у спільноті комп’ютерного зору [16].

1.4 Апаратна реалізація

Нещодавні прориви в розробці алгоритмів комп’ютерного зору зумовлені не лише технологіями глибокого навчання та великомасштабними наборами даних, але також покладаються на значні стрибки апаратного прискорення, яке забезпечує потужну паралельну обчислювальну архітектуру для ефективного навчання та висновків для масштабних, складних і багатошарових нейронних мереж.

Апаратне прискорення використовує переваги апаратного забезпечення комп’ютера для виконання обчислювальних завдань із меншою затримкою та вищою пропускною здатністю, ніж звичайна реалізація програмного забезпечення, що працює на загальному призначенні ЦП [17].

Історично склалося так, що орієнтовані на обчислення архітектури в основному розроблені для ефективних послідовних обчислень із складним плануванням завдань. Вони страждають від високого споживання енергії та низької пропускної здатності пам’яті для переміщення даних під час оцінки глибоких мереж CNN, які вимагають паралельних щільних обчислень, високої можливості повторного використання даних і великої пропускної здатності пам’яті [18].

Традиційні методи машинного навчання, такі як дерево рішень, SVM тощо. Вони є як правило, на основі розроблених вручну функцій.

Через обмежену здатність до навчання цих традиційних методів їхня точність не може безперервно зростати з масштабом даних/моделі. З іншого боку, глибокі нейронні мережі мають високу масштабованість у своїй здатності до навчання.

На практиці операція може бути обчислена швидше на апаратній платформі, яка розроблена та/або запрограмована спеціально для програми. Апаратне прискорення, таким чином, робить крок вперед для серйозних налаштувань можливостей обробки, дозволяючи великий паралелізм, маючи спеціальні шляхи даних для тимчасових варіантів і зменшуючи накладні витрати на контроль інструкцій [19].

1.5 Графічні процесори (GPUs)

Графічні процесори спочатку були розроблені для прискорення обробки графіки. Графічний процесор спеціально розроблений для інтегрованого перетворення, освітлення, трикутника налаштування/вирізання та рендеринг [9].

Сучасний графічний процесор – це не лише потужний графічний процесор, а й високопаралелізований обчислювальний процесор із високою пропускною здатністю та високою пропускною здатністю пам’яті для масивних паралельних алгоритмів, які називають обчисленнями GPU або обчисленнями загального призначення на GPU (GPGPU) [20].

На відміну від багатоядерних процесорів, які, як правило, не працюють,

виконують багато команд, працюють на високих частотах і використовують кеш-пам’ять великого розміру, щоб мінімізувати затримку одного потоку, GPGPU складаються з тисяч ядер, розташованих у порядку, працюючи на нижчих частотах і покладаючись на кеші меншого розміру.

Для створення високопродуктивних програм із прискоренням графічного процесора з паралельним програмуванням, різноманітні платформи розробки, наприклад обчислювальна уніфікована архітектура пристроїв (CUDA) і мова відкритих обчислень (OpenCL), вивчаються та використовуються для вбудованих систем із прискоренням GPU, настільних робочих станцій, корпоративних центрів обробки даних, хмарних платформ і серверів високопродуктивних обчислень (HPC).

Кілька виробників обладнання випустили графічні процесори. Серед них Intel, Nvidia та AMD/ATI були лідерами ринку.

Велика різноманітність графічних процесорів була розроблена для конкретного використання, наприклад Nvidia GeForce GTX і AMD Radeon HD GPU для потужних ігор, а також серії Nvidia Quadro і Titan X для професійних робочих станцій. Зовсім недавно поява технології глибокого навчання започаткувала значний прогрес уGPU обчислення.

Обчислювальна пропускна здатність, енергоспоживання та ефективність пам’яті є трьома важливими показниками при реалізації глибокого навчання на GPU.

При оцінці пам'яті ефективність графічних процесорів, розмір пам'яті та пропускна здатність пам'яті є двома важливими показниками. Сьогодні прийнято мати 11–12 ГБ пам’яті на найсучасніших ігрових картах. Для конкретного завдання глибокого навчання пікова продуктивність GPU часто далека від фактичної продуктивності.

У більшості практичних додатків пропускна здатність GPU становить близько 15–20% від його максимальної продуктивності [9]. Це, у свою чергу, означає, що оцінка DNN насправді обмежена пам’яттю натомість пропускну

здатність обчислювальної потужності.

Застосування графічних процесорів із високою пропускною здатністю пам’яті є правильною стратегією для прискорення як навчання, так і висновків.

Щоб вирішуючи фундаментальну проблему обмеженої обчислювальної пропускної здатності та пропускної здатності пам’яті, системи з декількома графічними процесорами допускають конфігурацію однієї машини та кількох графічних процесорів або навіть розподілених конфігурацій із кількома системами та кількома графічними процесорами. Для системи з однією машиною та декількома графічним процесорами, які працюють над окремими завданнями, можна отримати прямий доступ до будь-якого доступного графічного процесора без кодування в CUDA.

З іншого боку, для кількох графічних процесорів, які працюють над спільними завданнями, такими як навчання кількох моделей з різними гіперпараметрами, потрібне розподілене навчання. Nvidia має бібліотеку колективних комунікацій (NCCL) [21], який реалізує примітиви колективного зв’язку з декількома GPU та кількома вузлами, щоб повною мірою використовувати всі графічні процесори всередині і через кілька вузлів з максимальною пропускною здатністю.

Розподілене навчання тепер підтримується багатьма популярними фреймворками глибокого навчання, такими як Tensorflow, Caffe тощо. Ці методи скорочують обчислювальний час лінійно залежно від кількості GPU [22].

1.6 Програмовані вентильні матриці як засоби побудови апаратної архітектури для систем комп’ютерного зору

Незважаючи на те, що графічні процесори демонструють надзвичайно високу пропускну здатність і широко використовуються для апаратного прискорення мереж DNN, вони часто не є кращими для додатків з обмеженим енергоспоживанням, таких як пристрої Інтернету речей, через високе

енергоспоживання.

Таким чином, прискорення DNN рухається до альтернативного рішення на основі енергоефективних FPGA. FPGA дозволяє реалізувати нерегулярний паралелізм, індивідуальний тип даних і спеціальну апаратну архітектуру програми, пропонуючи велику гнучкість для адаптації останніх моделей DNN, які відрізняються підвищеною розрідженістю та компактними мережевими структурами.

Крім того, FPGA можна перепрограмувати після виготовлення для бажаних функцій і програм. Завдяки цим привабливим характеристикам для обох центрів обробки даних HPC було запропоновано велику кількість прискорювачів на основі FPGA і вбудовані програми.

FPGA – це компонент, який містить масив програмованих логічних блоків, з’єднаних через ієрархію реконфігурованих міжз’єднань. Сучасні FPGA зазвичай містять:

1) блоки обробки цифрових сигналів (DSP) для операцій множення-додавання-накопичення (MAC);

2) таблиці пошуку (LUT) для комбінаторних логічних операцій;

3) блок RAM для зберігання даних на кристалі.

Архітектура FPGA виконує модель DNN, виконавши кілька основних кроків. Він спочатку отримує вагові коефіцієнти DNN і вводить карти функцій у вбудований буфер (тобто ODM) від MDM.

Далі блок GEMM виконує матричні операції та передає результати до MLU для ReLU/нормалізації пакетів/об’єднання. Вихід MLU надходить до іншого блоку ODM і буде доступний для наступних згорткових та/або повністю підключених рівнів. Якщо вбудований буфер не має достатньо великої ємності, проміжні результати повинні бути тимчасово збережені на мікросхемі або позачипова пам'ять.

Історично система FPGA часто специфікувалася на рівні передачі регістрів (RTL) за допомогою мови опису обладнання (HDL), такої як Verilog або VHDL.

Ця методологія низькорівневого проектування потребує значних зусиль і досвіду в апаратному забезпеченні, щоб ретельно описати детальну архітектуру апаратного забезпечення, включаючи масову паралельність між різними апаратними модулями.

Нещодавно були успішно розроблені інструменти високорівневого синтезу (HLS) для полегшення ефективного проектування FPGA за допомогою мови програмування, наприклад C і C+, і автоматично компілювати високорівневий опис для створення низькорівневої специфікації (тобто HDL).

За допомогою вищезгаданого потоку синтезу вартість проектування прискорювачів FPGA може бути значно зменшена. Однак існує важливий компроміс між підходами RTL і HLS з точки зору дизайну вартість і продуктивність системи.

На практиці модель DNN часто навчається або налаштовується на високопродуктивній обчислювальній платформі, такій як GPU, тоді як прискорення FPGA реалізується для висновку DNN для обробки заданих вхідних даних на основі попередньо навченої моделі DNN.

Комп'ютер алгоритми бачення, засновані на DNN, часто пов’язані з високим обчислювальним навантаженням (тобто великою кількістю FLOP) і великим обсягом пам’яті (тобто великою кількістю параметрів мережі), тоді як пропускна здатність пам’яті FPGA часто становить менше 10% від цього графічних процесорів.

Отже, завдання гранту полягає в тому, щоб знайти ефективне відображення від попередньо навченої комплексної моделі DNN до обмежених апаратних ресурсів (тобто логіки високої щільності та блоків пам’яті), які пропонують FPGA.

Така технічна проблема була вирішена за допомогою зручних для апаратного забезпечення алгоритмів оптимізації, включаючи:

1) алгоритмічну роботу;
2) оптимізацію шляху даних;
3) стиснення моделі.

Алгоритмічна робота: обчислювальні перетворення, такі як GEMM, швидке перетворення Фур'є (ШПФ) і перетворення Вінограда, можна застосовувати до карти функцій та/або згорткові ядра, щоб зменшити кількість арифметичних операцій під час виведення.

GEMM – це популярний спосіб обробки DNN у процесорах і графічних процесорах, який векторизує обчислення згорткових і повнозв'язаних рівнів.

ШПФ перетворює двовимірну згортку на поелементне множення матриці, тим самим зменшуючи арифметичну складність. Він дуже ефективний для великого розміру ядра (>5) через велика кількість згорткових операцій між картами функцій і ядра.

Оптимізація шляху даних: щоб повністю використати паралелізм, шлях даних оптимізується шляхом розгортання згорткових шарів у CNN та відображаючи їх на обмежену кількість PE.

У ранніх реалізаціях FPGA елементи PE розташовані у двовимірній сітці як систолічний масив. Оскільки така проста архітектура обмежує розмір ядра CNN і не пропонує кешування даних, вона не може досягти надзвичайно високої продуктивності [23].

Нещодавно для вирішення вищезгаданої проблеми були запропоновані методи оптимізації циклу, включаючи перевпорядкування циклу, розгортання, конвеєрне розміщення та мозаїку. Перевпорядкування циклів намагається запобігти надлишковому доступу до пам'яті між циклами, щоб підвищити ефективність використання кешу. Розгортання циклу та конвеєрування максимізують використання FPGA джерела досліджуючи паралелізм ітерацій циклу [24].

Мозаїка циклу вирішує проблему, пов'язану з недостатньою пам'яттю на кристалі FPGA. Він розбиває карти функцій і ваги кожного шару, отриманого з пам'яті, на фрагменти, які також називають тайлами, щоб помістити їх у вбудовані буфери.

Стиснення моделі: DNN часто містять значну кількість надлишкових

параметрів і в основному використовуються для додатків, стійких до помилок. Тому було докладено багато зусиль для спрощення моделей DNN і, як наслідок, зменшення складності їх апаратної реалізації. Ці методи стиснення моделі можна загалом класифікувати на три різні категорії:

1) скорочення;
2) низькорангова апроксимація;
3) квантування [25].

По-перше, це обрізка зазвичай це перший крок до зменшення надмірності моделі шляхом видалення найменш важливих зв'язків та/або параметрів.

Беручи за приклад CNN, можна прибрати його вагомість, яка надзвичайно мала та/або спричинити високе споживання енергії [26]. Після скорочення модель CNN є дуже розрідженою і може бути ефективно реалізована за допомогою FPGA шляхом маскування нульових ваг для множень.

По-друге, апроксимація низького рангу розкладає вагову матрицю згорткового або повнозв'язного шару на набір фільтрів низького рангу, які можна оцінити з низькою обчислювальною вартістю. Нарешті, тому що арифметика з фіксованою комою вимагає менше обчислювальних ресурсів, ніж арифметика з плаваючою комою, карти функцій, вагові матриці та/або згорткові ядра можна квантувати за допомогою представлення з фіксованою комою для подальшого зниження вартості обчислень.

Простий підхід полягає в кодуванні кожного числового значення бажаною довжиною слова відповідно до його діапазону. Як альтернатива, динамічна схема може бути прийнята для призначення різних коефіцієнтів масштабування різним числовим параметрам в одній мережі [27].

Коли метод динамічного квантування застосовується як до згорткового, так і до повнозв'язаного шарів AlexNet без тонкого налаштування, точність класифікації майже не змінюється (<1%) [28].

У крайньому випадку DNN може використовувати двійкові ваги та активації, що призводить до надзвичайно компактного представлення, яке називають

бінарною нейронною мережею (BNN) [29].

BNN можна оцінити з надзвичайно низькими обчислювальними витратами, оскільки двійкове додавання та множення можуть бути реалізовані за допомогою простих логічних елементів.

Окрім BNN, потрійна DNN [30] встановлює свої ваги на 1, 0 або 1, дозволяючи кожній вазі бути представлено 2 бітами. З іншого боку, числові операції в усіх нейронах реалізуються за допомогою арифметики з плаваючою комою.

Продуктивність кількох типових моделей CNN, розгорнутих на FPGA з використанням різних методів оптимізації моделі.

Моделі CNN, засновані на квантованій арифметиці, є високоефективними з точки зору використання обладнання та енергоспоживання; однак їх точність часто скомпрометована.

З іншого боку, моделі CNN, засновані на низькоранговій апроксимації (наприклад, SVD) і скороченні, мають меншу кількість ваг, одночасно досягаючи високої точності класифікації.

Потрійний ResNet [31], реалізований за допомогою Intel StratixTM 10 FPGA, досягає пропускної здатності 12 TGOP/c, перевершуючи пропускну здатність TitanX Pascal GPU на 10%.

1.7 Дослідження спеціальних апаратних прискорювачів

Типова комп’ютерна система часто неоднорідна, складається з різнорідного набору процесорів, наприклад процесорів, до яких додаються інші несхожі процесори, щоб задовольнити конкретні вимоги до обчислень.

Процесори, які доповнюють центральні процесори, відомі як прикладні співпроцесори.

На додаток до відомих співпроцесорів, таких як графічні процесори та FPGA, існує кілька спеціалізованих апаратних блоків у формі автономних пристроїв або

співпроцесорів, які спеціально розроблені для глибокого навчання та/або інших додатків ШІ.

Тензорний процесор (TPU) [32], налаштований ASIC, розроблений Google, є автономним пристроєм, спеціально розробленим для нейронних мереж і налаштованим на фреймворк Google Tensorflow [33]. Він уже використовував багато програм Google, наприклад пошукову систему та AlphaGo [34].

Процесор нейронної мережі Intel Nervana (NNP) [35] призначений для забезпечення необхідної гнучкості примітивів глибокого навчання створюючи його основне обладнання.

Mobileye EyeQ – це сімейство пристроїв SoC, які спеціалізуються на обробці зображення під час автономного водіння. Він демонструє здатність обробляти складні та інтенсивні обчислювальні завдання зору, зберігаючи низьке енергоспоживання.

1.8 Висновки

У розділі розглянуто різні методи обробки та аналізу зображень, які дозволяють системам комп’ютерного зору виконувати різноманітні завдання, такі як виявлення об’єктів, розпізнавання образів, вимірювання параметрів тощо.

Одним із ключових факторів, що визначають продуктивність систем комп’ютерного зору, є їх апаратна архітектура. У наступних розділах буде розглянуто метод побудови апаратної архітектури на основі FPGA для систем комп’ютерного зору.

Побудова апаратної архітектури для систем комп’ютерного бачення на основі FPGA є привабливим рішенням, оскільки воно забезпечує гнучкість і масштабованість. FPGA можна програмувати та перепрограмувати для адаптації до конкретних потреб і вимог, що робить його чудовим вибором для додатків, які потребують настроювання.

Крім того, апаратні архітектури на основі FPGA можуть обробляти великі

обсяги даних у режимі реального часу, дозволяючи розробляти системи комп’ютерного зору в режимі реального часу, які можуть аналізувати складні ситуації та реагувати на них швидко й точно.

Крім того, використання FPGA може значно зменшити енергоспоживання систем комп’ютерного бачення, що робить його більш стійким і економічно ефективним рішенням порівняно з традиційними апаратними архітектурами. Це особливо важливо для таких застосувань, як безпілотні літальні апарати (БПЛА) і робототехніка, де енергоспоживання є критичним фактором.

Крім того, використання FPGA може уможливити реалізацію спеціальних алгоритмів і конвеєрів обробки, що може додатково оптимізувати швидкість системи. Розробивши архітектуру апаратного забезпечення, спеціально адаптовану до вимог системи комп’ютерного зору, можна досягти значно швидшого часу обробки та більшої пропускної здатності.

Ще одна перевага FPGA полягає в тому, що її можна використовувати для реалізації апаратних прискорювачів для певних алгоритмів або етапів обробки інформації.

Наприклад, апаратна архітектура на основі FPGA може включати апаратні прискорювачі для згорткових нейронних мереж (CNN), які зазвичай використовуються в програмах комп’ютерного зору. Використовуючи спеціальне обладнання для цих завдань, можна досягти значно швидшого часу обробки та більшої пропускної здатності [36].

Загалом побудова апаратної архітектури на основі FPGA може бути ефективним способом підвищення швидкості систем комп’ютерного бачення. Розробляючи спеціальну архітектуру, оптимізовану для конкретних вимог системи, можна досягти значного покращення швидкості обробки та пропускної здатності.

Підсумовуючи вищеописане, метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA пропонує кілька переваг, включаючи гнучкість, масштабованість, можливості обробки в реальному часі та знижене енергоспоживання.

Оскільки попит на ефективні та високопродуктивні системи комп’ютерного зору продовжує зростати, очікується, що використання FPGA стане більш поширеним у різних галузях промисловості.

2 FPGA ЯК ОСНОВА ДЛЯ РОЗРОБЛЕННЯ АПАРАТНИХ ПРИСКОРЮВАЧІВ

2.1 Дослідження елементної бази побудови апаратних прискорювачів

Основною функціональністю FPGA для реалізації поставленої задачі було здійснити реплікацію будь-якої логічної функції у спосіб, який можна перенастроїти. Як приклад наведемо наступну схему на рисунку 2.1.

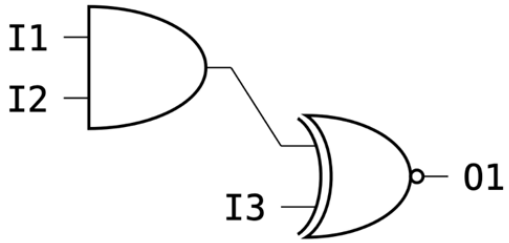


Рисунок 2.1 – Логічна функція з трьома входами та одним виходом. AND підключено до XNOR

Схему можна читати зліва направо. Функція приймає три входи і створює один вихід. Вентиль ліворуч називаються вентилями AND і видають істину на виході, лише якщо обидва вхідні дані також істинні. Другий вентиль — це вентиль XNOR, який видає значення true на виході, коли обидва входи рівні [37].

Обидва вентилі схеми на рисунку 2.1, а також їх сумарний вихід можуть бути представлені в окремих таблицях істинності, як показано в таблиці 2.1.

Комбінована таблиця істинності є таблицею з трьома входами та одним виходом. У світі FPGA така таблиця істинності називається таблицею пошуку (LUT). Усі логічні функції, які можна виразити, також можна представити у формі LUT, що робить їх ідеальними базовими будівельними блоками для FPGA.

Таблиця 2.1 - Таблиця істинності, отримана зі схеми на рисунку 2.1, де a) AND, b) XNOR, c) комбінована.

I1	I2	O1
0	0	0
0	1	0
1	0	0
1	1	1

а)

I1	I2	O1
0	0	1
0	1	0
1	0	0
1	1	1

б)

I1	I2	I3	O1
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	1	1

с)

LUT, що було використано в FPGA, набагато більша, ніж три входи та один вихід. Типова конфігурація – шість входів і один вихід. Як правило, ці LUT

реалізуються як дуже швидкі комірки SRAM, які втрачають свій стан після вимкнення живлення. Деякі FPGA використовують Flash, щоб уникнути порожнього пристрою під час завантаження за рахунок продуктивності та ефективності. Щоб підвищити гнучкість, такі LUT зазвичай можна об’єднувати для створення більших LUT або розділяти на окремі функції залежно від варіанту використання [38].

Другим ресурсом будь-якої сучасної FPGA є тригер (FF). FF — це однобітові елементи стану, які зберігають свій вхідний сигнал на передньому фронті тактового сигналу.

Кожна з LUT є висококонфігурованою та підтримує такі режими:

1) Будь-яка довільно визначена логічна функція з шістьма входами;

2) Дві довільно визначені булеві функції з п'ятьма входами, якщо ці дві функції мають спільні входи;

3) Дві довільно визначені булеві функції з трьома і двома входами або менше.

Відповідно, один CLB може бути як вісім LUT з шістьма входами, як шістнадцять LUT з п’ятьма входами (за умови, що кожна пара з двох має спільні входи) або будь-яка комбінація цих двох. LUT також можна використовувати як так звані мультиплексори, тип схеми, яка перенаправляє певний вхідний порт на вихід залежно від стану набору бітів вибору. Ці мультиплексори можна використовувати для маршрутизації сигналів всередині CLB [39].

Крім того, CLB містить різноманітні функціональні можливості, які можна використовувати для ефективної реалізації певних проектів. Наприклад, спеціальну логіку переносу можна використовувати для ефективної реалізації арифметичних операцій. Нарешті, LUT можна використати як пам’ять з довільним доступом на відміну від їх звичайної поведінки як пам’яті лише для читання.

За допомогою CLB можна представити будь-яку логічну функцію, яка відповідає ресурсам. Однак зазвичай логічна функція, наприклад множник 32 × 32

біта, набагато більша, ніж може вмістити один CLB. Відповідно, можна забезпечети спосіб підключення різних CLB.

2.2 Застосування HDLS

В роботі для забезпечення системного програмування було застосовано мови програмування: VHDL і Verilog. Обидві мови зазвичай використовуються на рівні реєстру-передачі (RTL) і зазвичай можуть бути транспільовані одна в іншу. Крім проектування схем за допомогою мов, також підтримується перевірка, і обидві мови було використано для створення тестових стендів. Їхній низький рівень абстракції дозволив реалізувати метод побудови апаратної архітектури для система комп'ютерного зору на основі FPGA. HCL, описані далі в цьому розділі, зазвичай компілюють свій код на будь-яку мову [40].

У деталях мови мають певні особливості, через які вони є досить різними, незважаючи на однакову силу. У VHDL дріт, який не підключений, призводить до попереджень і помилок далі в процесі інструментів. З іншого боку, Verilog не забезпечує виконання багатьох правил, які вимагаються VHDL.

Обидві мови мають певні застереження, пов'язані з їх походженням як мов перевірки системи. Відповідно, вони містять конструкції, які не можна перенести в апаратне забезпечення, оскільки вони застосовні лише для моделювання, що часто є незручністю для початківців, які намагаються потрапити в апаратне забезпечення. Зазвичай схеми добре симулюють, але інструменти, що використовуються для синтезу, не можуть працювати з кодом або створювати небажані представлення обладнання [41].

Незважаючи на використання деяких узагальнених концепцій, таких як цикли для копіювання обладнання, абстракція вищого рівня неможлива. Узагальнення в основному відбувається за допомогою параметрів, які можна використовувати для визначення різної ширини регістрів або додаткових вкладок

фільтрів. Основні функції, однак, є жорстко з’єднаними і потребують більш детальних змін для нових програм [42].

З роками з’явилися нові мови, спрямовані на покращення HDL шляхом додавання функцій високого рівня або різних методологій проектування. Ці мови все ще є HDL, але цей термін зазвичай асоціюється з Verilog і VHDL. З іншого боку, термін HLS також не підходить належним чином, оскільки мови, які використовуються для HLS, безпосередньо не представляють апаратне забезпечення, а зазвичай є мовами програмування DSL або переробленим програмним забезпеченням. Автори Chisel ввели термін HCL для своєї мови, щоб уникнути неправильного тлумачення традиційних нотацій [43].

2.3 Застосування мови побудови апаратного забезпечення

Мови побудови обладнання (англ. HCL – Hardware Construction Languages) – це підмножина HDL, яка спрямована на вдосконалення традиційних представників HDL кількома способами. Типовими для HCL є система типів і вищий рівень абстракції. Наприклад, велика частина функціональних можливостей для синхронізації, таких як рукостискання, може бути оброблена неявно.

У той час як HCL безпосередньо моделюють апаратне забезпечення, будь то на високому рівні абстракції, HLS використовують мови без будь-якого відношення до розробки апаратного забезпечення та виводять схеми з логіки мов (лістинг 1.1 у Додаток А) [44].

2.4 Синтез високого рівня (HLS – High Level Synthesis)

Як HDL, так і HCL мають круту криву навчання через їхню природу опису апаратного забезпечення. Розумний процес, що йде до розробки схеми, нелегко порівняти з розробкою програмного забезпечення. Особливо для постачальників FPGA це створює велику маркетингову проблему в галузі прискорювачів. Програмування багатопоточних процесорів і графічних процесорів дуже просте і в

значній мірі порівнянне одне з одним. Проте розкриття можливостей прискорення FPGA вимагає від розробника вивчення абсолютно нових мов із зовсім іншими характеристиками.

Виходячи з цієї потреби, усі основні виробники FPGA розробили так звані потоки інструментів HLS, орієнтовані на розробників програмного забезпечення без серйозних знань про апаратне забезпечення.

Схема описується не її успадкованою поведінкою, а через їх функціональність. Потім компілятор бере опис і намагається вивести робочу схему з очікуваною функціональністю. Для мов програмування загального призначення це не є тривіальним завданням, і багато поколінь вчених та інженерів однаково працюють над цією, все ще відкритою, проблемою.

Такі деталі, як правильне планування завдань над методами опису апаратних засобів задані ресурси, введення меж реєстрів для формування конвеєрів або обробка рекурсії та циклів роблять HLS дуже вимогливою темою для інженерів-компіляторів.

HLS зазвичай відноситься до синтезу обладнання з мов програмування загального призначення, таких як C і C++. Будучи найпоширенішим підходом, він значною мірою підтримується багатьма інструментами. HLS на основі DSL використовує інший підхід. Замість підтримки мов програмування загального призначення з супутніми проблемами в якості основи для синтезу використовуються спеціалізовані мови для вузької області.

Типові приклади доменів включають машинне навчання та біоінформатику. Вужчий набір мовних функцій покращує здатність до синтезу та покращує продуктивність кінцевої схеми. Крім того, представлено SystemC, суміш підходів HCL і HLS. Дотримуючись багатьох функцій, наявних у HLS, SystemC також підтримує опис деталей обладнання, таких як поведінка синхронізації [45].

2.4.1 Мова програмування на основі HLS

Синтезувати апаратне забезпечення безпосередньо з реалізацій програмного забезпечення, звичайно має свої переваги, які очевидні: програмне забезпечення, яке погано працює на апаратному забезпеченні загального призначення, можна просто скомпільувати для запуску безпосередньо на апаратному забезпеченні зі значним збільшенням продуктивності. Однак на практиці все по-іншому, незважаючи на численні спроби [29]. Академічні (і частково також промислові) інструменти, такі як LegUp або Nymble, можуть синтезувати підмножини відповідних базових мов, як правило, C, але їм бракує кількох аспектів, що робить їх нежиттєздатними для продуктивного використання. LegUp, наприклад, не підтримує доступ до основної пам’яті та працює лише з невеликими блокнотами.

У промисловості ситуація така ж. Незважаючи на високі заявки про збільшення продуктивності, комерційні інструменти, такі як Vivado HLS, не виправдовують багатьох аспектів.

Код можна легко синтезувати через VivadoHLS. Однак продуктивність дуже низька і не наближається до кастомного прискорювача. VivadoHLS може працювати краще, але потребує додаткової допомоги для досягнення належної продуктивності.

Однак оптимізована вручну версія є меншою програмною реалізацією, а скоріше близькою до апаратної реалізації. Регістр зсуву використовується для збереження стану останньої ітерації. Додаткові прагми використовуються, щоб вказати компілятору, як призначені певні цикли.

З роками стало зрозуміло, що важко досягти продуктивності спеціалізованих апаратних прискорювачів, використовуючи загальні мови програмування, такі як C, як основу. Їх цільова архітектура просто занадто відрізняється, щоб бути корисною.

З іншого боку, предметно-спеціальні мови надають неймовірні можливості, звужуючи простір проектування та допускаючи багато неявних оптимізацій для конкретного обладнання.

2.4.2 SystemC

SystemC – це бібліотека, написана мовою C++ для опису обладнання. Він розширює VHDL і Verilog, додаючи функції високого рівня на основі базової мови програмування. Наприклад, абстракція може бути виконана за допомогою класів і шаблонів. Крім того, доступна система типу C++. Бібліотека додає функції для підтримки апаратних конструкцій, таких як бітові типи та механізми інтерфейсу між різними модулями. На основі бібліотеки можна створити виконуваний файл моделювання за допомогою стандартних компіляторів C++.

Однак, порівняно з HCL, створити Verilog або VHDL з моделі SystemC непросто. Для цілей FPGA такі інструменти, як VivadoHLS, пропонують попередню підтримку, але зазвичай вона навіть слабша, ніж підтримка загального призначення C або C++. Теоретично мовні конструкції можуть надавати додаткові вказівки для компілятора, але на практиці застосовуються ті самі застереження, що й для HLS на основі мов програмування загального призначення [46].

Приклад фільтра FIR можна реалізувати в SystemC (лістинг 1.2 у Додаток А). Основні функції даного модуля надаються як модуль, який є класом C++ із додаванням синтаксичного цукру. Спеціалізовані типи використовуються для позначення входів і виходів модуля. Наприклад, для FIR-фільтру нам потрібен канал вхідних даних і канал вихідних даних, включаючи рукостискання та годинник/скидання, які виглядають так:

```
sc_in_clk clk;
sc_in<bool> reset;
sc_in<bool> input_valid;
 sc_in<sc_int<16>> input;
```

```
sc_out<bool> output_valid;
sc_out<sc_int<16>> output;
```

Сам модуль створюється за допомогою макросів у бібліотеці.

Функціональність модуля виглядає як будь-яка інша функція C++. Єдиним доповненням є те, що функція wait() доступна для очікування наступного такту.

Мова може бути використана для легкого вираження складних схем і широко використовується для цієї мети розробниками ASIC. Однак відсутність підтримки інструментів робить мову менш цікавою для цілей FPGA.

2.4.3 Доменно-спеціальна мова HLS

Розробка виділених мов для певної області не є новим підходом і широко використовується в багатьох областях, таких як машинне навчання, протягом багатьох років.

Замість того, щоб мати справу з деталями архітектури основного апаратного забезпечення, наприклад графічного процесора, розробник повинен мати справу лише зі своєю конкретною проблемною областю.

Розробники мови мають визначити, які функції необхідні, і можуть надати високооптимізовані апаратні версії необхідних основних будівельних блоків. Розробники мов також можуть виключити концепції, які не є необхідними для домену, такі як рекурсія, і зосередитися виключно на необхідних функціях. Така поведінка також має великий сенс для цілей FPGA.

DSL, призначені для FPGA, можуть використовувати високооптимізовані реалізації базових примітивів. Наприклад, обробка сигналу націлювання DSL може забезпечити спеціалізовані реалізації для фільтрів у різних форматах чисел.

Потім користувач може використовувати мову, щоб вказати, які фільтри слід використовувати в якому порядку.

Цю мову не можна буде використовувати для інших областей, таких як загальні обчислення лінійної алгебри, але вона буде дуже добре підходити для конкретних завдань обробки сигналів (лістинг 1.3 у Додаток А) [47].

Загалом, DSL добре підходять для використання для високорівневого синтезу.

Відповідно, багато різних проектів роблять саме це. Особливістю DSL є те, що симуляцію та перевірку зазвичай можна виконувати прямо на центральному процесорі.

Необхідний код може бути автоматично згенерований на основі того ж джерела DSL (лістинг 1.4 у Додаток А).

Таким чином, зміни необхідні для покращення можливостей застосування та зручності використання FPGA, особливо в середовищі центру обробки даних:

1) Доступність має зрости. Бажано розгортати як частину стандартних процесорів або як дешеві (і корисні) додаткові карти. Графічні процесори дійсно починалися як спеціалізовані графічні прискорювачі для аудиторії ентузіастів, а згодом стали потужними обчислювальними центрами, які ми знаємо сьогодні;

2) Проектування, схоже на асемблер, не повинно бути золотим стандартом. Навчання сучасних HDL має бути зосереджене як на наукових колах, так і на промисловості;

3) Перейти від невдалих підходів HLS до підходів DSL, які краще оптимізуються та прості у використанні;

4) Уніфікувати націлювання на різні пристрої. Багатьом додаткам не потрібні всі види спеціалізованих функцій, які пропонує сучасна FPGA, а просто потрібен доступ до пам’яті та засоби зв’язку;

5) Конкуренція повинна посилюватися завдяки сумісності. Блокування постачальників завдає шкоди всім залученим сторонам;

6) Необхідно зосередити увагу на розробці ланцюжків інструментів з відкритим кодом на рівні підтримки екосистеми GCC.

У наш час відносно легко націлюватися на різноманітні пристрої за допомогою безкоштовного відкритого програмного забезпечення.

Обмін кодом через такі платформи, як Github, є звичайним явищем, і навіть великі корпорації базують свій бізнес на програмному забезпеченні з відкритим кодом, а ще краще – вони активно фінансують розробку проектів з відкритим кодом відповідно до власних потреб, але також отримують користь від того, що спільнота ділиться своїми результатами на цих платформах.

Поки FPGA залишатиметься у своїй невеликій ніші з високою вартістю входу та обмеженою підтримкою, вони залишатимуться аутсайдерами на ринку прискорювачів [48].

Постачальники ланцюжків інструментів що працюють нарівні з традиційними інструментами FPGA, орієнтованими на кремній (ASIC), з їх надзвичайно високими бар'єрами входу.

Натомість основні гравці повинні почати діяти більше як компанії, що займаються програмним забезпеченням, і відкрити свої зусилля на благо багатьох.

Відносно невеликі виробники FPGA намагаються встигати за попитом на нові і покращені ланцюжки інструментів щоразу, коли надходить нова розробка FPGA.

Крім програмного забезпечення, вони також повинні підтримувати свій набір власних IP-ядер для різноманітних програм.

2.5 FPGA для експертів

Два великих постачальники FPGA, безперечно, знають про проблеми впровадження, з якими стикаються під час розгортання FPGA. Їх поточний погляд на цю проблему полягає в інструментах синтезу високого рівня, які використовують мови, що використовуються для націлювання на процесори або графічні процесори, такі як OpenCL, і спотворюють їх для націлювання на FPGA. Однак продуктивність, досягнута цими мовами, може бути низькою.

Крім того, код, який потрапляє в компілятор, зовсім не схожий на програмну реалізацію.

Відповідно, майбутній програміст FPGA повинен вивчити специфічний діалект мови, який добре працює при компіляції на FPGA.

Крім того, цей підхід не новий і дослідження в цій галузі майже такі ж старі, як і самі FPGA.

Незважаючи на величезні дослідницькі зусилля, серйозних проривів досягнуто не було [49].

Графічні процесори є основним використовуваним прискорювачем, але також доступна велика кількість інших спеціальних прискорювачів. З точки зору програміста, нічого не змінюється при використанні іншого типу прискорювача, крім продуктивності виконання. Код написаний не якоюсь мовою загального призначення, а натомість у вузькоспеціалізованих DSL, які підтримують саме ті функції, які потрібні для конкретного домену. Цей підхід вигідний розробникам апаратного забезпечення, оскільки вони можуть зосередитися на написанні високооптимізованих ядер для певних важливих завдань і використовуйте компілятор, щоб зшити ці ядра разом для даної програми. Цей підхід дає багатообіцяючі результати [50].

Одним з аспектів розробки DSL для цілей FPGA, який слід добре розглянути, є абстракція пристрою. Повинно бути якесь проміжне представлення, яке не залежить від структури FPGA та інтерфейсів із шаром абстракції.

2.6 Компоненти побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA

У даному розділі розглянуто компоненти побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA. З цією метою існує переміщення даних у FPGA та з нього за допомогою компонентів, які реалізують фундаментальні операції.

Для пристроїв на базі PCIe TaPaSCo використовує блок прямого доступу до пам’яті (DMA) для переміщення даних із пам’яті хоста безпосередньо в пам’ять пристрою. Схема його роботи представлена на рисунку 2.2.

Пам'ять хоста доступна через PCIe 3.0. Пам'ять DDR на пристрої доступна через контролери пам'яті. Механізм DMA переміщує дані з однієї пам’яті в іншу. Сині поля відсутні на FPGA.

Сірі коробки надає постачальник, у цьому випадку Xilinx.

Операція контролюється хостом через канал PCIe.

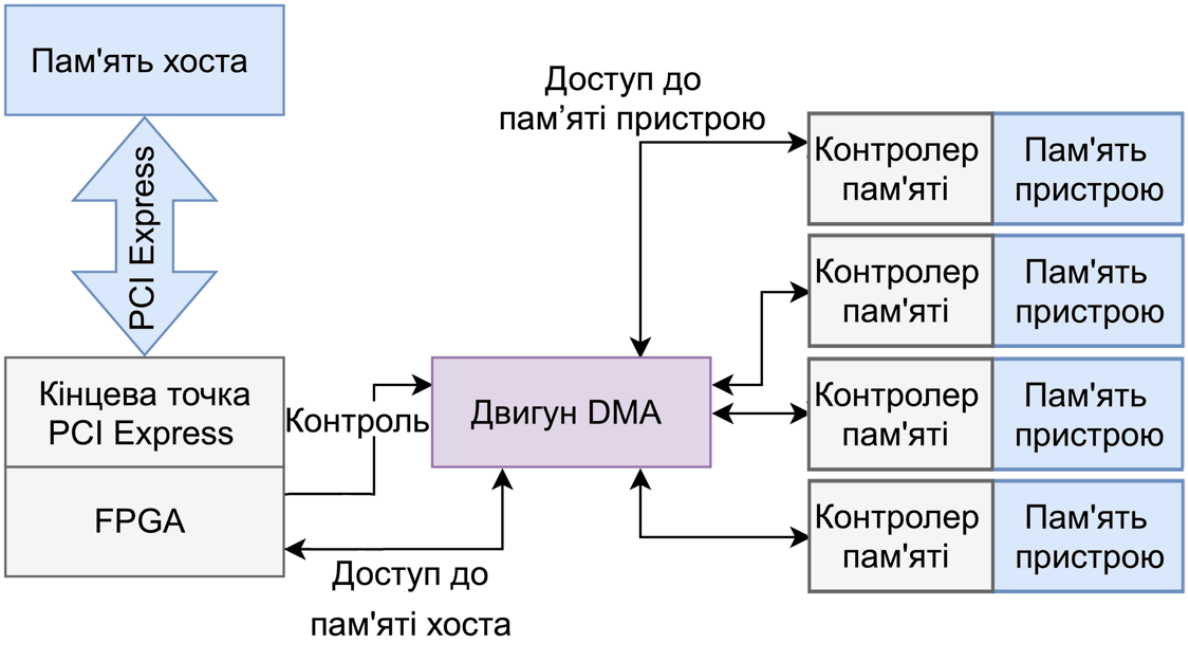


Рисунок 2.2 – Огляд системи DMA на FPGA

Механізм DMA розташований у пристрої між інтерфейсом PCIe та пам’яттю DDR. Передача здійснюється шляхом ініціалізації модуля DMA адресами пам’яті хоста та FPGA, напрямком і розміром передачі. Подальший процес може бути таким простим, як: почніть передавання на відповідних інтерфейсах і перемістіть дані в цільове розташування.

Linux не надає доступу до фізично безперервної пам’яті для простору користувача. Замість цього призначається віртуальна пам'ять користувача, яка

зазвичай не є фізично безперервною та нелегко доступною для підключеної FPGA PCIe.

Зокрема, ядро ОС не виділяє фізично безперервну пам'ять, оскільки це дуже дефіцитний ресурс. Відповідно, 4 МБ є типовим максимальним розміром цих областей пам’яті.

Двома часто використовуваними методами є буферизація відмов і списки розсіювання/збирання. Для останнього список сторінок у пам’яті простору користувача можна безпосередньо передати в FPGA, і пристрій має переконатися, що передаються правильні сторінки. Перевага полягає в тому, що копії між ділянками пам’яті хоста не відбуваються, але блок DMA має бути складнішим. TaPaSCo використовує інший підхід і використовує буфер відмов. Для цього підходу пам’ять користувача переміщується частинами до безперервної пам'яті, а потім механізмом DMA до пам'яті FPGA [51].

TPC використовує просту реалізацію DMA під назвою DualDMA, яка здатна обробляти одну передачу та не має черги для зберігання кількох запитів. TaPaSCo представляє BlueDMA як заміну з додатковими функціями та покращеною продуктивністю.

У наведеному нижче прикладі дослідження зосереджено на вивченні різних проектних рішень і наведено аргументи на те, щоб не винаходити колесо, оскільки досягнення оптимальної продуктивності на FPGA не є таким тривіальним, навіть у менш складних сценаріях.

Повертаючись до буферизації відмов. Конструкція для реалізації механізму DMA з використанням буферизації відскоків може працювати відповідно:

1) Виділити великий фрагмент (зазвичай об’ємом близько 4 МБ) безперервної пам’яті;

2) Перемістити фрагмент із пам’яті простору користувача в безперервну пам’ять;

3) Наказати механізму DMA перемістіть фрагмент у правильне місце в пам’яті FPGA;

4) Повторити пункт 2 до пункту 3, доки не буде передано всі фрагменти.

Цей підхід безсумнівно працює та досягає приблизно 17,8 % і 17,3 % для читання та запису відповідно від практичної пропускної здатності PCIe 3.0 x8 7306 МБ/с, як показано на рисунку 2.3.

Це пікове значення досягається при розмірі передачі лише 256 КБ і значно падає як для більших, так і для менших передач.

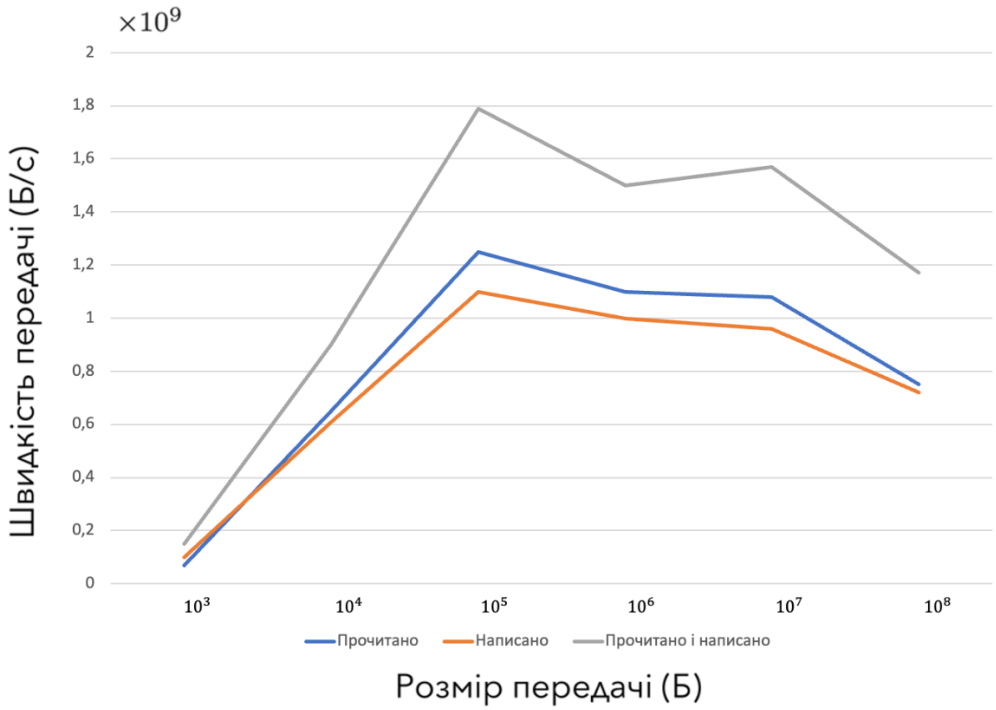


Рисунок 2.3 – Продуктивність для читання, запису та обох паралельно через PCIe 3.0 для буфера відскоку розміром 4 МБ.

Відповідно, продуктивність не відповідає тому, що можна очікувати від PCIe. З’ясування причини обмеженої продуктивності часто є одним із найскладніших завдань при націлюванні на FPGA.

Симуляція повної системи не є тривіальною, оскільки вимагає точних моделей усіх задіяних частин.

Для проектування під контролем користувача необхідні моделі доступні негайно.

Однак моделювання частин, які надає постачальник FPGA, можливе лише на милість творців IP.

Інтерфейс PCIe постачається з простою функціональною моделлю, яку не можна використовувати, оскільки вона недостатньо точна для виявлення проблем із продуктивністю, які виникли під час тестування пристрою.

Ще менше інформації доступно про решту ланцюга, що веде до хоста, наприклад про мости та кореневі комплекси.

Відповідно, в цьому випадку моделювання може знайти недоліки лише в логічній частині користувача, але не у всій системі. Випробування показали, що механізм DMA працює в очікуваному діапазоні та здатний перевищити продуктивність, необхідну для PCIe 3.0 x8 [52].

Оскільки симуляція не змогла вирішити цю проблему, послідував повільний і схильний до помилок процес налагодження апаратного забезпечення.

Перегляд FPGA під час обробки можливий за допомогою наданих постачальником логічних аналізаторів, які можна ввести в бітовий потік під час генерації. Ці логічні аналізатори можуть фіксувати певні сигнали під час виконання.

Вибірка починається на основі тригерів, таких як зміни сигналу або комбінація сигналів. Недоліком логічних аналізаторів є те, що вони часто збільшують час генерації бітового потоку. Крім того, один логічний аналізатор може охоплювати лише обмежену кількість сигналів і не зможе захопити всі цікаві сигнали одночасно, що призводить до кількох бітових потоків для ретельного дослідження [53].

Відповідно, апаратне налагодження слід розглядати як крайній засіб після того, як інші методи налагодження не виявили проблему.

Під час налагодження апаратного забезпечення справді виявлено одну проблему: під час читання міст PCIe надавав дані лише через кожний другий такт. Далі було глибоке занурення в технічну документацію всіх задіяних частин.

Зрештою виявилося, що хост-платформа не здатна підтримувати необхідні швидкості передачі даних.

Перехід на новішу платформу призвів до продуктивності, показаної на рисунку 2.4.

Однак графік, створений після переходу на ефективнішу платформу, також показує, що це не єдина проблема, яка призвела до низької продуктивності.

Копаючи глибше, висвітлюється інша проблема. Самі передачі є досить швидкими та часто підтримують очікувану пропускну здатність PCIe 3.0 x8. Однак хост не встигає за попитом, і час між двома передачами швидко збільшується. Ця проблема вже відома і раніше була вирішена за допомогою кількох незалежних механізмів DMA. Однак цей підхід є марнотратним, оскільки кожен блок DMA вже здатний досягти необхідної пропускної здатності. Будь-який інший механізм DMA повинен чекати в черзі, поки попередній не закінчиться, в основному не використовуючи ніякого паралелізму.

Натомість для BlueDMA використовується інший підхід. Двигун має чергу команд, яка зберігає запити, які можна обробляти якомога швидше один за одним.

Це доповнення також вимагає змін драйвера в головній системі. Початкові кроки буферизації відмов потрібно змінити. Замість виділення лише одного великого фрагмента розміром 4 МБ виділяється кілька фрагментів різного розміру. Драйвер автоматично вибирає найкращий розмір для хост-системи на основі попередніх тестів. Потім хост може заповнити наступний буфер і поставити його в чергу в FPGA, поки механізм DMA зайнятий обслуговуванням запитів [54].

На рисунку 2.5 показана швидкість передачі, досягнута для різних розмірів буфера для читання.

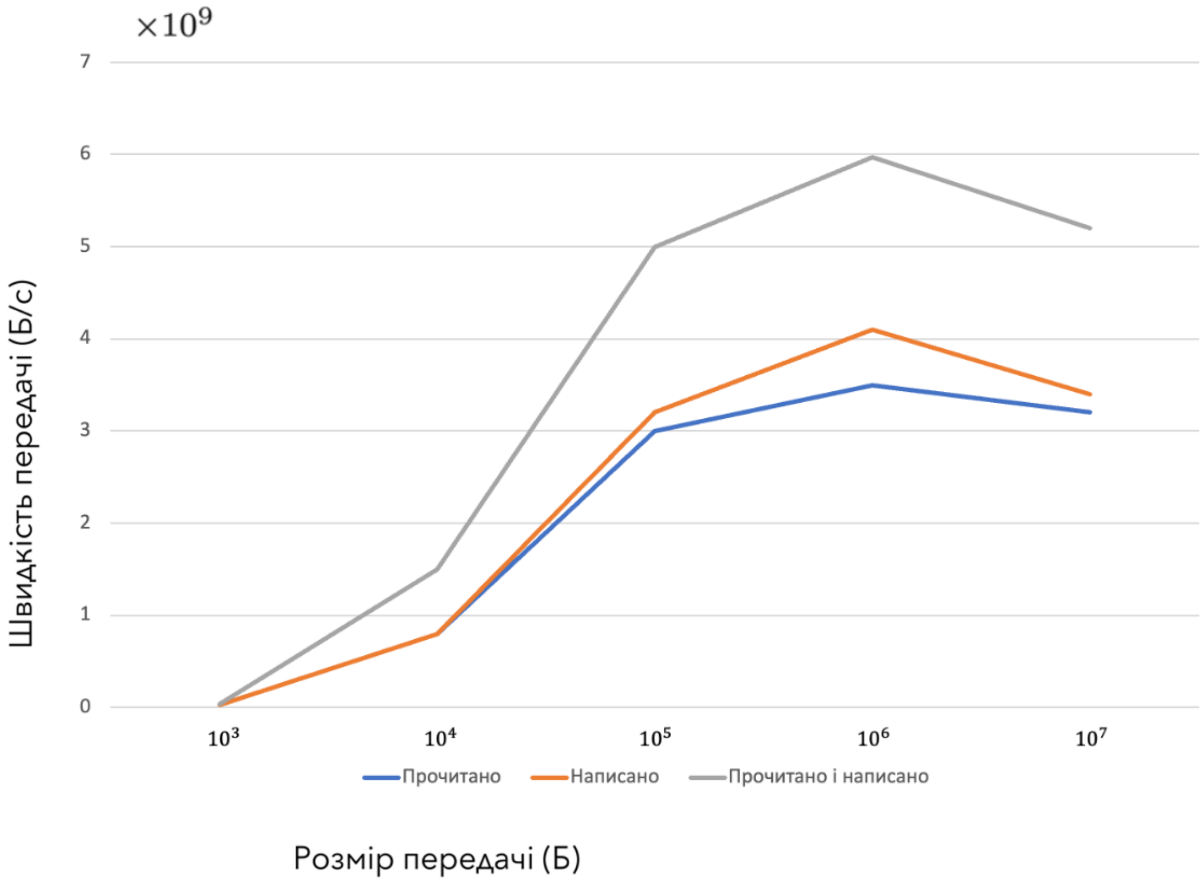


Рисунок 2.4 – Продуктивність для читання, запису та обох паралельно через PCIe 3.0 для буфера відскоку розміром 4 МБ

Запис і паралельне читання та запис покращено в одному рядку та не нанесено на графік для кращої читабельності. Продуктивність набагато краща при використанні багатьох малих буферів замість кількох великих буферів, оскільки навантаження краще розподіляється між залученими сторонами. Різниця між оптимальною пропускною здатністю та досягнутою пропускною здатністю зменшується до 78,9 % і 65,4 % для читання та запису відповідно. Для найефективнішої конфігурації з 16 фрагментів по 256 КБ швидкість передачі зменшується набагато менше для більших передач і залишається приблизно лінійною для передач до кількох ГБ. Все ще існує досить помітна різниця між обіцяною та досягнутою швидкостями, особливо на стороні запису, що вимагає подальшого дослідження.

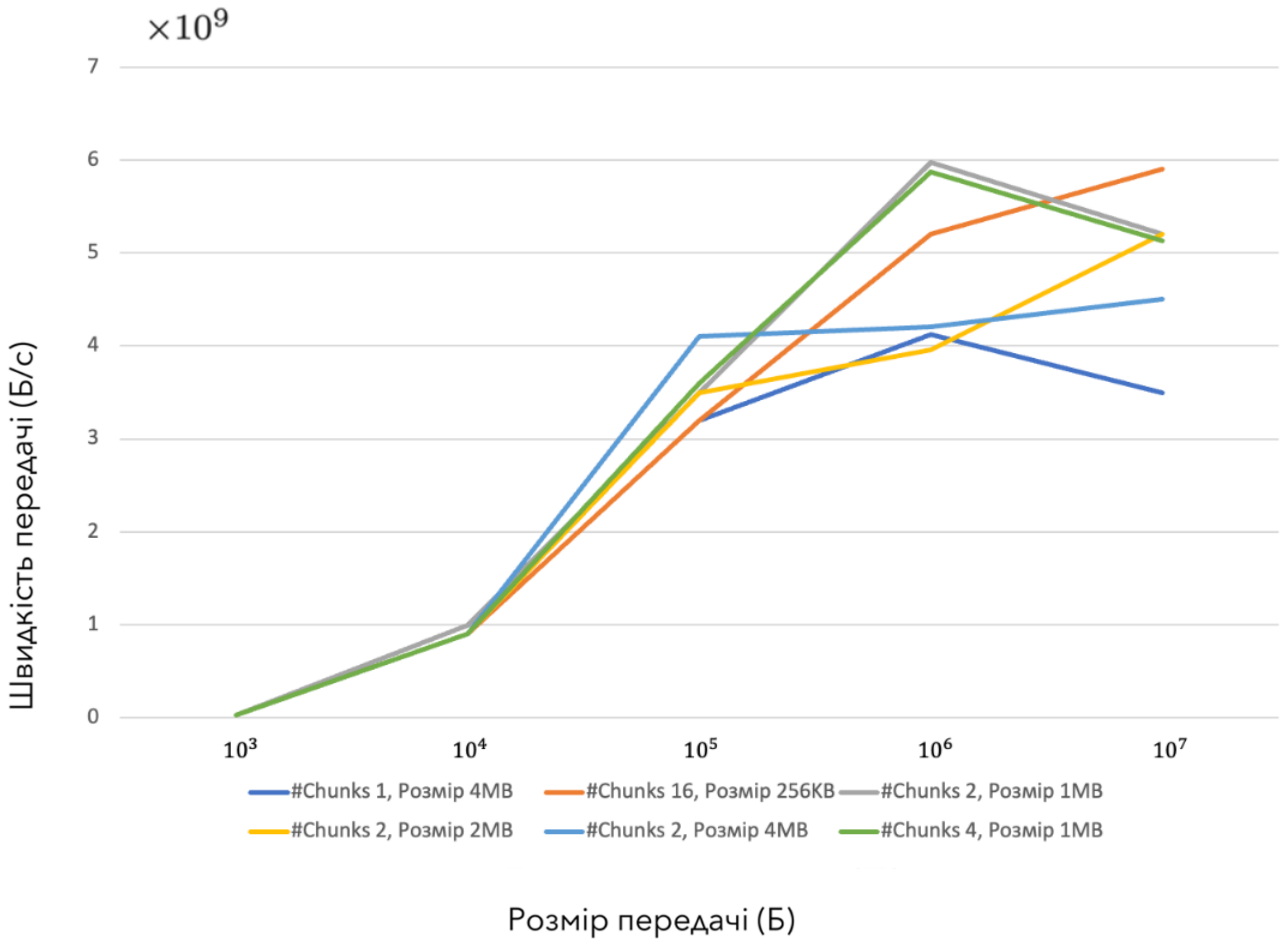


Рисунок 2.5 – Продуктивність для читання (переміщення даних із FPGA на хост) через PCIe 3.0 для різних конфігурацій буферів відскоку

Додавання додаткових методів для зменшення накладних витрат на хост, таких як буфери нульового копіювання замість буферів відмов, може додатково збільшити використання пропускної здатності [55].

Слабша система демонструє подібне підвищення продуктивності, використовуючи кілька маленьких блоків, порівняно з одним великим логічним блоком.

Однак слабший ЦП вимагає інших налаштувань і необхіднної кількості буферів відскоку.

Відповідно, потрібно використовувати ініціалізацію драйвера для конкретної платформи та проводити тестування на різних платформах, щоб знайти оптимальні конфігурації.

Цей приклад висвітлює певні аспекти розробки FPGA. Навіть завдання, які спочатку здаються простими, такі як переміщення пам’яті, мають цілу низку розробницьких рішень, які сильно впливають на продуктивність і часто їх важко відстежити.

Орієнтація на FPGA вимагає більше роботи, ніж просто реалізація бажаної функціональності як IP-ядра та натискання кнопки синтезу. Без необхідної інфраструктури на кристалі для зв’язку з FPGA неможливо контролювати виконання ядра IP. Крім того, доступ до зовнішньої пам’яті чи інших периферійних пристроїв недоступний за замовчуванням.

Отже, користувачеві потрібен якийсь рівень підтримки, який забезпечує всю навколишню логіку навколо самого ядра IP. Традиційно периферійна розробка створюється вручну для одного IP-ядра на одному пристрої.

Відповідно, проект важко перенести на іншу FPGA, оскільки цей крок вимагає повної зміни навколишньої інфраструктури [56].

Комерційні інструменти, які намагаються зробити IP портативним, такі як SDAccel або SDSoC, існують, але вони мають вузьку сферу застосування. Перший підтримує лише невелику кількість плат на основі PCIe, а другий працює лише на пристроях на базі Zynq.

2.7 Висновки

Даний розділ містить детальне дослідження елементної бази побудови апаратних прискорювачів для створення апаратної архітектури систем комп'ютерного зору на основі FPGA. Досліджено класичні мови побудови апаратного забезпечення, які дозволяють підвищити продуктивність та ефективність систем комп'ютерного зору на FPGA. Крім того, проведено дослідження синтезу високого рівня (HLS) та мови програмування на основі HLS, що дозволяє автоматизувати та прискорювати процес проектування апаратної архітектури на FPGA.

Також, в розділі розглянуто метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA та досліджено його застосування. Отримані результати дослідження стали основою для розробки апаратного методу побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA, який буде детально описаний у третьому розділі.

Загальна мета цього розділу – вивчити та дослідити можливості побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA, що дозволяє підвищити продуктивність та ефективність таких систем та забезпечити їх більш точну та швидку роботу.

3 МЕТОД ПОБУДОВИ АПАРАТНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA

Для досягнення мети підвищення швидкодії систем комп'ютерного зору було запропоновано метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA.

Дозволити комп'ютерам сприймати навколишнє середовище все ще залишається одним із найскладніших завдань комп'ютерного зору. Особливо стереозір, сприйняття глибини за допомогою двох камер, важливий для багатьох сфер, таких як робототехніка та автономне водіння. Стереозйомка використовує дві камери, розташовані на деякій відстані одна від одної по горизонталі, але на одному рівні по вертикалі. Таким чином, пікселі на зображеннях, знятих двома камерами, зміщуються лише в горизонтальному напрямку, причому максимальне зміщення пікселя (традиційно називається невідповідністю) обмежується відстанню між двома камерами [57].

Потім обчислену невідповідність для пікселя можна використовувати для отримання інформації про глибину зі стереозображень за допомогою тріангуляції (пікселі, розташовані ближче до камер, мають більшу невідповідність).

Алгоритм у центрі цієї роботи називається Semi-Global Block Matching (SGBM), який є одним із найшвидших алгоритмів, який також показує високу точність у тестах стереозору, таких як KITTI і Middlebury. Це призвело до його широкого використання в багатьох практичних застосуваннях [58].

Основою запропонованого методу є синтез нової апаратну архітектуру програмно-технічного засобу для комп'ютерного зору, що застосовує FPGA, а також використовує алгоритм SGBM.

Метод забезпечує синтез побудови параметризованої архітектури для систем комп'ютерного зору на основі FPGA архітектура, яка має високу масштабованість, що дозволяє легко впроваджувати її на малих малопотужних пристроях (наприклад, в автономних роботах), а також на великих високопродуктивних чіпах

(наприклад, у стаціонарних випадках використання для обробки кількох відеопотоків високої роздільної здатності).

Як показано на рисунку 3.1, основна увага приділяється обчисленню невідповідності; виправлення та фактичні інтерфейси камери обговорюватися не будуть.

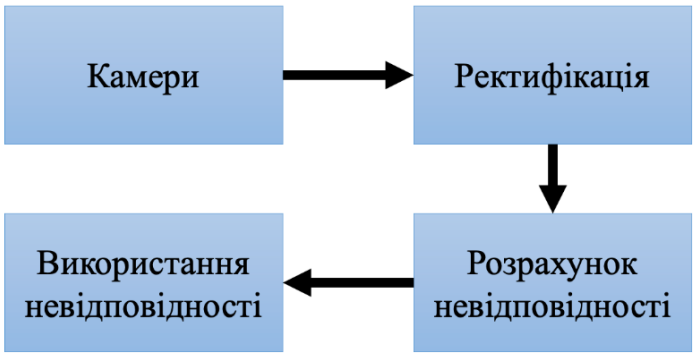


Рисунок 3.1 – Узагальнена схема програмно-апаратного засобу побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA. Типова система стереозору.

3.1 Напівглобальне зіставлення блоків

Метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA використовує алгоритм SGBM.

Він, в свою чергу, забезпечує хорошу точність при керованих обчислювальних зусиллях і надійний щодо вибору параметрів конфігурації.

Ключовою відмінністю від простих підходів на основі блоків, які зазвичай обчислюють збіги, прагнучи до мінімальних сум різниць між інтенсивністю пікселів у базовому та відповідних зображеннях, є використання більш складної функції вартості. Ця функція вартості не тільки має розширений діапазон (враховуючи не лише окремі пікселі чи локальні околиці, але й пікселі вздовж

шляхів по всьому зображенню), але також розглядає характеристики вищого рівня (наприклад, взаємну інформацію та перепис), а не інтенсивність пікселів.

Реалізація алгоритму SGBM використовує взаємну інформацію у функції вартості.

Перевага використання цієї характеристики, яку ввели Віола та Уеллс, полягає в її якості навіть у випадку невиправлених даних зображення. Однак він погано масштабується для зображень з більшою глибиною (більші координати S, особливо за глибинами, представленими як 8-бітні значення).

Подальші дослідники визначили перепис як надійний відповідний розрахунок вартості. Він ґрунтується на кодуванні відносин інтенсивності (наприклад, «темніший за центральний піксель») між пікселями в сусідстві в біт-векторах, а потім визначає вартість потенційного збігу на основі відстані Хеммінга між парою цих біт-векторів.

Використовується різновид цієї ідеї, яка розглядає різницю між прямими кількостями пікселів, що задовольняють відношення, а не відстань Хеммінга як вартість. Цей підхід, який називається непараметричним ранговим перетворенням, має якість відповідності, подібну до Census, але його легше реалізувати для високої продуктивності обчислень.

Функція $\mathrm{C}(\boldsymbol{p}, \mathrm{d}) \in N_0$ використовується для позначення вартості відповідності пікселя $\boldsymbol{p} = (x, y)$ у координатах (x, y) у базовому зображенні за припущеної невідповідності (зміщення) d у координатах $(x - d, y)$ на відповідному зображенні.

Відмінності в кількості пікселів, темніших за центральний піксель (параметричне рангове перетворення), можна використовувати для реалізації C.

Щоб визначити фактичну найкращу відповідність, ці витрати розраховуються для всіх потенційних розбіжностей $d < D_{max}$, де D_{max} є верхньою межею потенційної невідповідності (через фізичну відстань встановлення двох камер) [59].

У першому наближенні передбачалось, що відповідність із найнижчою вартістю вказує на справжню невідповідність $arg\ min_{d<D_{max}}\ C(\boldsymbol{p}, d)$ між базовим і відповідним зображеннями для окремого пікселя $\boldsymbol{p}$.

Щоб досягти кращої точності відповідності, (напів)глобальні підходи, такі як SGBM, було обчислено ці витрати на потенційні відповідності не лише між окремими (або околицями) пікселів, але й уздовж багатопіксельних шляхів, що простягаються через усе зображення [60].

Вартість узгодження вздовж усього шляху, описана відносним зсувом елементів шляху $r\ =\ (\Delta x, \Delta y)$, для припущеної невідповідності d позначається як $L_r(\boldsymbol{p}, d)$.

Ці шляхи рівномірно розподіляються для загального перегляду збігів:

$$L\boldsymbol{r}'(\boldsymbol{p}, d) = C(\boldsymbol{p}, d) + min \begin{cases} L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d) \\ L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d-1) + P_1 \\ L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d+1) + P_1 \\ min_i\ L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, i) + P_2 \end{cases} \quad (3.1)$$

Як правило, використовується щонайменше вісім рівномірно розподілених шляхів (рис. 3.1.б), але для оптимального покриття пропонувалось 16 [61].

Кількість і розташування шляхів має прямий вплив не тільки на точність узгодження, але також на обчислювальні зусилля і, в цьому випадку, на фактичну архітектуру апаратного прискорювача SGBM.

Метою є компроміс між продуктивністю та точністю. Скорочення від восьми до чотирьох шляхів $0°\ (r\ =\ (1,0)),\ 45°\ (r\ =\ (1,1)), 90°\ (r\ =\ (0,1))$ і $135°\ (r\ =\ (-1,1))$ (рисунок 3.2.а) призвело до втрати точності лише на 1,7% (збільшення кількості неправильно позначених розбіжностей) у тесті Middlebury, але дозволяє високоефективну апаратну архітектуру обчислень L_r для всіх цих шляхів паралельно.

Якщо навіть обмежена втрата точності є неприйнятною, запропоновану апаратну архітектуру було використано для виконання другого проходу по зображенню, обчислюючи решту шляхів $180°\ (r = (-1,0)), 225°\ (r = (-1,-1)), 270°\ (r = (0,-1))$ і $315°\ (r = (1,-1))$, починаючи з протилежного кута. Це зменшило частоту кадрів для відповідності вдвічі. Оскільки шляхи у вибраному розташуванні більше не розподіляються рівномірно по зображенню, у деяких тестах можна було виміряти деякі незначні (неізотропні) ефекти згладжування, але вони не впливають на зручність використання в сценаріях реального світу [62].

Ці необроблені витрати на шлях розраховувались для всіх вибраних шляхів, для всіх потенційних розбіжностей d до обмеження D_{max}. Для кожного пікселя оцінювалась як локальна вартість C, так і напівглобальний компонент.

Остання враховувала чотири характеристики, які спостерігаються в реальних зображеннях, мінімальна з яких додається до локальної вартості: перша характеристика – це вартість попереднього пікселя на шляху, другий і третій компоненти штрафують за невелику розбіжність. Зміни $|\Delta d| = 1$ за P_1, тоді як останній член штрафує більші зміни невідповідності (так звані розриви) за P_2. P_1 зазвичай визначається в автономному режимі експериментально шляхом аналізу вхідних зображень, типових для реального випадку використання стереозору. P_2, з іншого боку, динамічно коригується під час виконання: оскільки невідповідність часто також представляє розриви, коли інтенсивність пікселя змінюється, обчислення $P_2 = \frac{P_2'}{|\mathrm{I}_p - \mathrm{I}_{p-r}|}'$ компенсує різні інтенсивності пікселів I_p та I_{p-r} уздовж шляху r.

Що стосується P_1, P_2' є константою, визначеною експериментально на основі репрезентативних зразків зображень офлайн. Для подальшого обговорення розрахунку вартості шляху було розглянуто оригінальну роботу Гіршмюллера. Щоб визначити (напів) глобальну вартість узгодження, вартість шляху підсумували

по всіх шляхах. Однак для апаратної реалізації варто було розглянути дещо змінене формулювання [63].

Вісім рівномірно розподілених шляхів подано на рисунку 3.2.

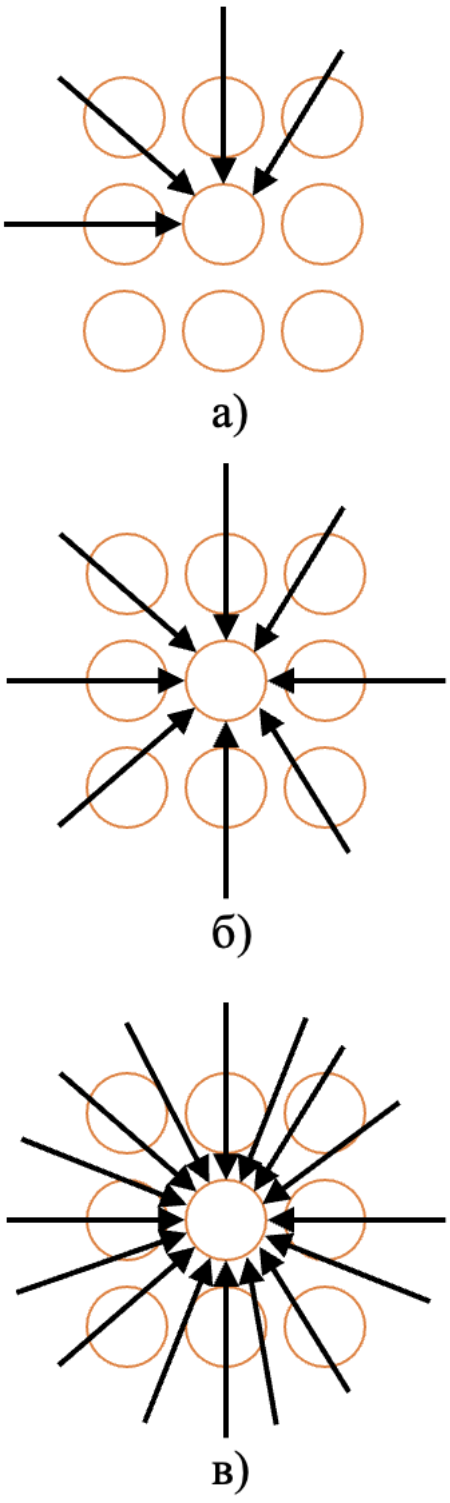


Рисунок 3.2 – Вісім рівномірно розподілених шляхів: а) чотири, б) вісім , в) 16 напрямків, що використовуються в напівглобальному зіставленні блоків

В апаратному забезпеченні ключовою характеристикою як з точки зору продуктивності, так і з точки зору використання області є ширина слова (у бітах) арифметичних операторів і типів даних.

Оскільки шляхи проходитимуть по всьому зображенню, вони можуть бути досить довгими (залежно від роздільної здатності камери), і підсумовування їх вартості могло призвести до великих значень, які потребують широких слів для обчислення та зберігання.

Цьому можна було протистояти, віднімаючи від необроблених витрат шляху для пікселя $L_{\boldsymbol{r}}{}'(\boldsymbol{p}, d)$ мінімальну вартість шляху для всіх припущених розбіжностей d для попереднього пікселя $p - r$ уздовж шляху $\boldsymbol{r}$.

Ефект кодування лише відмінностей між попередніми та поточними пікселями призвів до зменшення величини значень, які вимагають відповідно вужчих слів даних для зберігання та обчислення.

Таким чином, апаратно-оптимізоване обчислення вартості шляху стало:

$$L_r(\boldsymbol{p}, d) = C(\boldsymbol{p}, d) + min \begin{cases} L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d) \\ L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d-1) + P_1 \\ L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d+1) + P_1 \\ min_i L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, i) + P_2 \end{cases} - \min_j L_r(\boldsymbol{p} - \boldsymbol{r}, j) \qquad (3.2)$$

Вартість шляху L_r уздовж усіх шляхів r підсумувалось як $S(\boldsymbol{p}, d) = \sum_r L_r(\boldsymbol{p}, d)$.

Невідповідність d з мінімальною відповідністю $cost\ arg\ min_d S(\boldsymbol{p}, d)$ вважалась виграшною диспропорцією для пікселя $\boldsymbol{p}$. Ці виграшні відмінності вивелись прискорювачем для кожного пікселя як вхідні дані для подальшого обчислення фактичної глибини (положення осі Z, тут не обговорюється) [64].

Як зазначено вище, на практиці необхідні додаткові обмеження накладені для очищення викидів і позначення недійсних розбіжностей: результат аргументу $S(\boldsymbol{p}, d)$ може бути багатоелементним набором, що означає, що мінімальна вартість відповідності для пікселя $\boldsymbol{p}$ виникає для різних потенційних розбіжностей d. З

такою неунікальною вартістю алгоритм не можна було визначити одну виграшну невідповідність, а натомість зареєстровано невідповідність для цього пікселя як «недійсний».

Крім того, виконується так звана перевірка ліворуч/праворуч, яка порівнює результати алгоритму під час його виконання з поміняними ролями базового та відповідного зображень.

Цю перевірку також можна було ефективно здійснити (уникаючи перерахунку всіх розбіжностей для попереднього зображення збігу, яке зараз використовується як основа), повторно використовуючи попередньо обчислене $S(\boldsymbol{p}, d)$ уздовж епіполярної лінії як аргумент $S((x(\boldsymbol{p}) + d, y(\boldsymbol{p})), d)$, щоб вибрати виграшну невідповідність d для другого зображення.

Перевірка ліворуч/праворуч встановлює для невідповідності значення «недійсне», якщо відповідні невідповідності вихідного проходу та пропуску зі зміненими ролями відрізняються більш ніж на одиницю. Цей крок усуває фантомні диспропорції, що є результатом закритих поверхонь, які видно на одному зображенні, але приховані на іншому [65].

Нарешті, карта невідповідності обробляється з використанням основного медіанного фільтра 3 × 3 для придушення викидів [66].

3.2 Архітектура

Запропонований метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA уможливлює здійснити синтез архітектуру програмно-технічного засобу, яка підвищує швидкодію систем комп’ютерного зору за рахунок покращення не лише введення додаткового рівня детального паралелізму (наприклад, паралельне обчислення диспропорції та сортування), а й також завдяки реалізації з використанням найсучаснішого стилю розробки, нечутливого до затримок, у наступному – генерація мови опису обладнання. Як результат, він є значно більш масштабованим, легшим для розширення, але також

набагато швидшим, ніж оригінальна робота (навіть якщо компенсувати відмінності в цільових технологіях FPGA).

Базова архітектура вже може виконувати повне обчислення SGBM і, з деякою обережністю в реалізації, може бути повністю конвеєрною. Однак він паралельно виконує лише обчислення вартості шляху L_r. Це можна покращити як на дрібнозернистому, так і на крупнозернистому рівнях [67].

1) Дрібнозернисте розпаралелювання. Одним із ключових вкладів цієї роботи є обчислення витрат на піксель і на шлях для n потенційних значень невідповідності паралельно;

2) Грубе розпаралелювання. Базова архітектура також може бути розпаралелена на грубому рівні. Це вимагає m кількох екземплярів усієї базової архітектури (у цьому контексті називається процесором рядків). Кожен процесор рядків відповідає за обробку одного рядка вхідних зображень, що призводить до паралельної обробки смуги зображення висотою m рядків.

Продуктивність цього підходу залежить від того, наскільки швидко міжрядкові буфери можуть бути заповнені даними, оскільки процесори рядків, які очікують даних від своїх попередників, залишатимуться бездіяльними.

Подібним чином, процесор рядків зупиняється, якщо він не може передати свій вихід через те, що відповідний буфер до наступного рядка заповнений. Розглядаючи даний підхід як масив хвильового фронту (систолічний масив із квиткуванням замість покрокового розповсюдження даних), він ідеально піддається реалізації в новому застосованому апаратному стилі, нечутливому до затримки [68].

Обидва методи розпаралелювання можна комбінувати, щоб пом’якшити їхні відповідні недоліки (критична довжина шляху в дрібнозернистому підході, зупинка рядкових процесорів у грубозернистому).

Вибрані архітектурні деталі. Рангове перетворення складається з підрахунку кількості пікселів у околиці, які мають нижчу інтенсивність, ніж центральний піксель:

$$R(\boldsymbol{p}) = ||\boldsymbol{p}' \in N_S(\boldsymbol{p})|I(\boldsymbol{p}') < I(\boldsymbol{p})||, \quad (3.3)$$

Тут $R(\boldsymbol{p})$ є ранговим перетворенням пікселя $\boldsymbol{p}$, $N_S(\boldsymbol{p})$ околиці навколо пікселя $\boldsymbol{p}$, що охоплює всі пікселі з відстанню, меншою або рівною $(s-1)/2$ від центру, а $I(\boldsymbol{p})$ інтенсивність пікселя $\boldsymbol{p}$. Околиці, які також називають ядром, є квадратними із загальним розміром s^2 пікселів. Кожне перетворене значення рангу кодується в $[log_2(s^2)]$біт.

Розрахунок $R(\boldsymbol{p})$, який є вхідними даними для етапу розрахунку вартості, виконується паралельно для обох зображень. Процесори рядків подаються потоками $R_b(\boldsymbol{p})$ для базового зображення та $R_m(\boldsymbol{p})$ відповідного зображення, а також потоком для штрафних значень P_2. Якщо в системі присутній більше одного рядкового процесора, для розподілу вхідних даних по одному рядку за раз використовується циклічний процес (що призводить до обробки у смузі, що рухається вниз). Усі процесори рядків вимагають вхідних FIFO, які можуть зберігати повний рядок, щоб працювати як масив хвильового фронту.

Таким чином, обчислення фактичної вартості відповідності на піксель на основі рангового перетворення R є

$$C(\boldsymbol{p}, d) = |\left|R(\boldsymbol{p}) - R\left(\left(x(\boldsymbol{p}) - d, y(\boldsymbol{p})\right)\right)\right|| \quad (3.4)$$

Обгортковий буфер від останнього до першого процесора рядків більший, ніж звичайні буфери процесора між рядками, оскільки він має зберігати всі проміжні результати D_{max} для кожного пікселя в останньому рядку.

У конструкції з декількома рядковими процесорами обчислені диспропорції буферизуються в FIFO, доки їх не витягне модуль виводу. Потім модуль виводу об’єднує виходи процесорів рядків, використовуючи ту саму циклічну схему, що й модуль введення. Потім медіанний фільтр 3 × 3 застосовується до об’єднаного потоку для видалення викидів у обчислених диспропорціях.

Максимальне значення будь-якого L завжди менше $C_{max} + P_2$. Це обмежує ширину слова, необхідну для зберігання даних і арифметичних операторів в апаратній реалізації.

Інтеграція прискорювача в TPC потребує певних інтерфейсів для забезпечення зв'язку із зовнішнім світом. Інтерфейси використовуються для читання обох вхідних зображень паралельно.

Один з інтерфейсів додатково використовується для запису карти невідповідностей.

Інший інтерфейс використовується для взаємодії хоста, який контролює виконання, з прискорювачем. Такі параметри, як адреси вхідних і вихідних зображень, а також їхній розмір можна встановити перед виконанням. Нарешті, сигнал переривання сигналізує хосту про завершення виконання.

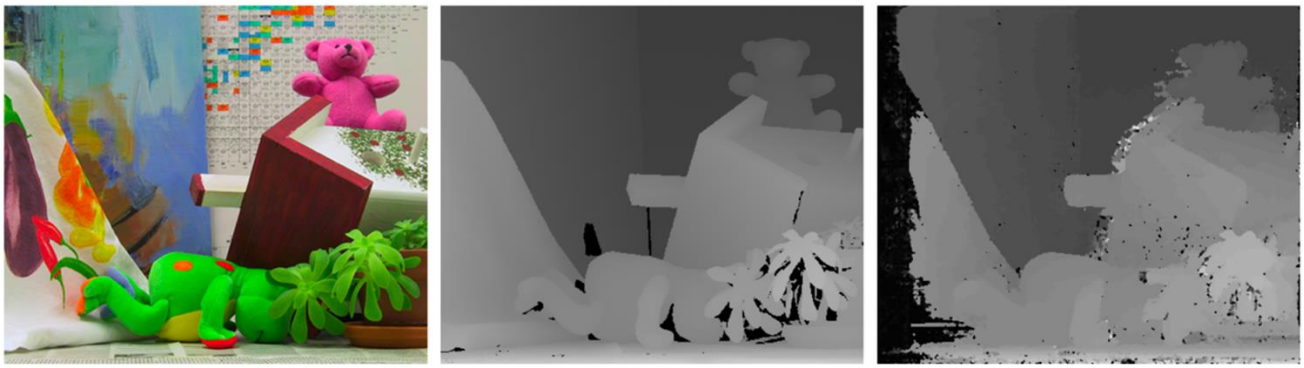
3.3 Оцінка підходу

Цей підхід оцінюється на трьох рівнях: першим критерієм є точність, потім модельована незалежна від цілі продуктивність апаратної архітектури в термінах тактових циклів і, нарешті, продуктивність настінного годинника на трьох фактичних платформах FPGA, що охоплюють вбудовану систему та центр обробки даних [43].

3.3.1 Точність

Використовуючи тест Middlebury, алгоритм створює в середньому 8,4 % розбіжностей, що перевищує поріг помилки в один піксель.

Різниця між результатами запропонованої архітектури та основною правдою в незакритих областях становить 9,5 % для Конусів, 13,3 % для Тедді, 6,8 % для Цукуби та 4,1 % для Венери. Зразок результату для тестового набору Тедді показано на рисунку 3.3.



а) б) в)

Рисунок 3.3 – Порівняння невідповідності для набору зображень Тедді:
а) Набір зображень Тедді зліва, б) Набір зображень Тедді, в) невідповідність, спричинена зображенням, що пояснює істини запропонованої архітектури

За потреби більшої точності можна досягти, виконавши повну перевірку ліворуч/праворуч (повне обчислення замість повторного використання епіполярних прогнозованих значень) або виконавши два проходи по зображенню для використання восьми шляхів. Обидва ці підходи приблизно вдвічі зменшують продуктивність прискорювача, але навіть тоді він буде корисним для обробки в реальному часі зі швидкістю 30 кадрів в секунду або вище [69].

3.3.2 Продуктивність, яка не залежить від платформи

Моделювання архітектури з точністю до циклу використовувалося для визначення характеристик часу виконання прикладів реалізацій.

Порівняння з фактичними реалізаціями апаратного забезпечення показує, що ці симуляції насправді є репрезентативними для кінцевої продуктивності.

Ядро оцінюється за трьома роздільними здатностями зображення: 640 × 480 пікселів (VGA), 1280 × 720 пікселів (720p) і 1920 × 1080 пікселів (1080p). Зображення з роздільною здатністю VGA оцінюються за Dmax = 64, для вищої роздільної здатності Dmax = 128.

Для всіх роздільних здатностей кількість тактів, необхідних для завершення одного кадру, визначається шляхом моделювання.

Експертиза містить різні композиції крупно- та дрібнозернистого паралелізму. Реалізація описується парою (**#p**, **#d**), яка вказує на використання рядкових процесорів **#p**, причому кожне обчислення **#d** передбачає невідповідності паралельно.

Для кожної з роздільних здатностей використовується автоматичне дослідження простору розробки для генерації 250 альтернатив реалізації, відображених на осі X у порядку збільшення площі або продуктивності.

Через обмеження простору лише підмножину альтернатив можна позначити тут за допомогою (**#p**, **#d**).

Як показано на рисунку 3.4 для зображень VGA, архітектура добре масштабується зі збільшенням кількості процесорів рядків, аж до нижньої межі 654644 циклів, після чого один піксель обчислюється за 2,11 циклу, а одна диспропорція вимагає 0,033 циклу.

Ця конструкція обмежена швидкістю заповнення вхідних буферів, яку можна збільшити ще більше, застосовуючи також методи дрібного розпаралелювання [44].

При передбачуваній тактовій частоті 200 МГц архітектура досягне до 306 кадрів в секунду, як показано на рисунку 3.5.

Такі великі та швидкі системи можна використовувати для розрахунку розбіжностей між кількома камерами для створення об’ємного огляду сцени [70].

Для додатків з низьким енергоспоживанням більш цікавим є мінімальна частота, необхідна для досягнення 30 кадрів на секунду (типова вимога для обробки в реальному часі).

Як показано на рисунку 3.6, архітектура здатна задовольнити цю вимогу на частоті до 30 МГц для зображень VGA [71, 72].

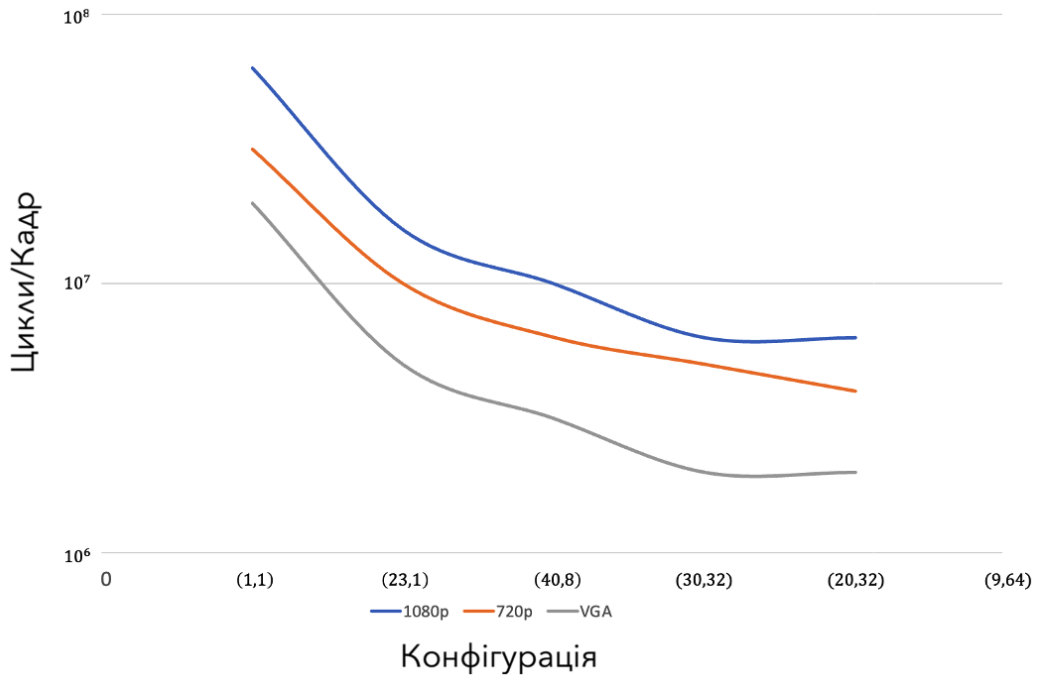


Рисунок 3.4 – Цикли, необхідні для обробки однієї карти невідповідності для різних ступенів паралельності

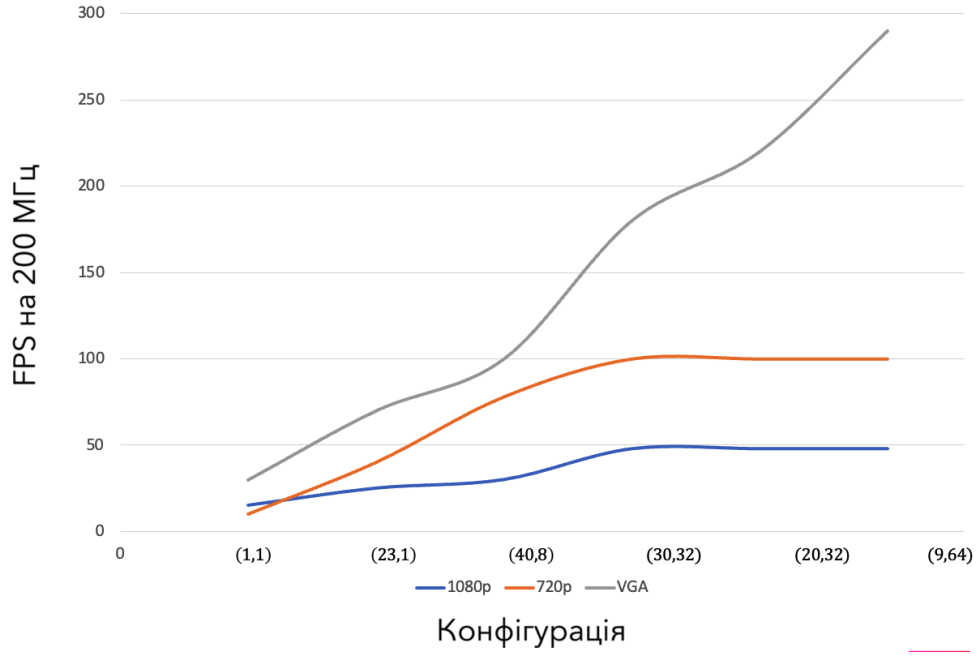


Рисунок 3.5 – Кількість кадрів в секунду досягається запропонованою архітектурою на тактовій частоті 200 МГц

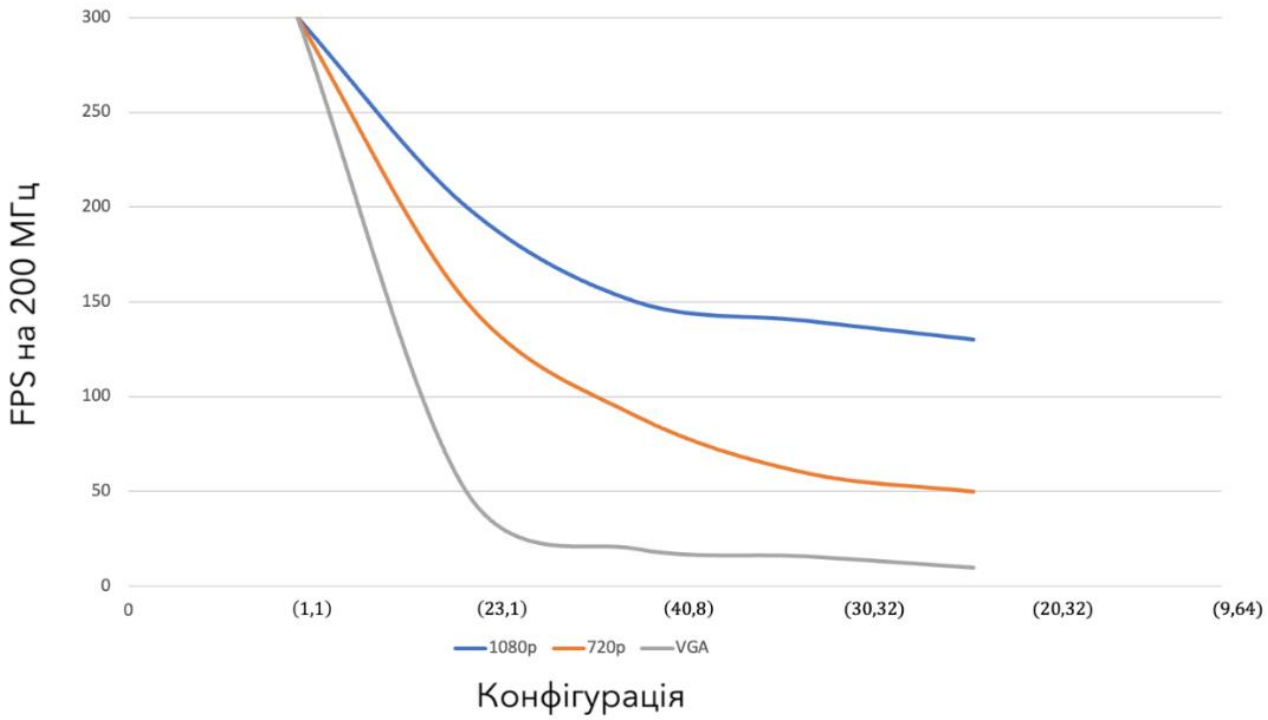


Рисунок 3.6 – Частота, необхідна для досягнення 30 кадрів на секунду на запропонованій архітектурі для різного ступеня паралелізму

3.4 Висновок

В розділі представлено метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA.

В осонві методу лежить застосування алгоритму Semi-Global Block Matching (SGBM), який забезпечує швидке виконання операцій та показує високу точність у тестах стереозору, таких як KITTI і Middlebury.

Основою запропонованого методу є синтез нової апаратну архітектуру програмно-технічного засобу для комп’ютерного зору, що застосовує FPGA, а також використовує алгоритм SGBM.

Метод забезпечує синтез побудови параметризованої архітектури для систем комп’ютерного зору на основі FPGA архітектура, яка має високу масштабованість, що дозволяє легко впроваджувати її на малих малопотужних пристроях (наприклад, в автономних роботах), а також на великих високопродуктивних чіпах

(наприклад, у стаціонарних випадках використання для обробки кількох відеопотоків високої роздільної здатності).

Запропонована архітектура для обчислення на FPGA добре працює в широкому діапазоні сценаріїв. Конфігурації VGA з низьким енергоспоживанням працюють зі швидкістю 30 кадрів в секунду з тактовою частотою до 74 МГц навіть на невеликих FPGA, таких як ті, які використовуються на Altera Cyclone V SoC 5CSXFC6D6F31C6N FPGA.

Для підвищення продуктивності архітектура пропонує кілька рівнів паралелізму та може бути налаштована TPC у автоматичному дослідженні простору проектування для виявлення оптимальних конфігурацій.

Введення дрібнозернистого паралелізму дозволяє набагато краще адаптувати прискорювачі до потреб індивідуального сценарію використання, оскільки просте збільшення кількості рядкових процесорів не завжди призводить до в найбільш ефективній реалізації.

4 ПРОГРАМНО-АПАРАТНИЙ ЗАСІБ ДЛЯ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA

4.1 Архітектура FPGA

FPGA є членами сімейства програмованих логічних пристроїв, їх можна розглядати як масив, що складається з конфігурованого логічного блоку (CLB). Ці CLB з'єднані програмованою мережею, разом вони керуються коміркою пам'яті, завдяки чому FPGA зручно задовольняють різні специфічні потреби.

До теперішнього часу структура FPGA базується на таких техніках: Flash, EPROM, SRAM і anti-fuse.

Найбільш широко використовувані FPGA базуються на SRAM і основними постачальниками яких є ALTERA і XILINX. Основні FPGA виготовляються за 65 нм, найвища тактова частота досягла 500 МГц, тим часом такі змінні FPGA, як частота та логічний шлюз, все ще швидко розвиваються.

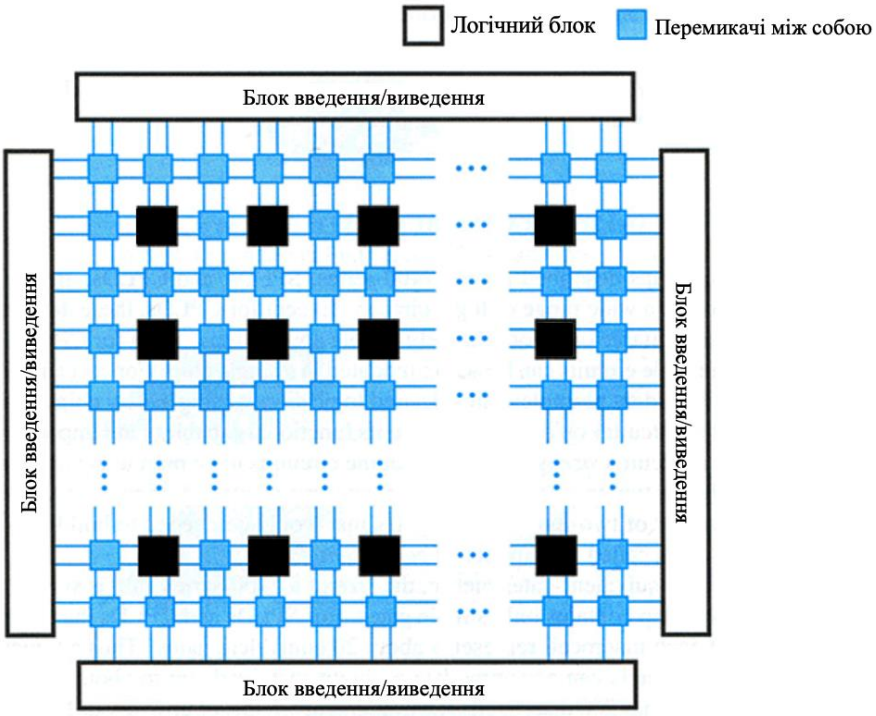
FPGA — це масив, що складається з CLB, найбільший CLB наразі має 192 рядки та 116 стовпців, програмовані входи/виходи формують поза кадром, іноді FPGA високого рівня може мати більше тисячі входів/виводів. Разом із взаємозв'язками вони є внутрішньою структурою FPGA, на рис. 4.1 показано CLB і приклад логіки користувача [73].

На схемі активовано чотири входи/виходи, а саме x1, x2, x3 і f, три логічні блоки, де чітко позначені входи та виходи. Сині лінії та хрестики означають з'єднання, а чорні – роз'єднання.

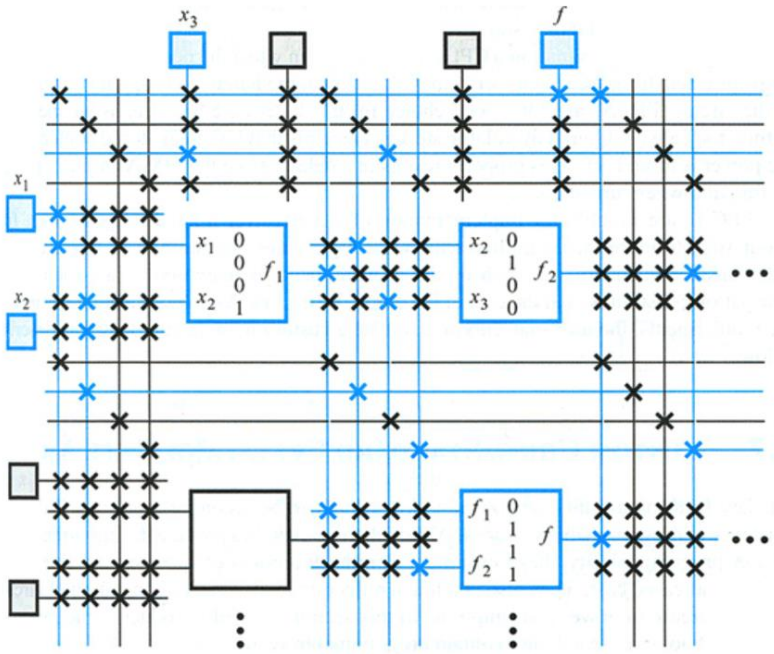
Взявши, наприклад, перший блок, x1 і x2 з'єднані з блоком за допомогою різних ліній, а вихід f1 переноситься на інший блок як вхід.

Щоб підвищити потужність FPGA, крім традиційних ресурсів у FPGA інтегровано деякі інші спеціальні модулі. Такі як RAM, DSP, вбудований жорсткий процесор (PowerPC або ARM) і програмний процесор (Nios або Microblaze). Це не означає замінити структуру традиційних FGPA, а доповнити можливості FPGA і

надати більше функцій і опцій, які можна застосувати до програм, які стають дедалі складнішими.



Загальна будова FPGA



Розділ програмованої FPGA

Рисунок 4.1 – Внутрішня структура FPGA

Кожен CLB може мати 2 або 4 або навіть більше логічних елементів (LE), що є базовим блоком FPGA, рис. 4.2 демонструє структуру LE.

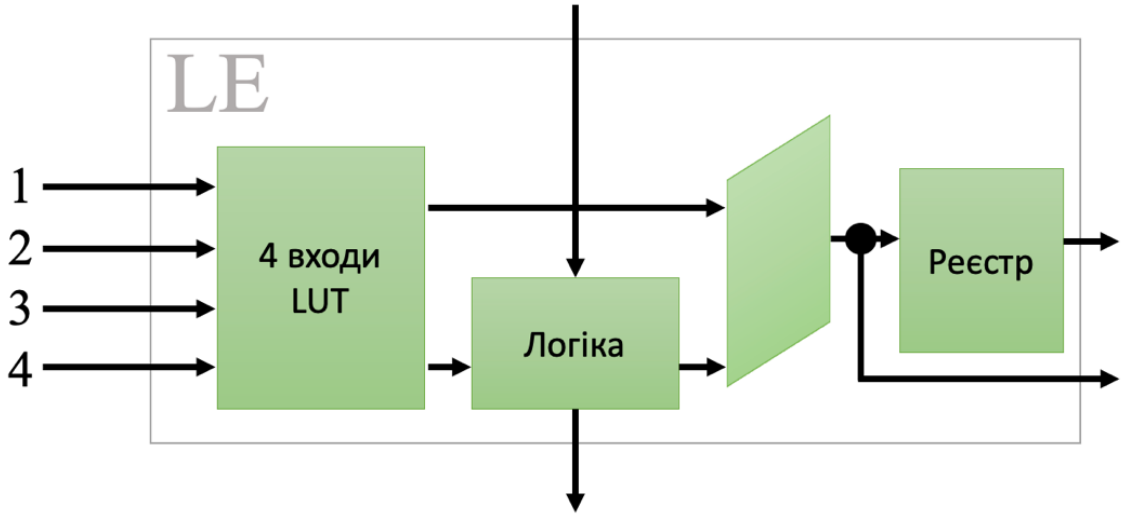


Рисунок 4.2 – Блок-схема LE

LE містить 4-розрядну таблицю пошуку (LUT), вона може бути виділена як 16-розрядна RAM або ROM або як комбінована логіка, яка виводить комбінаторний вихід [74].

Існує шлях перенесення, щоб LE міг виконувати певні відносні обчислення. Останньою частиною IE є D-тригер із сигналами ввімкнення, скидання та синхронізації, цей регістр може зберігати вихід LE. Тому що вихід LE міг бути встановлений як вхід іншого LE, так що LE також можна вважати мікростатичним автоматом [75].

Стандартна розробка програмування FPGA це:

1) логічне проектування;
2) моделювання;
3) компонування;
4) синтез;
5) налаштування;
6) налагодження.

Три з шести описаних вище процедур виконуються за допомогою інструментів, наданих виробниками, а саме симуляції, синтезу та налагодження. потужність інструменту синтезу безпосередньо визначає потужність усієї конструкції.

Отже, щоб отримати кращий результат синтезу, необхідно повністю враховувати спеціалізацію інструменту та розробити проект відповідно до структури мікросхеми, яка використовується.

На самому початку історії FPGA розробки були досить простими, тому логічні конструкції зазвичай виконуються за допомогою інструменту розробки схем.

Наприкінці 80-х років складність і масштаб дизайну зростають, так що проекти, які базуються на схемах, почали усуватись. завдяки розвитку технології EDA дизайн HDL стає все більш популярним і тепер також застосовується до FPGA [76].

Стандарт IEEE робить інструменти FPGA на основі HDL широко використовуваними в галузі мікроелектроніки, програмісти можуть вільно розробляти власні схеми за багатошаровістю або модульністю. Такі методи програмування є «зверху вниз» і їх можна класифікувати наступним чином:

1) Опис рівня системи, що описує специфікації дизайну всієї системи, який відображає взаємозв'язок між різними компонентами системи та їх функціональність;

2) Опис рівня поведінки, що описує математичну модель системи, тобто докладний розгляд різних аспектів поведінки системи, які можуть бути виражені у формі математичних функцій, рівнянь або алгоритмів;

3) Опис рівня RTL з використанням основних одиниць системи для характеристики математичної моделі. Опис рівня RTL може включати в себе детальний опис логіки, відповідальної за роботу системи, та її складові;

4) Фізичний рівень, що використовує прості схеми шлюзів для представлення системи, тобто це найнижчий рівень в архітектурі обчислювальної

системи, де електричні сигнали обробляються фізичними компонентами, такими як транзистори, резистори, конденсатори, тощо [77].

Загалом, при проектуванні системи зазвичай працюють над поведінкою і RTL-шарами, можна вибрати певний шар для роботи відповідно до складності алгоритму або знайомства з інструментом синтезу.

Важливим моментом використання HDL для опису схеми є повне розуміння зв'язку між мовою розробки апаратного забезпечення та апаратною схемою. Потрібно мати базову концепцію схеми, яку потрібно синтезувати під час написання коду.

4.2 Проектування логічних блоків

Проектування апаратного забезпечення чимось схоже на програмування програмного забезпечення на мові асемблера, у якому простий розрахунок складається з безлічі інструкцій машинного коду, а з іншого боку, багато логічних блоків утворюють єдиний апаратний модуль [78].

Рисунок 4.3 показує основну концепцію того, як працюють блоки FPGA. Нижче будуть розглянуті всі логічні блоки, які вводяться для реконфігурації часткового алгоритму.

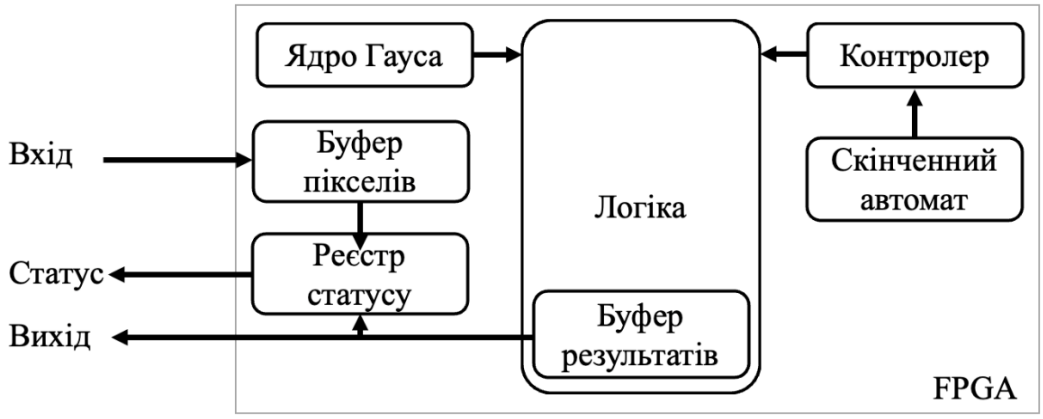


Рисунок 4.3 – Блок-схема реалізації FPGA

4.2.1 Ядро Гауса

На рисунку 4.4 показано ядро Гауса, яке використовується для отримання матриці Гарріса A. Цей блок є найпростішим, він не вимагає введення та виведення постійних змінних ядра.

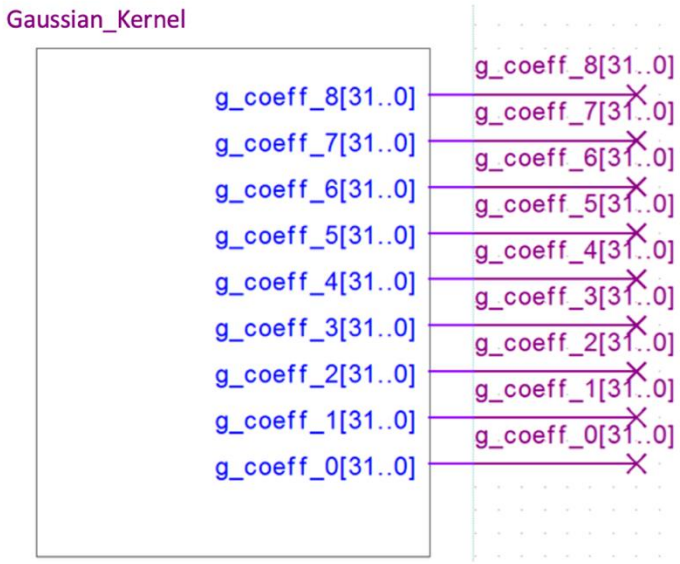


Рисунок 4.4 – Ядро Гауса

Кожен результат відповідає одному елементу матриці, коефіцієнт 8 представляє перший елемент 0,0001, коефіцієнт 5 є центральним елементом 0,3989 і так далі. Варто зауважити, що кожен параметр є числом з плаваючою точкою, і він отримує 32 біти пам’яті для збереження, тому всі виходи мають 32 біти [79].

Таблиця 4.1 – Матриця Харріса

0.0001	0.0044	0.0540
0.2420	0.3989	0.2420
0.0540	0.0044	0.0001

4.2.2 Вхід FIFO

Щоб максимізувати швидкість, було реалізоване апаратне забезпечення якомога паралельнішим. Всі IP-адреси FIFO, надані Altera, є механізмом «один вхід і один вихід», що не зовсім те, чого потрібно досягти, тому FIFO «один вхід і дев'ять виходів» реалізовано, щоб задовольнити потребу, щоб можна було запускати за один прохід цілу згортку з матрицею Гауса 3*3.

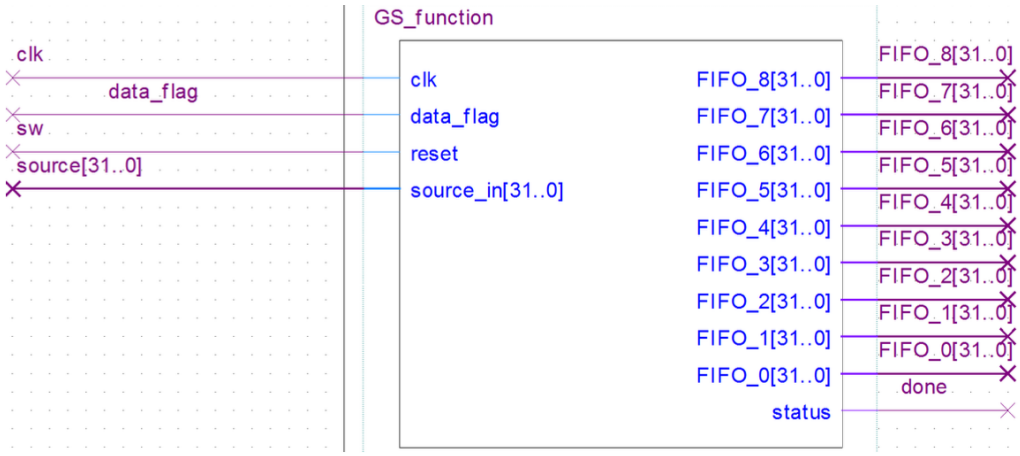


Рисунок 4.5 – вхід FIFO

Таблиця 4.2 – Сигнали вхідного FIFO

Назва сигналу	Опис
clk	Сигнал годинника
data flag	Перевертається, коли вводяться нові дані
sw	Перемикач, який керує функцією скидання
source[31..0]	Вхідні пікселі, які є 32-розрядним числом типу float
FIFO x[31..0]	Вихідний піксель No.x у парі з відповідним елементом матриці Харріса на подальших етапах
done	Встановлюється на високий рівень, коли FIFO заповнений, інакше залишається низьким

На рисунку 4.6 показано, як працює цей логічний блок.

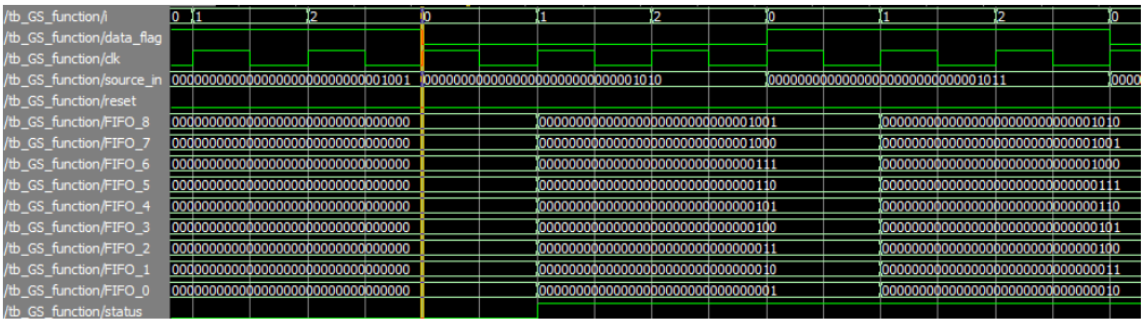


Рисунок 4.6 – Моделювання функціонального блоку GS

Спочатку всі 9 виходів встановлювались на нуль, а дані надсилались через один до дев’яти. Коли надсилаються 10-ті дані, усі вихідні дані отримують змінні в певному порядку, найстаріша розташована внизу, а найновіша – зверху. Для цього FIFO № 8 є верхнім, а № 0 – нижнім.

Коли 1-е дані (1010) отримані після трьох тактів, то в наступному такті найстаріші дані (0001) відкидались, а 10-ті дані додавались до FIFO та зберігались в регістрі № 8, і так далі.

Варто зауважити, що кожного разу, коли надходили нові дані, сигнал прапора даних перевертався, цей механізм гарантував, що система не переплутає два послідовних даних, які мають абсолютно однакове значення.

Крім того, завдяки збереженню даних у логіці вирішувалась проблема марнування ресурсів. Кожен окремий піксель використовувався багато разів у згортинах пікселів поруч, тому в програмному забезпеченні змінна має бути викликана кілька разів, але в апаратному забезпеченні вона буде викликана та надіслана до FIFO лише один раз, це триває, доки не завершяться всі пов’язані з нею роботи [80].

4.2.3 Додавання та множення

Згортка включає як FIFO, так і ядро Гауса, кожен вихід цих двох блоків було об’єднано в дев’ять різних груп. Зрештою всі вони були підсумовані в остаточний результат.

Оскільки алгоритм методу побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA використовує 32-розрядне число типу float для представлення пікселів, звичайні блоки обчислення не могли бути застосовані тут через складність значення float. Altera надає мегафункцію, яка повністю підтримує обчислення плаваючого значення, що робить її ідеальним вибором під час вибору методу побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA.

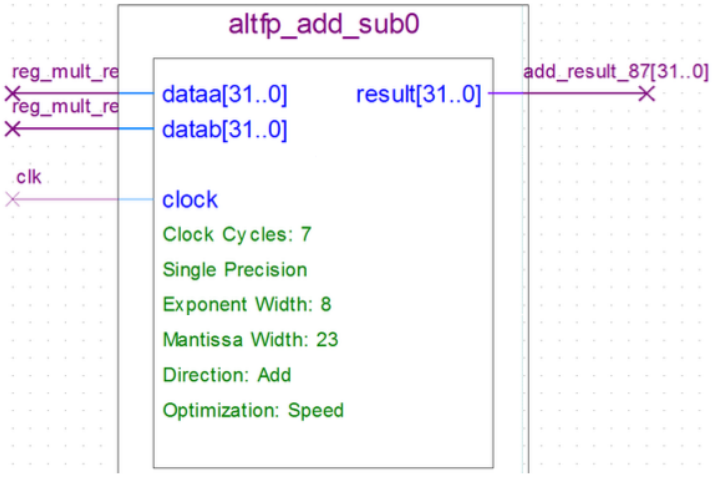


Рисунок 4.7 – Суматор

На рисунку 4.7 показано блок додавання після налаштування майстра мегафункцій. Оптимізацію швидкості вибрано, тому що, що є головним золотом дисертації, будь-які функції та можливості, крім чистого додавання, вимкнено з метою економії ресурсів (рисунок 4.8).

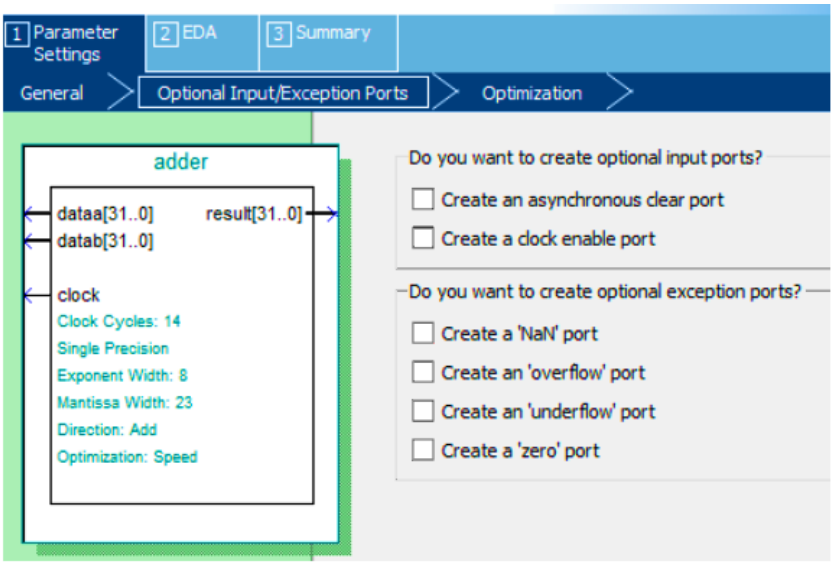


Рисунок 4.8 – Додаткові порти суматора

Такий блок потрібно дублювати багато разів, щоб сформувати функціональний алгоритм.

32-розрядне число з плаваючою точкою означає одинарну точність і має 8-розрядний експонент і також 23-розрядну мантису. Призначення портів досить зрозуміле, і глобальний сигнал clk буде підключений до суматора.

На рисунку 4.9 показано моделювання мега-функції суматора, яка отримує результат відразу після 6 тактів, оскільки джерела вводяться в такті «0», загальний період обробки точно такий же, як текст, вказаний на блоці, що становить 7 тактів.

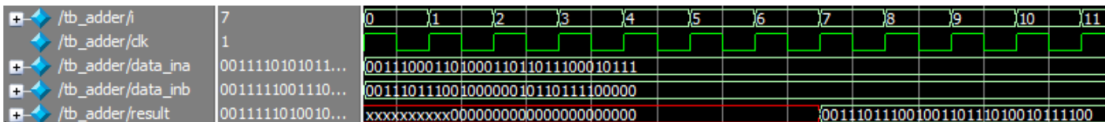


Рисунок 4.9 – Моделювання суматора

Модуль множення має таку саму процедуру налаштування та портів, за винятком того, що він виконує функцію множення, а для обчислення результату двох входів потрібно лише 5 тактів. Під час використання інструменту моделювання для симуляції таких заданих логічних блоків необхідно імпортувати

певну бібліотеку. Для двох вищезазначених блоків бібліотека «lpm ver» має бути включена до команди виконання, інакше виникнуть несподівані помилки.

4.2.4 Контейнер

Для однієї згортки двох матриць 3*3 реалізовано дев'ять множень і вісім суматорів. Кожен із блоків має період виконання, і останній обчислення може виконуватися лише тоді, коли перший виконає свою роботу. Для надійності всього цього процесу введено кінцевий контейнер.

На рисунку 4.10 представлена схема кінцевого скінченного автомат. Якщо отриманий сигнал є «високим», тоді спрацьовує арбітр, він починає рахувати всередині та виводить значення на роздільник, розділювач використовує це значення, щоб визначити, яким є поточний стан, і встановити відповідний вихід.

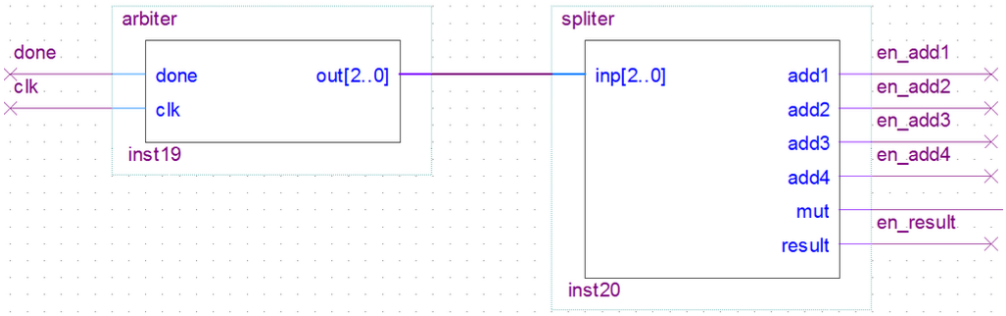


Рисунок 4.10 – Скінченний автомат

На рисунку 4.11 показано симуляцію арбітра, коли для сигналу done встановлено значення «високий», арбітр починає рахувати. Вихід арбітра – це лише числа в послідовності, кожне число представляє стан у розділювачі, а розділювач повідомляє, який регістр активний.

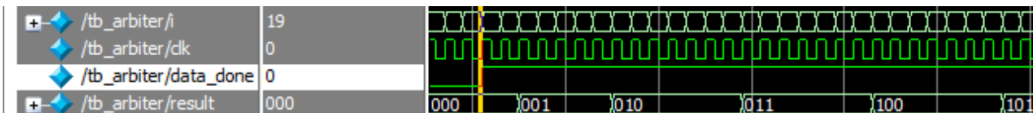


Рисунок 4.11 – Симуляція арбітра

4.2.4.1 Регістр суматора

Є чотири регістри додавання та регістр результату, які відіграють роль посередника між різними етапами згортки. Взявши, наприклад, регістр add1, входи підключені до попереднього етапу обчислення, незалежно від того, які значення є, регістр завжди зберігатиме те, що він отримує, коли отримано сигнал дозволу «позитивний фронт», вихід порти негайно виведуть дані, які наразі зберігаються, і залишатимуться виводом до наступного «позитивного фронту».

Останній етап обчислення, яким буде add2, використовує ці виходи як джерела вхідних даних і передає результат у регістр add2, а процес продовжує виконувати ту саму роботу для add3 і add4.

4.2.4.2 Реєстр результатів

Рисунок 4.12 – це регістр результатів, який не тільки зберігає та надсилає дані, але також має інші функції для підтримки надійності системи. Рисунок 4.13 відображає реєстр результатів.

Таблиця 4.3 описує функції додаткових портів регістра результату порівняно з регістрами суматора.

Причина, по якій реалізовано два вищезазначені сигнали, полягає в тому, що, на відміну від обчислювальної частини, між етапами немає перерв або дублікатів, попередній результат ніколи не зміниться, якби останній його не взяв.

Регістр результату взаємодіє з центральним процесором, який, в свою чергу, є повністю відокремленою системою.

Центральний процесор міг запитувати результат кілька разів, навіть якщо було обчислено лише один результат.

Через різницю в частоті між процесорами та FPGA, процесори завжди на рівні ГГц, а FPGA – у МГц, що досить повільно порівняно з процесорами.

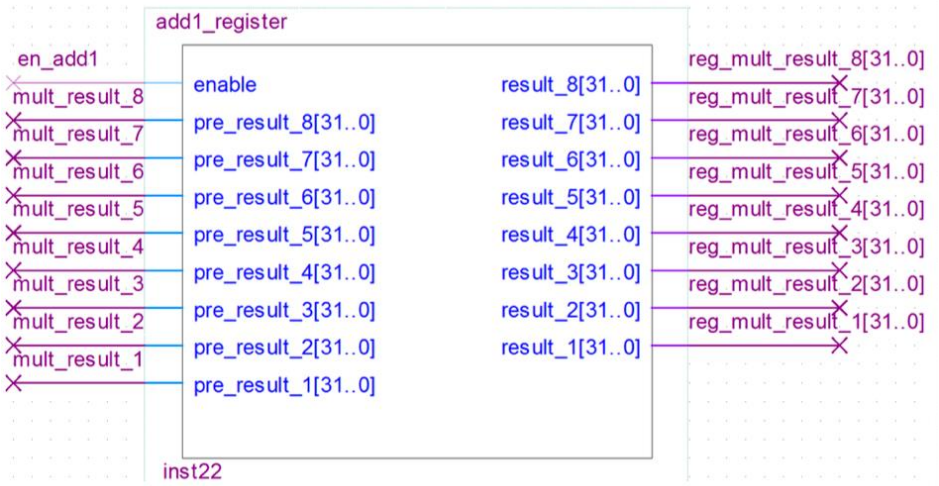
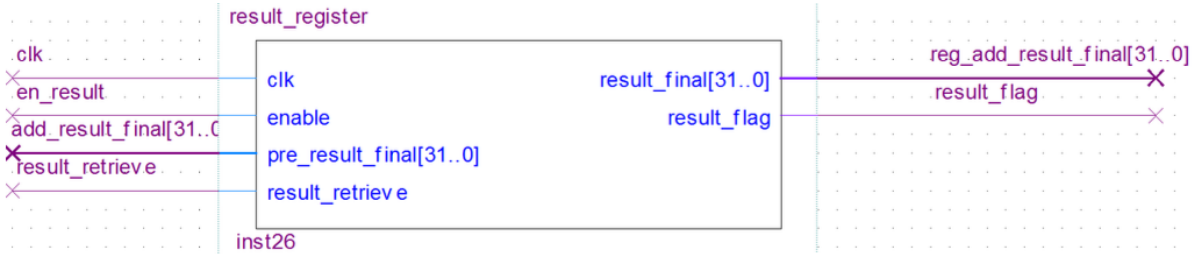


Рисунок 4.12 – Регістр суматора



Таблиця 4.13 – Реєстр результатів

Таблиця 4.3 – Сигнал регістру результату

Назва сигналу	Опис
result retrieve	Від центрального процесора надходить сигнал, який встановлюється на «високий» під час отримання результату та скидається на «низький» після завершення отримання.
result flag	Встановлення на «високий», коли сигнал увімкнення «позитивний фронт», вказує на те, що новий результат придатний для отримання. Встановіть значення «низький», коли сигнал отримання результату є «негативним фронтом».

Отже, має бути спосіб захисту кожного запиту, що стосується нових даних.

Сигнал прапора повідомляє центральному процесору про те, чи є поточні дані придатними та безпечними для отримання, і це завжди перший сигнал, який буде взаємодіяти в процесі зв'язку між центральним процесором і FPGA, коли процесор отримав «високий» сигнал.

Сигнал прапора почне отримувати дані, тим часом результат отримання буде надіслано до FPGA, що вказує на початок процесу. Після отримання результату ЦП знову встановлює «низький» сигнал отримання результату, повідомляє FPGA, що процес завершено.

Потім FPGA також встановлює сигнал прапора результату на «низький», щоб, дозволивши ЦП отримати сигнал прапора першим, він міг змусити ЦП чекати, поки не буде отримано новий результат.

Такий механізм може значно підвищити надійність системи.

4.2.5 Система PCIe

Окрім часткової реконфігурації, FPGA має спілкуватися із «зовнішнім світом», яким у даному випадку є центральний процесор.

На рис. 4.14 показано всі взаємозв'язки між різними модулями.

Єдиним можливим зв'язком є шина PCIe між процесором Atom і мікросхемою FPGA.

Використовуючи інструмент розробки Qsys, можна легко створити систему PCIe.

На рис. 4.15 показано структуру проектування Qsys, апаратні контролери з'єднані один з одним через інтерфейс Avalon Memory-Mapped, тому головні контролери можуть легко отримати доступ до підлеглого контролера за допомогою відображення пам'яті.

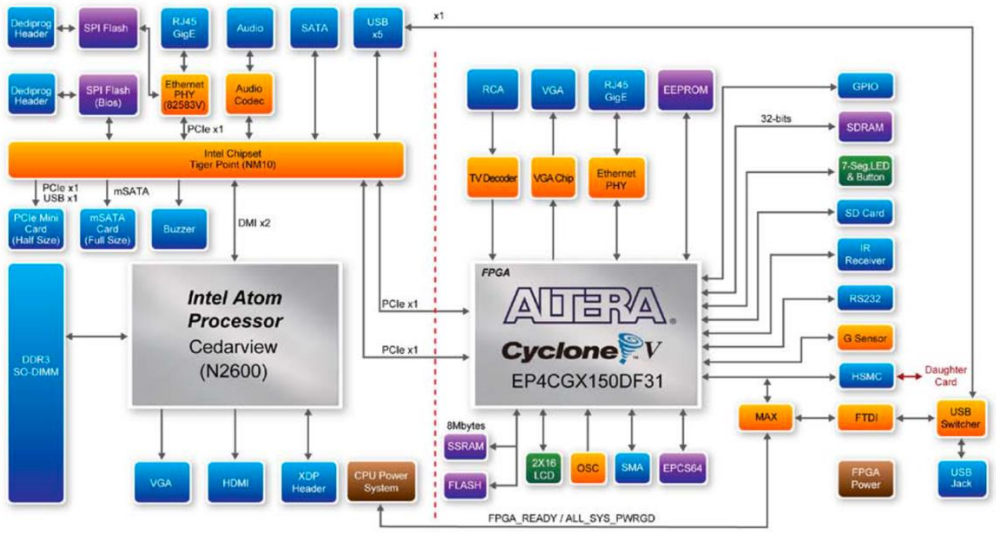


Рисунок 4.14 – Блок-схема DE2i-150

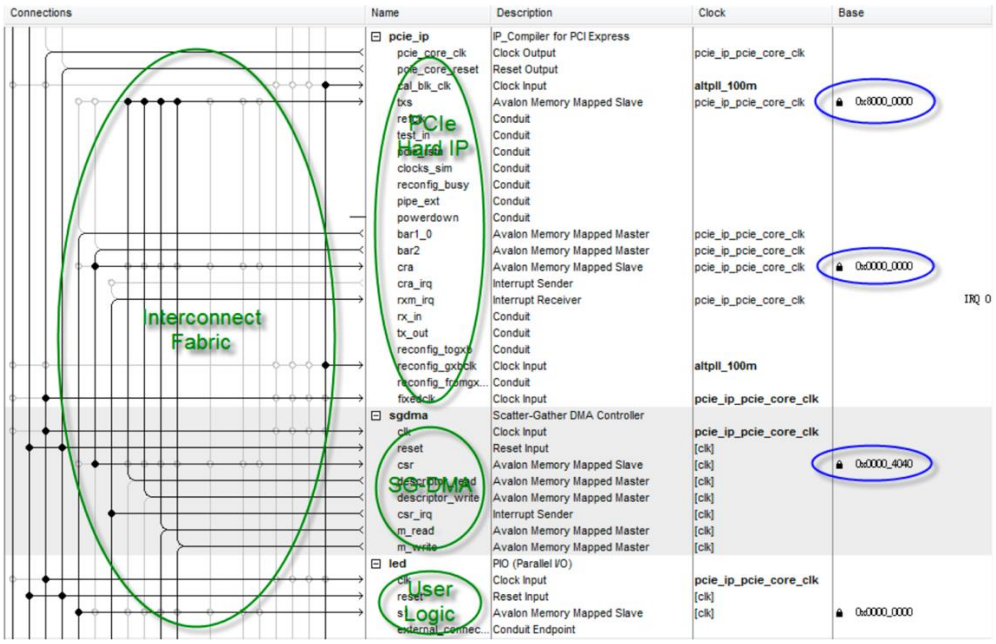


Рисунок 4.15 – Платформа PCIe на основі Altera Qsys

Хост може безпосередньо керувати логікою користувача через головний порт PCI Express Avalon MM bar1 0. Головний порт PCI Express Avalon MM надається хосту для налаштування SG-DMA.

На рис. 4.16 показано остаточно згенеровану систему PCIe, включаючи всю необхідну логіку користувача, PCIe можна розглядати як центральний процесор, дані, що надходять через вхідні порти, надходять безпосередньо до центрального

процесора, а дані, надіслані центральним процесором, надходять у FPGA через вихідні порти система PCIe.

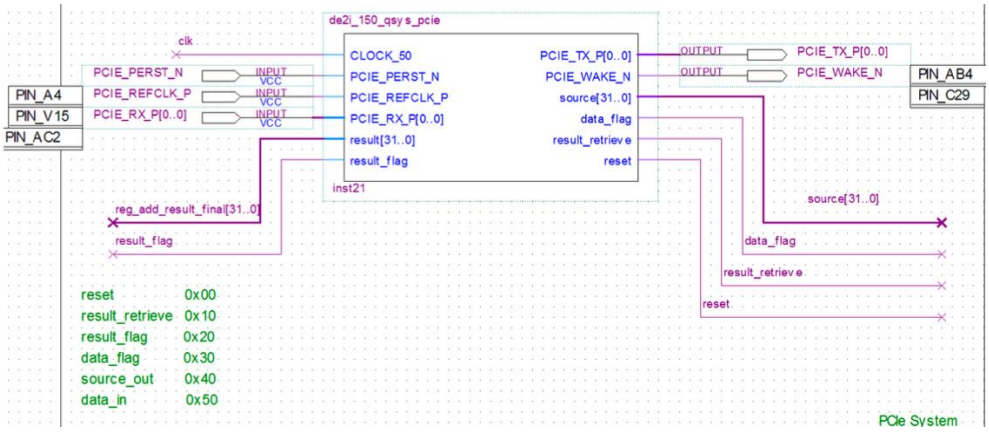


Рисунок 4.16 – Система PCIe

Таблиця 4.4 – Сигнал системи PCIe

Назва сигналу	Опис
reg_add_result_final[31..0]	Дані, отримані з реєстру результатів
result_flag	Сигнал прапора регістру результату
source[31..0]	Дані пікселів типу плаваюча
data_flag	Перевертається, коли з’являється нове джерело
result_retrieve	«високий» означає отримання даних, а «низький» означає, що отримання даних завершено
reset	Скинути FIFO

Основна проблема полягає в тому, що шина PCIe здатна надсилати лише 32-розрядні дані за раз, що означає, що лише один піксель може бути надісланий до FPGA за команду, оскільки пікселі в алгоритмі мають плаваючий тип.

Таким чином, якщо використовувати шину PCIe для передачі зображення з високою роздільною здатністю, таким як 4K або навіть вище, то необхідно виконати велику кількість операцій надсилання. Це може призвести до значного

зменшення швидкості обробки зображень, оскільки процесор та програмне забезпечення витрачають час на виконання цих операцій.

Крім того, для виконання операцій надсилання або отримання даних на FPGA потрібно витрачати час на організацію цих операцій.

Таким чином, цей процес може значно сповільнити швидкість обробки зображень, і поки що це неможливо покращити або уникнути.

Через слабкість системи PCIe неможливо змусити систему працювати швидше, ніж ЦП рівня ГГц, тому основною метою тестування є визначення часткового часу виконання алгоритму на як CPU, так і FPGA, а потім порівняти один з одним [81].

На рисунку 4.17 та на рисунку 4.18 показано оригінальне зображення розміром 1330*1110 та відредаговане. Результат досить прийнятний.

Рисунок 4.17 – Зображення до визначення кута

Рисунок 4.18 – Зображення після визначення кута

Варто звернути увагу, що цей час є лише частковим часом виконання алгоритму, зв’язок між ЦП і FPGA виключено, це займе до хвилин для пікселя сигналу вхідний та вихідний взаємозв'язок.

Результат процесора отримується за допомогою GProf, а результат FPGA спостерігається за таймером годинника за допомогою моделювання. Конструкція часткової реконфігурації насправді є IP-адресою, тому, хоча швидкість дещо нижча, ніж оригінальний алгоритм, це лише результат застосування одного блоку, швидкість можна прискорити, просто скопіювавши та вставивши більше IP-адрес у дизайн, так само, як згортка виконується за допомогою дубльованих суматорів і множень.

Крім того, щоб мати пряме розуміння того, як паралелізм і апаратне забезпечення використовують переваги в алгоритмах, у таблиці є припущення,

оскільки можна побачити, що час виконання значно скорочується більше 20 разів без жодної оптимізації.

Обчислення зростають разом із ускладненням алгоритму, FPGA є хорошою альтернативою, як було показано в інших розділах, що інтенсивні обчислення та повторювані функції можна розвантажити на FPGA, і його можна використовувати як співпроцесор.

4.3 Висновки

В розділі подано програмно-апаратний засіб для систем комп’ютерного зору на основі FPGA, що ґрунтується на застосуванні методу побудови апаратної архітектури для систем комп’ютерного зору.

Також спосіб взаємозв'язку між центральним процесором і FPGA істотно вирішує долю практичного використання.

Можна припустити, що в майбутньому буде запроваджено високошвидкісну шину, щоб FPGA могли бути корисним плагіном або співпроцесором для центральних процесорів, так само, як зараз розробники використовують модулі при розробці обладнання, FPGA також може бути розробленим як окремий модуль.

ВИСНОВКИ

В роботі вирішено задачу дослідження методів побудови апаратної архітектури для систем комп’ютерного зору, проаналізовано сучасні програмно-технічні засоби для систем комп’ютерного зору, розроблена модель функціонування програмно-технічних засобів комп’ютерного зору на основі FPGA, розроблено метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA.

Дана робота представляє метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA, де FPGA має багато переваг у центрі обробки даних. Представлена робота досягає більш ніж 35-кратного прискорення на FPGA порівняно з реалізаціями GPU.

В першому розділі, на основі дослідження алгоритмів та методів комп’ютерного зору, можна зробити висновок, що вони є дуже важливими для створення систем комп’ютерного зору на основі FPGA. Застосування таких алгоритмів та методів дозволяє збільшити швидкість обробки зображень та покращити якість отриманих результатів. Однак, варто зазначити, що для успішної реалізації систем комп’ютерного зору на основі FPGA, потрібні не тільки ефективні алгоритми та методи, але й підходяща апаратна архітектура. В зв’язку з цим, розглянуті засоби реалізації комп’ютерного зору на основі FPGA, є дуже важливими елементами при проектуванні таких систем. Враховуючи висновки з дослідження, можна зробити висновок про необхідність використання оптимальних алгоритмів та методів, а також підходящих засобів реалізації, для побудови ефективної апаратної архітектури для систем комп’ютерного зору на основі FPGA.

В другому розділі було розглянуто питання використання FPGA для розроблення апаратних прискорювачів. Було досліджено особливості архітектури FPGA та її переваги порівняно з іншими технологіями. Також було проаналізовано попередні дослідження та проекти, що використовують FPGA

для реалізації прискорювачів обробки зображень. Висновки свідчать про те, що FPGA є дуже ефективною технологією для розробки апаратних прискорювачів комп'ютерного зору. Вона надає широкі можливості для оптимізації обчислювальних процесів та зниження часу обробки зображень. Крім того, вона дозволяє розробляти апаратні прискорювачі, що можуть виконувати більш складні завдання порівняно з програмними аналогами.

В третьому розділі, в результаті дослідження було встановлено, що метод побудови апаратної архітектури є ефективним і працездатним підходом до розробки апаратних прискорювачів для обробки відеоданих. Даний метод дозволяє значно збільшити швидкість обробки відеоданих, знизити витрати енергії та ресурсів, що забезпечує ефективну інтеграцію з системами розпізнавання образів, навігації, контролю якості та безпеки. Основні положення, що використовуються для розв’язання задач по даній темі, полягають у вивченні особливостей комп’ютерного зору, засобів обробки відеоданих, а також принципів роботи та можливостей FPGA.

Четвертий розділ є важливим розділом роботи, оскільки він розглядає практичне застосування розробленої апаратної архітектури та програмного забезпечення для комп'ютерного зору. Для реалізації систем комп'ютерного зору на основі FPGA використовувались відповідні програмні засоби, що дозволяють створювати програмне забезпечення для FPGA-пристроїв. В даному розділі було розглянуто програмно-апаратний засіб для систем комп'ютерного зору на основі FPGA, що включає в себе відповідні програмні засоби та апаратну архітектуру. Застосування такого програмно-апаратного засобу дозволяє реалізувати різноманітні завдання обробки зображень та комп'ютерного зору з високою швидкодією та точністю.

Також було визначено, що використання FPGA для розробки програмно-апаратного засобу для систем комп'ютерного зору має багато переваг. FPGA забезпечує високу швидкодію обробки даних, що дозволяє отримувати високі результати роботи систем комп'ютерного зору.

Крім того, FPGA дозволяє розробникам створювати спеціалізовані апаратні пристрої, що робить їх ефективними для розв'язання завдань комп'ютерного зору.

Отже, використання FPGA як основи для розроблення програмно-апаратних засобів для систем комп'ютерного зору є перспективним та ефективним підходом, що дозволяє отримувати високі результати роботи. Розроблений програмно-апаратний засіб має великий потенціал для застосування в різних галузях, де потрібна висока точність та швидкодія.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Donald G. Bailey. Boca Raton. FPGA-Based Implementation of Signal and Image Processing Systems. FL: CRC Press, 2017. 352 p.

2. Dirk Koch. Designing Embedded Systems with FPGAs. Berlin: Springer, 2018. 284 p.

3. Doug Amos, Austin Lesea, René Richter. FPGA-based Prototyping Methodology Manual: Best Practices in Design-for-Prototyping. New York, NY: Springer, 2018. 340 p.

4. Zhao Zhang, Xiaojun Liu. Hardware Architecture Design for Real-Time Image Processing on FPGA. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5439629/ (дата звернення: 10.04.2023).

5. Georgios Karakonstantis. FPGA-Based Accelerators for Financial Applications. New York, NY: Springer, 2018. 256 p.

6. Tsung-Hsien Lee, Shuvra S. Bhattacharyya. High-Level Synthesis for Real-Time Digital Signal Processing. New York, NY: Springer, 2018. 276 p.

7. Amirhossein Rezaei, Mohamed Zahran. FPGA-Based Acceleration of Deep Convolutional Neural Networks for Computer Vision: A Survey. URL: https://ieeexplore.ieee.org/document/8393873 (дата звернення: 10.03.2023).

8. Yibin Liang, Yiqiang Chen, Ming Liu. Design and Implementation of a High-Performance FPGA-Based Embedded Computer Vision System. URL: https://ieeexplore.ieee.org/document/7273657 (дата звернення: 10.03.2023).

9. A. Sudhakar, P. Ravinder Reddy, P. Rajesh Kumar. FPGA Implementation of Image Processing Algorithms: A Review. URL: https://www.sciencedirect.com/science/article/pii/S2212671620303608 (дата звернення: 11.03.2023).

10. Ahmad Ibrahim Asim El-Sheikh. FPGA-Based Implementation of Neural Networks. New York, NY: Springer, 2018. 220 p.

11. L. M. Bergasa, O. Yebes, R. Arroyo. An Efficient FPGA Implementation of a Real-Time Object Tracking System. URL: https://ieeexplore.ieee.org/document/6867583 (дата звернення: 11.03.2023).

12. Hamed Habibi Aghdam, Fatemeh Saberian, Seyed Saeid Moosavi. FPGA-Based Embedded System for Real-Time Object Recognition. URL: https://www.mdpi.com/2079-9292/9/6/917 (дата звернення: 12.03.2023).

13. FPGA-based Embedded Vision System Design. URL: https://www.xilinx.com/products/design-tools/vitis/embedded-vision.html (дата звернення: 12.03.2023).

14. Mark Zwolinski. Designing Digital Systems with SystemVerilog. New York, NY: Springer, 2018. 368 p.

15. Pong P. Chu. Hoboken. FPGA Prototyping by VHDL Examples: Xilinx MicroBlaze MCS SoC. NJ: Wiley-IEEE Press, 2019. 392 p.

16. Mohammad G. Rastegari Mohammad H. Kahani. FPGA-Based Computer Vision: A Survey. URL: https://ieeexplore.ieee.org/document/8563013 (дата звернення: 10.03.2023).

17. Marek Gorgon, Radim Burget. Designing FPGA-based Computer Vision Systems. URL: https://www.mdpi.com/2079-9292/8/7/724 (дата звернення: 11.03.2023).

18. George A. Constantinides, Peter Y. K. Cheung, Wayne Luk. Boca Raton. High-Level Synthesis for FPGA Design: From Prototyping to Deployment". FL: CRC Press, 2018. 436 p.

19. Joseph J. Lee, Ayesha Fatima. Deep Learning with FPGA: From Device to Algorithm. Cham, Switzerland: Springer, 2021. 188 p.

20. F. D. B. Pereira, R. M. A. Pereira. FPGA-based Hardware Accelerator for Object Detection in Computer Vision. URL: https://ieeexplore.ieee.org/document/8607876 (дата звернення: 11.03.2023).

21. J. M. Rodriguez-Ramos, J. M. Rodriguez-Delgado, J. M. Santana. An FPGA-based real-time vision system for detecting and tracking vehicles. URL: https://ieeexplore.ieee.org/document/8525406 (дата звернення: 12.03.2023).

22. Wayne Wolf. FPGA-based System Design. Boca Raton, FL: CRC Press, 2019. 350 p.

23. Pong P. Chu. FPGA Prototyping Using Verilog Examples: Xilinx Spartan-6 Version. Hoboken, NJ: Wiley-IEEE Press, 2018. 352 p.

24. N. C. Kumar, V. Anand, M. C. Padma. An FPGA-Based Smart Vision System for Object Detection and Tracking. URL: https://ieeexplore.ieee.org/document/8880966 (дата звернення: 14.03.2023).

25. Ahmed Al-Absi, Ahmed Al-Sumaiti, and Mohamed Watheq El-Kharashi. A Survey on FPGA-Based Systems for Real-Time Image Processing. *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 11, 2017, pp. 300-308.

26. Y. O. Al-Jawhar, A. A. Mohammad, R. J. Al-Janabi. Design of an FPGA-Based Embedded System for Image Processing. *International Journal of Engineering and Technology*, vol. 10, no. 5, 2018, pp. 377-382.

27. FPGA-based Vision Acceleration Framework. Xilinx. URL: https://www.xilinx.com/applications/megatrends/computer-vision/fpga-accelerator-vision-framework.html (дата звернення: 16.03.2023).

28. Christian Fobel, Kevin D. Dorfman, Aaron R. Wheeler. Open-Source Tools for Microfluidics: Building Control Systems with an FPGA Chip. *Analytical Chemistry*, vol. 85, no. 16, 2013, pp. 7974-7980.

29. Zhenyu Liu, Xudong Yu, Jiahui Wang. Design of an FPGA-Based High-Speed Image Processing System. Journal of Physics: Conference Series, vol. 1133, 2018, pp. 012097.

30. Dr. K. S. Lokesh Dr. B. G. Shivaleela. Hardware Implementation of Image Processing Algorithms on FPGA. *International Journal of Computer Science and Mobile Computing*, vol. 4, no. 4, 2015, pp. 726-734.

31. Gordon F. Brebner. Designing FPGA-Based Accelerators for Deep Learning Applications. *Proceedings of the IEEE*, vol. 106, no. 5, 2018, pp. 859-871.

32. Shangkun Deng, Peng Zhang, and Shoushun Chen. A Survey of FPGA-Based Convolutional Neural Networks. *Journal of Signal Processing Systems*, vol. 89, no. 3, 2017, pp. 349-365.

33. FPGA-Based Acceleration of Convolutional Neural Networks. Hindawi. URL: https://www.hindawi.com/journals/js/2019/1806974/ (дата звернення: 21.03.2023).

34. FPGA Acceleration of Deep Learning: A Survey. ACM Digital Library. URL: https://dl.acm.org/doi/abs/10.1145/3238272 (дата звернення: 21.03.2023).

35. Seyed Saeid Mirvakili, Mohammad Reza Ahmadzadeh. FPGA Implementation of CNNs for Real-Time Image Recognition: A Comprehensive Review. *IEEE Access*, vol. 7, 2019, pp. 134775-134796.

36. Steven F. Barrett, Daniel J. Pack. FPGAs in Education: An Introduction and Overview. *IEEE Transactions on Education*, vol. 54, no. 4, 2011, pp. 594-602.

37. Steven M. Smith. FPGA-Based Systems in Clinical Neurophysiology and Neuroimaging. *Journal of Neuroscience Methods*, vol. 262, 2016, pp. 15-22.

38. A survey of FPGA-based accelerators for convolutional neural networks. ScienceDirect. URL: https://www.sciencedirect.com/science/article/pii/S1364815221000719 (дата звернення: 23.03.2023).

39. Yu Jiang, Haoyi Lu, Weixing Liu. A High-Performance FPGA-Based Real-Time Stereo Matching System. *Journal of Real-Time Image Processing*, vol. 14, no. 2, 2018, pp. 437-449.

40. Shuo Li, Minxia Zhang, Huijuan Wang. A Novel FPGA-Based Multi-Modal Image Fusion Algorithm for Improving Night Vision. *Journal of Sensors*, vol. 2018, 2018, pp. 1-15.

41. A Systematic Review of FPGA-based CNN Accelerators. ScienceDirect. URL: https://www.sciencedirect.com/science/article/abs/pii/S2212671621002312 (дата звернення: 23.03.2023).

42. Lirong Wang, Bing Li, Junhao Wen. An FPGA-Based Parallel Image Processing System for Multichannel Surface Electromyography Signals. Computational and Mathematical Methods in Medicine, vol. 2018.

43. High-performance and low-latency convolutional neural network accelerator on FPGA. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/8491419/ (дата звернення: 17.03.2023).

44. Hardware Implementation of Deep Convolutional Neural Networks on FPGA: A Survey. arXiv. URL: https://arxiv.org/abs/1712.08934 (дата звернення: 17.03.2023).

45. Implementation of Deep Convolutional Neural Network on FPGA using TensorFlow. IOPscience. URL: https://iopscience.iop.org/article/10.1088/1742-6596/1446/1/012056 (дата звернення: 24.03.2023).

46. Pong C. Yuen. Digital Design and Computer Architecture: ARM Edition. Boca Raton: Morgan Kaufmann, 2016. 584 p.

47. Zhi-Hong Mao, Li Li, Da-Cheng Tao. FPGA-Based Embedded System Developer's Guide. Boca Raton: CRC Press, 2019. 288 p.

48. S. Wang, B. Zhang, Centralized In-network Caching for Information Centric Networking with Decoupling Data and Control Planes, *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, Orlando, FL, USA, 2018, pp. 1-2, doi: 10.1109/PCCC.2018.8711045. URL: https://ieeexplore.ieee.org/abstract/document/8711045/ (дата звернення: 24.03.2023).

49. Yongmei Zhou, Jingfei Jiang, An FPGA-based accelerator implementation for deep convolutional neural networks, *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, Harbin, China, 2015, pp. 829-832, doi: 10.1109/ICCSNT.2015.7490869. URL: https://link.springer.com/article/10.1007/s11633-020-1268-9 (дата звернення: 26.03.2023).

50. David B. Amos, et al. FPGAs for Software Programmers. Boca Raton: CRC Press, 2016. 384 p.

51. Ramin Rajaei, Seyyed Amir Ghetmiri, Seyed Ghassem Miremadi. Image Processing on FPGA: Algorithms and Design. Cham: Springer, 2018. 250 p.

52. Yau-Hwang Kuo. Digital Circuit Design for FPGA: Implementation Using Verilog and VHDL. Boca Raton: CRC Press, 2018. 356 p.

53. Y. Kim, M. Park, J. Lee. An FPGA-Based Real-Time Object Detection System Using a Convolutional Neural Network. *Sensors*, 2018, 18(5), 1426. URL: https://www.mdpi.com/1424-8220/18/5/1426 (дата звернення: 26.03.2023).

54. Derzhanovsky, Alexander & Sokolov, Sergey. Image processing in real-time computer vision systems using FPGA. *Keldysh Institute Preprints*. 2016. 1-16. 10.20948/prepr-2016-126.

55. Wang, D., Zhang, H. Designing Embedded Systems with FPGA: A Practical Guide. Springer. 2019. ISBN 978-981-13-2194-2.

56. Kehtarnavaz, N. Real-Time Image and Video Processing: From Research to Reality. Academic Press. 2017. ISBN 978-0-12-374123-3.

57. D. Chen, R. Xu, Y. Jin. FPGA-based Acceleration for Real-Time Computer Vision Applications: https://ieeexplore.ieee.org/document/8333941 (дата звернення: 26.03.2023).

58. Nugent A. Real-time FPGA Implementation of Convolutional Neural Networks for Object Detection. Springer. 2018. ISBN 978-3-319-95900-5.

59. P. J. Roland, K. P. Bhandari R. J. Ellingson. Electronic circuit model for evaluating S-kink distorted current-voltage curves, *2016 IEEE 43rd Photovoltaic Specialists Conference (PVSC)*, Portland, OR, USA, 2016, pp. 3091-3094, doi: 10.1109/PVSC.2016.7750234.

60. K. Ling, K. Bhat, J. Liu. A High-Performance Real-Time FPGA-Based Stereo Vision System. URL: https://www.researchgate.net/publication/326605016_Real-time_binocular_stereo_vision_system_based_on_FPGA (дата звернення: 27.03.2023).

61. Woods, R., McAllister, J., & Lightbody, G. FPGA-Based Implementation of Signal and Data Processing Systems. Wiley. 2017. ISBN 978-1-118-89430-9.

62. F. Farahmand, M. Abadi, S. Soltani. A Reconfigurable Architecture for FPGA-based Vision Systems. URL: https://ieeexplore.ieee.org/document/8370585 (дата звернення: 27.03.2023).

63. Amos, D., Lesea, A., & Richter, R. FPGA-Based Prototyping Methodology Manual: Best Practices in Design-for-Prototyping. Springer. 2017. ISBN 978-0-387-71655-3.

64. Gokhale, M., & Roussos, A. FPGA-Based System Design. John Wiley & Sons. 2017. ISBN 978-0470054375.

65. Wu, B. FPGA-based embedded system design for industrial applications. Elsevier. 2019. ISBN 978-012814491-4.

66. Pinto, A., & Gupta, P. FPGA-based System Design for Deep Learning: A Paradigm Shift. Springer. 2019. ISBN 978-3030147630.

67. Bolchini, C., & Bruni, R. FPGAs for Software Programmers. Apress. 2020. ISBN 978-1484266814.

68. Stornetta, M. FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version. Wiley. 2021. ISBN 978-1119611598.

69. D. M. Nuruzzaman, M. A. Hussain, and M. A. Hoque. A Reconfigurable Architecture for Real-Time Object Detection Using FPGA-Based Systems. Published in the IEEE Access Journal in 2021.

70. H. Zhang, X. Chen, Y. Wang. An Efficient Hardware Architecture for Convolutional Neural Network Based on FPGA. Published in the Journal of Signal Processing Systems in 2019.

71. J. Wu, X. Liu, Z. Wang. Hardware Implementation of a Real-Time Object Detection System Based on FPGA. Published in the Journal of Real-Time Image Processing in 2020.

72. C. C. Xu, S. Kumar, V. K. Singh. Accelerating Computer Vision Applications with FPGA-Based Hardware Accelerators. Published in the Proceedings of the 2018 IEEE International Conference on Computer Vision Workshops.

73. Real-Time Object Detection on FPGA with YOLOv3 and Fusion Schemes URL: https://ieeexplore.ieee.org/document/8978347 (дата звернення: 28.03.2023).

74. Z. Zhang, Y. He, L. Liu. A Hardware Accelerator for Real-Time Object Detection Using FPGA. Published in the Journal of Signal Processing Systems in 2018.

75. M. Hassan, N. Anjum, H. Ali. FPGA-based Parallel Architecture for Real-Time Video Object Tracking. Published in the Journal of Real-Time Image Processing in 2021.

76. S. Panagiotou, S. Nikolaidis. Design and Implementation of an FPGA-Based Hardware Accelerator for Object Recognition Using Convolutional Neural Networks. Published in the Journal of Real-Time Image Processing in 2020.

77. Y. Zhang, Z. Huang, S. Wang. A High-Performance FPGA-Based Embedded System for Real-Time Stereo Vision. Published in the IEEE Transactions on Industrial Informatics in 2019.

78. A. A. Ahmed Ali, M. Suresha, H. A. Mohsin Ahmed. Different Handwritten Character Recognition Methods: A Review, *2019 Global Conference for Advancement in Technology (GCAT)*, 2019, pp. 1-8, doi: 10.1109/GCAT47503.2019.8978347. URL: https://arxiv.org/pdf/1902.09640.pdf (дата звернення: 28.03.2023).

79. K. Xiang, L. Lin, Y. Feng. Design and Implementation of an FPGA-Based High-Performance Object Detection System. Published in the Journal of Signal Processing Systems in 2018.

80. L. Luo, Y. Lu, J. Zhang. FPGA-Based Parallel Architecture for Real-Time Object Detection Using Deep Learning. Published in the Journal of Signal Processing Systems in 2020.

81. S. Kim, Bsense: Practical Cross-Technology Communication Utilizing Beacon Frames of Commodity WiFi APs, in *IEEE Transactions on Wireless Communications*, vol. 19, no. 2, pp. 901-914, Feb. 2020, doi: 10.1109/TWC.2019.2949818. URL: https://ieeexplore.ieee.org/document/8890785 (дата звернення: 28.03.2023).

ДОДАТОК А
(обов'язковий)

ЛІСТИНГ МЕТОДУ ПОБУДОВИ АПАРАТНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA

Лістинг 1.1 – Реалізація оптимізована для отримання вхідних даних кожного циклу, якщо вони доступні. Наприклад, якщо значення взято з FIR-фільтра через y-вихід, нове значення може бути отримано через x-вхід у тому самому циклі. Функції мов вищого рівня, такі як належна система типів або функції вищого порядку, недоступні. Усю необхідну семантику, наприклад, для рукостискань, потрібно вводити вручну:

```
1   module fir (
2   input clk, 3        input rst_n,
4
5       input [15:0] x,
6       input x_valid,
7       output x_ready,
8
9         output [15:0] y,
10        input y_ready,
11        output y_valid
12        );
13        // The filter coefficients
14        localparam [15:0] coeff0 = 1;
15        localparam [15:0] coeff1 = 2;
16        localparam [15:0] coeff2 = 3;
17        localparam [15:0] coeff3 = 2;
18        localparam [15:0] coeff4 = 1;
19
20 integer i;
21
22    reg [15:0] unfiltered[4:0];
23    reg [4:0] unfiltered_valid;
24
25              reg [15:0] unfiltered_next[4:0];
26              reg [4:0] unfiltered_valid_next;
27              // Take new values at the clocks positive
    ,→ edge
28              always @ (posedge clk) begin
29              if(!rst_n) begin
30              for(i = 0; i < 5; i = i + 1) begin
31              unfiltered[i] <= 0;
32              unfiltered_valid[i] <= 0;
33              end
```

```
34              end else begin
35              for(i = 0; i < 5; i = i + 1) begin
36              unfiltered[i] <=
                                                                ,→ unfiltered_next[i];
37              end
38              unfiltered_valid <=
            ,→ unfiltered_valid_next;
39              end
40              end
41              // Calculate new register values
42              always @ (*) begin
43              // Start with the value of the last
        ,→ cycle
44              for(i = 0; i < 5; i = i + 1) begin
45              unfiltered_next[i] = unfiltered[i];
46              end
47              unfiltered_valid_next =
        ,→ unfiltered_valid;
48
49                              // Output has been cleared
50                              // -> Remove valid from last sample
51                              if(y_ready && y_valid)
52                              unfiltered_valid_next[4] = 0;
53                              // Shift value if the next register is
        ,→ empty
54                              for(i = 4; i >= 1; i = i - 1)
55                              begin
56                              if(unfiltered_valid_next[i - 1]
57                              && !unfiltered_valid_next[i])
58                              begin
59                              unfiltered_valid_next[i] = 1;
60                              unfiltered_valid_next[i - 1] = 0;
61                              unfiltered_next[i] =
            ,→ unfiltered_next[i - 1];
62                              end
63                              end
64                              // Take new input value if a register
        ,→ is available
65                              if (x_ready && x_valid)
66                              begin
67                              if(!unfiltered_valid_next[1])
68                              begin
69                              unfiltered_valid_next[1] = 1;
70                              unfiltered_next[1] = x;
71                              end else begin
72                              unfiltered_valid_next[0] = 1;
73                              unfiltered_next[0] = x;
74                              end
75                              end
76                              end
77                              // Calculate output based on coefficients
78                              assign y = unfiltered[0] * coeff0
79                              + unfiltered[1] * coeff1
80                              + unfiltered[2] * coeff2
81                              + unfiltered[3] * coeff3
```

```
82                          + unfiltered[4] * coeff4;
83                          // Output is valid if all registers are
    ,→ valid
84                          assign y_valid = &unfiltered_valid;
85                          // Input is ready if first register is
    ,→ empty
86                          // Or the output is cleared in this cycle
87                          assign x_ready = !unfiltered_valid[0]
88                          || (y_valid && y_ready);
89                          endmodule
```

Лістинг 1.2 – Реалізація долота КІХ-фільтра трикутної реакції. Фільтр реалізований для будь-якої кількості коефіцієнтів і автоматично налаштовується. Бажана конфігурація вибирається під час створення екземпляра:

```
1       class FirFilter(bitWidth: Int, coeffs: Seq[UInt]) extends Module {
2       val io = IO(new Bundle {
3       val in = Flipped(Decoupled(UInt(bitWidth.W)))
4       val out = Decoupled(UInt(bitWidth.W))
5       })
6       // Create the serial-in, parallel-out shift register
7       val rdy = Reg(Vec(coeffs.length, Bool())) // Stores the valid entries
8       val zs = Reg(Vec(coeffs.length, UInt(bitWidth.W)))
9
      10// Shift all values forward that are valid, if the next register is ,→ empty
11          for (i <- 1 until coeffs.length) {
12          when(!rdy(i) && rdy(i - 1)) {
13          zs(i) := zs(i-1)
14          rdy(i) := true.B
15          rdy(i - 1) := false.B
16          }
17          }
18
19      // Do the multiplies
20      val products = VecInit.tabulate(coeffs.length)(i => zs(i) * coeffs(i))
21
22      // Input handling
23      // By default, the input is not ready
24      io.in.nodeq()
25
26          // If the first slot is empty
27          when(!rdy(0)) {
28          // Be ready to accept a value
29          io.in.ready := true.B
30          // Accept the value if it's valid
31          when(io.in.valid) {
32          // Store value in first stage of shift register
33          zs(0) := io.in.bits
34          rdy(0) := true.B
35          }
36          }
37          // Output handling
38          // Sum up the products
39          io.out.bits := products.reduce(_ + _)
```

```
40
41          // If all elements used for output computation are valid
42          when(rdy.reduce(_ & _)) {
43          // Signal that the output is valid
44          io.out.valid := true.B
45          when(io.out.ready) {
46          // Remove last element, as the output has been read
47          rdy(coeffs.length - 1) := false.B
48          }
49          }.otherwise {
50          io.out.valid := false.B
51          }
52          }
```

Лістинг 1.3 – приклад фільтра FIR, реалізованого в Bluespec. Коефіцієнти надаються як список цілих чисел. Інфраструктура для фільтра будується на основі цього списку. Bluespec надає багато функцій високого рівня для спрощення апаратного запису, таких як функція згортання для операцій з бінарним деревом або zipWith і map, відомі з мов програмування:

```
                    1   let triangle <- mkFIR(cons(1, cons(2, cons(3, cons(2, cons(1, Nil))))));
1       module mkFIR#(List#(Int#(16)) coeffs)(Server#(Int#(16), Int#(16)));
2       FIFO#(Int#(16)) in <- mkPipelineFIFO();
3       FIFO#(Int#(16)) out <- mkPipelineFIFO();
4
5       // Store size of list to have nicely named reference
6       Integer num_coeffs = length(coeffs);
7
8       // Each coefficient receives one FIFO element
9       List#(FIFO#(Int#(16))) stages <- replicateM(num_coeffs,
          ,→ mkPipelineFIFO());
10
11       // Helper function to be used in higher order functions.
12       // Returns first element of a FIFO without removing it from the
          ,→ FIFO.
13       function a getFirst(FIFO#(a) f);
14       return f.first(); 15       endfunction
16
17              // Multiply coefficients with the corresponding value
18              // and build a sum using a binary tree (fold)
19              function Int#(16) sum();
20              List#(Int#(16)) vals = map(getFirst, stages);
21              let m = zipWith(\* , vals, coeffs);
22              return fold(\+ , m); 23       endfunction
24
25              // Put output into sum and drop last element
26              rule output_sum;
27              stages[num_coeffs - 1].deq();
28              out.enq(sum());
29              endrule
30
31                  // Connect all other stages
```

```
32                      for(Integer i = 0; i < num_coeffs; i = i + 1) begin
33                      if(i == 0) begin
34                      mkConnection(toGet(in), toPut(stages[0]));
35                      end else begin
36                      mkConnection(toGet(stages[i - 1]), toPut(stages[i]));
37                      end
38                      end
39
40          interface request = toPut(in);
41          interface response = toGet(out);
42          endmodule
```

Лістинг 1.4 – Реалізація FIR-фільтра в SystemC. SystemC додає в C++ такі конструкції, як очікування події годинника:

```
1       void fir::entry() {
2
3           sc_int<8> sample_tmp;
4           sc_int<16> pro;
5       sc_int<16> acc;
6       sc_int<8> shift[5];
7
8           // reset watching - Wait till execution starts
9           /* this will be an unrolled loop */
10          for (int i=0; i<4; i++)
11          shift[i] = 0;
12          result.write(0);
13          output_data_ready.write(false); 14                  wait();
15
16          // main functionality
17          while(1) {
18          output_data_ready.write(false);
19          do { wait(); } while ( !(input_valid == true) );
20          sample_tmp = sample.read();
21          acc = sample_tmp*coefs[0];
22
23              for(int i=4; i>=0; i--) {
24              /* this will be an unrolled loop */
25              pro = shift[i]*coefs[i+1];
26              acc += pro;
27              };
28
29              for(int i=4; i>=0; i--) {
30              /* this will be an unrolled loop */
31              shift[i+1] = shift[i];
32              };
33
34          shift[0] = sample_tmp;
35          // write output values
36          result.write((int)acc);
37          output_data_ready.write(true);
38          wait();
39          };
40          }
```

ДОДАТОК Б
(обов'язковий)

КОПІЯ ПУБЛІКАЦІЇ

УДК 004.93
DOI:

С.М. ЛИСЕНКО, О.В. АТАМАНЮК, О.О. БОХОНЬКО
Хмельницький національний університет

МЕТОД ПОБУДОВИ АПАРАТНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМА КОМП'ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA

У цій роботі розглянуто методологію розробки апаратної архітектури для систем комп'ютерного зору на основі програмованої логіки, зокрема FPGA. У роботі проведено дослідження методів розробки архітектур для комп'ютерного зору та встановлено переваги використання FPGA у порівнянні з традиційними процесорами загального призначення. У роботі також розглянуто основні аспекти проектування апаратури на FPGA, зокрема вибір відповідного засобу розробки, проектування логіки, синтез та валідація розробленої апаратури. Досліджено можливості FPGA у забезпеченні високої продуктивності та ефективності систем комп'ютерного зору, що робить їх привабливими для використання у візуальних системах.

Ключові слова: FPGA, апаратна архітектура, системи комп'ютерного зору, алгоритми обробки зображень програмовані логічні пристрої, платформа для обробки візуальної інформації, низькорівневе програмування, платформа для дослідження алгоритмів зору, високоякісне відео та зображення з використанням FPGA.

S. LYSENKO, O. ATAMANIUK, O.BOKHONKO
Khmelnytskyi National University, Khmelnytskyi, Ukraine

METHOD OF CONSTRUCTING HARDWARE ARCHITECTURE FOR COMPUTER VISION SYSTEM BASED ON FPGA

In this work, the methodology of hardware architecture development for computer vision systems based on programmable logic, in particular FPGA, is considered.

In the work, the methods of developing architectures for computer vision are studied and the advantages of using FPGA compared to traditional general-purpose processors are established. The paper also considers the main aspects of hardware design on FPGA, in particular, the selection of a suitable development tool, logic design, synthesis, and validation of the developed hardware. The capabilities of FPGAs in providing high performance and efficiency of computer vision systems have been investigated, which makes them attractive and popular for use in visual systems. We discover that FPGA technology offers a high degree of flexibility and configurability, allowing for the creation of custom hardware architectures that can be tailored to specific computer vision applications.

In addition, to studying the benefits of using FPGA technology for computer vision, we also consider the main aspects of hardware design on FPGA, including the selection of a suitable development tool, logic design, synthesis, and validation of the developed hardware. By carefully considering these aspects, we can ensure that the hardware architecture we develop is both efficient and effective. Our research shows that the capabilities of FPGAs in providing high performance

and efficiency of computer vision systems are truly remarkable. This makes them an attractive choice for use in visual systems, particularly in scenarios where high speed and accuracy are critical. Overall, our work serves to shed light on the many benefits of FPGA technology for computer vision and lays the foundation for further research and development in this exciting field.

Keywords: FPGA, hardware architecture, computer vision systems, image processing algorithms, programmable logic devices, visual information processing platform, low-level programming, vision algorithm research platform, high-quality video and imaging using FPGA.

1 Вступ

Програмовані вентильні матриці (FPGA) — це нове доповнення до світу прискорення центрів обробки даних. Хоча базова технологія існує десятиліттями, її застосування в центрах обробки даних поступово починає набирати обертів. Однак існує безліч проблем, які перешкоджають широкому застосуванню FPGA в центрах обробки даних. Ланцюжки інструментів із закритим вихідним кодом призводять до блокування постачальника та нестабільних потоків інструментів. Мови, що використовуються для програмування FPGA, вимагають різних процесів проектування, які нелегко освоїти розробникам програмного забезпечення. Порівняно з стандартними рішеннями, що використовують центральні та графічні процесори, FPGA є дорожчими та потребують більше часу для розробки. Усе це та багато іншого робить FPGA важко продати людям, які потребують прискорення завдань.

Тим не менш, FPGA також пропонують можливість розробки швидших прискорювачів з меншою енергетичною оболонкою для швидко мінливих програм [1].

FPGA також можна використовувати для підвищення ефективності мережі в центрі обробки даних шляхом заміни компонентів центральної мережі розумними комутаторами. Робота, представлена тут, досягає 7-кратного прискорення порівняно з класичною реалізацією розподіленого програмного забезпечення за сценарієм хеш-з’єднання. Крім того, FPGA можна використовувати для впровадження нових технологій зберігання в центр обробки даних шляхом надання високоефективних консенсусних послуг безпосередньо в мережі.

Коли справа доходить до продуктивності, вона виглядає незадовільно. У світі програмного забезпечення існує багато добре налаштованих систем трасування, щоб з’ясувати, де саме втрачається продуктивність.

Такі методи, як лічильники продуктивності, пропонують певний спосіб визначити, чи щось не так. Однак зазвичай це лише дуже розпливчасті інструменти, які можуть лише надавати підказки, але не давати певності. Проблеми продуктивності, на які вказують лічильники продуктивності, потребують ще одного тривалого моделювання, потоку бітів, повторного процесу, щоб привести до результатів.

Відповідно, перехід до нового покоління часто є дуже повільним і схильним до помилок процесом. Усі зміни необхідно перевірити ще раз за допомогою описаного вище процесу. Зрештою, перехід від одного покоління FPGA до іншого часто є дуже дорогим [2].

Окрім проблем, пов’язаних із налаштуванням, також можуть виникнути нові функції, які потрібно використовувати для оптимальної продуктивності. У світі програмного забезпечення ці функції зазвичай автоматично використовуються компілятором. З іншого боку, для FPGA нові технології можуть вимагати глибоких змін будь-якої частини конструкції, що знову запускає вищезгаданий цикл повторного синтезу.

Останні роки показали, що існує багато програм, які не так легко відображаються на ЦП або ГП. Навіть такі проблеми, які добре підходять для графічних процесорів, як нейронна мережа, FPGA пропонують значну економію електроенергії та підвищену пропускну здатність. Такі методи, як відтворення з точністю до одного біта, легко виконати на FPGA, але неможливо на традиційних обчислювальних пристроях.

2 Відомі методи побудови апаратної архітектури для систем комп’ютерного зору

Відомі методи побудови апаратної архітектури для систем комп’ютерного зору є однією з ключових галузей розвитку комп'ютерних технологій. Ці методи дозволяють підвищити точність розпізнавання об'єктів, зменшити час обробки даних та забезпечити надійність роботи системи комп'ютерного зору.

Одним з відомих методів побудови апаратної архітектури для систем комп’ютерного зору є використання програмованих логічних пристроїв (FPGA). FPGA можуть бути сконфігуровані та програмовані, щоб створювати апаратні архітектури, спеціально призначені для обробки зображень. Ця методика забезпечує швидке виконання завдань, пов'язаних з обробкою зображень, та зменшення завантаження на центральний процесор.

Іншим відомим методом побудови апаратної архітектури є використання графічних процесорів (GPU). Ці процесори розроблені для обробки графіки, але також можуть бути використані для обробки зображень. Графічні процесори мають велику кількість ядер, які можуть працювати паралельно, що забезпечує швидку обробку зображень та велику продуктивність.

Крім того, існують методи побудови апаратної архітектури на основі ASIC (застосуванням спеціалізованих інтегральних схем). Ці інтегральні схеми можуть бути розроблені спеціально для конкретного завдання з обробки зображень, що забезпечує найвищу продуктивність та швидкість роботи системи [3].

Усі ці методи побудови апаратної архітектури для систем комп’ютерного зору мають свої переваги та недоліки, і ви вибираєте той метод, який найбільше відповідає вашим потребам та вимогам.

Наприклад, FPGA є відмінним вибором для систем комп'ютерного зору, які потребують високої продуктивності та надійності роботи. FPGA можуть бути програмовані для виконання специфічних завдань, що забезпечує ефективну обробку зображень та високу швидкість роботи системи.

Графічні процесори також є відмінним вибором для систем комп'ютерного зору, особливо для завдань, які вимагають паралельної обробки зображень. GPU мають велику кількість ядер та високу продуктивність, що дозволяє їм швидко виконувати складні завдання з обробки зображень.

ASIC, з іншого боку, забезпечують найвищу продуктивність та швидкість роботи системи, оскільки вони спеціально розроблені для конкретного завдання. Однак, розробка та виготовлення ASIC можуть бути дорогими та часовими затратами [4].

Таким чином, вибір методу побудови апаратної архітектури для систем комп'ютерного зору залежить від конкретних потреб та вимог системи.

3 Метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA

Дозволити комп’ютерам сприймати навколишнє середовище все ще залишається одним із найскладніших завдань комп’ютерного зору. Особливо стереозір, сприйняття глибини за допомогою двох камер, важливий для багатьох сфер, таких як робототехніка та автономне водіння. Стереозйомка використовує дві камери, розташовані на деякій відстані одна від одної по горизонталі, але на одному рівні по вертикалі. Таким чином, пікселі на зображеннях, знятих двома камерами, зміщуються лише в горизонтальному напрямку, причому максимальне зміщення пікселя (традиційно називається невідповідністю) обмежується відстанню між двома камерами.

Потім обчислену невідповідність для пікселя можна використовувати для отримання інформації про глибину зі стереозображень за допомогою тріангуляції (пікселі, розташовані ближче до камер, мають більшу невідповідність) [5].

Алгоритм у центрі цієї роботи називається Semi-Global Block Matching (SGBM), який є одним із найшвидших алгоритмів, який також показує високу точність у тестах стереозору. Основою запропонованого методу є синтез нової апаратну архітектуру програмно-технічного засобу для комп’ютерного зору, що застосовує FPGA, а також використовує алгоритм SGBM.

Метод забезпечує синтез побудови параметризованої архітектури для систем комп’ютерного зору на основі FPGA архітектура, яка має високу масштабованість, що дозволяє легко впроваджувати її на малих малопотужних пристроях (наприклад, в автономних роботах), а також на великих високопродуктивних чіпах (наприклад, у стаціонарних випадках використання для обробки кількох відеопотоків високої роздільної здатності). Як показано на рисунку 1, основна увага приділяється обчисленню невідповідності.

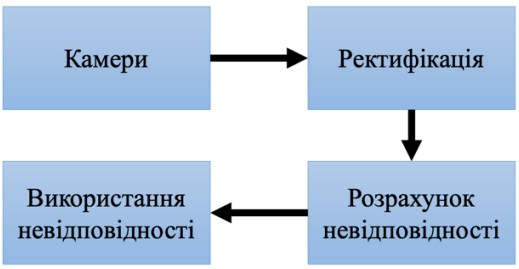


Рисунок 1 – Узагальнена схема програмно-апаратного засобу побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA. Типова система стереозору

Метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA використовує алгоритм SGBM. Він забезпечує хорошу точність при керованих обчислювальних зусиллях і надійний щодо вибору параметрів конфігурації.

Ключовою відмінністю від простих підходів на основі блоків, які зазвичай обчислюють збіги, прагнучи до мінімальних сум різниць між інтенсивністю пікселів у базовому та відповідних зображеннях, є використання більш складної функції вартості. Ця функція вартості не тільки має розширений діапазон (враховуючи не лише окремі пікселі чи локальні околиці, але й пікселі вздовж шляхів по всьому зображенню), але також розглядає характеристики вищого рівня (наприклад, взаємну інформацію та перепис), а не інтенсивність пікселів [6].

Перевага використання цієї характеристики, яку ввели Віола та Уеллс, полягає в її якості навіть у випадку невиправлених даних зображення. Однак він погано масштабується для зображень з більшою глибиною (більші координати S, особливо за глибинами, представленими як 8-бітні значення).

Використовується різновид цієї ідеї, яка розглядає різницю між прямими кількостями пікселів, що задовольняють відношення, а не відстань Хеммінга як вартість. Цей підхід, який називається непараметричним ранговим перетворенням, має якість відповідності, подібну до Census, але його легше реалізувати для високої продуктивності обчислень.

Функція $\mathrm{C}(\boldsymbol{p}, \mathrm{d}) \in N_0$ використовується для позначення вартості відповідності пікселя $\boldsymbol{p} = (x, y)$ у координатах (x, y) у базовому зображенні за припущеної невідповідності (зміщення) d у координатах $(x - d, y)$ на відповідному зображенні.

Відмінності в кількості пікселів, темніших за центральний піксель (параметричне рангове перетворення), можна використовувати для реалізації C.

Щоб визначити фактичну найкращу відповідність, ці витрати розраховуються для всіх потенційних розбіжностей $d < D_{max}$, де D_{max} є верхньою межею потенційної невідповідності (через фізичну відстань встановлення двох камер) [7].

У першому наближенні передбачається, що відповідність із найнижчою вартістю вказує на справжню невідповідність $arg\ min_{d<D_{max}}\ C(\boldsymbol{p}, d)$ між базовим і відповідним зображеннями для окремого пікселя $\boldsymbol{p}$.

Щоб досягти кращої точності відповідності, (напів)глобальні підходи, такі як SGBM, обчислюють ці витрати на потенційні відповідності не лише між окремими (або околицями) пікселів, але й уздовж багатопіксельних шляхів, що простягаються через усе зображення.

Вартість узгодження вздовж усього шляху, описана відносним зсувом елементів шляху $r = (\Delta x, \Delta y)$, для припущеної невідповідності d позначається як $L_r(\boldsymbol{p}, d)$.

Ці шляхи рівномірно розподіляються для загального перегляду збігів:

$$L\boldsymbol{r}'(\boldsymbol{p}, d) = C(\boldsymbol{p}, d) + min \begin{cases} L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d) \\ L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d-1) + P_1 \\ L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, d+1) + P_1 \\ min_{\ i}\ L\boldsymbol{r}'(\boldsymbol{p} - \boldsymbol{r}, i) + P_2 \end{cases} \quad (1)$$

Кількість і розташування шляхів має прямий вплив не тільки на точність узгодження, але також на обчислювальні зусилля і, в цьому випадку, на фактичну архітектуру апаратного прискорювача SGBM.

Метою є компроміс між продуктивністю та точністю. Скорочення від восьми до чотирьох шляхів $0°\,(r = (1,0))$, $45°\,(r = (1,1)), 90°\,(r = (0,1))$ і $135°\,(r = (-1,1))$ призводить до втрати точності лише на 1,7% (збільшення кількості неправильно позначених розбіжностей) у тесті Middlebury, але дозволяє високоефективну апаратну архітектуру обчислень L_r для всіх цих шляхів паралельно [8].

Якщо навіть обмежена втрата точності є неприйнятною, запропоновану апаратну архітектуру можна використати для виконання другого проходу по зображенню, обчислюючи решту шляхів $180°\,(r = (-1,0)), 225°\,(r = (-1,-1)), 270°\,(r = (0,-1))$ і $315°\,(r = (1,-1))$, починаючи з протилежного кута. Це зменшить частоту кадрів для відповідності вдвічі. Оскільки шляхи у вибраному розташуванні більше не розподіляються рівномірно по зображенню, у деяких тестах можна виміряти деякі незначні (неізотропні) ефекти згладжування, але вони не повинні впливати на зручність використання в сценаріях реального світу.

Необроблена вартість $L\boldsymbol{r}'(\boldsymbol{p}, d)$ для узгодження пікселів $\boldsymbol{p}$ уздовж шляху r для припущеної невідповідності d обчислюється за формулою. Ці необроблені витрати на шлях розраховуються для всіх вибраних шляхів, для всіх потенційних розбіжностей d до обмеження D_{max}. Для кожного пікселя оцінюється як локальна вартість C, так і напівглобальний компонент. Остання враховує чотири характеристики, які спостерігаються в реальних зображеннях, мінімальна з яких додається до локальної вартості: перша характеристика – це вартість попереднього пікселя на шляху, другий і третій компоненти штрафують за невелику розбіжність. Зміни $|\Delta d| = 1$ за P_1, тоді як останній член штрафує більші зміни невідповідності (так звані розриви) за P_2. P_1 зазвичай визначається в автономному режимі експериментально шляхом аналізу вхідних зображень, типових для реального випадку використання стереозору. P_2, з іншого боку, динамічно коригується під час виконання: оскільки невідповідність часто також представляє розриви, коли інтенсивність пікселя змінюється, обчислення $P_2 = \frac{P_2'}{|\mathrm{I}_p - \mathrm{I}_{p-r}|}'$ компенсує різні інтенсивності пікселів I_p та I_{p-r} уздовж шляху r. Що стосується P_1, P_2' є константою, визначеною експериментально на основі репрезентативних зразків зображень офлайн. Для подальшого обговорення розрахунку вартості шляху зверніться до оригінальної роботи Гіршмюллера. Щоб визначити (напів) глобальну вартість узгодження, вартість шляху підсумовується по всіх шляхах. Однак для апаратної реалізації варто розглянути дещо змінене формулювання [9].

В апаратному забезпеченні ключовою характеристикою є ширина слова (у бітах) арифметичних операторів і типів даних. Оскільки шляхи проходитимуть по всьому зображенню, вони можуть бути досить довгими (залежно від роздільної здатності камери), і підсумовування їх вартості може призвести до великих значень, які потребують широких слів для обчислення та зберігання. Цьому можна протистояти, віднімаючи від необроблених витрат шляху для пікселя $L\boldsymbol{r}'(\boldsymbol{p}, d)$ мінімальну вартість шляху для всіх припущених розбіжностей d для попереднього пікселя $p - r$ уздовж шляху $\boldsymbol{r}$.

Ефект кодування лише відмінностей між попередніми та поточними пікселями призводить до зменшення величини значень, які вимагають відповідно вужчих слів даних для зберігання та обчислення.

Невідповідність d з мінімальною відповідністю $cost\ arg\ min_d\ S(\boldsymbol{p}, d)$ вважається виграшною диспропорцією для пікселя $\boldsymbol{p}$. Ці виграшні відмінності виводяться прискорювачем для кожного пікселя як вхідні дані для подальшого обчислення фактичної глибини (положення осі Z, тут не обговорюється).

На практиці необхідні додаткові обмеження накладені для очищення викидів і позначення недійсних розбіжностей: результат аргументу $S(\boldsymbol{p}, d)$ може бути багатоелементним набором, що означає, що мінімальна вартість відповідності для пікселя $\boldsymbol{p}$ виникає для різних потенційних розбіжностей d. З такою неунікальною вартістю алгоритм не може визначити одну виграшну невідповідність, а натомість реєструє невідповідність для цього пікселя як «недійсний» [8].

Крім того, виконується так звана перевірка ліворуч/праворуч, яка порівнює результати алгоритму під час його виконання з поміняними ролями базового та відповідного зображень. Цю перевірку також можна ефективно

здійснити (уникаючи перерахунку всіх розбіжностей для попереднього зображення збігу, яке зараз використовується як основа), повторно використовуючи попередньо обчислене $S(\boldsymbol{p}, d)$ уздовж епіполярної лінії як аргумент $S((x(\boldsymbol{p}) + d, y(\boldsymbol{p})), d)$, щоб вибрати виграшну невідповідність d для другого зображення. Перевірка ліворуч/праворуч встановлює для невідповідності значення «недійсне», якщо відповідні невідповідності вихідного проходу та пропуску зі зміненими ролями відрізняються більш ніж на одиницю. Цей крок усуває фантомні диспропорції, що є результатом закритих поверхонь, які видно на одному зображенні, але приховані на іншому.

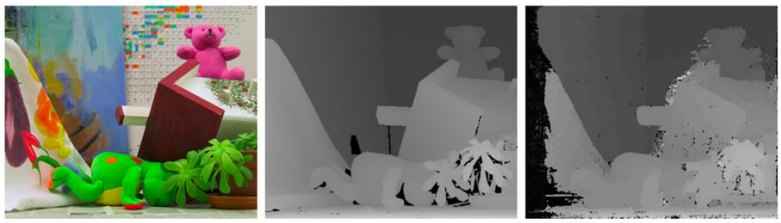
Запропонований метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA уможливлює здійснити синтез архітектуру програмно-технічного засобу, яка підвищує швидкодію систем комп'ютерного зору за рахунок покращення не лише введення додаткового рівня детального паралелізму (наприклад, паралельне обчислення диспропорції та сортування), а й також завдяки реалізації з використанням найсучаснішого стилю розробки, нечутливого до затримок, у наступному – генерація мови опису обладнання. Як результат, він є значно більш масштабованим, легшим для розширення, але також набагато швидшим, ніж оригінальна робота (навіть якщо компенсувати відмінності в цільових технологіях FPGA) [10].

У конструкції з декількома рядковими процесорами обчислені диспропорції буферизуються в FIFO, доки їх не витягне модуль виводу. Потім модуль виводу об'єднує виходи процесорів рядків, використовуючи ту саму циклічну схему, що й модуль введення. Потім медіанний фільтр 3 × 3 застосовується до об'єднаного потоку для видалення викидів у обчислених диспропорціях.

Максимальне значення будь-якого L завжди менше $C_{max} + P_2$. Це обмежує ширину слова, необхідну для зберігання даних і арифметичних операторів в апаратній реалізації [11].

Цей підхід оцінюється на трьох рівнях: першим критерієм є точність, потім модельована незалежна від цілі продуктивність апаратної архітектури в термінах тактових циклів і, нарешті, продуктивність настінного годинника на трьох фактичних платформах FPGA, що охоплюють вбудовану систему та центр обробки даних.

Використовуючи тест Middlebury, алгоритм створює в середньому 8,4 % розбіжностей, що перевищує поріг помилки в один піксель. Різниця між результатами запропонованої архітектури та основною правдою в незакритих областях становить 9,5 % для Конусів, 13,3 % для Тедді, 6,8 % для Цукуби та 4,1 % для Венери. Зразок результату для тестового набору Тедді показано на рисунку 2.



а) б) в)

Рисунок 2 – Порівняння невідповідності для набору зображень Тедді:
а) Набір зображень Тедді, б) Набір зображень Тедді, в) Невідповідність, спричинена зображенням

За потреби більшої точності можна досягти, виконавши повну перевірку ліворуч/праворуч або виконавши два проходи по зображенню для використання восьми шляхів. Обидва ці підходи приблизно вдвічі зменшують продуктивність прискорювача, але навіть тоді він буде корисним для обробки в реальному часі.

Моделювання архітектури з точністю до циклу використовувалося для визначення характеристик часу виконання прикладів реалізацій. Порівняння з фактичними реалізаціями апаратного забезпечення показує, що ці симуляції насправді є репрезентативними для кінцевої продуктивності.

Ядро оцінюється за трьома роздільними здатностями зображення: 640 × 480 пікселів (VGA), 1280 × 720 пікселів (720p) і 1920 × 1080 пікселів (1080p). Зображення з роздільною здатністю VGA оцінюються за Dmax = 64, для вищої роздільної здатності Dmax = 128. Для всіх роздільних здатностей кількість тактів, необхідних для завершення одного кадру, визначається шляхом моделювання [12].

Експертиза містить різні композиції крупно- та дрібнозернистого паралелізму. Реалізація описується парою (#p, #d), яка вказує на використання рядкових процесорів #p, причому кожне обчислення #d передбачає невідповідності паралельно. Для кожної з роздільних здатностей використовується автоматичне дослідження простору розробки для генерації 250 альтернатив реалізації, відображених на осі X у порядку збільшення площі або продуктивності. Через обмеження простору лише підмножину альтернатив можна позначити тут за допомогою (#p, #d). Як показано на рисунку 3 для зображень VGA, архітектура добре масштабується зі збільшенням кількості процесорів рядків, аж до нижньої межі 654644 циклів, після чого один піксель обчислюється за 2,11 циклу, а одна диспропорція вимагає 0,033 циклу. Ця конструкція обмежена швидкістю заповнення вхідних буферів, яку можна збільшити ще більше, застосовуючи також методи дрібного розпаралелювання [13]. При передбачуваній тактовій частоті 200 МГц архітектура досягне до 306 кадрів в секунду, як показано на рисунку 4. Такі великі та швидкі системи можна використовувати для розрахунку розбіжностей між кількома камерами для створення об'ємного огляду сцени. Для додатків з низьким енергоспоживанням більш цікавим є мінімальна частота, необхідна для досягнення 30 кадрів на секунду (типова вимога для обробки в реальному часі) [14]. Як показано на рисунку 5, архітектура здатна задовольнити цю вимогу на частоті до 30 МГц для зображень VGA.

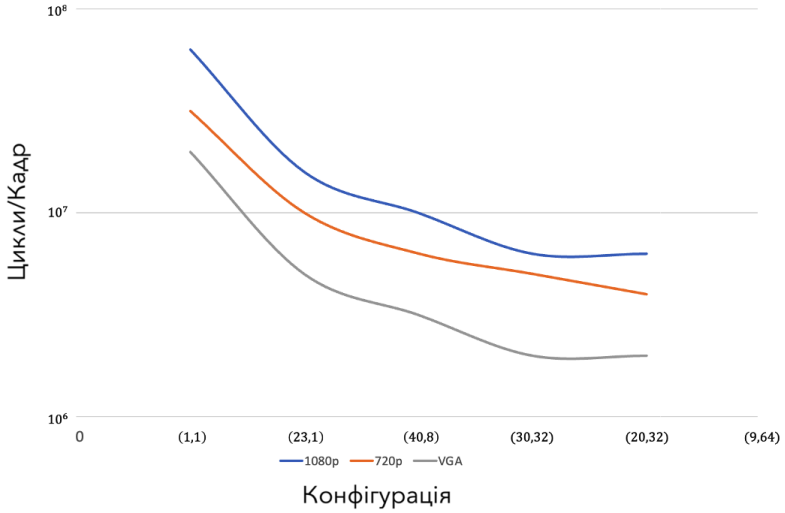


Рисунок 3 – Цикли, необхідні для обробки однієї карти невідповідності для різних ступенів паралельності

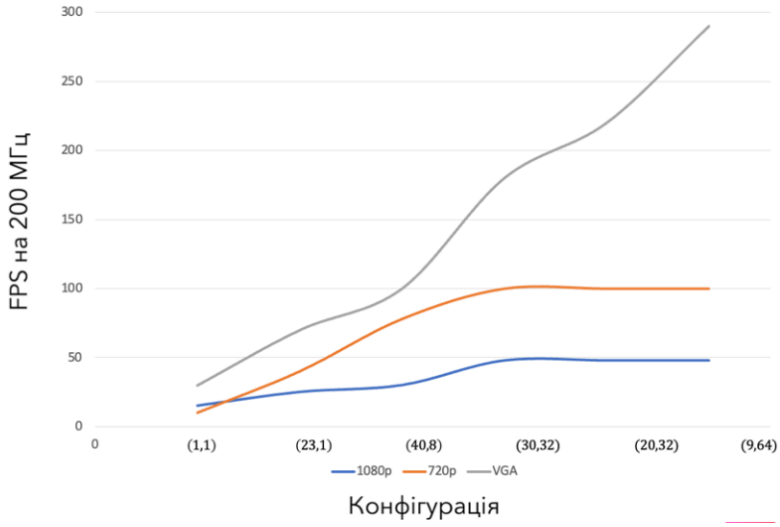


Рисунок 4 – Кількість кадрів в секунду досягається запропонованою архітектурою на тактовій частоті 200 МГц

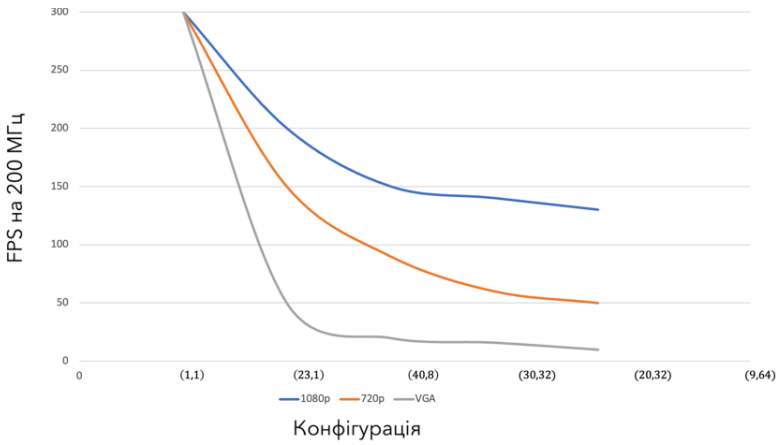


Рисунок 5 – Частота, необхідна для досягнення 30 кадрів на секунду на запропонованій архітектурі для різного ступеня паралелізму

Експериментальні дослідження методу та програмно-апаратна реалізація

Проектування апаратного забезпечення чимось схоже на програмування програмного забезпечення на мові асемблера, у якому простий розрахунок складається з безлічі інструкцій машинного коду, а з іншого боку, багато логічних блоків утворюють єдиний апаратний модуль. Рисунок 6 показує основну концепцію того, як працюють блоки FPGA. Нижче будуть розглянуті всі логічні блоки, які вводяться для реконфігурації часткового алгоритму [15].

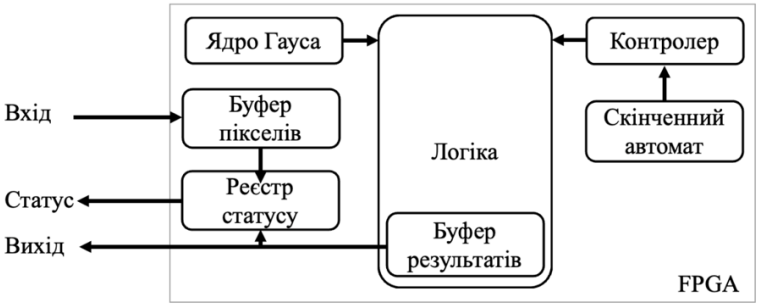


Рисунок 6 – Блок-схема реалізації FPGA

На рисунку 7 показано ядро Гауса, яке використовується для отримання матриці Гарріса A. Цей блок є найпростішим, він не вимагає введення та виведення постійних змінних ядра.

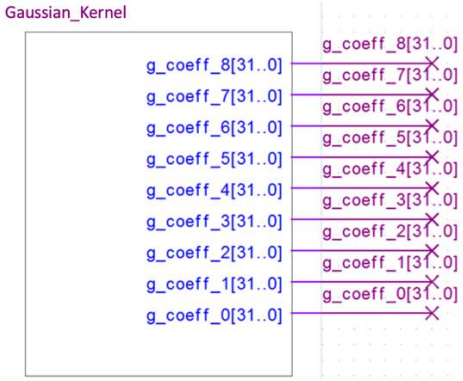


Рисунок 7 – Ядро Гауса

Кожен результат відповідає одному елементу матриці, коефіцієнт 8 представляє перший елемент 0,0001, коефіцієнт 5 є центральним елементом 0,3989 і так далі. Варто зауважити, що кожен параметр є числом з плаваючою точкою, і він отримує 32 біти пам'яті для збереження, тому всі виходи мають 32 біти.

Щоб максимізувати швидкість, було реалізоване апаратне забезпечення якомога паралельнішим. Всі IP-адреси FIFO, надані Altera, є механізмом «один вхід і один вихід», що не зовсім те, чого потрібно досягти, тому FIFO «один вхід і дев'ять виходів» реалізовано, щоб задовольнити потребу, щоб можна було запускати за один прохід цілу згортку з матрицею Гауса 3*3 [16]. Сигнали вхідного FIFO подано в таблиці 1.

Таблиця 1.1 – Сигнали вхідного FIFO

Назва сигналу	Опис
clk	Сигнал годинника
data flag	Перевертається, коли вводяться нові дані
sw	Перемикач, який керує функцією скидання
source[31..0]	Вхідні пікселі, які є 32-розрядним числом типу float
FIFO x[31..0]	Вихідний піксель No.x у парі з відповідним елементом матриці Харріса
done	Встановлюється на високий рівень, коли FIFO заповнений, інакше залишається низьким

Спочатку всі 9 виходів встановлювались на нуль, а дані надсилались через один до дев'яти. Коли надсилаються 10-ті дані, усі вихідні дані отримують змінні в певному порядку, найстаріша розташована внизу, а найновіша – зверху. Для цього FIFO № 8 є верхнім, а № 0 – нижнім.

Коли 1-е дані (1010) отримані після трьох тактів, то в наступному такті найстаріші дані (0001) відкидались, а 10-ті дані додавались до FIFO та зберігались в регістрі № 8, і так далі.

Варто зауважити, що кожного разу, коли надходили нові дані, сигнал прапора даних перевертався, цей механізм гарантував, що система не переплутає два послідовних даних, які мають абсолютно однакове значення [17]. Згортка включає як FIFO, так і ядро Гауса, кожен вихід цих двох блоків було об'єднано в дев'ять різних груп. Оскільки алгоритм методу побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA використовує 32-розрядне число типу float для представлення пікселів, звичайні блоки обчислення не могли бути застосовані тут через складність значення float. Altera надає мегафункцію, яка повністю підтримує обчислення плаваючого значення, що робить її ідеальним вибором під час вибору методу побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA [18] (рис.8).

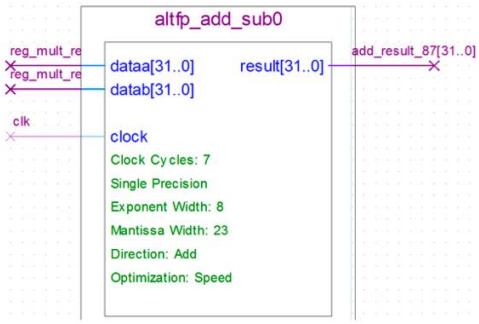


Рисунок 8 – Суматор

На рисунку 8 показано блок додавання після налаштування майстра мегафункцій. Оптимізацію швидкості вибрано, тому що, що є головним золотом дисертації, будь-які функції та можливості, крім чистого додавання, вимкнено з метою економії ресурсів. Такий блок потрібно дублювати багато разів, щоб сформувати функціональний алгоритм. 32-розрядне число з плаваючою точкою означає одинарну точність і має 8-розрядний експонент і також 23-розрядну мантису. Призначення портів досить зрозуміле, і глобальний сигнал clk буде підключений до суматора [19].

Для однієї згортки двох матриць 3*3 реалізовано дев'ять множень і вісім суматорів. Кожен із блоків має період виконання, і останній обчислення може виконуватися лише тоді, коли перший виконає свою роботу. Для надійності всього цього процесу введено кінцевий контейнер.

На рисунку 9 представлена схема кінцевого скінченного автомату. Якщо отриманий сигнал є «високим», тоді спрацьовує арбітр, він починає рахувати всередині та виводить значення на роздільник, розділювач використовує це значення, щоб визначити, яким є поточний стан, і встановити відповідний вихід на «високий» під час налаштування інші на «низький» [20].

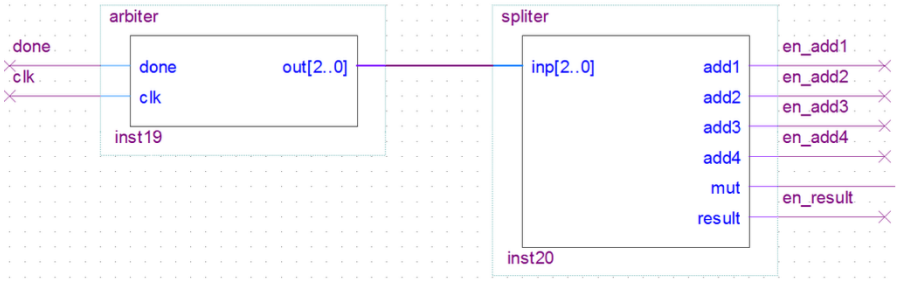


Рисунок 9 – Скінченний автомат

Є чотири регістри додавання та регістр результату. Взявши, регістр add1, входи підключені до попереднього етапу обчислення, незалежно від того, які значення є, регістр завжди зберігатиме те, що він отримує, коли отримано сигнал дозволу «позитивний фронт», вихід порти негайно виведуть дані, які наразі зберігаються, і залишатимуться виводом до наступного «позитивного фронту». Останній етап обчислення, яким буде add2, використовує ці виходи як джерела вхідних даних і передає результат у регістр add2, а процес продовжує виконувати роботу для add3 і add4 [21].

Регістр результату взаємодіє з центральним процесором, який, в свою чергу, є повністю відокремленою системою. Центральний процесор міг запитувати результат кілька разів, навіть якщо було обчислено лише один результат. Через різницю в частоті між процесорами та FPGA, процесори завжди на рівні ГГц, а FPGA – у МГц, що досить повільно порівняно з процесорами [22].

Отже, має бути спосіб захисту кожного запиту, що стосується нових даних. Сигнал прапора повідомляє центральному процесору про те, чи є поточні дані придатними та безпечними для отримання, і це завжди перший сигнал, який буде взаємодіяти в процесі зв’язку між центральним процесором і FPGA, коли процесор отримав «високий» сигнал [23]. Сигнал прапора почне отримувати дані, тим часом результат отримання буде надіслано до FPGA, що вказує на початок процесу. Після отримання результату ЦП знову встановлює «низький» сигнал отримання результату, повідомляє FPGA, що процес завершено. Потім FPGA також встановлює сигнал прапора результату на «низький», щоб, дозволивши ЦП отримати сигнал прапора першим, він міг змусити ЦП чекати, поки не буде отримано новий результат. Такий механізм може значно підвищити надійність системи [24].

Таким чином, цей процес може значно сповільнити швидкість обробки зображень, і поки що це неможливо покращити або уникнути.

Висновки

В роботі представлені результати дослідження, зокрема, розроблено метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA.

В результаті досліджень було встановлено, що метод побудови апаратної архітектури є ефективним підходом до розроблення апаратних прискорювачів для обробки відеоданих.

Даний метод дозволяє значно збільшити швидкість обробки відеоданих, знизити витрати енергії та ресурсів, що забезпечує ефективну інтеграцію з системами розпізнавання образів, навігації, контролю якості та безпеки. Основні положення, що використовуються для розв’язання задач по даній темі, полягають у вивченні особливостей комп’ютерного зору, засобів обробки відеоданих, а також принципів роботи та можливостей FPGA.

Також в роботі подано аспекти практичногго застосування розробленої апаратної архітектури та програмного забезпечення для комп'ютерного зору. Для реалізації систем комп'ютерного зору на основі FPGA використовувались відповідні програмні засоби, що дозволяють створювати програмне забезпечення для FPGA-пристроїв. В роботі представлено програмно-апаратний засіб для систем комп'ютерного зору на основі FPGA, що включає в себе відповідні програмні засоби та апаратну архітектуру. Застосування такого програмно-апаратного засобу дозволяє реалізувати різноманітні завдання обробки зображень та комп'ютерного зору з високою швидкодією та точністю.

Література

1. Donald G. Bailey. Boca Raton. FPGA-Based Implementation of Signal and Image Processing Systems. FL: CRC Press, 2017. 352 p.
2. Dirk Koch. Designing Embedded Systems with FPGAs. Berlin: Springer, 2018. 284 p.
3. Doug Amos, Austin Lesea, and René Richter. FPGA-based Prototyping Methodology Manual: Best Practices in Design-for-Prototyping. New York, NY: Springer, 2018. 340 p.
4. Zhao Zhang and Xiaojun Liu. Hardware Architecture Design for Real-Time Image Processing on FPGA. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5439629/ (дата звернення: 10.04.2023).
5. Georgios Karakonstantis. FPGA-Based Accelerators for Financial Applications. New York, NY: Springer, 2018. 256 p.
6. Tsung-Hsien Lee and Shuvra S. Bhattacharyya. High-Level Synthesis for Real-Time Digital Signal Processing. New York, NY: Springer, 2018. 276 p.
7. Amirhossein Rezaei and Mohamed Zahran. FPGA-Based Acceleration of Deep Convolutional Neural Networks for Computer Vision: A Survey. URL: https://ieeexplore.ieee.org/document/8393873 (дата звернення: 10.03.2023).
8. Yibin Liang, Yiqiang Chen, and Ming Liu. Design and Implementation of a High-Performance FPGA-Based Embedded Computer Vision System. URL: https://ieeexplore.ieee.org/document/7273657 (дата звернення: 10.03.2023).
9. A. Sudhakar, P. Ravinder Reddy, and P. Rajesh Kumar. FPGA Implementation of Image Processing Algorithms: A Review. URL: https://www.sciencedirect.com/science/article/pii/S2212671620303608 (дата звернення: 11.03.2023).
10. Ahmad Ibrahim and Asim El-Sheikh. FPGA-Based Implementation of Neural Networks. New York, NY: Springer, 2018. 220 p.
11. L. M. Bergasa, O. Yebes, and R. Arroyo. An Efficient FPGA Implementation of a Real-Time Object Tracking System. URL: https://ieeexplore.ieee.org/document/6867583 (дата звернення: 11.03.2023).
12. Hamed Habibi Aghdam, Fatemeh Saberian, and Seyed Saeid Moosavi. FPGA-Based Embedded System for Real-Time Object Recognition. URL: https://www.mdpi.com/2079-9292/9/6/917 (дата звернення: 12.03.2023).
13. FPGA-based Embedded Vision System Design. URL: https://www.xilinx.com/products/design-tools/vitis/embedded-vision.html (дата звернення: 12.03.2023).
14. Mark Zwolinski. Designing Digital Systems with SystemVerilog. New York, NY: Springer, 2018. 368 p.
15. Pong P. Chu. Hoboken. FPGA Prototyping by VHDL Examples: Xilinx MicroBlaze MCS SoC". NJ: Wiley-IEEE Press, 2019. 392 p.
16. Mohammad G. Rastegari and Mohammad H. Kahani. FPGA-Based Computer Vision: A Survey. URL: https://ieeexplore.ieee.org/document/8563013 (дата звернення: 10.03.2023).
17. Marek Gorgon and Radim Burget. Designing FPGA-based Computer Vision Systems. URL: https://www.mdpi.com/2079-9292/8/7/724 (дата звернення: 11.03.2023).
18. George A. Constantinides, Peter Y. K. Cheung, and Wayne Luk. Boca Raton. High-Level Synthesis for FPGA Design: From Prototyping to Deployment. FL: CRC Press, 2018. 436 p.

19. Joseph J. Lee and Ayesha Fatima. Deep Learning with FPGA: From Device to Algorithm. Cham, Switzerland: Springer, 2021. 188 p.
20. F. D. B. Pereira and R. M. A. Pereira. FPGA-based Hardware Accelerator for Object Detection in Computer Vision. URL: https://ieeexplore.ieee.org/document/8607876 (дата звернення: 11.03.2023).
21. J. M. Rodriguez-Ramos, J. M. Rodriguez-Delgado, and J. M. Santana. An FPGA-based real-time vision system for detecting and tracking vehicles. URL: https://ieeexplore.ieee.org/document/8525406 (дата звернення: 12.03.2023).
22. Wayne Wolf. FPGA-based System Design. Boca Raton, FL: CRC Press, 2019. 350 p.
23. Pong P. Chu. FPGA Prototyping Using Verilog Examples: Xilinx Spartan-6 Version. Hoboken, NJ: Wiley-IEEE Press, 2018. 352 p.
24. N. C. Kumar, V. Anand, and M. C. Padma. An FPGA-Based Smart Vision System for Object Detection and Tracking. URL: https://ieeexplore.ieee.org/document/8880966 (дата звернення: 14.03.2023).

ДОДАТОК В
(обов'язковий)

ПРЕЗЕНТАЦІЯ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Кафедра комп'ютерної інженерії та інформаційних систем

Атаманюк Ольга

МЕТОД ПОБУДОВИ АПАРАТНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA

Науковий керівник – д.т.н. проф. Лисенко С.М.

Хмельницький - 2023

МЕТА І ЗАДАЧІ ДОСЛІДЖЕННЯ

Метою роботи є підвищення швидкодії систем комп’ютерного зору.

Об'єкт дослідження – комп’ютерних зір.

Предмет дослідження – модель, метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA.

МЕТА І ЗАДАЧІ ДОСЛІДЖЕННЯ

Поставлена мета досягається розв'язанням таких основних **задач**:

- дослідити методи побудови апаратної архітектури для систем комп'ютерного зору;
- проаналізувати сучасні програмно-технічні засоби для систем комп'ютерного зору;
- розробити модель функціонування програмно-технічних засобів комп'ютерного зору на основі FPGA;
- розробити метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA;
- реалізувати метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA.

ПЕРЕДБАЧУВАНА НАУКОВА НОВИЗНА ОТРИМАНИХ РЕЗУЛЬТАТІВ

1) Удосконалено метод побудови апаратної архітектури для систем комп'ютерного зору, який на відміну від відомих використовує FPGA та грунтується на алгоритмі **SGBM**, і який забезпечує високу швидкодію систем комп'ютерного зору.
2) Набули подальшого розвитку програмно-технічні засоби комп'ютерного зору на основі FPGA.

ПРАКТИЧНЕ ЗНАЧЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

В результаті виконаного наукового дослідження було розроблено дослідження буде розроблено апаратно-програмні засоби комп’ютерного зору на основі FPGA.

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

Дослідження методу побудови апаратної архітектури для систем комп'ютерного зору на основі **FPGA** має велике значення для подальшого розвитку комп'ютерного зору та застосування його у різних галузях. Результати таких досліджень можуть сприяти покращенню якості та ефективності роботи систем комп'ютерного зору, що, в свою чергу, дозволить розширити сферу їх застосування.

УДОСКОНАЛЕНИЙ МЕТОД ПОБУДОВИ АПАРАТНОЇ АРХІТЕКТУРИ ДЛЯ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ НА ОСНОВІ FPGA

Удосконалений метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA може ґрунтуватися на методах оптимізації, таких як генетичні алгоритми, а також на методах машинного навчання, таких як нейронні мережі і дерева рішень. Використання таких методів дозволяє підібрати найкращі параметри для апаратної архітектури, що забезпечує найбільшу продуктивність при роботі системи комп'ютерного зору на основі FPGA.

ПУБЛІКАЦІЇ ЗА МАТЕРІАЛАМИ ДИПЛОМНОЇ РОБОТИ

Атаманюк А.В., Лисенко С.М. Метод побудови апаратної архітектури для систем комп’ютерного зору на основі FPGA. Журнал «Вісник Хмельницького національного університету» №2 за 2023 рік.

ВИСНОВКИ

В першому розділі, на основі дослідження алгоритмів та методів комп'ютерного зору, можна зробити висновок, що вони є дуже важливими для створення систем комп'ютерного зору на основі FPGA. Застосування таких алгоритмів та методів дозволяє збільшити швидкість обробки зображень та покращити якість отриманих результатів. Однак, варто зазначити, що для успішної реалізації систем комп'ютерного зору на основі FPGA, потрібні не тільки ефективні алгоритми та методи, але й підходяща апаратна архітектура. В зв'язку з цим, розглянуті засоби реалізації комп'ютерного зору на основі FPGA, є дуже важливими елементами при проектуванні таких систем. Враховуючи висновки з дослідження, можна зробити висновок про необхідність використання оптимальних алгоритмів та методів, а також підходящих засобів реалізації, для побудови ефективної апаратної архітектури для систем комп'ютерного зору на основі FPGA.

ВИСНОВКИ

В другому розділі було розглянуто питання використання FPGA для розроблення апаратних прискорювачів. Було досліджено особливості архітектури FPGA та її переваги порівняно з іншими технологіями. Також було проаналізовано попередні дослідження та проекти, що використовують FPGA для реалізації прискорювачів обробки зображень. Висновки свідчать про те, що FPGA є дуже ефективною технологією для розробки апаратних прискорювачів комп'ютерного зору. Вона надає широкі можливості для оптимізації обчислювальних процесів та зниження часу обробки зображень. Крім того, вона дозволяє розробляти апаратні прискорювачі, що можуть виконувати більш складні завдання порівняно з програмними аналогами.

ВИСНОВКИ

В третьому розділі, в результаті дослідження було встановлено, що метод побудови апаратної архітектури є ефективним і працездатним підходом до розробки апаратних прискорювачів для обробки відеоданих. Даний метод дозволяє значно збільшити швидкість обробки відеоданих, знизити витрати енергії та ресурсів, що забезпечує ефективну інтеграцію з системами розпізнавання образів, навігації, контролю якості та безпеки. Основні положення, що використовуються для розв'язання задач по даній темі, полягають у вивченні особливостей комп'ютерного зору, засобів обробки відеоданих, а також принципів роботи та можливостей FPGA.

ВИСНОВКИ

Четвертий розділ розглядає практичне застосування розробленої апаратної архітектури та програмного забезпечення для комп'ютерного зору. Для реалізації систем комп'ютерного зору на основі FPGA використовувались програмні засоби, що дозволяють створювати програмне забезпечення для FPGA-пристроїв. В розділі було розглянуто програмно-апаратний засіб, що включає в себе відповідні програмні засоби та апаратну архітектуру. Застосування такого програмно-апаратного засобу дозволяє реалізувати різноманітні завдання обробки зображень та комп'ютерного зору з високою швидкодією та точністю.

Отже, використання FPGA як основи для розроблення програмно-апаратних засобів для систем комп'ютерного зору є перспективним та ефективним підходом, що дозволяє отримувати високі результати роботи. Розроблений програмно-апаратний засіб має великий потенціал для застосування в різних галузях, де потрібна висока точність та швидкодія.

UNICHECK

User name:
Кафедра КІ

Check date:
28.04.2023 12:15:59 EEST

Report date:
28.04.2023 12:28:49 EEST

Check ID:
1014845945

Check type:
Doc vs Internet + Library

User ID:
100005591

File name: **Атаманюк_Метод побудови апаратної архітектури для система комп'ютерного зору на основі FPGA**

Page count: **89** Word count: **17127** Character count: **132704** File size: **7.93 MB** File ID: **1014546614**

1.05% Matches

Highest match: **0.44%** with Library source (File ID: **1011224875**)

0.8% Internet sources 63 ..... Page 91

0.32% Library sources 66 ..... Page 91

0.09% Quotes

Quotes 1 ..... Page 92

References 1 ..... Page 92

0% Exclusions

No exclusions

Modifind

Text modifications detected. Find more details in the online report.

Replaced characters 3

Fri Apr 28 11:21:02 EEST 2023, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

The maximum coincidence with one document 24.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 13%**

	Document		Sum coincidence on the DB	
ID: 112692 Title: Метод побудови апаратної архітектури для система комп'ютерного зору на основі FPGA Added in a DB: 2023-04-28 Authors: Атаманюк О.В. Heads: Лисенко С.М. Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	114347	896	27854 (24%)	194 (22%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes
112079	Title: ЗВІТ з науково-дослідної практики "Метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA Added in a DB: 2023-03-21 Authors: О. В. Атаманюк Heads: Бобровнікова К.Ю. Consultants: Opponents:	27220 (24.0%)	183 (20.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач: Атаманюк Ольга Вадимівна

Тема: Метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень _-_; кількість сторінок записки _81_

1. Короткий зміст роботи та прийнятих рішень _У роботі запропоновано метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA

2. Висновок про відповідність роботи дипломному завданню _____ Кваліфікаційна робота магістра відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: _У першому розділі проведено огляд алгоритмів та методів, а також засобів реалізації комп'ютерного зору. Досліджено відомі рішення та засоби в цій сфері. У другому розділі запропоновано FPGA як основу для розроблення апаратних прискорювачів. У третьому розділі запропоновано метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA. У четвертому розділі запропоновано програмно-апаратний засіб для систем комп'ютерного зору на основі FPGA.

4. Позитивні сторони роботи: В результаті виконаного наукового дослідження було розроблено апаратно-програмні засоби комп'ютерного зору на основі FPGA. Удосконалено метод побудови апаратної архітектури для систем комп'ютерного зору, який на відміну від відомих використовує FPGA та ґрунтується на алгоритмі SGBM, і який забезпечує високу швидкодію систем комп'ютерного зору.

5. Негативні сторони роботи: ___В першому розділі роботи не розглянуто апаратні рішення FPGA останнього покоління_____

6. Оцінка графічного оформлення та пояснювальної записки роботи: __=____

_____

_____

_____

_____

_____

7. Відгук про роботу в цілому: __В загальному робота виконана на високому рівні._____

_____

_____

_____

_____

8. Інші зауваження: ___=_____

_____

_____

_____

_____

_____

9. Оцінка кваліфікаційної роботи:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи вважаю, що робота заслуговує оцінки «відмінно» 4,80 (А)_____

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) ___д. т. н. професор, Мартинюк В. В. завідувач кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки Хмельницького національного університету____

"28"квітня 2023 р.

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Атаманюк Ольги Вадимівни
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-21-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

03 травня 2023 року

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Метод побудови апаратної архітектури для систем комп'ютерного зору на основі FPGA»

Автор: Атаманюк Ольга Вадимівна

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Лисенко Сергій Миколайович, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах є збіг зі звітом з науково-дослідної практики автора Атаманюк Ольги «Метод побудови апаратної архітектури для системи комп'ютерного зору на основі FPGA», який було додано в репозиторій ХНУ 21 березня 2023 року;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості Unicheck, складає 1,05% і адресується до 63 першоджерела; а системою Anti-Plagiarism складає 24,0%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ С. М. Лисенко

Гарант ОП _____ О. С. Савенко

Завідувач кафедри КІПСч _____ Т. О. Говорущенко