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

КВАЛІФІКАЦІЙНА РОБОТА

Програмно-технічний засіб прискорювача openCL на FPGA
Назва теми

Рівень вищої освіти *перший (бакалаврський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІ 22050.22.03.16 ПЗ

Виконав здобувач IV курсу, група КІ2-22-3 _____ Віталій ЛЕВУНЕЦЬ
Підпис Ініціали, прізвище

Керівник _____ Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

Нормоконтролер _____ Сергій ЛИСЕНКО
Науковий ступінь, учене звання Підпис Ініціали, прізвище

До захисту допускаю:
завідувач кафедри КІІС
_____ Ольга ПАВЛОВА
підпис Ініціали, прізвище
«04» червня 2026 р.
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ПЕРШИЙ (БАКАЛАВРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 10 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Левунець Віталій Олександрович
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмно-технічний засіб прискорювача openCL на FPGA

Керівник проекту (роботи) Лисенко Сергій Миколайович, д.т.н., проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 27.04.2026 р. № 5

2. Термін подання здобувачем роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Програмно-технічний засіб прискорювача openCL на FPGA та постановка задачі щодо її удосконалення

Проектування системи обробки великих масивів інформації із використанням паралельних обчислень

Програмно-технічна реалізація кіберфізичного засобу прискорювача обчислень на платі FPGA

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

Архітектура ПЗ проєкту

Архітектура ПЗ для кіберфізичної системи

Апаратне забезпечення проєкту

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_10_»_____01_____2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2026	виконано
2	Ознайомлення з цілями завдання, ознайомлення з основами FPGA, створення плану виконання.	01.02.2026	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2026	виконано
4	Робота над розділом 2 – вибір компонентів для проєктування системи адаптивного застосування моніторингових елементів розвідувального	01.04.2026	виконано
5	Робота над розділом 3 – проектування прискорювача на Cyclone V DE10 Standard	29.04.2026	виконано
6	Оформлення пояснювальної записки відповідно до вимог	25.05.2026	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2026 року	

Здобувач_____ Віталій ЛЕВУНЕЦЬ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи_____ Сергій ЛИСЕНКО
Підпис Ім'я, ПРІЗВИЩЕ

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 22050.22.03.16 ПЗ	Пояснювальна записка	56		
			Графічні матеріали			
2		КвРКІ 22050.22.03.16 Е8	Архітектура ПЗ проєкту	1		
3		КвРКІ 22050.22.03.16 Е8	Архітектура ПЗ для	1		
			Прискорювача обчислень			
4		КвРКІ 22050.22.03.16 Е8	Апаратне забезпечення	1		
			проєкту			

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 22050.22.03.16 ВП		
Розробив		Левунець		01.06	Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Лисенко		01.06		У	1	1
Н. контр.		Лисенко		01.06		ХНУ, КІ2-22-3		
Затв.		Павлова		01.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмно-технічний засіб прискорювача обчислень openCL на FPGA».

Автор роботи: Віталій ЛЕВУНЕЦЬ.

Керівник роботи: Сергій ЛИСЕНКО.

Пояснювальна записка: 56 с., 16 рис., 1 табл., 3 дод., 54 джерел.

Графічна частина: 3 креслення.

МНОЖЕНЯ МАТРИЦЬ, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, ПЛІС, , ПРИСКОРЮВАЧ, ПРОГРАМНО ТЕХНІЧНИЙ ЗАСІБ.

Кваліфікаційна робота бакалавра присвячена розробці та дослідженню Програмно-технічного засобу що реалізує обчислювальну систему яка прискорює комп'ютерну швидкість обробки великих масивів даних, та проведення математичних обчислень, на базі програмованої логічної інтегральної схеми.

Програмно-технічні засоби, поєднують високорівневий опис алгоритмів із апаратною реалізацією. Завдяки своїй гнучкості вони дозволяють адаптувати архітектуру під конкретні потреби користувача, забезпечуючи оптимальну продуктивність. Ефективність таких засобів досягається можливість зміни прошивки що дозволяє налаштувати плату працювати саме з конкретною задачею, крім того, можливість паралельної обробки даних на рівні бітів забезпечує значне скорочення часу виконання складних алгоритмів.

Метою роботи є розробка, та тестування програмно-технічного засобу що візьме на себе прості обчислення та великі об'єми даних звільняючи обчислювальні можливості основного комп'ютера для виконання складніших задач це дозволить прискорити обробку інформації та швидкість її обчислень.

Підпис здобувача

30.05.2026

Дата

ЗМІСТ

ВСТУП

У сучасному світі доводиться обчислювати великі обсяги інформації які з року в рік постійно збільшуються. Для наукових досліджень, Інтернету речей, Фінансові обчислення та інші. Зростає потреба у складних обчисленнях та швидкій і ефективній роботі із великими обсягами даних. Традиційні Центральні процесори та графічні прискорювачі не завжди можуть забезпечити високу продуктивність паралельних обчислень та низьке енергоспоживання для складних обчислень.

Вирішенням цієї проблеми є програмно технічний засіб FPGA. Головною перевагою є можливість перепрограмувати плату багато разів навіть після впровадження в кінцевий продукт. Це дозволяє оновлювати алгоритми, виправляти помилки або додавати нові функції без заміни пристрою, що дозволяє створювати конфігурації що спеціально оптимізовані для проведення паралельних обчислень які забезпечать вищу швидкість ніж у CPU. Також завдяки конфігурації що не виконує надлишкових дій, що зменшує енергоспоживання більше ніж у GPU.[8]

Мій проект спрямований на створення програмно-технічного засобу прискорювача OpenCL на FPGA, який виступає рішенням для сучасних задач обробки великих обсягів даних, який дозволяє проводити паралельні обчислення описані мовою OpenCL-C та через відповідні API.

1 АНАЛІЗ ПРОГРАМНО-ТЕХНІЧНИХ ЗАСОБІВ ПРИСКОРЮВАЧА ТА ПРИНЦИПУ ЇХ РОБОТИ НА ОСНОВІ FPGA

1.1 Проблема необхідності у програмно технічних засобах

Цифровізація сьогодні постає не просто як технологічний прогрес, а як одна з найбільш фундаментальних та визначальних тенденцій розвитку суспільства та світової економіки. Цей процес є всеохоплюючим, оскільки він інтегрується у найрізноманітніші пласти людського життя, від повсякденних побутових інструментів та персональних мобільних застосунків до складних промислових комплексів, високотехнологічних транспортних систем та розгалужених структур державного управління. Стратегічною метою цифровізації є впровадження інновацій, та системне розповсюдження адаптації цифрових технологій у ключових сферах життєдіяльності, що кардинально змінює підходи до ведення бізнесу, методологію освіти, принципи адміністрування та якість щоденного побуту кожної особистості.[11]

У сучасному секторі послуг, комунікацій та розваг цифрові рішення виступають інструментом подолання фізичних обмежень. Завдяки їм з'являється унікальна можливість здійснювати професійну та особисту діяльність незалежно від географічного розташування суб'єктів. Відеоконференції високої чіткості, функціонування глобальних маркетплейсів для миттєвої купівлі товарів першої потреби та розвиток інтерактивних месенджерів перетворюють світ на єдиний цифровий простір. Окрім комерційного аспекту, цифровізація відіграє критичну роль у вирішенні глобальних соціальних викликів. Вона забезпечує демократичний доступ до життєво необхідних сервісів: створення інтелектуальних систем електронної охорони здоров'я дозволяє проводити дистанційну діагностику та моніторинг; впровадження платформ для дистанційного навчання робить якісну освіту доступною у найвіддаленіших регіонах; а фінансові фінтех-сервіси забезпечують швидкість та безпеку транзакцій.

Це глибока трансформація культури та мислення, що передбачає повну інтеграцію інтелектуальних рішень у стратегію розвитку будь-якої структури. Цей процес супроводжується відчутним економічним ефектом перехід на електронний документообіг мінімізує операційні витрати та бюрократичне навантаження, а використання аналітики великих даних дозволяє оптимізувати виробничі цикли та децентралізувати виробництво. Цифрові рішення надають можливість приймати рішення в режимі реального часу, спираючись на актуальні показники, а не на застарілу статистику. це означає перехід до створення екологічно сталих та інтелектуальних продуктів, суттєве скорочення термінів розробки, підвищення контролю якості на кожному етапі та здатність бізнесу майже миттєво адаптуватися до динамічних коливань ринкової кон'юнктури.

Проте для підтримання стрімкого темпу постійно зростаючого рівня глобальної цифровізації та достатню кількість обчислювальних потужностей і життєздатності сучасних інформаційних екосистем виникає, потреба у надшвидкій обробці колосальних масивів даних. Вирішення складних обчислювальних задач, що виникають у процесі цифрової трансформації, вимагає принципово нових підходів до архітектури систем обробки інформації. Робота з великими масивами неструктурованих даних, виконання криптографічних перетворень у реальному часі або аналіз потокового відео високої чіткості вимагають залучення надзвичайно потужних та енергоефективних обчислювальних ресурсів.

Традиційно одним із базових рішень для задоволення цих потреб вважається нарощування потужностей шляхом експлуатації високопродуктивних персональних комп’ютерів, розгортання масивних хмарних дата центрів та використання багатопроцесорних серверних ферм. Однак, незважаючи на їхню значну потужність, такі системи далеко не завжди демонструють належну ефективність, особливо в умовах обмежених енергетичних бюджетів або жорстких вимог до габаритів пристроїв.

Головним недоліком універсальних процесорів CPU та графічних прискорювачів GPU є їхня фіксована архітектура, яка орієнтована на широкий спектр загальних завдань, що призводить до зайвих витрат ресурсів при виконанні вузькоспеціалізованих операцій.

У сучасних умовах значення набуває не тільки універсальність підходу, а здатність до динамічної зміни пристрою безпосередньо під час конкретного завдання. Виникає потреба в архітектурній модульності, яка дозволяє не просто виконувати код, а фізично змінювати структуру обчислювача на апаратному рівні. Це дозволить об’єднати пристрої з іншими сумісними системами, створюючи цілісні програмно технічні комплекси, адаптовані під унікальні вимоги кожної конкретної задачі.

Рішенням цієї проблеми стали програмно-технічні засоби, які поєднують високорівневий опис алгоритмів із апаратною реалізацією. Завдяки своїй гнучкості вони дозволяють адаптувати архітектуру під конкретні потреби користувача, забезпечуючи оптимальну продуктивність у вузькоспеціалізованих задачах. Енергоефективність таких засобів досягається завдяки тому, що вони виконують лише необхідні операції без зайвих обчислень, що особливо важливо у сучасних умовах зростання енергоспоживання дата-центрів. Крім того, можливість паралельної обробки даних на рівні бітів забезпечує значне скорочення часу виконання складних алгоритмів.

Програмно-технічний засіб - це комплекс що складається з двох частин програмної та технічної. Програмна частина стосується інформації, команд та процесів з якими працює обчислювальна машина вона складається з алгоритмів, інструкцій, программного забезпечення, яке визначає, як саме має працювати система, як зберігати дані, які дії виконувати, та як реагувати на зовнішні сигнали. Технічна система це фізична основа що відповідає за зв'язок з іншими пристроями канали зв’язку, джерела живлення та інтерфейси за допомогою яких користувач взаємодіє з системою.

Вона включає в себе: датчики, контролери, електронні плати, механізми чи інші пристрої, які фізично реалізують роботу системи. Поєднання інформаційної та фізичної частин дозволяє створити інструмент що може вирішувати різні практичні завдання

Основними перевагами програмно-технічних засобів є їхня комплексність, адже вони не обмежуються лише фізичними можливостями пристроїв, а отримують гнучкість і функціональність, які забезпечує програмне забезпечення. Такий підхід дозволяє адаптувати одну й ту саму апаратну платформу до різних сфер застосування, змінюючи лише програмну частину. Програмно-технічні засоби відзначаються універсальністю, оскільки можуть працювати як у промислових системах керування, так і в побутових пристроях чи високопродуктивних обчислювальних комплексах. Також важливою перевагою є масштабованість: вони легко інтегруються у більші інформаційні системи, підтримують взаємодію з мережею та іншими пристроями, що робить їх придатними для сучасних технологій, таких як Інтернет речей чи хмарні обчислення. Крім того, програмно-технічні засоби забезпечують енергоефективність та оптимізацію ресурсів, адже програмна складова дозволяє змінити конфігурацію пристрою щоб раціонально розподіляти навантаження між апаратними компонентами.

Перспективи розвитку програмно-технічних засобів визначаються сучасними тенденціями у сфері інформаційних технологій та обчислювальних систем. З кожним роком зростає потреба у більшій продуктивності, енергоефективності та гнучкості, що стимулює інтеграцію апаратних і програмних рішень у єдині комплекси. наступним ключовим вектором розвитку є застосування у сфері штучного інтелекту та машинного навчання, де програмно-технічні засоби стають основою для реалізації складних алгоритмів аналізу даних. Не менш важливим є їхнє використання в Інтернеті речей, де потрібна компактність, енергоефективність та здатність працювати у реальному часі.

1.2 Аналіз видів прискорювачів обчислень та можливих типів створення

При необхідності виконати велику кількість однотипних операцій, таких як обробка масивів, матричні обчислення, аналіз сигналів чи навчання моделей машинного навчання, класичні процесори виконують з низкою обмежень, вони мають обмежену паралельність, адже кількість потоків команд які може виконувати залежить від кількості ядер що не дозволяє ефективно виконувати тисячі однотипних операцій одночасно. Високе енергоспоживання, зумовлене масовим виконання повторюваних обчислень призводить до значних витрат енергії, що критично для дата-центрів та вбудованих систем; демонструють затримки при масштабуванні, адже зі збільшенням обсягу даних час обробки зростає непропорційно, що робить систему менш придатною для роботи в реальному часі; а також є неефективними для спеціалізованих задач, де операції з великими матрицями, обробка графів чи алгоритми глибинного навчання виконуються значно повільніше, ніж на архітектурах, оптимізованих під паралельні обчислення.

Прискорювачі обчислень, беруть на себе виконання масових та однотипних нескладних операцій, звільняючи центральні процесори для більш складних і багаторівневих завдань. Завдяки архітектурі, оптимізованій під паралельні обчислення, такі пристрої як GPU, FPGA чи ASIC забезпечують велику продуктивність та значне скорочення часу обробки великих масивів даних, зменшують енергетичні затрати системи завдяки апаратній оптимізації та дозволяють працювати з високою швидкістю у реальному часі. Після завершення обробки інформації результати повертаються назад у пам’ять компютера. Такий підхід дозволяє суттєво зменшити навантаження на процесор і підвищити загальну продуктивність системи та довговічність так як процесор буде працювати не на повну потужність, і менше нагріватись. Приклад прискорювача реалізованого на FPGA зображено на рисунку 1.1.

Рисунок 1.1 – приклад FPGA прискорювача з чіпом Intel Altera [52]

Існує кілька основних видів прискорювачів обчислень, кожен з яких був сконструйований по різним принципам та для різних ситуацій які дозволяють ефективно працювати на конкретні типи задач. Кожен вид прискорювачів завдяки своїм особливостям має свої переваги та недоліки які можуть підходити для різних умов, під час вивчення принципу роботи прискорювачів було обрано декілька самих розповсюджених типів.

Graphics Processing Unit - Графічні спроектований для надзвичайно швидкої маніпуляції та зміни пам'яті з метою прискорення створення зображень у буферу кадру та для рендерингу відео, але їх архітектура з тисячами паралельних потоків виявилася ідеальною для масових обчислень. Центральний процесор здатний одночасно виконувати величезну кількість однотипних операцій, що робить їх гарним рішенням у задачах машинного навчання, глибинних та нейронних мереж, обробки великих масивів даних та наукових симуляцій. Вони забезпечують значне прискорення порівняно з CPU, особливо у випадках, коли потрібно виконати матричні множення чи інші лінійно-алгебраїчні операції.

Application-Specific Integrated Circuit - це спеціалізовані інтегральні схеми, розроблені для виконання вузького набору завдань. Вони не мають гнучкості як у FPGA, але натомість забезпечують максимальну швидкість та мінімальне енергоспоживання. ASIC широко застосовуються у, мобільних пристроях, мережевому обладнанні та інших сферах, де потрібна висока ефективність при виконанні конкретних алгоритмів. Їхня головна перевага — оптимізація «під одне завдання», що робить їх надзвичайно продуктивними.

Field-Programmable Gate Array - це програмовані логічні матриці, які дозволяють створювати апаратні конфігурації під конкретні алгоритми. Інженер може перепрограмувати FPGA для виконання різних завдань, оптимізуючи архітектуру під конкретні обчислення.

Tensor Processing Unit - це спеціалізовані процесори, створені компанією Google для роботи з тензорними операціями, які є основою сучасних нейронних мереж. Вони оптимізовані для виконання операцій множення матриць та тензорів, що дозволяє прискорити навчання моделей штучного інтелекту. TPU використовуються у дата-центрах для обробки великих обсягів даних та забезпечують високу продуктивність при відносно низькому енергоспоживанні. Їхня архітектура спеціально адаптована під потреби машинного навчання, що робить їх одним із ключових інструментів у розвитку штучного інтелекту.

У даній дипломній роботі буде спроектовано та реалізовано програмно-технічний засіб прискорювача обчислень використовуючи в якості онови саме Terasic Altera Cyclone V DE10 Standard, який за допомогою свого поєднання HPS модуля керування обчислювальними потужностями та FPGA частини що буде переконфігурована для виконання матиматичних обчислень, що допоможе ефективніше використовувати обчислювальні потужності компютера до якого підключений прискорювач, для виконання паралельних матиматичних обчислень над простими числами та прискорить виконання задач.

1.3 Пояснення FPGA та принцип його роботи

Field Programmable Gate Array - це напівпровідниковий пристрій що виступає гнучкою платформою, яку можна адаптувати під різні задачі безпосередньо в процесі експлуатації. Головною перевагою FPGA є можливість повної зміни внутрішньої конфігурації пристрою, що дозволяє динамічно оновлювати апаратну частину системи відповідно до нових вимог чи стандартів. Якщо виникає потреба у зміні функціоналу, достатньо лише оновити прошивку, без необхідності фізичного втручання у пристрій. Крім того, FPGA вирізняються високим рівнем модульності, що дозволяє підключати різні види пристроїв, FPGA мають розвинену систему інтерфейсів вводу-виводу, що дає змогу підключатися до різноманітних зовнішніх пристроїв. Вони підтримують стандартні цифрові інтерфейси SPI, UART, PCIe, Ethernet, HDMI, LVDS, що дозволяє інтегрувати їх із мікроконтролерами, процесорами, датчиками, пам’яттю, та іншими компонентами. У багатьох випадках FPGA можуть виступати як центральний вузол системи, координуючи обмін даними між кількома пристроями одночасно що дає змогу інтегрувати його у більші системи та поєднувати з іншими пристроями для виконання конкретних завдань.

Відміність FPGA між центральним процесором у тому що звичайний процесор після завершення виробництва має фіксовану структуру яка не може бути змінена та виконує команди послідовно за допомогою універсальних блоків, то FPGA складається з великої кількості логічних елементів і комутаційних ліній, які можна перепрограмувати так, щоб вони утворювали потрібну апаратну схему для оптимального виконання задачі. Можливість перебудовувати внутрішню архітектуру під конкретні алгоритми, забезпечують паралельність виконання операцій. Коли інші програмно-технічні засоби навіть багатоядерні виконують команди чергою що сповільнює роботу коли велика кількість команд має виконуватись послідовно

FPGA здатні виконувати команди паралельно що дозволяє одночасно обробляти велику кількість потоків даних, що забезпечує високу продуктивність у задачах реального часу. На рисунку 1.2 зображено приклад плати SoC altera з FPGA.

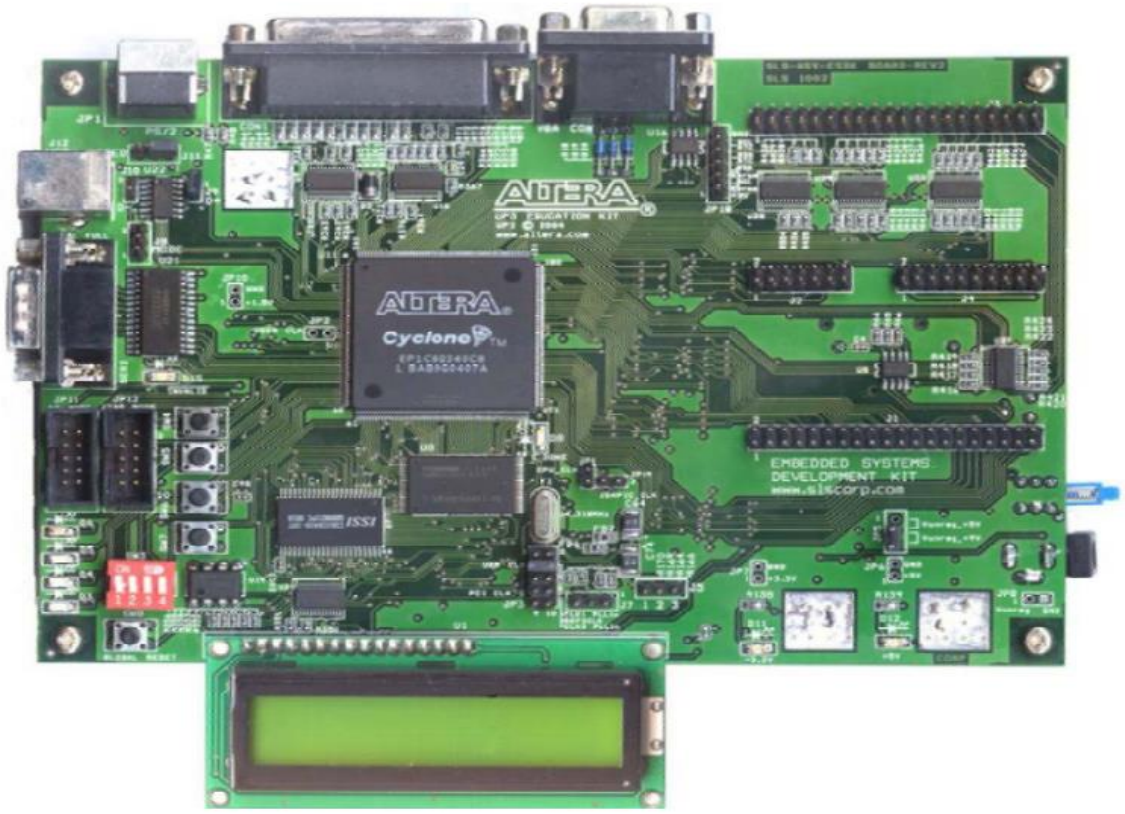


Рисунок 1.2 – Плата з FPGA у кристалі Cyclone [53]

Архітектура FPGA плат базується на кристалі SoC, яка інтегрує майже всі компоненти комп'ютера на одному напівпровідниковому чіпі. Ключовою особливістю є тісна інтеграція логічної матриці з апаратною процесорною системою. Кристал забезпечує внутрішню взаємодію між центральним процесором та матрицею програмованої логіки за допомогою високошвидкісних сполучних з’єднань. Ця архітектура дозволяє об'єднувати обчислювальні потужності в єдину систему, де передача даних між апаратною частиною та програмним середовищем відбувається з мінімальними затримками.

Важливим напрямом є машинне навчання та штучний інтелект, де FPGA використовуються для прискорення нейронних мереж, комп'ютерного зору та аналізу великих масивів даних, забезпечуючи баланс між продуктивністю та енергоефективністю. У наукових дослідженнях вони допомагають моделювати фізичні процеси та вирішувати задачі, що потребують високої точності та паралельних обчислень. Крім того, FPGA знаходять застосування у робототехніці, де потрібна швидка реакція систем у реальному часі, а також у побутових пристроях та IoT рішеннях, де важлива компактність і низьке енергоспоживання. Таким чином, сфери використання FPGA охоплюють як високотехнологічні галузі, так і повсякденні застосування, що робить їх універсальним інструментом сучасних програмно технічних засобів.[12-14]

Принцип роботи FPGA базується на трьох основних елементах програмовані логічні блоки CLB - Configurable logic blocks, матриці зв'язків Interconnections matrix та інтерфейсні блоки для вводу і виводу Input Output blocks та елементів маршрутизації. Основою для будь яких логічних блоків є логічні елементи та регістри памяті. Логічні елементи бувають типів AND, NOT, OR, XOR, та їх інвертовані версії. Для зберігання інформації використовують регістри що при отримані команди на запис зберігають значення на вході, аналогічно для читання. Використовуючи велику кількість логічних елементів можна будувати складні логічні схеми які обробляють інформацію на вході і виводять результат на виході, після чого інформацію можна буде зберегти у памяті.

Для прикладу розглянемо просту базову операцію логічного множення AND над двома операндами A та B, результат якої записується у змінну X мовою OpenCL C. На рівні вихідного коду це виглядає як проста інструкція, однак у випадку FPGA її виконання суттєво відрізняється від класичної процесорної моделі

У традиційній архітектурі центрального процесора виконання операції побітового додавання або порівняння є незміним послідовним циклом інструкцій, де арифметико-логічний пристрій виступає універсальним елементом із незміною архітектурою. Процесор змушений витрачати ресурси на вибірку команди, її декодування та звернення до регістрового файлу, що створює значні витрати енергії та часу на кожну окрему операцію. На відміну від цього, реалізація на базі програмованої схеми за допомогою фреймворку OpenCL дозволяє не виконувати обмежені команди закладені виробником і створити власні потокової обробки даних на рівні топології напівпровідника.

Спеціалізований компілятор виконує високорівневий синтез, перетворюючи алгоритмічний опис на мові програмування у розгалужену мережу взаємопов’язаних апаратних вузлів CLB. Замість того, щоб команди виконувалися послідовно на універсальному обчислювачі, компілятор створює в структурі програмованої схеми виділений конвеєр, де кожен логічний крок алгоритму має власну фізичну ділянку на кристалі. Конфігураційний файл, що завантажується у систему, фактично перепрограмує статичну пам'ять, перетворюючи їх на спеціалізовані апаратні логічні вентилі, що ідеально відповідають структурі коду.

Процес фізичної конфігурації ресурсів мікросхеми перетворює матрицю логічних блоків на унікальний обчислювальний пристрій, де внутрішня мережа програмованих з’єднань виконує роль гнучкої системної шини. Коли операнди надходять із пам’яті, команди відбуваються одночасно, і безпосередньо проходять крізь сформовану апаратну логіку, що забезпечує виконання операцій за мінімальну кількість тактів. Така архітектура дозволяє досягти масивного паралелізму, оскільки в одному циклі роботи системи можуть одночасно оброблятися тисячі незалежних потоків даних, що фізично неможливо для стандартних процесорів з послідовною логікою виконання. Дана архітектура зображенана рисунку 1.3.

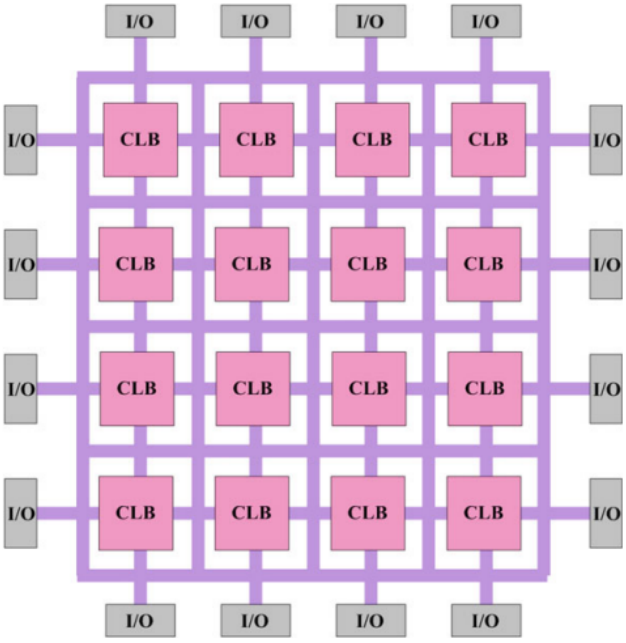


Рисунок 1.3 - базова архітектура FPGA[5]

Процес обробки інформації всередині Архітектури конфігурованого логічного блоку CLB зображено на рисунку 1.4 обчислення розпочинається на етапі надходження цифрових сигналів із глобальної мережі з'єднань до вхідних каскадів вузла. Первинну роль у розподілі цих потоків відіграє блок декодування та керування, який на основі завантаженого бітстріму визначає маршрутизацію сигналів до конкретних обчислювальних ресурсів. Дані потрапляють на вхідні порти таблиць пошуку LUT, де відбувається основна стадія комбінаційної обробки. Кожна таблиця виконує роль апаратної моделі логічної функції, миттєво генеруючи результат на основі попередньо записаних значень істинності, що дозволяє перетворювати вхідні вектори операндів на проміжні обчислювальні результати.[2]

Після первинної обробки в LUT 1 та LUT 2 сигнали спрямовуються до системи проміжних мультиплексорів, які виконують роль динамічних комутаторів. На цьому етапі архітектура дозволяє реалізувати багаторівневу логіку: виходи перших таблиць можуть стати вхідними даними для наступних клкментів, або комбінуватися з прямими сигналами з шин даних.

Таке каскадування забезпечує можливість побудови складних функцій та арифметичних операцій, де результат одного вузла безпосередньо впливає на обчислення в наступному без виходу сигналу в загальну комутаційну матрицю кристала, що мінімізує часові затримки.

На завершальному етапі сформовані потоки даних проходять крізь блоки керування скиданням та вихідної інверсії де відбувається корекція сигналу відповідно до алгоритмічних вимог, включаючи можливість синхронізації з тактовим генератором або примусового обнулення стану. Кінцевий результат через вихідні порти повертається у мережу програмованих з'єднань.[1]

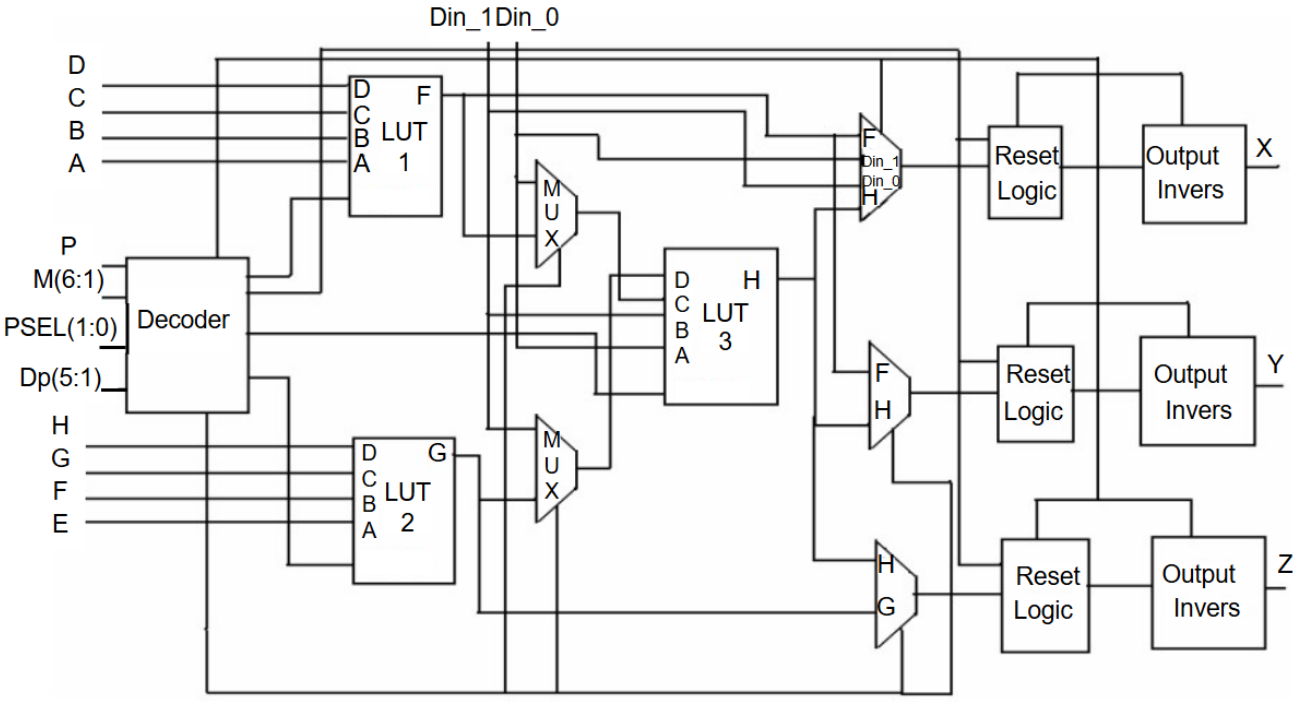


Рисунок 1.4 - архітектура CLB[23]

На вході дані потрапляють у LUT який зображено на рисунку 1.5, він містить таблицю істинності для операції AND. LUT це невелика пам'ять, що зберігає таблицю істинності певної логічної функції. Для операції AND таблиця істинності містить вісім комбінацій входів, що працюють як адреса у пам'яті для потрібного результату.

Коли на входи LUT подаються вхідні значення, вони використовуються як адреса цієї пам’яті, а на виході подається інформація що відповідає роботі логічного елементу AND з’являється відповідне значення. Таким чином, LUT фактично реалізує будь-яку логічну функцію без потреби у фізичній зміні транзисторної структури достатньо лише змінити її вміст під час конфігурації.

Вхід			Вихід	
A	B	Cin	Sim	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

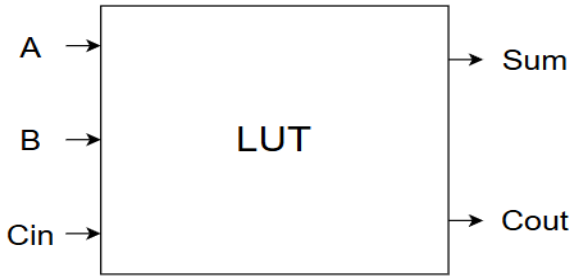


Рисунок 1.5 - таблиця істинності яку зберігає LUT

Коли обчислювальний цикл усередині логічного вузла завершується, сформований результат залишає CLB через вихідні порти і потрапляє у глобальну матрицю програмованих з’єднань. На цьому етапі цифрові дані перестають бути абстрактною логічною операцією і перетворюються на стабільний електричний потенціал, який транспортується через розгалужену мережу металевих доріжок кристала. Завдяки заздалегідь конфігурованим транзисторним ключам у вузлах комутації, цей сигнал спрямовується до своєї кінцевої точки призначення з мінімальними затримками розповсюдження.

1.4 Пояснення завдання дипломної роботи

Головною метою даної дипломної роботи є проектування, реалізація та дослідження програмно-технічного засобу прискорювача обчислень на базі FPGA з підтримкою стандарту OpenCL. У межах роботи передбачається комплексне вивчення принципів функціонування програмованих логічних інтегральних схем, особливостей їх конфігурації та методів програмування з використанням високорівневих засобів опису паралельних алгоритмів. Апаратною платформою для реалізації обрано відлагоджувальну плату DE10-Standard, що побудована на базі мікросхеми Intel Cyclone V.

Розроблюваний програмно-технічний засіб орієнтований на виконання різних теоретичних завдань як масивно-паралельних обчислень над великими обсягами даних, зокрема операцій над векторами та матрицями. Такі операції є фундаментальними для алгоритмів цифрової обробки сигналів. Математичні перетворення потребують одночасної обробки значної кількості елементів даних. Традиційні процесорні архітектури загального призначення, незважаючи на високу тактову частоту та багато ядерність, не завжди забезпечують достатній рівень продуктивності та енергоефективності при виконанні подібних задач через обмеження послідовної моделі виконання та універсальність архітектури.

У рамках роботи передбачається перенесення масивних та непотребуючих складних обчислень операцій із центрального процесора до програмованої логіки FPGA. На відміну від класичного процесора, який виконує інструкції переважно послідовно, FPGA дозволяє реалізувати справжню апаратну паралельність шляхом створення спеціалізованих обчислювальних блоків, що працюють одночасно. Такий підхід дає можливість значно скоротити час виконання алгоритмів, мінімізувати затримки та підвищити пропускну здатність системи. Крім того, апаратна реалізація конкретного алгоритму дозволяє оптимізувати структуру обчислень під специфіку задачі, усуваючи зайві операції та підвищуючи енергоефективність.

Програмно-технічний засіб реалізує апаратні обчислювальні пристрої, які підтримують стандарт OpenCL і можуть виконувати паралельні обчислення, описані мовою OpenCL C через відповідні API. Використання OpenCL забезпечує переносимість алгоритмів і спрощує розробку, оскільки дозволяє описувати паралельні ядра на високому рівні абстракції без необхідності детального проєктування на рівні регістрових передач. Водночас компілятор формує апаратну реалізацію обчислювальних ядер, які інтегруються в логічну структуру FPGA. Особлива увага приділяється побудові апаратних конвеєрів, організації локальної та глобальної пам’яті, а також оптимізації обміну даними між хост-процесором і FPGA через відповідні інтерфейси.

Важливим етапом роботи є моделювання, синтез та тестування розробленої системи. Тестування продуктивності є ключовим етапом для перевірки роботи прискорювача. Аналізуються показники пропускної здатності, затримки доступу до пам’яті, ефективність використання логічних ресурсів, та стабільність роботи при тривалому навантаженні.

Розроблюваний прискорювач повинен відповідати базовим вимогам ефективності, серед яких забезпечення високого рівня паралельних обчислень, оптимальне використання енергоресурсів та модульність архітектури.

Виконання даної роботи дозволить набути комплексних теоретичних знань і практичних навичок у сфері проєктування програмно-технічних засобів на основі FPGA, засвоїти принципи їх конфігурації та адаптації до конкретних прикладних задач. У процесі розробки буде отримано досвід створення масивно-паралельних обчислювальних систем, оптимізації архітектури FPGA та інтеграції апаратних модулів із програмним середовищем. Таким чином, результатом роботи стане функціональна система прискорення обчислень OpenCL на платформі FPGA, що демонструє переваги апаратної паралельної обробки даних порівняно з традиційними архітектурами загального призначення.

1.5 Висновки першого розділу

У результаті аналізу зібраної інфомації в першому розділі було обґрунтовано потребу у обчислювальних пристроїв як головного рушія трансформації сучасних економічних та соціальних процесів. Встановлено, що стрімке впровадження цифрових технологій у всі сфери людського життя генерує колосальні обсяги даних, обробка яких вимагає принципово нових підходів до архітектури обчислювальних систем.

Дослідження існуючих видів прискорювачів обчислень показало, що традиційні центральні процесори поступово втрачають ефективність при потребі у виконанні масових паралельних завдань, таких як матричні обчислення чи навчання нейронних мереж, через обмежену кількість фізичних ядер та високе питоме енергоспоживання. На противагу їм, технологія FPGA визначена як найбільш адаптивна та перспективна платформа. Головною перевагою якого є можливість формування унікальної апаратної конфігурації безпосередньо під топологію конкретного алгоритму, що забезпечує максимальну продуктивність при високій енергоефективності.

На основі проведеного аналізу було сформульовано та деталізовано завдання дипломної роботи. Визначено, що найбільш раціональним шляхом реалізації сучасного прискорювача обчислень є використання FPGA у поєднанні зі стандартом високорівневого опису для полегшення створення прискорювача. Такий підхід дозволяє нівелювати складність низькорівневого проектування, зосередившись на оптимізації алгоритмічної частини, для створення оптимізованого прискорювача обчислень.

2 ПРОЄКТУВАННЯ СИСТЕМИ ПРИСКОРЮВАЧА

2.1 Ознайомлення з можливостями і функціоналом платформи Terasic DE10-Standard

Сучасні потреби у гнучких обчислювальних машин у сфері напівпровідникових технологій дозволили створити плату Terasic DE10-Standard що є універсальною апаратною платформою, призначеною для розробки та конструювання високопродуктивних вбудованих систем. Ця плата використовується як інженерний інструмент, що об'єднує потужні обчислювальні ресурси з розширеним набором периферійних інтерфейсів для взаємодії з навколишнім середовищем.

Конструктивно плата спроектована таким чином, щоб забезпечити максимальну гнучкість при розробці проектів різного ступеня складності. Основна увага в архітектурі DE10-Standard приділена створенню надійного зв'язку між обчислювальним ядром та зовнішніми пристроями. Для цього функціоналу передбачено Cyclone V SoC апаратний процесорний блок на базі двоядерної архітектури ARM Cortex-A9 який є базовим компонентом, що функціонує як повноцінний комп’ютерний вузол, здатний працювати незалежно від ресурсів FPGA. Процесорна підсистема містить інтегровані блоки кеш-пам’яті першого та другого рівнів, апаратний контролер оперативної пам’яті DDR3, а також розширений набір контролерів сучасних інтерфейсів, таких як Gigabit Ethernet, USB та UART, реалізація яких на базі стандартних логічних елементів була б ресурсномісткою та менш ефективною. Особливе значення має система внутрішньо системних зав’язків на основі високошвидкісних мостів AXI, що забезпечують прозорий доступ до адресного простору та спільних ресурсів пам’яті між процесорними ядрами та логічними модулями користувача.[33]

У таблиці 2.1 вказано точний об’єм памяті3, обчислювальні потужності та роз’єми Terasic DE10-Standard.

Таблиця 2.1 – Технічні характеристики [54]

№.	Категорія	Компонент та детальні характеристики
1.	Обчислювальне ядро SoC	Cyclone V SX 5CSXFC6D6F31C6N інтегрована система, що поєднує FPGA-фабрику та апаратний процесор.
2.	Процесорна система HPS	ARM Cortex-A9 MPCore: 2 ядра, 925 МГц; 512 КБ кешу L2; 8-канальний DMA-контролер.
3.	Програмована логіка FPGA	Ресурси: 110 тис. логічних елементів LE; 41 509 адаптивних логічних модулів ALM; 6 FPGA PLL та 3 HPS PLL.
5.	Оперативна пам'ять	HPS: 1 ГБ DDR3 SDRAM; FPGA: 64 МБ SDRAM; Внутрішня: 5761 Кбіт вбудованої пам'яті кристала.
6.	Зберігання даних та конфігурація	MicroSD: гніздо на HPS для ОС; EPCS128: пристрій конфігурації FPGA; USB Blaster II: вбудований програматор.
7.	Мережа та зв'язок	Ethernet: 10/100/1000 Мбіт/с; USB: 2 хост-порти USB 2.0; UART: USB-to-UART для консольного доступу.
8.	Аналого-цифрова обробка	АЦП: 8 каналів, 12 біт, частота 500 KSPS; діапазон напруги 0 ~ 4,096 В

DE10-Standard має достатньо високі характеристики та набір ресурсів порівняно з аналогічними платами з FPGA у своєму класі. Завдяки розширеним обчислювальним можливостям та значному обсягу інтегрованої пам'яті, вона є хорошою основою для роботи з великими обсягами данних та проведеню паралельних обчислень що задовільняє усі вимоги для розробки прискорювача обчислень на FPGA.

Terasic DE10-Standard сприяє наявність розвиненої екосистеми програмних засобів, зокрема середовища Quartus Prime та інструментарію Platform Designer, які дозволяють інтегрувати інтелектуальні IP-ядра в єдиний проект у стислі терміни. Завдяки такій інтеграції, обчислювальна платформа дозволяє виконувати повний цикл розробки складних систем, включаючи апаратне прискорення алгоритмів, налагодження низькорівневих драйверів та розгортання операційних систем реального часу.[44]

DE10 Standard зображено на рисунку 2.1 де вказано компоненти плати та до якої частини плати вони відносяться.

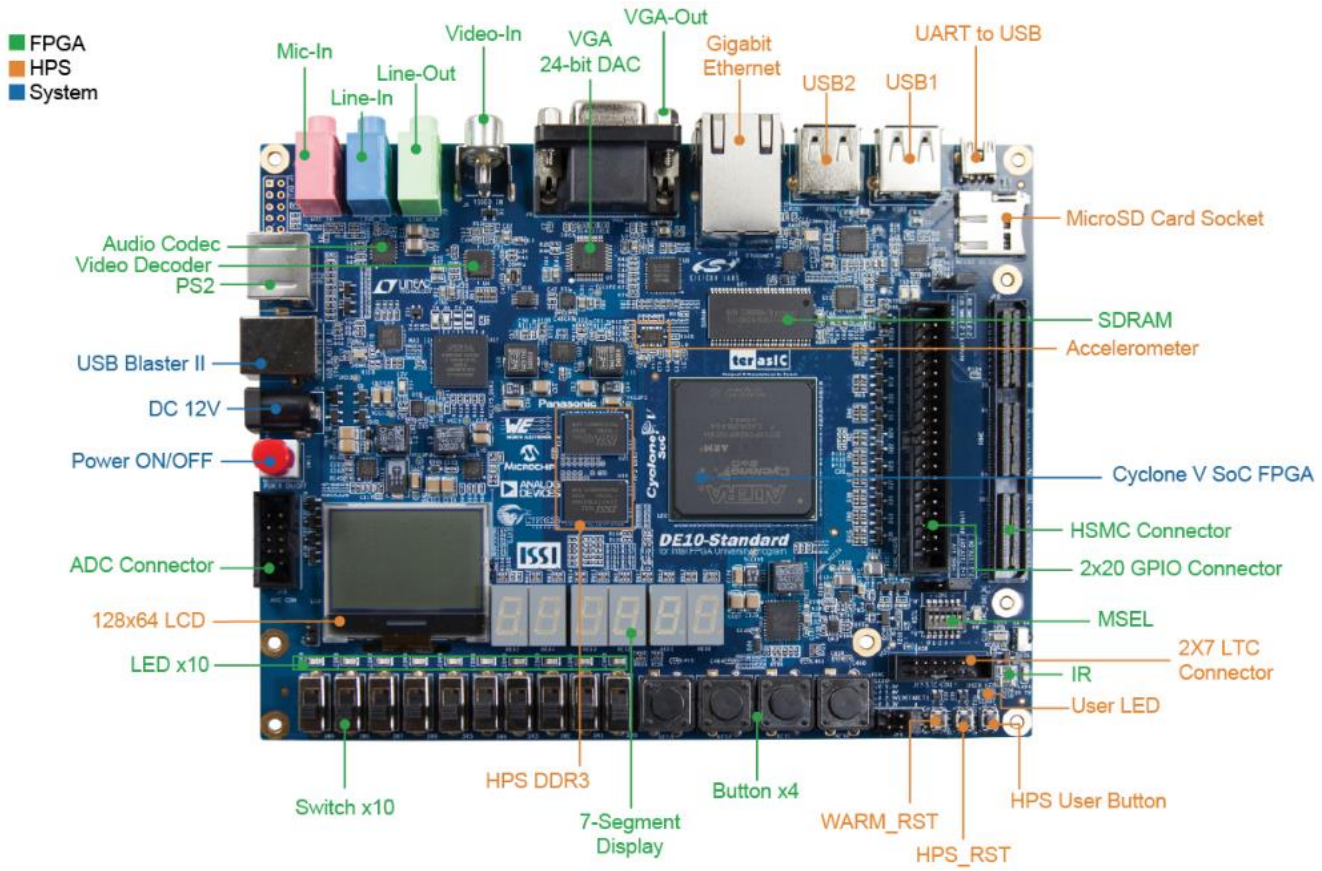


Рисунок 2.1 - Terasic DE10-Standard [14]

Terasic DE10-Standard базується на апаратній базі Intel Cyclone V, яка є представником класу систем на кристалі SoC. System-on-Chip це архітектурна концепція що передбачає повну інтеграцію обчислювальних потужностей, та інтерфейсів, безпосередньо на одному напівпровідниковому кристалі.

Коли усі логічні компоненти розміщенні у одному місці це дає високу швидкість взаємодії між модулями, і збільшує швидкодію завдяки компактному розміщенню. Це також покращує продуктивність, та енергоспоживання системи за рахунок використання внутрішніх високошвидкісних шин замість зовнішніх ліній зв'язку, також використання єдиного кристалу підвищує загальну надійність та спрощує проектування вбудованих систем.[46]

Сам кристал Cyclone V SoC розділений на дві головні частини апаратну процесорну систему HPS та програмовану логіку FPGA, на рисунку 2.2 показано це розділення і частини плати які ці сегменти контролюють.

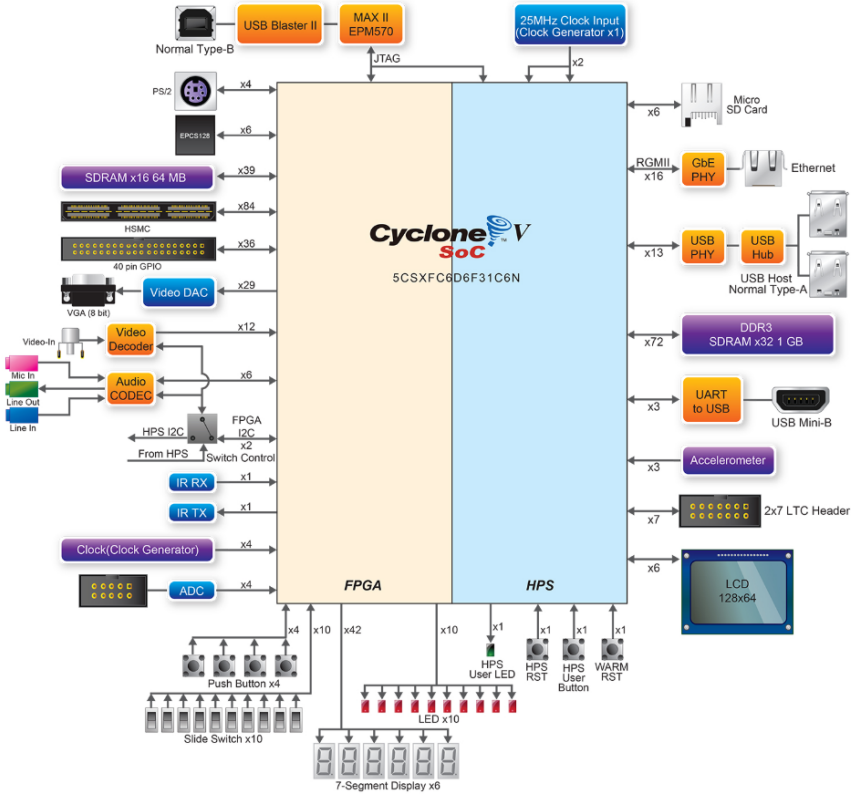


Рисунок 2.2 - Схема Intel Cyclone V [14]

Hard Processor System - це інтегрована апаратна обчислювальна підсистема, подібна до потужного одноплатного комп'ютера. Вона забезпечує повноцінне функціонування операційних систем загального призначення, що дозволяє використовувати звичні інструменти розробки, високорівневі мови програмування та складні бібліотеки для обробки даних.

Процесорна підсистема виконує функцію головного координатора, який керує мережевими протоколами, файловими системами та складними користувацькими інтерфейсами без необхідності заглиблення в особливості апаратної реалізації. HPS працює незалежно від частини FPGA але може з нею взаємодіяти, через перетворювачі для читання та запису даних за певними адресами в оперативній пам'яті. під керуванням складного програмного забезпечення. На відміну від FPGA частини яка конфігуруються безпосередньо в логічних елементах, HPS являє собою незміну архітектуру яка не може бути змінена і виконує лише попередньо задані команди.[47-48]

Field Programmable Gate Array - Програмована логічна інтегральна схема, яка на відміну від HPS може змінювати свою внутрішню конфігурацібю та принцип роботи, що надає можливість створювати власні обчислювальні інструкції та модулі обробки даних. Також завдяки своїй структурі схема дозволяє опрацьовувати порерації паралельно що для розробника програмного забезпечення ця частина системи може використовуватись у якості надпотужного суброцесора, здатного виконувати масивно-паралельні операції, які були б занадто повільними для центрального процесора. Замість написання послідовного коду, на цьому рівні відбувається опис структури потоків даних, де кожна операція виконується безпосередньо в "залізі" за один або кілька тактів.

Вся взаємодія з цією частиною з боку операційної системи абстрагується через драйвери та спеціальні інтерфейси доступу до пам'яті, що дозволяє інтегрувати апаратні модулі у звичайні програмні додатки. FPGA сприймається як динамічне розширення системи, де за необхідності можна повністю змінити функціональне призначення пристрою шляхом оновлення конфігураційного файлу. Це забезпечує унікальну комбінацію високої продуктивності спеціалізованих мікросхем із гнучкістю та швидкістю розробки, притаманною програмним рішенням.[49]

Схема роботи HPS та FPGA і їх взаємодія показана на рисунку 2.3.

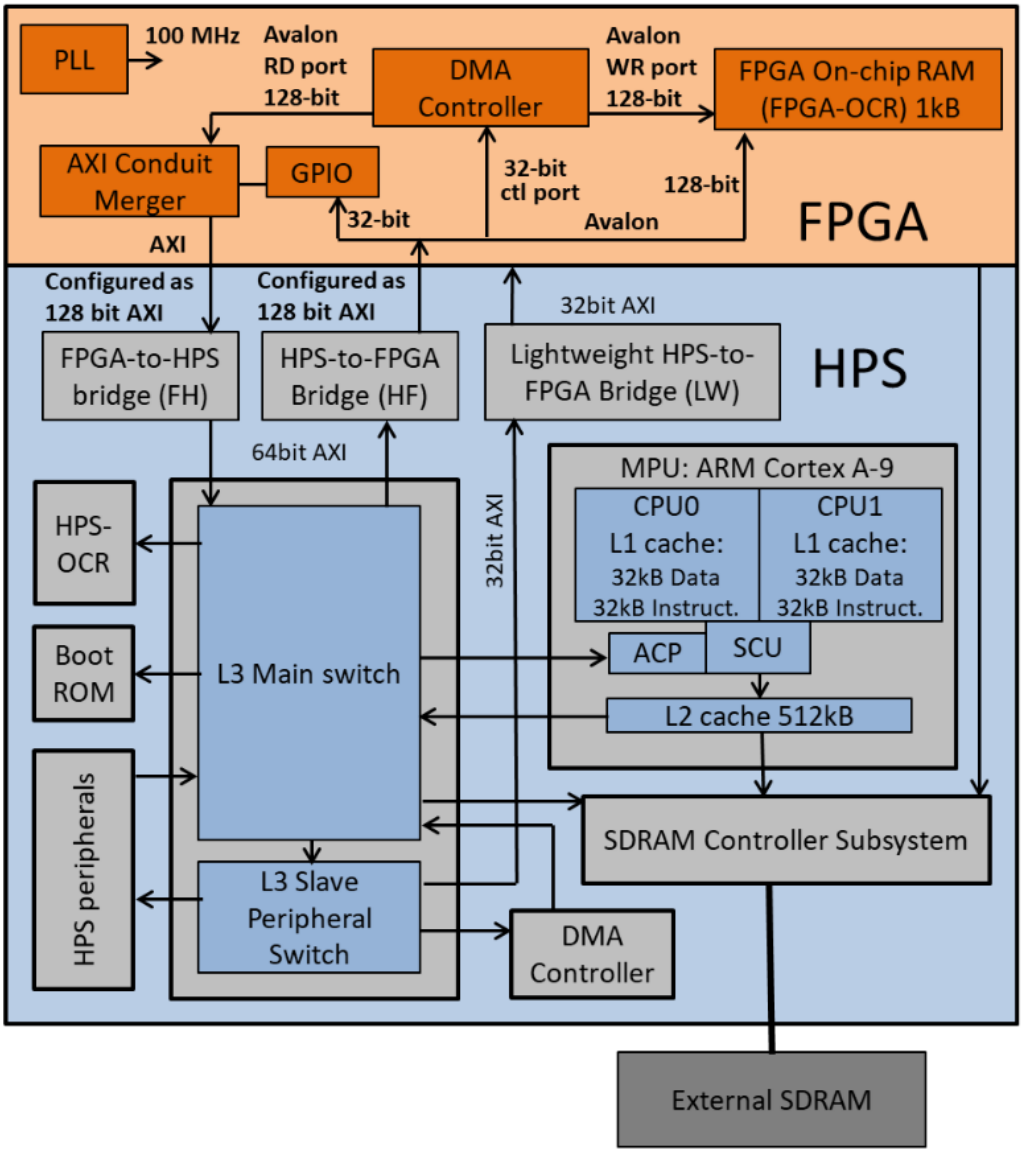


Рисунок 2.3 – логіка HPS та FPGA [18]

Низькорівнева організація SoC починається у просторі користувача Операційної системи Linux, де програма на мові опису OpenCL C готує масив даних у зовнішній пам'яті External SDRAM. На цьому етапі дані існують як набір логічних адрес у віртуальній пам'яті процесора. Для прискорення процесу ядра процесор Cortex A9 задіюють блок DMA передаючи йому початкову адресу та розмір блоку. [42]

Контролер DMA ініціює серію переказів команд на внутрішній шині AXI дані проходять крізь L3 Main Switch, де відбувається маршрутизація команд від усіх елементів плати, де комутатор надає DMA пріоритетний доступ до контролера пам'яті. Тут дані трансформуються з послідовності команд контролера SDRAM у пакети протоколу AXI адреса, і сигнали контролю. Після чого дані потрапляють на міст HPS to FPGA, який змінює фомат даних з яким може працювати FPGA частина так як HPS працює у своєму форматі данних. Після проходження HPS to FPGA команди у низькому рівні абстракції відбувається синхронізація: дані, що йдуть із частотою шини HPS, перехоплюються регістрами мосту та видаються в FPGA вже синхронно з локальним тактовим сигналом від PLL 100 МГц. Ширина шини може розширюватися до 128 біт для забезпечення пікової пропускної здатності.

Всередині FPGA дані потрапляють до модуля. На цьому етапі відбувається фізична зміна інформації: Вхідний потік бітів рівномірно розподіляється на декілька обчислювальних потоків данних. Логічні елементи виконують булеві операції, заміну LSB-біта або криптографічне перетворення безпосередньо над електричними сигналами. Оброблена інформація тимчасово зберігається у FPGA On-chip RAM, що дозволяє згладити нерівномірність швидкості обробки та передачі назад у HPS.

Модуль у FPGA ініціює запис через міст FPGA-to-HPS знову змінюючи формат данних але тепер для HPS частини. Дані знову проходять крізь систему комутації L3 який спрямовує оброблені дані які записуються в External SDRAM за новими адресами. Одночасно з цим через простий міст LW FPGA надсилає сигнал переривання до процесора ПК, сповіщаючи про завершення операції.

Таким чином данні з ПК обчислюються використовуючи потужності прискорювача не задіюючи обчислювальні можливості самого комп’ютера.

2. 2 Програмно обчислювальні методи реалізації

Реалізація програмно-технічного засобу для швидких паралельних обчислень базується на комбінації декількох промислових стандартів та відкритих специфікацій, що використовуються для можливості надати повну сумісність високорівневого програмного коду з архітектурою низькорівневої програмованої логіки. Використання стандартизованих протоколів взаємодії дозволяє замість проектування на складних в освоєні низькорівневих апаратних особливостей FPGA, зосередитись на основній алгоритмічній ефективності системи.

Стандарт OpenCL - це відкритий стандарт для розробки програмного забезпечення, що забезпечує виконання паралельних обчислень на платі standart DE10. Даний набір інструментів та бібліотек надає єдиний універсальний інтерфейс програмування, який дозволяє ефективно розподіляти обчислювальне навантаження між центральним процесором, та прискорювачем обчислень та додатковими спеціалізованими чіпами, у випадку якщо для системи прискорювача обчислень використати декілька пристроїв в межах однієї системи. В основі технології лежить використання мови програмування С, що є легкою в освоєні та розповсюдженої що значно спрощує розробку та редагування код. Для високопродуктивного ядра та вирішення ресурсномістких завдань. Стандарт OpenCL дозволяє трансформувати математичні операції множення матриць у спеціалізовану паралельну архітектуру, де обчислення виконуються одночасно на апаратному рівні. Завдяки можливості запускатись на великій кількості пристроїв, OpenCL забезпечує високий рівень адаптованості коду, дозволяє значно скоротити час проєктування для того чи іншого апаратного засобу для оптимізації системи, із збереженням максимальної обчислювальної потужності.

Керуюче програмне забезпечення, що виконує роль головної host програми, розробляється на базі міжнародного стандарту мови С. Вибір даного стандарту є стратегічно обумовленим, оскільки він забезпечує прямий доступ до системних ресурсів SoC та дозволяє реалізувати прецизійне управління пам'яттю з мінімальними накладними витратами. Це необхідно для забезпечення високої швидкості передачі великих масивів даних, зокрема двомірних матриць, з оперативної пам'яті до апаратного прискорювача, уникаючи зайвих рівнів абстракції, що могли б уповільнити процес.

У межах функціонування системи програма на мові С виконує повний цикл керування обчислювальним процесом, починаючи від ініціалізації пристрою та створення контексту обчислень до завантаження бінарного файлу конфігурації у логічну матрицю FPGA. Данна мова програмування надає усі необхідні інструменти для ефективної диспетчеризації завдань, формування черг команд та точного профілювання часу виконання операцій.

Для забезпечення ефективного обміну даними всередині системи на кристалі використовується архітектурний стандарт AXI4. У межах розробки даний протокол виступає базовим інтерфейсом, що виконує функцію фізичного та логічного інтерпритатора який дозволяє процесору HPS керувати периферією та інтелектуальною власністю IP-ядрами, що реалізовані у програмованій логіці FPGA. Завдяки цій шині процесор може звертатися до регістрів або пам'яті всередині FPGA так само просто, як до власної оперативної пам'яті. між обчислювальною потужністю процесорної підсистеми HPS та спеціалізованою логікою у FPGA.[22]

Специфікація AXI4 дозволяє організувати високоефективні з'єднання для передачі даних всередині кристала. Ці канали забезпечують повністю паралельне читання та запис значних обсягів інформації без взаємного блокування завдяки роздільним лініям адрес та даних. Взаємодія між різними компонентами системи реалізується через вбудовані мости зв'язку та блоки комутації, які оптимізують трафік і мінімізують затримки при доступі до пам'яті.

2.3 Програмні засоби розробки та синтезу

Проєктування та запуск на платі обчислювального прискорювача на базі стандарту OpenCL потребує розгортання спеціалізованого програмного стека, що охоплює всі етапи життєвого циклу розробки. Для написання високорівневого синтаксису і його попереднє тестування під час розробки ядра до апаратного синтезу та фінальної компіляції на платформі. Для забезпечення ефективної взаємодії між процесорною частиною та програмованою логікою використовується великий комплекс утиліт, компіляторів та пакетів системної підтримки.

Для розробки керуючої логіки та обчислювальних модулів системи використовується інтегроване середовище Microsoft Visual Studio, яке виступає центральним хабом для написання, редагування та верифікації вихідного коду. У межах даного проєкту IDE забезпечує комплексний цикл створення Host-програми на мові C++, що відповідає за ініціалізацію пристрою, керування чергами команд та менеджмент пам'яті, а також слугує основним інструментом для опису архітектури обчислювальних ядер Kernel мовою OpenCL C. Завдяки розширеним можливостям статичного аналізу коду та гнучким інструментам налагодження, Visual Studio дозволяє мінімізувати кількість синтаксичних та логічних помилок на етапі пре-компіляції, що є критично важливим перед ресурсомістким процесом апаратного синтезу в середовищі Quartus.

Для забезпечення надійності розроблюваного програмно-технічного засобу використовується фреймворк NVIDIA CUDA Toolkit, який відіграє ключову роль на етапі попередньої верифікації та функціонального тестування алгоритмів. Завдяки кросплатформній природі стандарту OpenCL, даний інструментарій дозволяє виконувати ідентичний код обчислювальних ядер на графічному процесорі GPU локальної робочої станції, що забезпечує швидку ітераційну розробку та налагодження логіки обчислень.

Такий підхід дозволяє переконатися у коректності роботи математичних моделей та алгоритмів у високопаралельному середовищі перед переходом до ресурсомісткого та тривалого за часом процесу повного апаратного синтезу для архітектури FPGA, значно скорочуючи загальний цикл розробки та ймовірність виникнення критичних помилок на фінальних етапах проєктування.

Основним інструментарієм для проєктування апаратної складової прискорювача є система Quartus, яка забезпечує повний цикл фізичного синтезу. Розміщення та трасування Place & Route.

Quartus автоматично розподіляє обчислювальні ресурси ALM, блоки пам'яті M10K, DSP блоки всередині кристала та прокладає оптимальні маршрути передачі сигналів між ними для мінімізації затримок.

Часовий аналіз система перевіряє, чи встигають електричні сигнали пройти через усі логічні ланцюги протягом одного такту заданої частоти 100 МГц від PLL, що гарантує стабільну роботу прискорювача без критичних збоїв під специфіку розробки.

Також для виконання завдання дипломного проєкту середовище quartus також додатково необхідно доповнити специфічними модулями, що забезпечують роботу за стандартом OpenCL.

Пакет Intel FPGA SDK for OpenCL AOCL виступає спеціалізованим середовищем високорівневого синтезу, яке забезпечує трансформацію абстрактних алгоритмів у детерміновану апаратну архітектуру. Основним функціональним елементом цього розширення є офлайн-компілятор aoc, який, на відміну від стандартних засобів розробки для центральних процесорів, не просто генерує набір інструкцій, а фактично проєктує унікальну топологію обчислювальної системи всередині кристала. Процес починається з глибокого аналізу потоків даних у вихідному коді ядра, на основі якого компілятор будує складні конвеєри обробки, де кожна математична операція або ітерація циклу отримує виділені апаратні ресурси в логічній матриці FPGA.

Під час синтезу SDK автоматично вирішує завдання оптимізації та розподілу ресурсів між адаптивними логічними модулями, блоками вбудованої пам’яті та апаратними множниками, намагаючись досягти максимальної паралельності обчислень. Завдяки цьому підходу розробник може ігнорувати складність низькорівневого проєктування інтерфейсів та часової синхронізації сигналів, оскільки ці аспекти автоматизуються на етапі компіляції. Фінальним продуктом роботи середовища є файл конфігурації з розширенням .aocx, який об’єднує в собі апаратний біт-потік для програмування логіки та необхідні метадані для взаємодії з керуючою програмою на базі процесора ARM. Таким чином, використання AOCL дозволяє зосередити основні зусилля на алгоритмічній ефективності прискорювача, забезпечуючи при цьому продуктивність, порівнянну з ручною розробкою на мовах опису апаратури.

Спеціалізований набір інструментів SoC FPGA Embedded Design EDS є сполучної частини між програмним забезпеченням та апаратною платформою, забезпечуючи розробку та компіляцію керуючого коду для процесорної підсистеми HPS. Оскільки цільова архітектура базується на ядрах ARM Cortex A9, пакет надає необхідні засоби крос-компіляції, зокрема інструментарій ARM GCC. Це дозволяє створювати та збирати виконувані файли на робочій станції під керуванням Windows для їх подальшого запуску в середовищі вбудованої ОС Linux.

Окрім компілятора, набір містить утиліти для генерації завантажувачів Preloader та U-Boot, які необхідні для початкової ініціалізації периферії SoC.Функціональність SoC EDS не обмежується лише компіляцією, оскільки пакет містить критично важливі бібліотеки рантайму, необхідні для реалізації API OpenCL на стороні Host програми. Це програмне забезпечення забезпечує низькорівневу взаємодію з драйверами операційної системи, дозволяючи процесору керувати завантаженням апаратних ядер у логічну матрицю FPGA, координувати передачу даних через мости AXI та обробляти переривання від обчислювальних модулів.

Завдяки інтеграції цього інструментарію у загальний цикл розробки стає можливим створення стабільного зв'язку між високорівневим кодом на C та апаратними прискорювачами. SoC EDS фактично формує операційне середовище, де Host-програма може ефективно виконувати роль головного координатора обчислень, делегуючи найбільш ресурсомісткі завдання паралельним структурам у програмованій логіці.

Пакет підтримки плати є необхідним компонентом, що забезпечує апаратну адаптацію універсального стека OpenCL під специфічну архітектуру плати Terasic DE10-Standard. Оскільки стандарт OpenCL оперує абстрактними поняттями обчислювальних пристроїв та глобальної пам'яті, саме BSP є інструментом, який транслює данні у конкретні фізичні ресурси для DE10 standart друкованої плати.

Основним завданням цього розширення є автоматичне конфігурування системних інтерфейсів та периферійних блоків SoC. Пакет містить готові описи топології шин, які налаштовують параметри мостів AXI для забезпечення високошвидкісного обміну даними між процесором та логікою. Крім того, BSP відповідає за ініціалізацію контролерів зовнішньої пам'яті DDR3, розподіляючи адресний простір таким чином, щоб OpenCL-ядра мали прямий та ефективний доступ до масивів даних.

Завдяки використанню BSP, не потрібно вручну описувати розпіновку мікросхеми або часові параметри зовнішніх інтерфейсів у коді VHDL. Пакет створює стабільний фізичний рівень конфігурування плати, що дозволяє синтезованому обчислювальному ядру безперешкодно взаємодіяти з усією апаратною інфраструктурою плати. Це гарантує, що прискорювач зможе повноцінно використовувати пропускну здатність усіх наявних шин та об'єм встановленої пам'яті, зберігаючи дані щоб надати цілісність системі синхронізації та передачі сигналів.

2. 4 Висновок другого розділу

У другому розділі було проведено детальний аналіз функціональних можливостей платформи Terasic DE10 Standard яку було обрано для виконання дипломної роботи, її використання задовільняє усі вимоги для створення прискорювача. Встановлено, що ключовою перевагою обраної платформи є наявність системи на кристалі SoC, яка поєднує гнучку логічну матрицю FPGA та стабільну процесорну підсистему HPS на базі архітектури ARM. Процесорна частина бере на себе функції високорівневого керування та мережевої взаємодії, тоді як FPGA забезпечує апаратну реалізацію критичних за часом паралельних обчислень.

Обґрунтовано вибір програмно обчислювальних методів реалізації, в основі яких лежить стандарт OpenCL. Доведено, що використання даного фреймворку дозволяє абстрагуватися від низькорівневих апаратних складнощів проєктування ПЛІС, зосередивши основну увагу на алгоритмічній оптимізації. Завдяки універсальному інтерфейсу OpenCL забезпечується повна сумісність високорівневого коду з архітектурою обчислювальних ядер, що інсталюються в логічну матрицю, забезпечуючи високу пропускну здатність системи.

Було обрано набір програм та розширень, необхідних для повної розробки проєкту, від написання початкового коду та його емуляції на ПК до фінального апаратного синтезу. Визначено, що інтеграція таких засобів, як Intel FPGA SDK for OpenCL та Intel SoC EDS, є важливою частиною для забезпечення взаємодії між HPS та FPGA частинами через швидкісні інтерфейси. Була сформована проєктна база створює всі необхідні передумови для подальшої реалізації та тестування обчислювального прискорювача на фізичному обладнанні.

3 РОЗРОБКА ФІЗИЧНО-ПРОГРАМНОГО ПРИСКОРЮВАЧА ОБЧИСЛЕНЬ

3.1 Розробка архітектури прискорювача обчислень

Архітектура системи SoC на рисунку 3.1, базується на принципі об’єднання усіх логічних частин обчислювальної машини на одному кварцевому чіпі. Ліва сторона відповідає за . HPS частину, а права FPGA. HPS це процесор, що керує командами усієї плати. Він відповідає за функціонування операційної системи Linux, виконання Host програми, керування файловою системою та координацію загального логічного циклу обчислень.

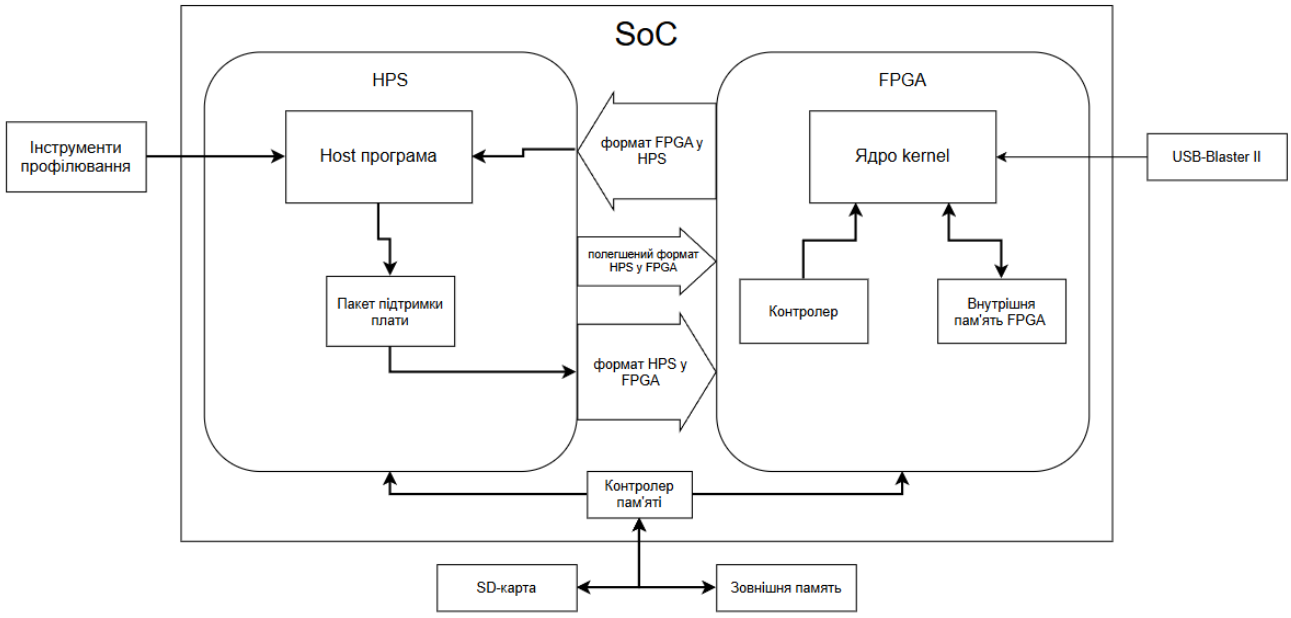


Рисунок. 3.1 – Архітектура системи SoC

FPGA це перепрограмовувана частина що використовується для обчислення матриць. За допомогою універсальних логічних блоків система налаштовується на проведення паралельного множення виконуючи логіку ядра Kernel.

На відміну від традиційних процесорів, FPGA дозволяє створювати спеціалізовані паралельні архітектури, які фізично перебудовуються під конкретний алгоритм, у цьому проекті для матричних обчислень.

Процес ініціалізації системи та взаємодія із зовнішніми інтерфейсами поділяється на два паралельні потоки підготовку обчислювальної логіки та запуск керуючої оболонки. На першому потоці виконується завантаження бінарного файлу прошивки в матрицю FPGA, а на другому ініціалізація ядер процесора, запуск завантажувача та розгортання операційної системи.

Конфігурація апаратної частини USB-Blaster II через інтерфейс JTAG за допомогою програматора USB Blaster II у FPGA завантажується бінарний файл конфігурації з розширенням .aocx. Цей процес фізично перебудовує логічні комірки ПЛІС, формуючи всередині кристала архітектуру обчислювального ядра Kernel, оптимізовану під конкретну задачу.

Завантаження програмного середовища через SD-карту, процесор HPS використовує SD-карту як основний накопичувач, з якого зчитується образ операційної системи Linux та скомпільований виконуваний файл Host програми. Це забезпечує розгортання високорівневого середовища, необхідного для керування ресурсами OpenCL та виконання користувацького коду.

Організація пам'яті та зберігання даних DR3 SDRAM де зовнішня оперативна пам'ять DDR3 виступає головним сховищем та спільним об’ємом пам’яті для обох частин HPS та FPGA до якого доступ має уся система. У ній Host програма резервує простір для вхідних матриць, які згодом зчитуються прискорювачем, і саме сюди FPGA записує фінальний результат обчислень, роблячи його доступним для подальшого аналізу та виводу.

Взаємодія між ARM-процесором та логікою FPGA реалізується через систему спеціалізованих комунікаційних мостів, які забезпечують швидку передачу даних між компонентами SoC. Ця інфраструктура підтримує звязок між компонентами кристалу і взаємодію його компонентів.

AXI HPS to FPGA та FPGA to HPS – це високошвидкісні шини, розраховані на масивну передачу даних між оперативною пам'яттю та прискорювачем. HPS та FPGA використовують різні мови і щоб вони могли розуміти один одного мости також конвертують дані для того щоб візні частини розуміли команди і дані які вони передають. Завдяки цим каналам ядро Kernel може зчитувати елементи матриць великого обсягу безпосередньо з пам'яті та записувати обчислені результати в режимі реального часу, забезпечуючи високу пропускну здатність системи.[44]

Lightweight HPS to FPGA це додатковий особливий виділений канал який виділений спеціально для передачі швидких команд керування та налаштування регістрів. Через цей канал Host записує параметри обчислень та фізичні адреси матриць у керуючі регістри FPGA, що дозволяє точно визначити джерела даних у спільній пам'яті. Після підготовки конфігурації процесор надсилає через міст сигнал активації, який запускає паралельну обробку алгоритму безпосередньо в апаратній логіці пристрою. Протягом роботи прискорювача Host код використовує цей міст для опитування статусних регістрів, щоб зафіксувати момент завершення розрахунків та перевірити готовність системи до видачі результату.

Контролер пам'яті це спеціалізований апаратний блок, який керує потоками даних між зовнішнім модулем оперативної пам'яті та внутрішніми компонентами чипа. Він контролює доступ до пам’яті усього SoC, чергує запити на читання та запис, що надходять одночасно від процесора HPS та обчислювального ядра FPGA. Контролер гарантує, що прискорювач і процесор працюють зі спільною пам'яттю без конфліктів, запобігаючи одночасному перезапису однієї і тієї ж комірки різними частинами системи. Він перетворює високорівневі команди доступу на низькорівневі електричні сигнали, специфічні для стандарту DDR3, забезпечуючи максимальну пропускну здатність шини без простоїв.[53]

Рівень програмного забезпечення, що функціонує на базі HPS, забезпечує інтелектуальне керування апаратними ресурсами та реалізує інтерфейс взаємодії з користувачем. Він складається з двох ключових компонентів, які працюють у тісній зв'язці для забезпечення цілісності обчислювального процесу:

Host це диспетчер, що керує усіма процесами від початкового виділення буферів пам'яті в ОЗП до збору фінальних результатів. Він відповідає за завантаження вхідних даних, ініціює запуск апаратного ядра та після завершення роботи прискорювача проводить аналітичну обробку отриманих значень.

Бібліотека середовища виконання OpenCL слугує важливим посередником, який перекладає високо рівневі команди Host коду у низько рівневі апаратні команди. Бібліотека фізично керує потоками даних через шини AXI та мости комунікації, забезпечуючи коректне відображення програмних запитів на фізичні операції всередині SoC.

Останній етап роботи системи зосереджений на виконанні математичних обчислень в FPGA та аналізі отриманих результатів роботи плати. Що дозволяє прискорити швидкість роботи комп’ютера, після чого оцінити його оптимізацію та ефективність прискорювача.

Обчислювальне ядро Kernel використовує власну виділену локальну пам'ять для зберігання поточних сегментів матриць. Доступ до внутрішньої пам'яті відбувається значно швидше, ніж до зовнішньої. Це зменшує затримки при зверненні до зовнішньої пам'яті та уникнути затримок. [54-55]

Інструменти профілювання відповідає за збір телеметрії безпосередньо з апаратних лічильників під час роботи ядра. Інструменти фіксують точний час виконання операцій та розраховують показники продуктивності, GIOPS мільярди цілочисельних операцій на секунду. Отримані звіти передаються користувачеві для аналізу даних.

3.2 Опис процесу створення баз даних

Спочатку необхідно побудувати логічно-фізичну модель даних за допомогою CASE-засобу AllFusion ERWin Data Modeler. Фізична модель наведена нижче на рисунку 3.2.

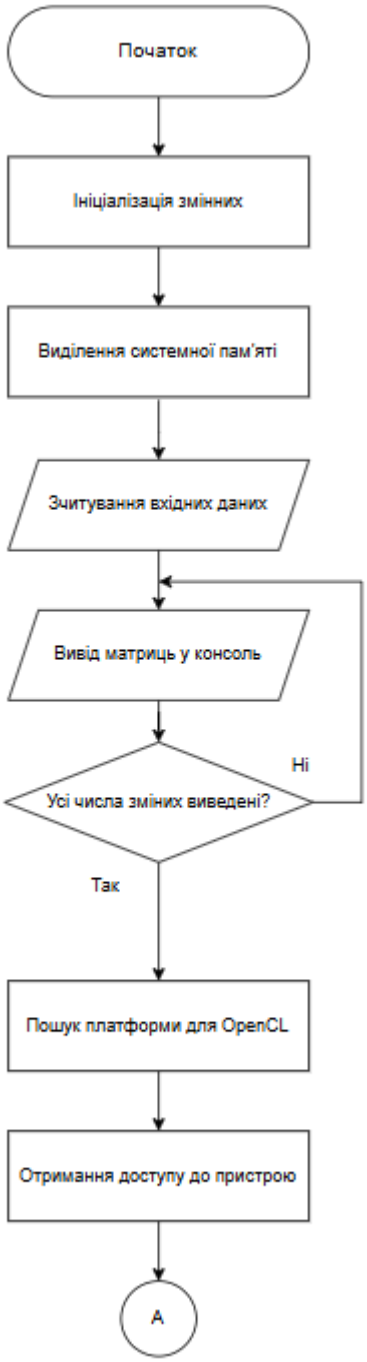


Рисунок. 3.2 – Блок схема принципу роботи Host програми, аркуш 1

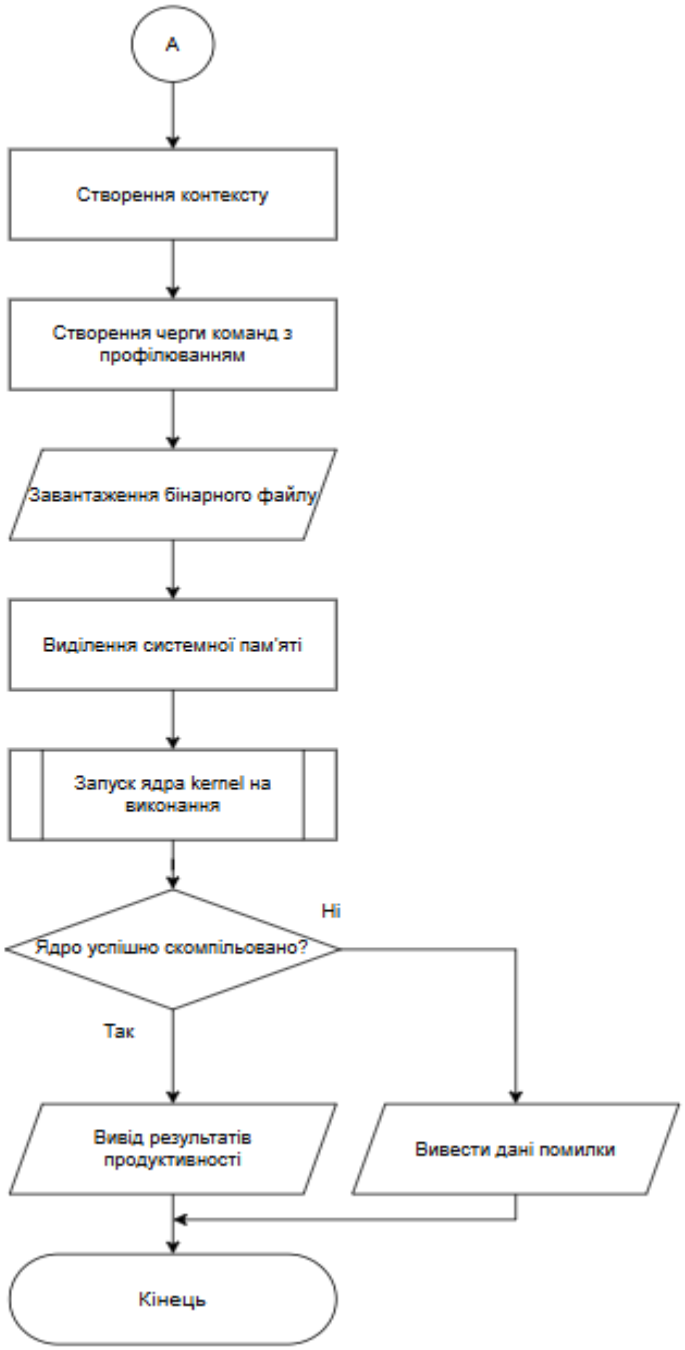


Рисунок. 3.3 – Блок схема принципу роботи Host програми, аркуш 2

Етап ініціалізації змінних де створюються основні параметри обчислювальної задачі. На цьому кроці визначається ключова константа N, яка задає розмірність матриць і безпосередньо впливає на масштаб паралелізму, що буде реалізований апаратною логікою FPGA.

Програма розраховує загальну кількість елементів як матрицю N x N, а також обчислює точний обсяг пам’яті у байтах, необхідний для зберігання даних.

Етап виділення системної пам'яті в Host-програмі відповідає за фізичне резервування ресурсів в оперативній пам'яті ARM-процесора HPS для роботи з даними. Використовуючи стандартну функцію malloc, ця частина коду виділяє три окремі області пам'яті для масивів A, B та C. Блок зчитування вхідних даних заповнює підготовлені масиви конкретними значеннями. Цей процес автоматизовано проходить по кожній комірці пам'яті та присвоює ціле число.

Етап виводу матриць у консоль виконує функцію візуального моніторингу та верифікації стану системи перед активацією апаратного прискорювача. Ця частина коду організована як вкладений цикл, який форматує одновимірні масиви A та B у вигляді зручних для сприйняття таблиць, відображаючи їх у терміналі. Що допомагає провести ручну перевірку коректності генерації даних та переконатися, що всі елементи матриць відповідають очікуваним значенням. Пошук платформи для OpenCL ініціює взаємодію з драйверами шляхом виклику. Програма проводить сканування доступних у системі середовищ виконання та ідентифікує платформу, яка здатна забезпечити стабільний канал зв'язку з логічною матрицею FPGA. Результатом виконання блоку є отримання унікального ідентифікатора, що підтверджує готовність обраного програмного стека до керування апаратним прискорювачем.

Отримання доступу до пристрою проводить операцію конкретизації об'єкта обчислень. Оскільки сучасні SoC системи можуть містити кілька обчислювальних модулів, програма цілеспрямовано звертається до акселератора, ігноруючи інші доступні ресурси, такі як інтегрована графіка чи додаткові ядра. Host код прив’язується до конкретної фізичної адреси пристрою. Ця прив'язка реалізується шляхом мапування регістрів керування акселератора у віртуальний адресний простір користувача за допомогою системного виклику через драйвер символьного пристрою.

Створення контексту це процес формування програмної оболонки, всередині якої відбуваються будь-які операції OpenCL. Контекст створює захищений простір, де зберігаються всі об'єкти, пов'язані з конкретною обчислювальною задачею. Це гарантує, що операції над матрицями в одному контексті не будуть конфліктувати з іншими процесами в системі. Програма використовує цей об'єкт як контейнер, який має доступ до інформації про всі доступні ресурси: від фізичного чипа Cyclone V до виділених ділянок пам'яті.

В межах контексту визначаються правила створення та видалення черг команд, програмних об’єктів Kernel та буферів даних. Контекст виступає гарантом того, що обчислювальне ядро зможе отримати доступ до вказаних буферів пам'яті, оскільки вони належать до одного логічного домену. Без сформованого контексту неможливо ініціалізувати передачу даних через інтерфейс AXI, оскільки система не матиме точки координації адресних просторів.

Створення черги команд з профілюванням виконує організування логічного потоку команд до прискорювача. Використання особливого прапорця активує спеціальний апаратний механізм відстеження подій. Завдяки цьому Host програма отримує можливість фіксувати точні моменти початку та завершення операцій, що необхідно для обчислення реальної швидкодії системи та її пропускної здатності.

Завантаження бінарного файлу реалізує процес зчитування скомпільованого образу ядра з диска в пам'ять Host системи. На відміну від стандартного програмного коду, цей файл містить повну цифрову схему архітектури прискорювача. Виклик функції clCreateProgramWithBinary ініціює передачу цієї конфігурації безпосередньо в логічні комірки ПЛІС, що фактично перебудовує внутрішню структуру чипа під виконання конкретної математичної операції.

Виділення системної пам'яті відбувається створення спеціалізованих об’єктів. На відміну від звичайного виділення пам’яті в оперативному запам’ятовуючому пристрої процесора, цей крок формує логічні канали доступу до пам'яті самого пристрою. Код визначає права доступу для кожного буфера, позначаючи матриці A та B як об'єкти тільки для читання, а матрицю C для результату обчислення як буфер лише для запису. Це дозволяє драйверу оптимізувати трафік на шині зв’язку між HPS та FPGA і підготувати адресний простір прискорювача до прийому або відправки пакетів даних.

Запуск ядра kernel ініціює перехід від програмного керування до прямих апаратних обчислень на платі через виклик необхідної функції. На цьому етапі відбувається обчислення матриці, програма передає пристрою параметри розмірності обчислювальної сітки у двовимірному простору, що дозволяє FPGA миттєво розгорнути тисячі одночасних операцій множення. Під час виконання цього блоку HPS процесор відправляє спеціальний сигнал керування через міст що перетворює данні на зрозумілий формат для FPGA, вказуючи адреси вхідних буферів та цільову адресу для результату. Оскільки ядро працює в асинхронному режимі, Host код може продовжувати моніторинг або очікувати завершення операції, тоді як апаратна частина виконує по елементну обробку матриць на рівні транзисторних зв’язків. Цей етап є головним моментом роботи системи, де реалізується основна перевага архітектури у високій швидкості обчислення завдяки паралельних розрахунків при низьких енерговитратах.

Вивід результатів продуктивності виконує аналітичну частину яка відповідає за обчислювання загальної ефективності роботи прискорювача, перетворюючи сирі дані апаратних лічильників у зрозумілі метрики ефективності. Використовуючи функцію, програма вилучає часові мітки початку та завершення роботи обчислювального ядра. Процес оцінки базується на трьох ключових параметрах.

Першим параметром є час виконання обчислювального ядра, який відображає чистий інтервал роботи апаратного прискорювача без урахування програмних затримок. Цей показник розраховується як різниця між моментом постановки завдання в чергу та сигналом про його завершення, що дозволяє точно визначити тривалість обробки даних безпосередньо в логічних комірках FPGA. Такий підхід забезпечує об’єктивну оцінку продуктивності алгоритму, оскільки виключає вплив операційної системи та накладні витрати на ініціалізацію середовища.

Другим показником виступає пропускна здатність, яка виражається у кількості виконаних операцій за одиницю часу та демонструє обчислювальну потужність системи. Вона обчислюється шляхом ділення загального обсягу проведених математичних операцій на час роботи ядра, що дозволяє кількісно оцінити ефективність паралельної архітектури. Ця метрика є ключовою для порівняння апаратної реалізації з програмними аналогами, оскільки вона наочно показує перевагу розгалужених обчислювальних конвеєрів над послідовним виконанням інструкцій.

Третім важливим параметром є час передачі даних, який фіксує тривалість копіювання інформації між системною пам’яттю процесора ARM та внутрішніми буферами FPGA через мости AXI. Оскільки обмін даними часто займає найбільше часу у багаторівневих системах, вимірювання цих затримок дозволяє оцінити вплив комунікаційних накладних витрат на загальну швидкість роботи. Аналіз цього показника дає можливість оптимізувати протоколи обміну та визначити реальний поріг ефективності використання апаратного прискорення для завдань різної складності.

3.3 Розробка архітектури обчислювального ядра

Процес розробки починається з визначення структури ядра kernel, яке буде виконуватися на боці пристрою FPGA. На відміну від стандартного процесора, де цикли виконуються послідовно, архітектура OpenCL дозволяє визначити масив робочих елементів, кожен з яких обробляє окрему частину даних паралельно. Блок схема роботи kernel показана на рисунку 3.3.

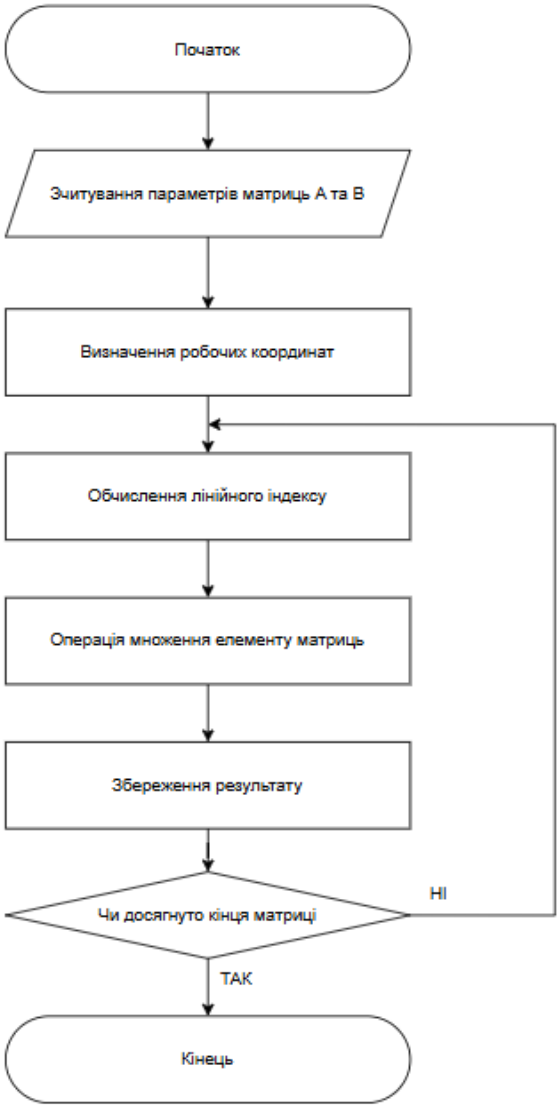


Рисунок. 3.3 – Блок схема принципу роботи ядра

Зчитування параметрів матриць A та B є вхідною точкою роботи обчислювального ядра kernel і відповідає за підготовку даних безпосередньо перед початком математичних операцій. На відміну від стандартних програм для CPU, у контексті FPGA та OpenCL цей процес має декілька критичних аспектів. Передача аргументів: Обчислювальне ядро отримує не самі масиви даних, а вказівники на їх розташування у глобальній пам’яті пристрою Global Memory, це здійснюється через Host програму, яка ініціалізує буфери.

Параметри матриць A та B зчитуються з використанням особливого специфікатора. Це вказує компілятору Intel FPGA SDK, що дані знаходяться у зовнішній оперативній пам'яті DDR3 і призначені лише для читання. Такий підхід дозволяє оптимізувати доступ до пам'яті, використовуючи кешування з боку апаратного контролера. Зчитування розмірності окрім самих матриць, на етапі зчитується скалярна величина N що використовується для збереження визначеної розмірності матриці. Вона є важливою для коректного позиціонування в одновимірному масиві та контролю циклу обчислень. Підготовка конвеєра на апаратному рівні ініціює завантаження перших порцій даних у вхідні регістри обчислювального блоку. Завдяки конвеєрній архітектурі FPGA, зчитування наступних елементів відбуватиметься паралельно з обробкою поточних, що дозволяє уникнути простоїв обчислювальних одиниць.

На моменті визначення робочих координат необхідно обчислювати міце розташування кожного елементу матриці через неможливість FPGA роботи з двомірними масивами. У архітектурі FPGA пам'ять організована як лінійний масив байтів, а ми оперуємо концепцією двовимірних матриць, цей етап є критично важливим для правильної адресації даних. Математичне перетворення на основі отриманих раніше робочих координат x та y які відповідають за адресу кожного елементу матриці, обчислюється унікальний порядковий номер елемента в пам'яті за формулою математичного перетворення двовимірних декартових координат елемента матриці у лінійний індекс одновимірного масиву.

Це дозволяє ядру точно визначити, до якої саме комірки масиву необхідно звернутися. Апаратна реалізація: В FPGA це операція виконується за допомогою вбудованих блоків множення та додавання. Використання стандарту OpenCL дозволяє автоматично конвеєризувати це обчислення, тому розрахунок індексу для наступного елемента відбувається одночасно з обробкою поточного елементу.

Оптимізація доступу завдяки якій архітектура прискорювача намагається згрупувати запити до пам'яті. Якщо сусідні робочі потоки звертаються до послідовних лінійних індексів, контролер пам'яті виконує одне широке читання замість багатьох дрібних, що значно підвищує загальну пропускну здатність системи. Підготовка зміщення це обчислений індекс який використовується як зміщення відносно базової адреси вказівників на усіх трьох матриць, які були зчитані на першому етапі.

Операція множення елементу матриць це крок математичного обчислення матриць прискорювача, де відбувається перетворення вхідних даних у обчислений результат за допомогою апаратних ресурсів ПЛІС.

Апаратне обчислення де цифрова логіка FPGA використовує спеціалізовані блоки DSP, Для чипа, що встановлений на платі DE1-SoC, це дозволяє виконувати операцію множення з надзвичайно високою швидкістю, оскільки вона реалізується на рівні транзисторних зв'язків, а не через виконання програмних інструкцій процесором. Паралелізм даних оскільки ядро OpenCL розгорнуте на великій кількості обчислювальних одиниць, ця операція виконується одночасно для багатьох елементів матриці. Поки один блок завершує обчислення для індексу, ID інший блок вже приймає дані що формує безперервний потік обробки. Також точність і швидкість виконання операції чітко відповідає стандарту ISO IEC, що гарантує ідентичність результатів апаратного обчислення та програмної моделі на мові C. Це важливо для подальшої верифікації правильності роботи прискорювача.

Збереження результату відповідає за фіксацію обчислених значень та їх повернення у загальну ієрархію пам'яті системи. Отримане значення після операції множення записується у вихідну матрицю C, що розташована в глобальній пам'яті пристрою. Цей процес є останньою дією, саме з цього буфера Host програма згодом зчитає фінальний результат для подальшого аналізу.

В архітектурі FPGA запис результату відбувається через інтерфейс AXI4, що забезпечує цілісність даних при передачі від обчислювального ядра до контролера пам'яті після чого результати обчислень будуть доступні для процесорної частини HPS без помилок чи втрат. Завершення обчислень для конкретного робочого елемента є фінальною апаратною командою.

Після успішного запису ресурс, який виконував обчислення, звільняється або переходить до обробки наступного пакету даних, у випадку якщо матриця перевищує кількість доступних паралельних блоків. Збереження результату у визначеному лінійному індексі ID дозволяє надати повну відповідність структурі вихідної матриці.

Розроблене обчислювальне ядро паралельного множення елементів матриць безпосередньо в апаратній логіці ПЛІС. Завдяки використанню глобальних ідентифікаторів та розрахунку лінійного індексу, надає точне відображення обчислень на структуру пам'яті пристрою. Описана логіка дозволяє задіяти виділені DSP блоки чипа для виконання арифметичних операцій, що значно підвищує швидкість обробки даних. Фінальне збереження результату через інтерфейс AXI4 гарантує цілісність передачі інформації між прискорювачем та процесорною частиною HPS для подальшої верифікації.

3.4 Реалізація алгоритму прискорювача на платі

Для запуску розробленого алгоритму на платформі Terasic DE10 standart необхідно скомпілювати описане ядро алгоритму в апаратну конфігурацію. Вихідний код ядра, написаний мовою OpenCL, проходить через процес високорівневого синтезу, результатом якого є файл з розширенням aocx.

AOCX являє собою фінальний бінарний образ, який містить повну апаратну та програмну частину для реалізації алгоритму всередині системи. Він вміщує цифровий зліпок цільової архітектури, що фізично визначає стан мільйонів логічних комірок, тригерів та комутаційних з'єднань у матриці FPGA для її перебудови під конкретне обчислювальне завдання. Окрім апаратної конфігурації, цей файл зберігає в собі детально прописану логіку паралельних конвеєрів обробки даних, яка регламентує виконання операцій у кожному такті системного годинника. AOCX містить необхідні метадані для бібліотек OpenCL Runtime та описи інтерфейсних мостів, що дозволяє процесору HPS коректно взаємодіяти з апаратним прискорювачем через шини передачі даних.

Повний цикл синтезу апаратної логіки для системи на кристалі зображено на рисунку3.4. На початковому етапі компілятор перевіряє коректність синтаксису вихідного коду kernel.cl і будує граф потоків даних. Потім виконується аналіз аргументів ядра та типів даних. Математичні операції перетворюються на архітектуру паралельних потоків команд, де прописано, які дії виконуються синхронно в кожному такті системного годинника.[43]

Після побудови логічної схеми компілятор виконує оптимізацію та попередній розрахунок необхідних апаратних ресурсів. Лінкування з IP блоками зв'язується з готовими бібліотеками. Система генерує звіт, що показує відсоткове завантаження кристала, скільки ядро може займати 19% загальної логіки, 12% блоків пам'яті та 4% DSP модулів.

```
D:\fpga_projects>aoc kernel.cl -o bin/kernel.aocx -board=de10_standard_sharedonly -report -v
aoc: Environment checks are completed successfully.
aoc: Cached files in C:\Users\т|єры|щ\AppData\Local\aocl may be used to reduce compilation time
You are now compiling the full flow!!
aoc: Selected target board de10_standard_sharedonly
aoc: Running OpenCL parser....
d:/fpga_projects/kernel.cl:1:46: warning: declaring kernel argument with no 'restrict' may lead to low kernel perform
__kernel void matrix_add(__global const int* A,
                                             ^
d:/fpga_projects/kernel.cl:2:25: warning: declaring kernel argument with no 'restrict' may lead to low kernel perform
    __global const int* B,
                        ^
d:/fpga_projects/kernel.cl:3:19: warning: declaring kernel argument with no 'restrict' may lead to low kernel perform
    __global int* C,
                  ^
3 warnings generated.
aoc: OpenCL parser completed successfully.
aoc: Optimizing and doing static analysis of code...
aoc: Linking with IP library ...
Checking if memory usage is larger than 100%

!===========================================================================
! The report below may be inaccurate. A more comprehensive
! resource usage report can be found at kernel/reports/report.html
!===========================================================================

+--------------------------------------------------------------------+
; Estimated Resource Usage Summary                                  ;
+---------------------------------------+----------------------------+
; Resource                              + Usage                      ;
+---------------------------------------+----------------------------+
; Logic utilization                     ;   19%                      ;
; ALUTs                                 ;   11%                      ;
; Dedicated logic registers             ;    9%                      ;
; Memory blocks                         ;   12%                      ;
; DSP blocks                            ;    4%                      ;
+---------------------------------------+----------------------------;
aoc: First stage compilation completed successfully.
Compiling for FPGA. This process may take a long time, please be patient.
```

Рисунок. 3.4 – Компіляція файлу ядра

Для перенесення розроблених компонентів на апаратну платформу використовується карта пам'яті MicroSD. Вона є основним накопичувачем, де зберігається образ вбудованої операційної системи Linux, необхідні драйвери та файлова система. Процес підготовки носія починається з форматування та розміщенні бінарного файлу апаратного ядра разом із скомпільованим виконуваним host файлом у спільній робочій директорії користувача. Коли host ініціює обчислення, вона автоматично звертається до поточного каталогу на карті пам'яті для пошуку потрібного образу апаратної логіки. Цей файл необхідний для запуску внутрішнього процесу конфігурування та прошивки програмованої матриці FPGA безпосередньо під час роботи системи. Паралельно з цим виконується автоматичне налаштування та задання системних шляхів, що дозволяє операційній системі зв'язати програмну частину із завантаженою апаратною логікою.

Взаємодія з розгорнутим середовищем здійснюється через термінальний доступ за протоколами UART, Інстанціація обчислювального прискорювача в матриці ПЛІС реалізується шляхом виконання системної команди aocl program, яка спрямовує бінарний потік даних із файлу aocx до контролера конфігурації. У цей момент через інтерфейс USB Blaster 2 або внутрішні шини SoC відбувається фізичне заповнення комірок пам'яті FPGA, що перетворює універсальну логіку на спеціалізовану архітектуру. Успішне завершення цього процесу підтверджується активацією світлодіода, що свідчить про повну перебудову кристала та готовність апаратних конвеєрів до роботи. Завершення компілювання зображено на фото 3.5.

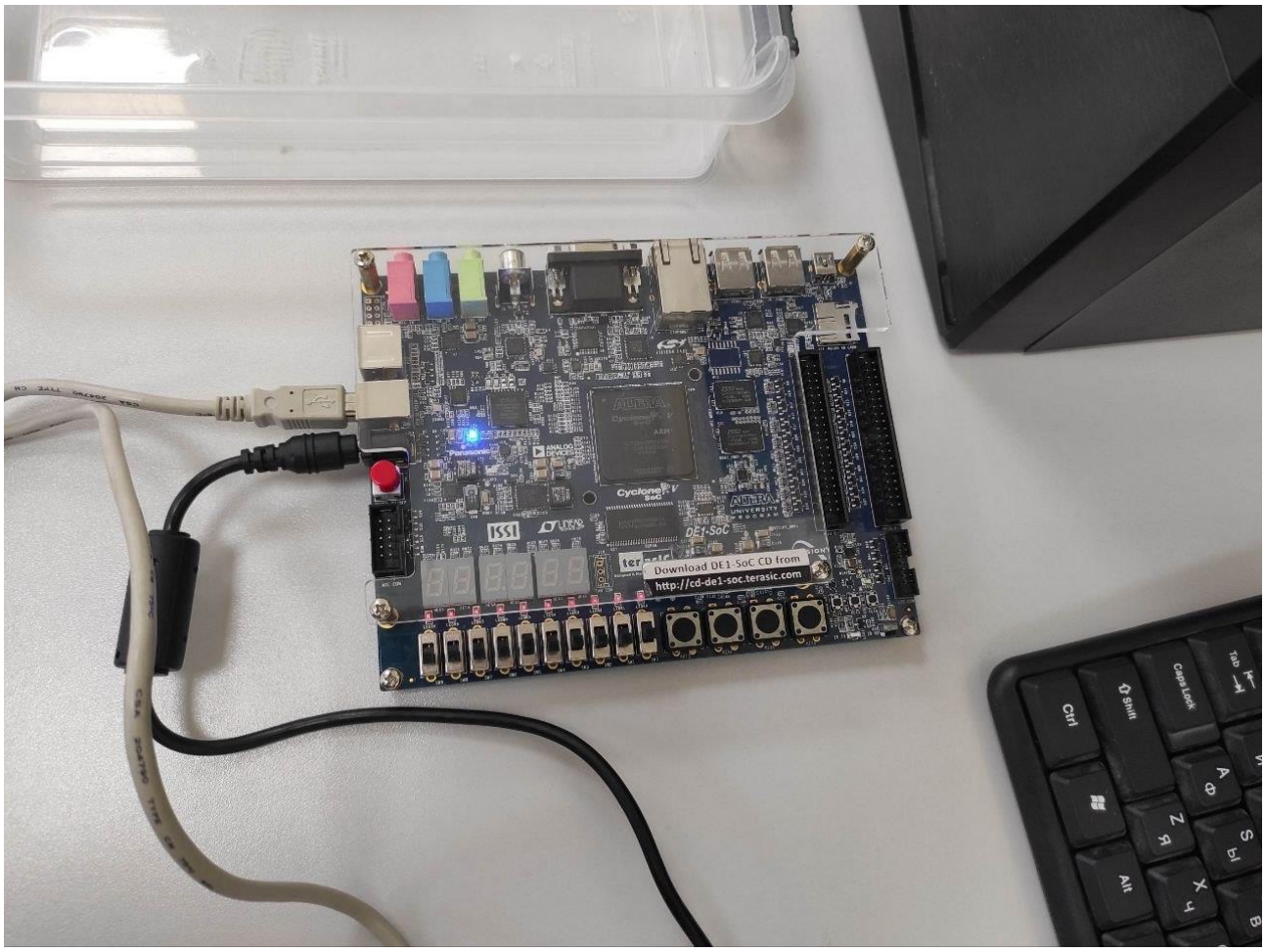


Рисунок. 3.5 – Фото скомпільованої FPGA частини

Наступним етапом запускається Host програми, яка керує процесом виділення памяті і запуском FPGA частини для проведення обчислювального процесу програма виконує сканування доступних апаратних платформ, виділяє необхідні обсяги оперативної пам'яті DDR3 для формування вхідних і вихідних буферів, а також налаштовує параметри передачі даних. Через інтерпритатор інформації, host транслює ядру фізичні адреси масивів та подає сигнал до початку обробки.

Завершальний етап охоплює виконання алгоритму в залізі та подальшу обробку отриманих результатів. Поки логічна матриця здійснює паралельну обробку даних, програмна частина перебуває в стані очікування, контролюючи прапор готовності прискорювача. Після завершення циклу обчислень результати зчитуються із пам'яті для виведення в консоль користувача. Паралельно з цим, за допомогою засобів інструментального аналізу, проводиться розрахунок параметрів ефективності, таких як чистий час виконання та продуктивність у GIOPS, що дозволяє об’єктивно оцінити переваги апаратного прискорення.

Результат роботи програми рисунок 3.6.

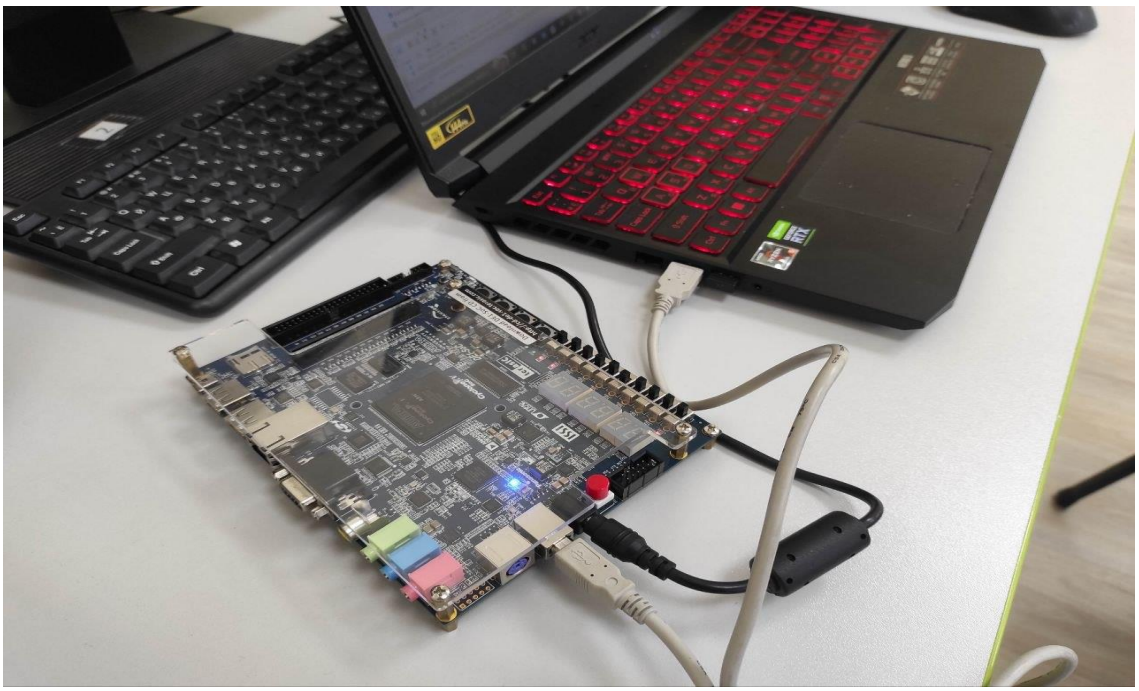


Рисунок. 3.6 – Завершення виконання обчислення данних

3.5 Тестування та аналіз результатів запуску прискорювача

Аналіз ефективності системи без апаратного прискорення зображено на рисунку 3.7. Для отримання базових показників продуктивності буловедено контрольний запуск обчислень без задіяння прискорювача обчислень FPGA матриці. Результати цього тестування демонструють значно нижчу обчислювальну ефективність порівняно з паралельним режимом роботи, оскільки процесорні ядра змушені виконувати всі операції послідовно.

Низька пропускна здатність: Показник продуктивності склав 4.394531 GIOPS. Це підтверджує, що при виконанні завдання виключно програмними засобами або при відсутності оптимізованого апаратного конвеєра, система здатна обробити значно меншу кількість операцій за одиницю часу. Центральний процесор стає вузьким місцем системи через високу щільність арифметичних команд.

Час чистої роботи Kernel work time зафіксовано на рівні 0.005120 ms. При невеликих обсягах даних цей час фактично зрівнюється з часом роботи ядра в оптимізованих сценаріях, проте через відсутність обчислення великої кількості параметрів одночасно, загальна продуктивність Throughput залишається низькою. Ситуація критично погіршується при масштабуванні завдання, коли відсутність апаратної конвеєризації призводить до лінійного зростання затримок.

Витрати на пам'ять: Час запису в пам'ять становить 0.015616 ms, що у три рази перевищує час самих обчислень. Це вказує на те, що навіть без апаратного прискорювача обмін даними з пам’яттю залишається домінуючим фактором, що обмежує загальну швидкість системи.

Аналіз показує, що без використання прискорювача система демонструє у декілька разів меншу потужність порівняно з піковими значеннями 10 - 12 GIOPS, отриманими при повній апаратній підтримці на базі CycloneV. вирішення ресурсомістких завдань у вбудованих системах.

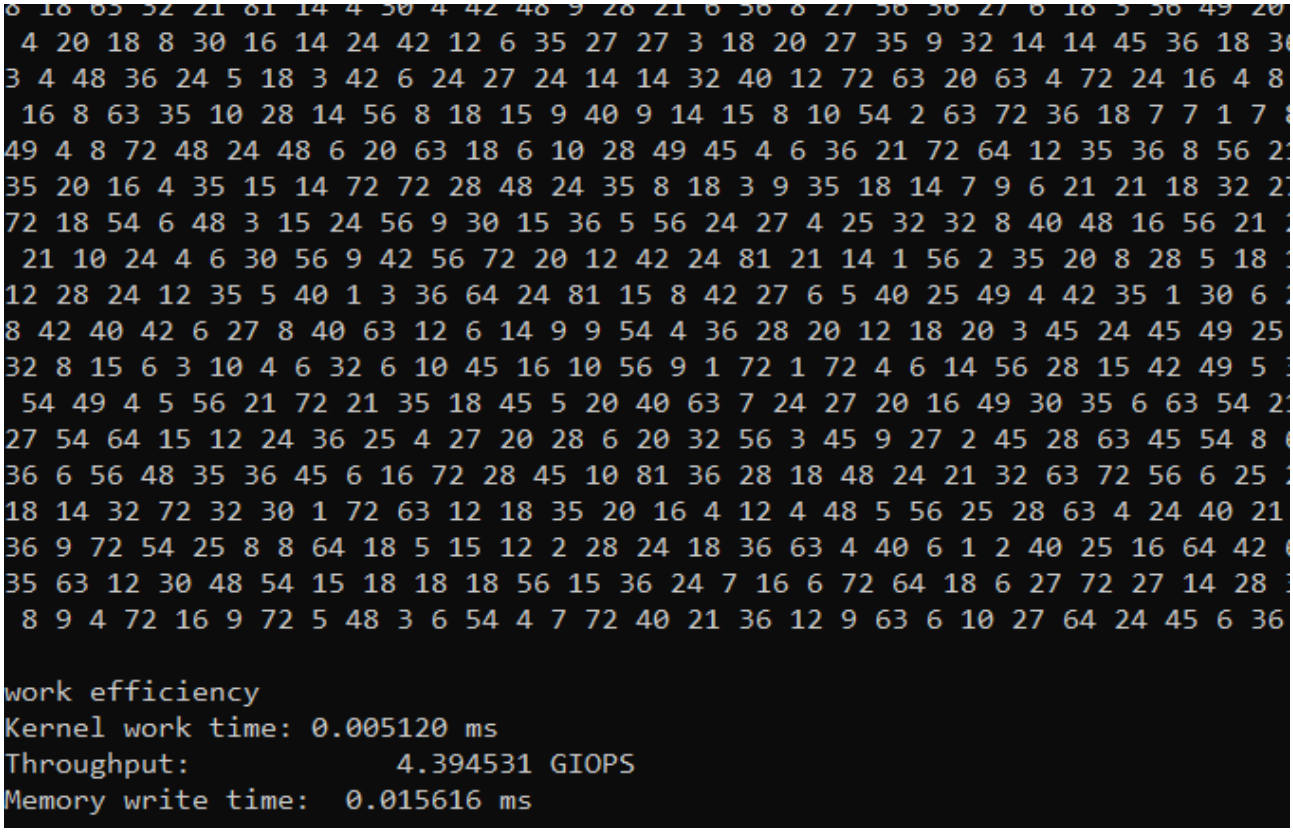


Рисунок. 3.7 Аналіз швидкості обчислення без прискорювача обчислень

Аналіз ефективності системи з використанням апаратного прискорювача зображено на рисунку 3.8. Використання розробленого апаратного ядра в межах паралельної системи дозволяє кардинально змінити динаміку обчислень за рахунок великої системи паралельних обчислень де кожен елемент може працювати незалежно один від одного, притаманного архітектурі FPGA. Результати тестування, демонструють оптимальний баланс між швидкістю обробки та часом доступу до пам'яті.

Оптимізація часу виконання: Час безпосередньої роботи обчислювального ядра Kernel work time було скорочено до 0.004952 ms. Це свідчить про високу швидкість проходження даних через апаратні блоки множення. Завдяки глибокій конвеєризації вдалося досягти оптимізованого обчислювального процесу, для виконуваня арифметичних операції над новими елементами масиву на кожному такті.

Стабільна пропускна здатність показник продуктивності на рівні 11. 680108 GIOPS при мінімальних часових витратах демонструє ефективну ініціалізацію обчислювальних ресурсів. Хоча це значення не є піковим для даної архітектури, воно демонструє стабільність роботи прискорювача при збалансованому навантаженні.

Мінімальні затримки пам'яті під час запису в пам'ять Memory write time склав лише 0.005144 ms. У порівнянні зі сценарієм без прискорювача де цей показник сягав 0.015616 ms, спостерігається покращення швидкодії обміну даними у 2.5 рази. Це досягається завдяки ефективному використанню контролера пам'яті та прямих зв'язків між обчислювальним ядром і шиною AXI.

Загальний аналіз підтверджує, що активація апаратного прискорювача дозволяє не тільки скоротити час обробки алгоритму, але й значно оптимізувати роботу підсистеми пам'яті. Таке поєднання апаратної гнучкості FPGA та керуючої логіки процесора HPS забезпечує високу паралельність часу виконання, вбудованих систем.

```
18 45 10 24 8 15 18 4 18 54 24 9 2 14 10 72 30 35 14 21 27 6 64 35 24 42 21 16 9 45 12 12
 63 8 7 5 4 20 27 16 21 72 5 36 36 56 24 5 18 5 42 32 12 10 24 9 45 3 56 6 2 24 6 8 6 24
1 36 72 20 27 24 24 64 15 30 42 3 48 54 18 36 16 24 9 20 24 49 9 24 16 30 15 72 10 49 9 3
0 18 48 14 9 35 4 24 9 14 20 42 16 36 18 6 20 28 8 72 4 54 5 5 48 12 54 4 32 30 35 42 48
35 18 42 42 36 4 8 18 16 8 8 30 18 3 40 24 8 16 4 14 6 2 56 32 21 10 81 3 42 35 16 72 16
56 8 28 16 18 5 3 56 49 7 6 5 24 72 10 72 15 10 32 56 49 32 45 36 8 72 54 6 20 28 12 14 2
56 8 45 45 21 3 21 24 21 9 15 10 40 15 54 12 21 7 64 56 8 14 9 24 10 40 8 9 45 56 54 14 2
15 9 20 5 42 24 7 8 3 24 30 49 25 40 72 4 40 21 8 18 48 9 10 8 48 42 6 49 2 14 36 15 35 4
40 48 18 6 4 30 32 21 56 20 32 54 81 12 20 25 48 32 5 8 63 30 9 3 25 12 56 72 9 32 4 48 9
25 81 7 35 63 42 45 14 10 20 10 45 48 27 54 20 25 9 7 8 28 32 15 28 28 10 7 5 8 8 6 45 6
21 16 63 32 12 14 64 54 72 8 36 24 18 30 27 5 14 21 2 16 5 35 15 20 8 30 14 27 7 9 12 2 1
20 28 16 12 14 14 48 18 18 4 32 2 36 48 24 8 16 4 6 16 28 12 16 9 45 27 6 35 40 16 15 16
72 36 18 12 15 72 32 10 2 10 32 18 40 24 18 18 9 35 9 25 30 27 9 3 35 8 18 24 64 15 28 18
4 14 30 7 63 20 4 7 8 15 36 16 16 9 18 4 30 36 3 4 5 81 6 30 5 40 32 27 21 24 8 45 27 18
72 12 30 7 72 10 36 30 35 18 20 21 9 16 18 24 45 63 32 4 45 15 6 56 16 12 72 35 2 1 45 5
 5 5 72 15 12 21 12 9 2 24 49 20 18 4 14 4 56 36 8 45 54 42 36 42 32 49 4 54 72 54 40 8 1
9 10 8 54 9 5 4 40 64 28 18 28 32 4 30 42 1 32 27 6 9 28 54 6 12 56 27 48 63 48 18 40 28
4 14 10 20 3 35 30 20 54 32 21 7 18 12 3 36 18 25 21 6 12 3 6 18 56 16 56 36 6 49 28 18 2

work efficiency
Kernel work time: 0.004952 ms
Throughput:            11.680108 GIOPS
Memory write time:  0.005144 ms
```

Рисунок. 3.8 Аналіз швидкості обчислення з прискорювачем обчислень

Проведені випробування розробленого програмно технічного засобу продемонстрували, що інтеграція обчислювальних потужностей FPGA у загальну структуру системи забезпечує перевагу над традиційними архітектурами, побудованими виключно на базі центральних процесорів. Ключовий ефект досягається завдяки зміні методу обробки інформації.

Показники продуктивності апаратного прискорювача під час роботи з великими масивами даних підтвердили здатність системи підтримувати високу інтенсивність обчислень, що раніше було доступно лише для стаціонарних робочих станцій.

При цьому розроблене рішення зберігає всі критично важливі переваги вбудованих систем: надзвичайно компактні габарити, відсутність потреби у складних системах охолодження та низьке енергоспоживання, що робить його ідеальним для автономних об’єктів.

Завдяки застосуванню інструментарію FPGA SDK та тонкому налаштуванню параметрів компіляції, час чистого виконання обчислювального ядра було мінімізовано до вражаючого показника 0.005144 ms. Окрім швидкості безпосередньо математичних операцій, вагомим досягненням стала оптимізація роботи з підсистемою пам'яті. Апаратна інтеграція та використання локальної кеш-пам'яті всередині ПЛІС дозволили скоротити затримки при записі та читанні даних у 2.5 рази порівняно з базовими, неоптимізованими режимами роботи.

Розроблена архітектура ефективно нівелює накладні витрати на передачу інформації через швидкісні шини AXI. Використання дворівневої буферизації та механізмів прямого доступу до пам’яті дозволило мінімізувати вплив інтерфейсних затримок на загальний цикл обчислень. Таким чином, створений прискорювач не лише демонструє високу сиру потужність, а й забезпечує високу детермінованість процесів, що підтверджує надійність та ефективність запропонованого підходу до проєктування систем.

3.5 Висновок третього розділу

У третьому розділі було виконано практичну частину розробки, імплементації та тестування фізично програмованого прискорювача обчислень на базі архітектури SoC. Встановлено, що розроблена архітектура забезпечує високоефективну взаємодію між інтелектуальним центром керування HPS під управлінням ОС Linux та обчислювальною потужністю матриці FPGA. Для успішної реалізації необхідно використання високошвидкісних шин та мостів AXI, які виконують роль апаратних конверторів даних, що дозволило подолати мовний та структурний бар'єр між процесорною частиною та програмованою логікою.

Процес проектування обчислювального ядра kernel базувався на створенні масиву паралельних робочих елементів. На відміну від послідовного виконання в класичних CPU, розроблена архітектура ядра на базі OpenCL дозволила реалізувати пряме звернення до глобальної пам'яті Global Memory через систему вказівників, ініціалізованих Host програмою. Це забезпечило максимально швидку підготовку даних для матричних операцій та задіяння виділених DSP-блоків чипа для виконання арифметичних дій.

Етап високорівневого синтезу та формування бінарного образу, став цифровою реалізацією архітектури, що на фізичному рівні визначив стан логічних комірок ПЛІС, перебудувавши їх під конкретне завдання паралельного множення матриць.

У фінальній стадії було проведено порівняльний аналіз який показав, що прискорювач значно підвищував швидкість обробки даних. Впровадження розробленого прискорювача дозволило побудувати оптимізований апаратний процес, який значно підвищив показник пропускної здатності. Практична реалізація підтвердила теоретичні розрахунки та довела високу ефективність обраного стеку технологій для вирішення складних обчислювальних задач.

ВИСНОВКИ

У межах даної дипломної роботи було проведено комплексний аналіз програмно-технічного інструментарію, необхідного для розробки та функціонування сучасних прискорювачів обчислень. На основі отриманих теоретичних знань та практичних навичок було спроєктовано та реалізовано авторський апаратний прискорювач на базі високорівневої мови опису OpenCL. Для практичної апробації та розгортання розробленої архітектури було використано платформу terasic DE10 standart на базі чипа intel cyclone V для реалізації більш складної і оптимізованої системи.

У першому розділі проведено аналіз програмно технічних засобів прискорювача їх видів та принципів їх роботи на основі FPGA. У ході збору інформації було детально розглянуто проблему гострої необхідності у спеціалізованих обчислювальних засобах, зумовлену постійним зростанням обсягів даних та складністю сучасних алгоритмів, які вимагають значних ресурсів, що часто виходять за межі можливостей класичних центральних процесорів. Було виконано порівняльний аналіз різних видів прискорювачів обчислень, зокрема графічних процесорів GPU, спеціалізованих інтегральних схем ASIC та програмованих логічних інтегральних схем FPGA, що дозволило визначити найбільш перспективні типи архітектур для створення вбудованих систем.

Було досліджено внутрішню структуру FPGA та принципи його роботи, заснованих на використанні конфігурованих логічних блоків та систем між з’єднань. Ця теоретична основа дозволила виділити ключову перевагу FPGA е можливjcns формування апаратної структури, що максимально точно відповідає топології конкретного алгоритму, забезпечуючи тим самим високу паралельність обчислень. Проведений аналіз став основою для подальшого вивчення матеріалу.

У другому розділі проведено детальне проєктування архітектури системи прискорювача та обґрунтовано вибір інструментарію для її реалізації. Першочергово було здійснено глибоке ознайомлення з технічними можливостями та функціоналом платформи Terasic DE 10 standart Cyclone V, що дозволило визначити оптимальні шляхи використання ресурсів системи на кристалі, зокрема інтеграцію двоядерного процесора Cortex-A9 з програмованою логікою через швидкісні мости AXI.

У ході роботи було досліджено програмно-обчислювальні методи реалізації, що базуються на багато потоковій моделі обчислень, де основне навантаження розподіляється між хост процесором та апаратним ядром. Також було оцінено різні види програмних засобів розробки та синтезу, зокрема середовищу Intel FPGA SDK for OpenCL та Intel SoC EDS, які забезпечують високорівневий синтез алгоритмів у бітстрім для FPGA.

У третьому розділі виконано практичну реалізацію та експериментальне дослідження фізично програмного прискорювача обчислень. У рамках цього етапу було розроблено архітектуру системи, що базується на тісній інтеграції апаратних ресурсів FPGA та керуючого програмного забезпечення на стороні хост. Було описано процес підготовки та структурування вхідних масивів даних, що забезпечило їхню коректну обробку.

Розробка архітектури обчислювального ядра, оптимізованого для виконання паралельних операцій, дозволила ефективно задіяти логічні блоки FPGA. Після завершення синтезу було проведено практичну реалізацію алгоритму, де було підтверджено працездатність створеного рішення у реальних умовах. Заключним етапом стало комплексне тестування та аналіз результатів, за результатами якого зафіксовано пікову продуктивність. Отримані дані підтвердили кратне скорочення затримок при зверненні до пам'яті у 2.5 рази порівняно з роботи комп’ютера без прискорювача, що доводить ефективність.

Поставлена мета проєкту була повністю досягнута, а розроблене технічне рішення демонструє високу ефективність у межах масивних обчислень. Було успішно досліджено паралельне проєктування апаратних прискорювачів. Завдяки ґрунтовній теоретичній підготовці та практичній реалізації, було детально засвоєно принципи роботи високорівневого синтезу HLS та специфічні особливості використання мови OpenCL для генерації високоефективної апаратної логіки.

Реалізація власного прискорювача дозволила на практиці вивчити складну архітектуру взаємодії між процесорною системою HPS та програмованою логікою FPGA через спеціалізовані інтерфейси. Також керування ресурсами кристала, і розподіл пам'яті

Отриманий досвід розробки, інсталяції та налагодження системного оточення у середовищі Intel SoC EDS дозволив глибше зрозуміти фундаментальні принципи роботи комп’ютерних систем та програмованих плат і оптимізації обчислювальних процесів. Це надало навички для виконання прикладних задач де виникає потреба у швидкій обробці великих масивів даних при обмеженому енергоспоживанні. Вивчення технології OpenCL для FPGA відкрило нові перспективи у сфері автоматизації проєктування, дозволивши поєднати програмну гнучкість із апаратною швидкістю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1) Куцевський С. М., Ратайчук П. Є. Прискорювачі штучного інтелекту: сучасний стан та галузі застосування. *Сучасна молодь в світі інформаційних технологій* : матеріали [конференції]. 2021. С. 55.

2) Boutros A., Betz V. FPGA architecture: Principles and progression. *IEEE Circuits and Systems Magazine*. 2021. Vol. 21, no. 2. P. 4–29.

3) Li Z. [та ін.]. A survey of FPGA design for AI era. *Journal of Semiconductors*. 2020. Vol. 41, no. 2. P. 021402.

4) Cong J. [та ін.]. FPGA HLS today: successes, challenges, and opportunities. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*. 2022. Vol. 15, no. 4. P. 1–42.

5) Farooq U., Marrakchi Z., Mehrez H. FPGA architectures: An overview. *Tree-Based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*. 2012. P. 7–48.

6) Seng K. P., Lee P. J., Ang L. M. Embedded intelligence on FPGA: Survey, applications and challenges. *Electronics*. 2021. Vol. 10, no. 8. P. 895.

7) Zhang B., Kannan R., Prasanna V. BoostGCN: A framework for optimizing GCN inference on FPGA. *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* : [Proceedings of the International Symposium]. 2021.

8) Wan Z. [та ін.]. A survey of fpga-based robotic computing. *IEEE Circuits and Systems Magazine*. 2021. Vol. 21, no. 2. P. 48–74.

9) Bobda C. [та ін.]. The future of FPGA acceleration in datacenters and the cloud. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*. 2022. Vol. 15, no. 3. P. 1–42.

10) Zha Y., Li J. Virtualizing FPGAs in the cloud. *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020. P. 845–858.

11) Wang C., Luo L. A review of the optimal design of neural networks based on FPGA. *Applied Sciences*. 2022. Vol. 12, no. 21. P. 10771.

12) Fang J. [та ін.]. In-memory database acceleration on FPGAs: a survey. *The VLDB Journal*. 2020. Vol. 29, no. 1. P. 33–59.

13) Elkhatib R. [та ін.]. Cryptographic engineering a fast and efficient SIKE in FPGA. *ACM Transactions on Embedded Computing Systems*. 2024. Vol. 23, no. 2. P. 1–25.

14) Carayol Rodríguez R. *Developement of a SOC acquisition system* : [dissertation / thesis]. 2025.

15) Cock D. [та ін.]. Enzian: an open, general, CPU/FPGA platform for systems software research. *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2022. P. 434–451.

16) Li A., Wentzlaff D. PRGA: An open-source FPGA research and prototyping framework. *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2021. P. 127–137.

17) Lin W. [та ін.]. SuperNIC: An FPGA-based, cloud-oriented SmartNIC. *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 2024. P. 130–141.

18) Molanes R. F., Rodríguez-Andina J. J., Fariña J. Performance characterization and design guidelines for efficient processor–FPGA communication in Cyclone V FPSoCs. *IEEE Transactions on Industrial Electronics*. 2017. Vol. 65, no. 5. P. 4368–4377.

19) Vaarandi R., Mäses S. How to Build a SOC on a Budget. 2022 IEEE International Conference on Cyber Security and Resilience (CSR) : [Proceedings of the International Conference]. 2022. P. 171–177.

20) Chakravarthi V. S. A practical approach to VLSI system on chip (SoC) design. Cham : Springer International Publishing, 2020.

21) Erdogan A. T. [та ін.]. A high dynamic range 128×120 3-D stacked CMOS SPAD image sensor SoC for fluorescence microendoscopy. IEEE Journal of Solid-State Circuits. 2022. Vol. 57, no. 6. P. 1649–1660.

22) Tianxu Y. Convolutional neural network FPGA-accelerator on Intel DE10-standard FPGA. 2021.

23) Design and implementation of FPGA configuration logic block using asynchronous static NCL / I. P. Dugganapally, W. K. Al-Assadi, T. Tammina, S. Smith. 2008 IEEE Region 5 Conference : conference paper, 17–20 April 2008, Kansas City, MO, USA. IEEE, 2008. P. 1–6.

24) Razi K. F., Schmid A. Epileptic seizure detection with patient-specific feature and channel selection for low-power applications. IEEE Transactions on Biomedical Circuits and Systems. 2022. Vol. 16, no. 4. P. 626–635.

25) Sukarno A. N. [та ін.]. High Accuracy LPC Encoder Architecture for FPGA. 2023 International Conference on Artificial Intelligence, Blockchain, Cloud Computing, and Data Analytics (ICoABCD) : [Proceedings of the International Conference]. 2023. P. 186–191.

26) Sprang F., Latzel T., Englberger F. FPGA and RISC-V integration in electrical engineering curriculum. ICERI2024 Proceedings. 2024. P. 2560–2568.

27) Abushanab S., Hadoush A. Hardware Implementation of Newton-Raphson Method for Nonlinear Stress Estimation. Wadi Alshatti University Journal of Pure and Applied Sciences. 2025. P. 244–250.

28) Hsu J. Y., Jiang T. Y., Chao P. C. P. A fast FPGA hardware accelerator for remote heart rate detection based on RGB vision. IEEE Transactions on Biomedical Circuits and Systems. 2024. Vol. 18, no. 3. P. 592–607.

29) Lebedev M., Belecky P. A survey of open-source tools for FPGA-based inference of artificial neural networks. 2021 Ivannikov Memorial Workshop (IVMEM) : [Proceedings of the Workshop]. 2021. P. 50–56.

30) Campaña de la Cadena J. F. Diseño e implementar una plataforma digital Real Time-HIL en base a FPGA de la línea Altera para aplicações de sistemas electrónicos de potência. 2026.

31) Madhavan P. V. Fpga based ultrasonic non-destructive testing. 2022.

32) Špaček M., Sedláček R., Roztročil J. Implementation of a time-to-digital converter inside FPGA. 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS) : [Proceedings of the International Conference]. 2023. P. 79–83.

33) Dong D. [та ін.]. Cost-optimized heterogeneous FPGA architecture for non-iterative hologram generation. Applied Optics. 2020. Vol. 59, no. 25. P. 7540–7546.

34) Mendes L. A., Figueiredo M., Menotti R. Análise de técnicas de predição de desvio sob a arquitetura RISC-V. Simpósio em Sistemas Computacionais de Alto Desempenho (SSCAD). 2024. P. 9–16.

35) Johnson C. W. FPGA-Based Accelerator Generation for Artificial Neural Networks. The Florida State University, 2024.

36) Pan S. T., Wu H. J. FPGA Chip Design of Sensors for Emotion Detection Based on Consecutive Facial Images by Combining CNN and LSTM. Electronics. 2025. Vol. 14, no. 16. P. 3250.

37) Шевченко В. О. Програмно-технічний модуль перетворення даних в систему залишкових класів на базі FPGA. 2024.

38) Величко Н. В. Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA. 2026.

39) Шпуляр Є. М. Метод створення SVM-класифікатора для аналізу даних на базі FPGA. 2024.

40) Ванярха О. С. Метод побудови архітектури вбудованих систем обробки зображень на основі FPGA. 2023.

41) Атаманюк О. Метод побудови апаратної архітектури для системи комп’ютерного зору на основі FPGA. 2023.

42) Караченцев К. В. Апаратний прискорювач математичних операцій на платформі SoC. 2025.

43) Тецький А. Г., Перепелицин А. Є. Можливості використання апаратних прискорювачів у системах виявлення та запобігання вторгненням. 2024.

44) Snider R. Chapter 6: Introduction to intel quartus prime. Advanced Digital System Design using SoC FPGAs: An Integrated Hardware/Software Approach. Cham : Springer International Publishing, 2022. P. 55–86.

45) Xie Y., Zhou J., Kang S. Design and application of digital phase locked loop based on Quartus II. IOP Conference Series: Earth and Environmental Science. 2020. P. 012026.

46) Alpay A., Heuveline V. SYCL beyond OpenCL: The architecture, current state and future direction of hipSYCL. Proceedings of the International Workshop on OpenCL. 2020. P. 1–1.

47) Nishitsuji T. Basics of OpenCL. Hardware Acceleration of Computational Holography. Singapore : Springer Nature Singapore, 2023. P. 83–95.

48) Breyer M., Van Craen A., Pflüger D. A comparison of sycl, opencl, cuda, and openmp for massively parallel support vector machine classification on multi-vendor hardware. Proceedings of the 10th International Workshop on OpenCL. 2022. P. 1–12.

49) Young-Schultz T. [та ін.]. Using OpenCL to enable software-like development of an FPGA-accelerated biophotonic cancer treatment simulator. Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2020. P. 86–96.

50) Solanti J. [та ін.]. PoCL-R: A scalable low latency distributed OpenCL runtime. International Conference on Embedded Computer Systems. Cham : Springer International Publishing, 2021. P. 78–94.

51) Ahmed U. [та ін.]. A load balance multi-scheduling model for OpenCL kernel tasks in an integrated cluster. Soft Computing. 2021. Vol. 25, no. 1. P. 407–420.

52) FPGA-прискорювачі злітають у «хмари». APEPS КПІ. URL: https://apeps.kpi.ua/downloads/FPGA.pdf (дата звернення: 16.05.2026).

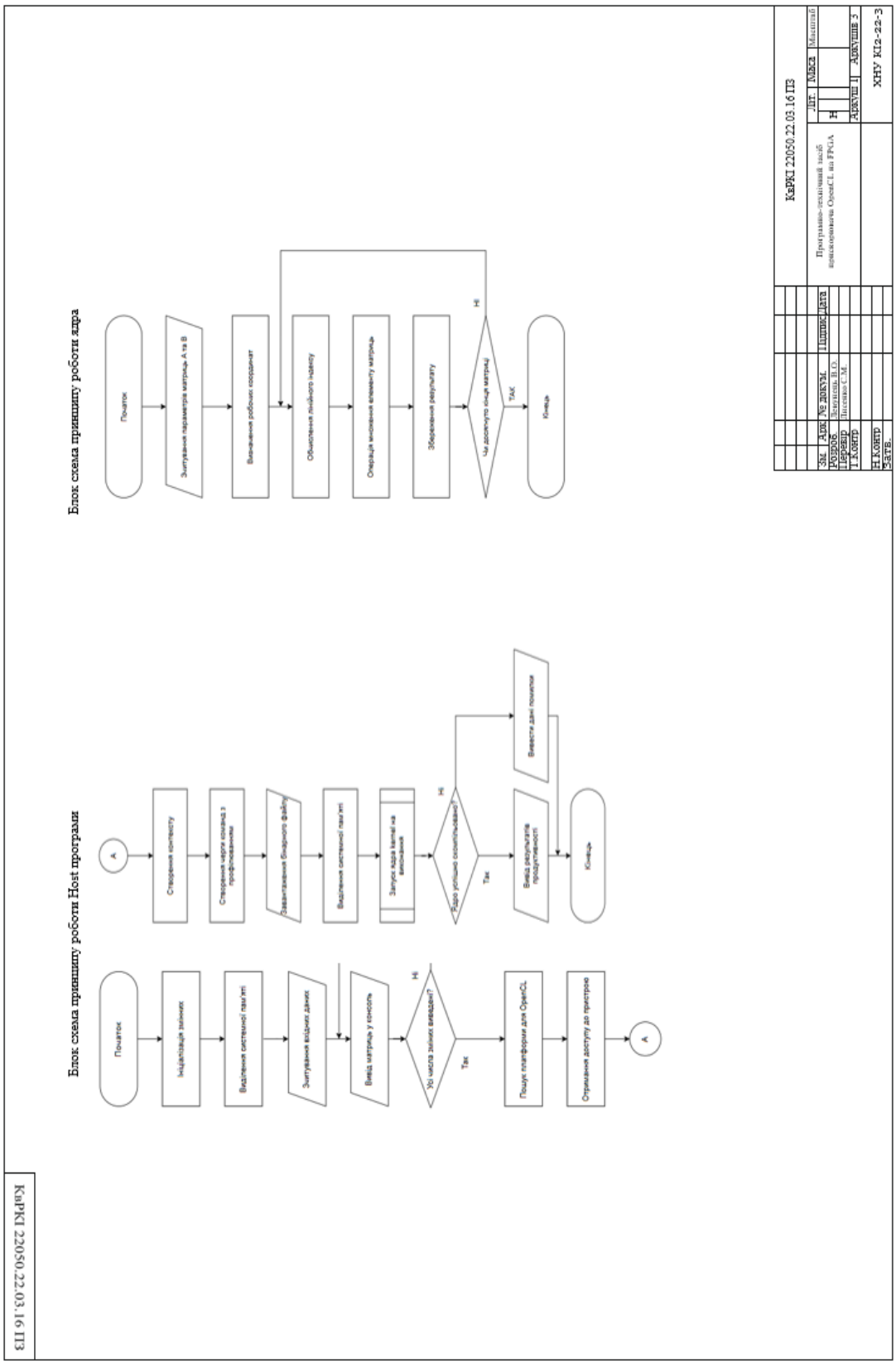
53) Hamblen J. O. Using second generation SOPC boards for student design projects. *2005 IEEE International Conference on Microelectronic Systems Education* : proceedings of the international conference. IEEE, 2005. P. 69–70.

54) DE10-Standard Development and Education Kit. *Terasic*. URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=1081 (дата звернення: 16.05.2026).

ДОДАТОК А

(обов’язковий)

Копія креслення «Архітектура ПЗ Прискорювача обчислень»

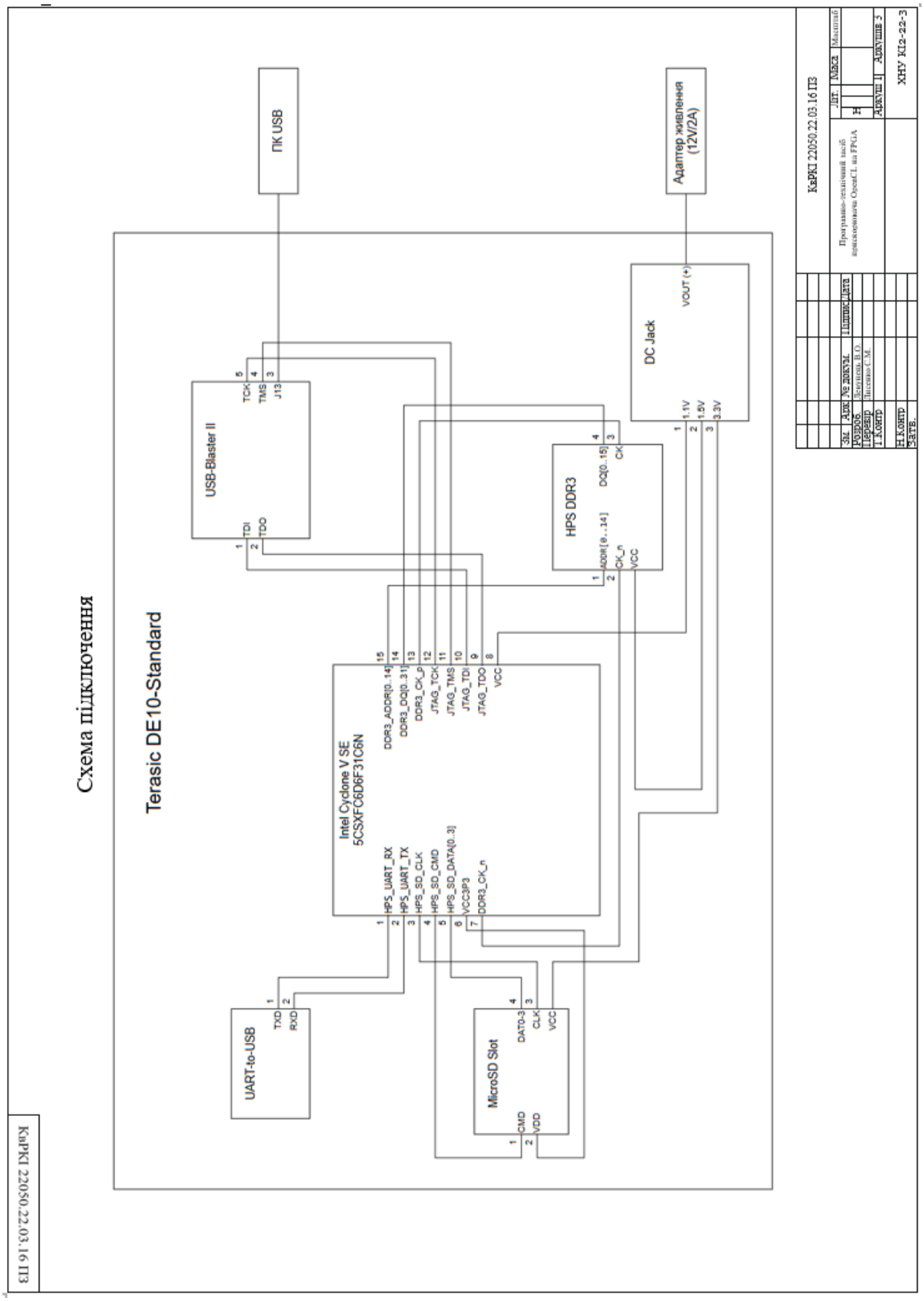


КвРКІ 22050.22.03.16 ПЗ

ДОДАТОК Б

(обов’язковий)

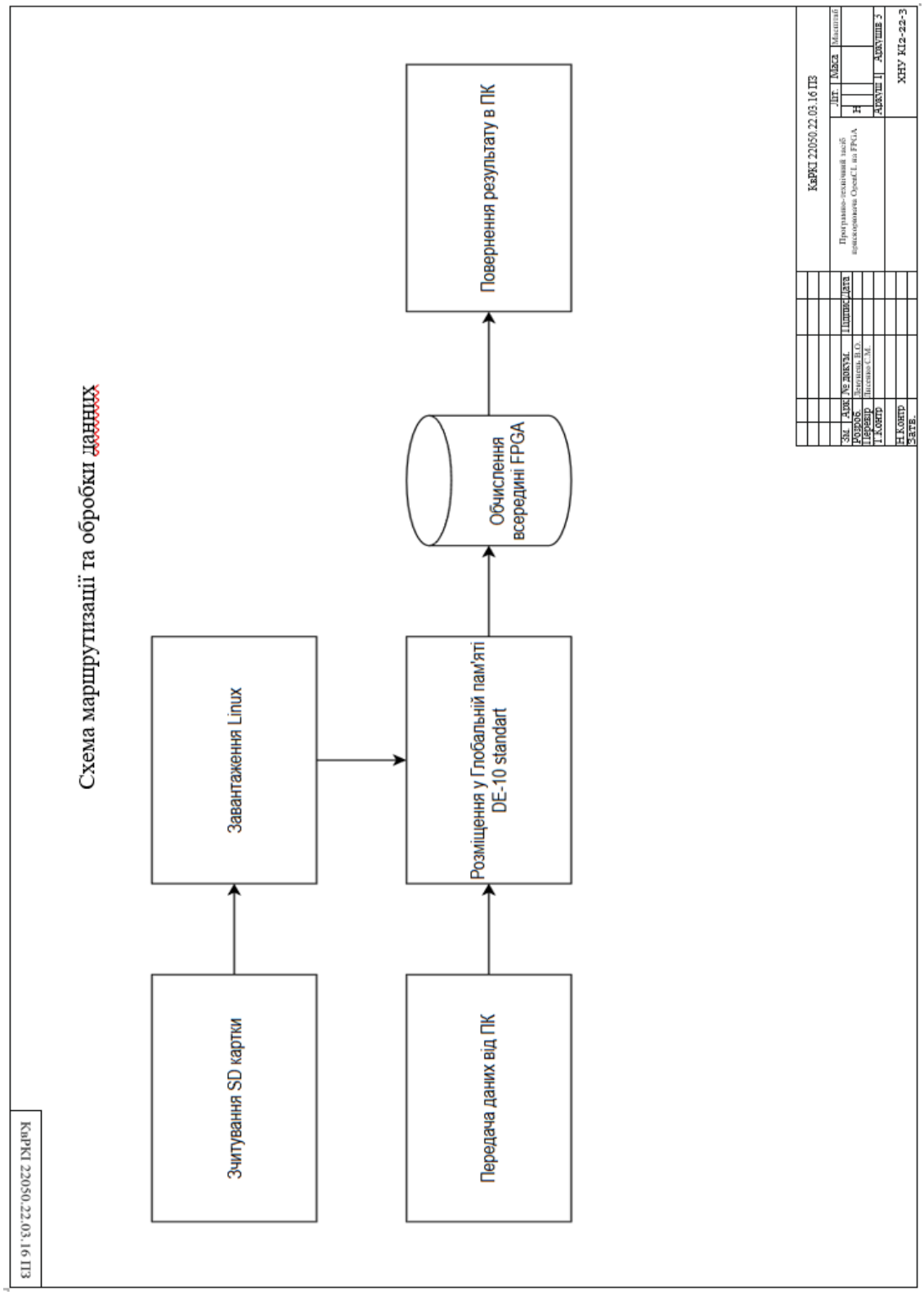
Копія креслення «Схема підключення програмного забезпечення»



ДОДАТОК В

(обов’язковий)

Копія креслення «Схема маршрутизації та обробки данних»



ДОДАТОК Г

(обов’язковий)

Копія креслення «Лістинг коду Host програми»

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#include <time.h>

#define MAX_SOURCE_SIZE (0x100000)

int main() {
    srand((unsigned int)time(NULL));
    const int N = 100;
    const int LIST_SIZE = N * N;
    size_t size_in_bytes = sizeof(int) * LIST_SIZE;

    // Виділення пам'яті
    int* A = (int*)malloc(size_in_bytes);
    int* B = (int*)malloc(size_in_bytes);
    int* C = (int*)malloc(size_in_bytes);

    for (int i = 0; i < LIST_SIZE; i++) {
        A[i] = rand() % 9 + 1;
        B[i] = rand() % 9 + 1;
    }

    printf("Matrix A:\n");
    for (int j = 0; j < N; j++) {
        for (int i = 0; i < N; i++) printf("%d ", A[j * N + i]);
        printf("\n");
    }

    printf("\nMatrix B:\n");
    for (int j = 0; j < N; j++) {
        for (int i = 0; i < N; i++) printf("%d ", B[j * N + i]);
        printf("\n");
    }

    cl_int ret;
    cl_platform_id platform_id;
    cl_device_id device_id;
    clGetPlatformIDs(1, &platform_id, NULL);
    clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_ALL, 1, &device_id, NULL);
```

```
cl_context context = clCreateContext(NULL, 1, &device_id, NULL, NULL, &ret);

// Створення черги з підтримкою профілювання
cl_command_queue queue = clCreateCommandQueue(context, device_id, CL_QUEUE_PROFILING_ENABLE, &ret);

// Завантаження бінарного файлу .aocx для FPGA
printf("Loading kernel.aocx for FPGA...\n");
FILE* fp = fopen("kernel.aocx", "rb");
if (fp == NULL) {
    fprintf(stderr, "Error: Could not find kernel.aocx file!\n");
    return -1;
}

fseek(fp, 0, SEEK_END);
size_t binary_size = ftell(fp);
rewind(fp);

unsigned char* binary_buf = (unsigned char*)malloc(binary_size);
fread(binary_buf, 1, binary_size, fp);
fclose(fp);

cl_int binary_status;
cl_program program = clCreateProgramWithBinary(
    context, 1, &device_id, &binary_size,
    (const unsigned char**)&binary_buf, &binary_status, &ret
);

// Обов'язковий виклик для ініціалізації бінарного коду
clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
free(binary_buf);

// 3. Створення буферів та пересилка даних
cl_event event_write, event_kernel, event_read;

cl_mem a_mem = clCreateBuffer(context, CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, size_in_bytes, A,
&ret);
// Використовуємо event_write для відстеження часу запису другого буфера
cl_mem b_mem = clCreateBuffer(context, CL_MEM_READ_ONLY, size_in_bytes, NULL, &ret);
ret = clEnqueueWriteBuffer(queue, b_mem, CL_TRUE, 0, size_in_bytes, B, 0, NULL, &event_write);

cl_mem c_mem = clCreateBuffer(context, CL_MEM_WRITE_ONLY, size_in_bytes, NULL, &ret);

// 4. Запуск обчислень
cl_kernel kernel = clCreateKernel(program, "matrix_mul_elementwise", &ret);
clSetKernelArg(kernel, 0, sizeof(cl_mem), &a_mem);
clSetKernelArg(kernel, 1, sizeof(cl_mem), &b_mem);
```

```
clSetKernelArg(kernel, 2, sizeof(cl_mem), &c_mem);
clSetKernelArg(kernel, 3, sizeof(int), &N);

size_t global_size[2] = { (size_t)N, (size_t)N };
clEnqueueNDRangeKernel(queue, kernel, 2, NULL, global_size, NULL, 0, NULL, &event_kernel);

// 5. Читання результату
clEnqueueReadBuffer(queue, c_mem, CL_TRUE, 0, size_in_bytes, C, 0, NULL, &event_read);

printf("\nResult Matrix C:\n");
for (int j = 0; j < N; j++) {
    for (int i = 0; i < N; i++) {
        printf("%d ", C[j * N + i]);
    }
    printf("\n");
}

// 6. Розрахунок метрик
cl_ulong start, end;
clGetEventProfilingInfo(event_kernel, CL_PROFILING_COMMAND_START, sizeof(cl_ulong), &start, NULL);
clGetEventProfilingInfo(event_kernel, CL_PROFILING_COMMAND_END, sizeof(cl_ulong), &end, NULL);
double time_kernel = (double)(end - start) / 1.0e6;

clGetEventProfilingInfo(event_write, CL_PROFILING_COMMAND_START, sizeof(cl_ulong), &start, NULL);
clGetEventProfilingInfo(event_write, CL_PROFILING_COMMAND_END, sizeof(cl_ulong), &end, NULL);
double time_write = (double)(end - start) / 1.0e6;

double operations = (double)N * N;
double giops = (operations / (time_kernel / 1000.0)) / 1.0e9;

printf("\nWork efficiency\n");
printf("Kernel work time: %f ms\n", time_kernel);
printf("Throughput: %f GIOPS\n", giops);
printf("Memory write time: %f ms\n", time_write);

// 7. Очищення
clReleaseEvent(event_kernel);
clReleaseEvent(event_write);
clReleaseEvent(event_read);
clReleaseMemObject(a_mem);
clReleaseMemObject(b_mem);
clReleaseMemObject(c_mem);
clReleaseKernel(kernel);
clReleaseProgram(program);
clReleaseCommandQueue(queue);
clReleaseContext(context);
```

```
    free(A); free(B); free(C);

    return 0;
}
```

Копія креслення «Лістинг коду Kernel програми»

```
__kernel void matrix_mul_elementwise(
    __global const int* A,
    __global const int* B,
    __global int* C,
    const int N)
{
    int x = get_global_id(0);
    int y = get_global_id(1);

    if (x < N && y < N) {
        int id = y * N + x;
        C[id] = A[id] * B[id];
    }
}
```

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Віталій ЛЕВУНЕЦЬ

Співавтор:

Назва: Програмно-технічний засіб прискорювача openCL на FPGA

Експерт: Сергій ЛИСЕНКО

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:3.42%

Коефіцієнт подібності 2:0.72%

Мікропробіли: 9

Заміна букв: 2

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-20 06:09:12.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-05-20 Доцент Андрій Нічепорук

Дата експерт

Wed May 20 06:54:58 EEST 2026, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 271757 Назва: БКР Програмно-технічний засіб прискорювача openCL на FPGA Додано в БД: 2026-05-20 Автора: Віталій ЛЕВУНЕЦЬ Керівники: Сергій ЛИСЕНКО Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	93408	630	2051 (2%)	29 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Левунець Віталій Олександрович

Тема: Програмно-технічний засіб прискорювача openCL на FPGA

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___56___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є створення програмно технічного пристрою на платі FPGA для прискорення математичних обчислень із великим обсягом даних.

2. Висновок про відповідність роботи дипломному завданню: Було успішно створено програмно технічний прискорювач обчислень математичних операцій.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проведено теоретичний аналіз сучасних засобів прискорення обчислень. Переваги програмованих логічних інтегральних схем, а також вивчено внутрішню структуру й переваги паралельних обчислень на базі програмованої логіки. У другому розділі здійснено проєктування архітектури прискорювача та обрано інструменти для реалізації. Проведено аналіз технічних можливостей цільової апаратної платформи. У третьому розділі виконано практичну частину роботи та проведено експерименти. Реалізовано архітектуру обчислювального ядра, та налаштовано структурування вхідних даних та інтеграцію апаратної логіки з керуючою програмою. Вкінці проведено тестування системи, яке підтвердило працездатність прискорювача та зафіксувало значне підвищення продуктивності під час роботи з пам'яттю.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: недостатня увага аналізу предметної області; недостатньо чітко описано процес складання програмно-технічного засобу.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на достатньому технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: задовільно (D / 70)

Рецензент (прізвище, ім`я, по батькові, посада, місце роботи) _____

Сєдлюк Микола Васильович, PhD, ст. викладач, к. к. Зарубежжя

"28" 05 2026 р. _____ (підпис)

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Віталій ЛЕВУНЕЦЬ
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-22-3

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

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

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Програмно-технічний засіб прискорювача openCL на FPGA
Автор Віталій ЛЕВУНЕЦЬ
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти перший (бакалаврський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: д.т.н., професор Сергій ЛИСЕНКО

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

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

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 3,42%; та системою Anti-Plagiarism складає 1.0%, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

01.06.2026

Завідувач кафедри — Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми — Підпис — Андрій НІЧЕПОРУК (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи — Підпис — Сергій ЛИСЕНКО (Ім'я, ПРІЗВИЩЕ)