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

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему: «Спеціалізована система сортування даних на базі ПЛІС»

КвРКІ. 202130.22.02.32 ПЗ

Виконав: студент 2 курсу, група КІ2м-22-2 Горбач Г.С.
Ініціали, прізвище

Керівник: кандидат техн. наук, доцент Грига В.М.
Науковий ступінь, вчене звання Підпис Ініціали, прізвище

До захисту допускаю:
Зав. кафедри КІІС, д.т.н., проф.
Т.О. Говорущенко
__ _____ 2024 р.

Хмельницький, 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень магістр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

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

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О. Говорущенко

"01" 09 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Горбач Галині Степанівні

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Спеціалізовані системи сортування даних на базі ПЛІС

Керівник проекту (роботи) Грига М.В. к.т.н. доцент

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

Затверджена наказом ректора університету від 09.01.2024 р. №1

2. Строк подання студентом проекту (роботи) на кафедру 01.05.2024 р.

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

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

Аналіз відомих методів та засобів сортування даних на базі ПЛІС

Моделювання процесів сортування за допомогою спеціалізованих систем

Розроблення удосконаленого спеціалізованого методу сортування на базі ПЛІС

Розроблення інформаційних технологій та засобів сортування на базі ПЛІС

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

6. Консультанти розділів кваліфікаційної роботи магістра

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

7. Дата видачі завдання «_01_»_____09_____2023р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	01.09.2023	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.10.2023	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	01.11.2023	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	01.12.2023	виконано
5	Робота над науковою статтею	01.02.2024	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2024	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	01.04.2024	виконано
8	Оформлення пояснювальної записки згідно вимог	18.04.2024	виконано
9	Попередній захист ДРМ	25.04.2024	виконано
10	Захист ДРМ на засіданні ЕК	До 23.05.2024	

Студент _____ Горбач Г.С.
Підпис Ініціали, прізвище

Керівник (проекту) роботи _____ Грига В.М.
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Спеціалізована система сортування на базі ПЛІС.

Автор роботи: Горбач Галина Стеанівна

Керівник роботи: кандидат техн. наук, доцент Грига Володимир Михайлович.

Пояснювальна записка: 80 с., 38 рис., 1 табл., 6 дод., 80 джерел.

ПЕРЕЛІК КЛЮЧОВИХ СЛІВ: системи й алгоритми сортування, сортування даних на базі ПЛІС, потоковий граф, рекурсивний граф, автомат Мура, сортування «бульбашкою», керуючий автомат, апаратні системи сортування.

Об'єктом дослідження є процес проектування спеціалізованої системи сортування даних на основі просторово-часових графів.

Предметом дослідження є методи та алгоритми сортування масивів двійкових даних.

Метою кваліфікаційної роботи магістра є проектування і дослідження спеціалізованої системи сортування даних на базі ПЛІС.

Для розв'язання поставлених задач використовувалися методи сортування даних, аналіз ефективності цих методів, експериментальні дослідження сортувальних систем, систематичний огляд літератури та дослідів.

Наукова новизна отриманих результатів:

− набув подальшого розвитку рекурсивний метод сортування масивів двійкових даних на базі просторово-часових графів, який на відміну від відомих дозволяє ефективно оптимізувати кількість операцій сортування, що забезпечує низьку апаратну складність спеціалізованої системи;

− удосконалено структуру спеціалізованої системи сортування двійкових даних на основі графів, яка завдяки використанню удосконалених методів, функціональних моделей паралельно-потокового сортування та урахуванню інтенсивності надходження даних, розмірів масивів даних і

можливостей реалізації дозволяє здійснювати сортування даних у реальному часі з високою ефективністю використання обладнання.

Практична значимість отриманих результатів полягає у розроблені спеціалізованої системи сортування даних на базі ПЛІС.

Рекомендації з використання результатів роботи. Отримані результати можуть бути використані у розробці системи сортування даних на базі ПЛІС для оптимізації швидкодії та ефективності обробки великих обсягів даних.

Важливість роботи і висновки. Робота має значення для систем сортування даних на основі ПЛІС. Це дозволяє значно зменшити кількість операцій алгоритму сортування, що є критичним у багатьох областях, таких як обробка великих обсягів даних, комп'ютерні додатки або мережеві протоколи. Використання такої системи дозволяє ефективно використовувати ресурси обладнання та прискорювати обчислення в реальному часі.

Публікації. За темою кваліфікаційної роботи опубліковано одну публікацію [1].

Структура та об’єм кваліфікаційної роботи. Кваліфікаційна робота складається з вступу, чотирьох розділів, висновку та додатків, її повний зміст сторінок, основний зміст викладено на 80 сторінці, 6-х додатках, містить 38 рисунків, 1 таблицю, включає 80 найменувань вітчизняної та зарубіжної літератури.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЛІС (FPGA) –програмовані логічні інтегральні схеми

ВСС (HLS) – високорівневий синтез

БД – база даних

ЦП – центральний процесор

ГПУ (GPU) – графічний процесор

ЦП (CPU) – центральний процесор

ОП – операційний пристрій

ШІ – штучний інтелект

ОС – операційна система

ПЗ – програмне забезпечення

АЛП (ALU) - арифметико-логічний пристрій

НВІС - надвелика інтегральна схема

ПЗ - програмне забезпечення

ФО - функціональні оператори

ПГА - потоковий граф алгоритму

ВСТУП

Постійний прогрес у розвитку обчислювальної техніки веде до постійного прагнення до підвищення швидкості та ефективності обробки даних. Сортування даних відіграє ключову роль у багатьох обчислювальних задачах, таких як пошук, аналіз і обробка інформації. У цьому контексті виникає потреба у розробці спеціалізованих систем сортування, що забезпечують високу швидкість та ефективність при обробці великих обсягів даних.

Сортування є важливою операцією для багатьох вбудованих додатків, які працюють у реальному часі. Це одна з найбільш вивчених проблем у сфері інформатики. Для деяких додатків, таких як авіоніка та системи підтримки прийняття рішень, це може вважатися великою перевагою. Однак сортування великої кількості елементів і прийняття рішень у реальному часі вимагає надзвичайної швидкості обробки. Одним з потенційних рішень для реалізації спеціалізованих систем сортування є використання програмованих логічних інтегральних схем (ПЛІС). ПЛІС надають можливість створення спеціалізованих апаратних реалізацій, оптимізованих під конкретні завдання, що дозволяє досягти значних приростів у продуктивності та швидкодії обчислень.

Метою кваліфікаційної роботи є проектування і дослідження спеціалізованої системи сортування даних на базі ПЛІС.

Поставлена мета досягається розв’язанням таких основних задач:

- дослідити відомі методи сортування даних;
- дослідити відомі методи сортування даних, їх переваги та недоліки;
- дослідити бази ПЛІС як засіб для вирішення задачі;
- розробити спеціалізовану систему сортування даних на базі ПЛІС;
- протестувати та оцінити ефективність системи сортування даних на базі ПЛІС, написаною на мові програмування VHDL.

Об’єктом дослідження є процес проектування спеціалізованої системи сортування даних на основі просторово-часових графів.

Предметом дослідження є методи та алгоритми сортування масивів

двійкових даних.

Наукова новизна отриманих результатів:

− набув подальшого розвитку рекурсивний метод сортування масивів двійкових даних на базі просторово-часових графів, який на відміну від відомих дозволяє ефективно оптимізувати кількість операцій сортування, що забезпечує низьку апаратну складність спеціалізованої системи;

− удосконалено структуру спеціалізованої системи сортування двійкових даних на основі графів, яка завдяки використанню удосконалених методів, функціональних моделей паралельно-потокового сортування та урахуванню інтенсивності надходження даних, розмірів масивів даних і можливостей реалізації дозволяє здійснювати сортування даних у реальному часі з високою ефективністю використання обладнання.

Практичне значення отриманих результатів проявляється в можливості створення оптимальної системи сортування даних на основі ПЛІС. Це дозволяє значно зменшити кількість операцій алгоритму сортування, що є критичним у багатьох областях, таких як обробка великих обсягів даних, комп'ютерні додатки або мережеві протоколи. Використання такої системи дозволяє ефективно використовувати ресурси обладнання та прискорювати обчислення в реальному часі.

За темою кваліфікаційної роботи магістра опубліковані тези в конференції «ISCM–2023 Грига В.М., Горбач Г.С. Спеціалізована система сортування двійкових даних» [1].

1 АНАЛІЗ ВІДОМИХ МЕТОДІВ ТА ЗАСОБІВ СОРТУВАННЯ МАСИВІВ ДАНИХ

1.1 Теоретичне визначення системи сортування: аспекти та особливості

Сортування наразі важливе для організації даних у більш зручний і структурований спосіб. Це процес, що допомагає систематизувати дані шляхом розташування їх у певному логічному порядку [2]. Під час сортування елементи однотипної послідовності даних, такі як масив, список або файл, перегруповуються таким чином, щоб вони відповідали певному критерію порядку.

Алгоритм сортування - це конкретний метод, за допомогою якого елементи розташовуються в певному порядку. Цей порядок може бути числовим (за зростанням або спаданням значень), лексикографічним (за алфавітом) або будь-яким іншим, який задає користувач. Мета сортування полягає в полегшенні подальшої роботи з даними, такої як пошук певного елемента, оновлення значень, виключення або включення нових елементів у структуру даних.

Відсортовані дані полегшують виконання багатьох операцій. Наприклад, на відсортованих даних легше виявити пропущені елементи або перевірити, чи всі елементи присутні [3]. Також, вони дозволяють швидше знаходити спільні елементи між двома або більше структурами даних, а також об'єднувати їх для подальшого використання.

Загалом, сортування є важливим інструментом для оптимізації роботи різних алгоритмів, особливо тих, які часто працюють з великим обсягом даних. Відсортовані дані дозволяють більш ефективно виконувати операції, які вимагають доступу до певних елементів у структурі даних.

Існує безліч алгоритмів сортування, проте вибір конкретного для певного випадку залежить від ряду факторів. Ефективність алгоритму сортування може значно варіюватися залежно від таких чинників:

– кількість сортованих елементів: деякі алгоритми можуть бути ефективнішими для невеликої кількості елементів, тоді як інші можуть працювати

краще для великих наборів даних;

− діапазон та розподіл значень елементів: Деякі алгоритми можуть бути ефективнішими для певних розподілів значень, наприклад, якщо значення розподілені рівномірно або якщо вони мають особливі властивості, такі як багато повторюваних значень;

− ступінь початкової відсортованості елементів: Деякі алгоритми можуть працювати швидше на вже відсортованому масиві, тоді як інші можуть бути ефективнішими при неупорядкованому наборі даних;

− характеристики алгоритму: Складність алгоритму, вимоги до пам'яті та інші характеристики можуть впливати на вибір. Наприклад, деякі алгоритми можуть мати кращу часову складність, а інші - кращі вимоги до пам'яті;

− місце розміщення елементів: Розміщення елементів, чи то в оперативній пам'яті (у вигляді масиву або списку), чи на диску (у вигляді файлу), також може впливати на вибір алгоритму.

Визначення найефективнішого методу сортування у кожному конкретному випадку залежить від цих параметрів та встановлення пріоритетів для конкретної задачі [4]. Наприклад, алгоритм, який добре працює з великими обсягами даних, може бути більш вигідним для застосування у додатках з великою кількістю записів, тоді як інший метод може бути оптимальним для використання у випадках обмежених ресурсів пам'яті.

Згідно з останнім фактором, всі методи сортування можна розділити на два класи: внутрішнє сортування та зовнішнє сортування. Класифікація цих методів відбувається на основі того, де розташовані дані, які піддаються сортуванню, і чи можуть вони одночасно поміститися в оперативну пам'ять комп'ютера.

Внутрішнє сортування використовується тоді, коли весь обсяг даних, що потребує сортування, може бути оброблений безпосередньо в оперативній пам'яті комп'ютера. Таким чином, всі елементи, які необхідно відсортувати, зберігаються в оперативній пам'яті. Це дозволяє застосовувати різноманітні алгоритми сортування, такі як швидке сортування (quicksort), сортування злиттям (merge sort), або сортування вставками (insertion sort), забезпечуючи ефективну обробку

даних без перебільшень обсягу пам'яті.

У випадку зовнішнього сортування, весь обсяг даних, який необхідно відсортувати, не може бути одночасно розміщений в оперативній пам'яті комп'ютера. Ці дані можуть зберігатися на зовнішньому носії, такому як жорсткий диск або інший зовнішній пристрій зберігання. Сортування проводиться частинами: блоки даних зчитуються зі зовнішнього носія, сортуються в оперативній пам'яті, а потім записуються назад на зовнішній носій. Для цього типу сортування застосовуються спеціальні алгоритми, такі як зовнішнє сортування злиттям (external merge sort) або зовнішнє сортування методом впорядкування нульових перетворень (distribution sort). Такий підхід дозволяє ефективно сортувати великі обсяги даних, які не поміщаються повністю в оперативну пам'ять [5].

Основними вимогами до алгоритмів сортування, як і до будь-яких інших алгоритмів, є здатність оптимально використовувати пам'ять та ефективно керувати часом виконання. Це означає, що при внутрішньому сортуванні елементів масиву процес сортування має відбуватися без створення додаткового масиву, тобто елементи сортування мають бути оброблені "на місці".

Одним з ключових показників ефективності алгоритму сортування є кількість порівнянь ключів (C) та кількість пересилань (M), які визначаються за кількістю сортованих елементів (n). Ці параметри є важливими для оцінки продуктивності алгоритму та порівняння різних методів сортування. Зменшення кількості порівнянь та пересилань може призвести до значного покращення часу виконання алгоритму, особливо при роботі з великими обсягами даних. Таким чином, важливо розробляти та вибирати алгоритми, які оптимізовані з точки зору використання ресурсів пам'яті та часу виконання.

Дані можуть бути організовані за допомогою певної властивості, яка визначається як ключ сортування. Цей ключ може складатися з одного або кількох полів у записах даних. У практичних застосунках часто виникає необхідність сортувати масив записів за допомогою кількох ключів. Це стає важливим тоді, коли одного ключа недостатньо для чіткої ідентифікації записів.

Наприклад, у великих компаніях може знадобитися впорядкувати список співробітників, спочатку за їхнім відділом, а потім в межах кожного відділу — за алфавітним порядком їхніх імен. Такий підхід допомагає зберегти структурованість та зрозумілість даних для подальшого використання.

Інші приклади сортування за допомогою декількох ключів можуть включати:

- телефонні довідники: Довідники, де імена впорядковані спочатку за місцем розташування (наприклад, за містом), після чого за категорією (наприклад, бізнес або громадяни) і, нарешті, в алфавітному порядку;
- бібліотечний каталог: інформація про книги може бути впорядкована в алфавітному порядку за назвою книги, а потім за іменами авторів;
- адресні списки клієнтів: адреси клієнтів можуть бути відсортовані спочатку за назвою населеного пункту, далі за назвою вулиці, а потім за номером будинку.

Таким чином, повне сортування та його ключ можуть бути визначені за допомогою двох або більше ключів часткового сортування. У таких випадках перший ключ відомий як первинний ключ сортування, тоді як наступні ключі називаються вторинними ключами сортування. Використання декількох ключів дозволяє досягти більшої точності та адаптабельності у впорядкуванні даних, адже враховуються різні аспекти і умови сортування.

1.2 Відомі методи й алгоритми сортування масивів даних

Однією з основних передумов сортування даних є потреба у швидкій обробці великих обсягів інформації, зокрема в системах моніторингу запасів та фінансових програмах, де дані генеруються в режимі реального часу і обробляються без зберігання у сховищі[6]. За зростання обсягів даних потрібно використовувати спеціалізовані прискорювачі, такі як ГПУ, ПЛІС або багатоядерні процесори, для оптимізації обробки.

Сортування є важливим етапом в обчислювальних завданнях, і для відповіді

на вимоги продуктивності останнім часом активно вивчаються швидкі прискорювачі на основі різних технологій. Крім того, мережі сортування знаходять широке застосування в обробці даних, що дозволяє виконувати порівняння та обмін даними паралельно [6].

У випадку застосування платформи ПЛІС потокова обробка зазвичай інтегрується з традиційними потоковими процесорами на рівні сервера. Проблема, що виникає, полягає у розробці основної апаратно-програмної платформи виконання, яка сприятиме оптимізації розміщення операторів сортування для досягнення різних цілей, таких як пропускна здатність та затримка. Ключовим атрибутом тут є продуктивність згенерованої апаратної схеми, яка має бути чітко визначена.

На різних етапах розробки програми використовуються алгоритми сортування, такі як системи підтримки прийняття рішень, планування маршрутів, планування тощо. Однак складність і цільова платформа (платформи) виконання є основними критеріями продуктивності для таких алгоритмів.

Різні платформи, такі як ЦП (одно чи багатоядерний), ГПУ (графічний процесор) і ПЛІС (програмовна логічна інтегральна схема), можуть бути використані для виконання алгоритмів сортування. ПЛІС є найбільш популярною платформою для реалізації процесу сортування, оскільки вона забезпечує підвищену продуктивність у реальному часі та ефективне використання енергії. За допомогою ПЛІС можна створювати складні програми з високою продуктивністю, що базуються на масивно-паралельних архітектурах. Використання інструментів високорівневого синтезу (HLS) також сприяє підвищенню продуктивності розробки проектів на основі ПЛІС [7].

Зокрема, інтеграція гетерогенних систем на кристалах (SoC) та прискорювачів дозволяє підвищити продуктивність проектування, зменшити обмеження часу виходу на ринок та відокремити алгоритм від архітектури. Мова програмування високого рівня, така як C / C ++, сприяє створенню апаратних прискорювачів. Алгоритм сортування є важливим процесом для багатьох вбудованих програм реального часу, існує багато алгоритмів (рисунок 1.1), таких

як BubbleSort, InsertionSort, SelectionSort, MergeSort, які можна використовувати в залежності від вимог до продуктивності та обсягу даних.

Рисунок - 1.1. Класифікація методів сортування

Кожна цільова платформа, така як процесори, графічні процесори, ПЛІС та гібридна платформа, має свої власні переваги. ПЛІС визначається як найкраща платформа з точки зору енергоефективності, тоді як центральний процесор вважається простішою платформою для програмування. Графічний процесор представляє собою середнє рішення.

Вибір алгоритму сортування залежить від кількості сортованих елементів. Наприклад, якщо кількість елементів невелика, рекомендується використовувати InsertionSort, в іншому випадку рекомендується застосовувати MergeSort.

Результати з різних досліджень, спрямованих на паралельне виконання

алгоритмів сортування на різних архітектурах (ЦП, ГПУ та ПЛІС) [8], показують, що швидкість збільшується пропорційно кількості процесорів. Ці дослідження демонструють, що реалізація на апаратному рівні на ПЛІС демонструє кращу продуктивність як з точки зору часу, так і енергозбереження. Проте в деяких випадках максимальне прискорення може бути обмеженим через обмежену можливість розпаралелювання, особливо коли ми маємо справу з невеликою кількістю елементів для сортування.

Аналіз відомих алгоритмів сортування включає вивчення їхніх теоретичних властивостей, ефективності, складності та придатності для конкретних завдань сортування даних. Великий обсяг досліджень у цій області спрямований на розуміння різних аспектів таких алгоритмів, включаючи їхню часову та просторову складність, стійкість до різних типів вхідних даних та їх вплив на загальну продуктивність систем. Отож, розглянемо певні з них.

Бульбашкове сортування (BubbleSort) [9] є відомим алгоритмом у світі обчислень, але воно стає неефективним при сортуванні великої кількості елементів через свою квадратичну складність $O(n^2)$ у середньому та найгіршому випадку. Процес бульбашкового сортування можна розділити на чотири етапи. Першим кроком є висування всіх елементів на вихідні дані. Другий етап полягає у порівнянні двох сусідніх елементів та їх обміні місцями, якщо вони не відсортовані. Наступною дією є порівняння двох елементів. Нарешті, застосовується суматор для визначення більшого числа у порівнянні. Ці кроки повторюються до тих пір, поки масив не буде відсортованим.

Алгоритм сортування вибором відрізняється своєю простотою та легкістю розуміння, порівняно з іншими методами сортування. Це робить його привабливим для розв'язання завдань з невеликою кількістю елементів. Проте, він виявляється неефективним у випадку великих об'ємів даних через квадратичну складність, що означає, що кількість порівнянь залежить від квадрату кількості елементів у масиві. Його складність становить $O(n^2)$, де n – це кількість елементів. Цей алгоритм, також відомий як SelectionSort [10], працює шляхом вибору мінімального елемента на кожному кроці сортування. Однією з ключових

особливостей є фіксація мінімального значення у початковому індексі. Алгоритм знаходить мінімальний елемент у списку та переміщає його на відповідне місце. Після цього процес повторюється, збільшуючи мінімальний індекс, доки масив не буде впорядкованим.

Сортування вставками (InsertionSort) [10] - це ще один простий алгоритм сортування, який часто використовується для сортування невеликої кількості елементів. Хоча, він має кращу продуктивність порівняно з сортуванням бульбашкою і сортуванням вибором, це алгоритм все ж залишається менш ефективним у випадку сортування великої кількості елементів. У таких випадках варто використовувати більш оптимізовані алгоритми, такі як швидке сортування (QuickSort), пірамідальне (HeapSort) або сортування злиттям (MergeSort), оскільки його складність дуже важлива $O(n^2)$ у середньому та гіршому випадку.

Алгоритм сортування вставками використовується для інтеграції нового елемента в кожній ітерації порівняти значення елементів у списку. значення елемента менше поточного значення цього елемента, тоді виконується перемикання. Повторюється цей крок до $n-1$ елемента.

Сортування Шелла (ShellSort) [11] є алгоритмом сортування, який вважається дуже ефективним для середньої кількості елементів. Він є вдосконаленням алгоритму сортування вставками, оскільки дозволяє переміщати елементи, розташовані на відстані. Середня та найгірша складність цього алгоритму становлять $O(n(\log n)^2)$. Основна ідея полягає в тому, щоб обчислити значення h і розділити список на менші підсписки з однаковими інтервалами h. Потім кожен підсписок, що містить багато елементів, сортується за допомогою сортування вставками. Цей процес повторюється, доки список не буде впорядкований. На відміну від деяких інших алгоритмів, цей алгоритм не зустрічається часто у практиці й літературі.

Швидке сортування [12] базується на поділі масиву на два підмасиви: нижчі елементи та вищі елементи. Алгоритм можна розділити на декілька кроків. По-перше, обирається елемент з масиву , який називається опорним (pivot). Наступний крок - розділення: всі менші елементи переміщуються перед опорним,

а всі більші - після нього. Після цього опорний елемент опиняється на своєму кінцевому місці, що є операцією розділення. Наступний крок полягає в рекурсивному повторенні цих кроків для двох підмасивів з меншими та більшими значеннями елементів. Таким чином, це алгоритм "розділяй і володарюй". На практиці швидке сортування виявляється швидшим за інші алгоритми, такі як сортування буьбашкою або сортування вставками, оскільки його складність становить $O(n\ log\ n)$. Проте, його реалізація не є стабільною і вимагає складних дій. Також вибір опорного елемента може впливати на продуктивність алгоритму. Якщо на кожному кроці швидке сортування вибирає медіану як опорний елемент, то складність буде $O(n\ log\ n)$, але неправильний вибір опорних точок може призвести до поганої продуктивності (часова складність $O(n^2)$).

Пірамідальне сортування (HeapSort) базується на тому ж принципі, що й сортування вибору , оскільки він шукає максимальний елемент у списку і розміщує його у кінці. Цей процес повторюється для решти елементів. Проте HeapSort є кращим алгоритмом сортування, оскільки його складність становить $O(n\ log(n))$. Алгоритм HeapSort можна розділити на два основні етапи:

− створення структури так званої купи (Max-Heap або Min-Heap) з першим елементом купи, який є найбільшим або найменшим (залежно від типу купи). Цей етап діє за принципом "розділяй і володарюй";

− повторення процесу з рештою елементів, щоб знову вибрати перший елемент купи та перемістити його у кінці масиву, поки весь масив не буде відсортованим;

− хоча HeapSort є швидким алгоритмом сортування, він не є стабільним. Цей алгоритм часто використовується для сортування великих масивів даних.

У 1945 році Джон фон Нейман [13] створив алгоритм сортування злиттям (MergeSort). Цей алгоритм зберігає порядок елементів від введення до виведення, що робить його ефективним і стабільним. Він ґрунтується на відомій парадигмі "розділяй і володарюй". Основними кроками цього алгоритму є: 1) Розділення масиву на два підмасиви; 2) Рекурсивне сортування цих двох підмасивів; 3) Об'єднання двох відсортованих підмасивів, щоб отримати результат. Сортування

злиттям вважається кращим алгоритмом сортування, ніж пірамідальне , швидке, Шилла і гібридне (TimSort) сортування, оскільки його складність у середньому та найгіршому випадках дорівнює $O(n\,log(n))$.

Гібридне сортування (TimSort), заснований на алгоритмах сортування злиттям і сортування включення (Insertion-Sort), відрізняється своєю спроможністю перемикатися між цими двома алгоритмами в залежності від оптимального параметра (ОП). Для архітектури процесора Intel i7 цей параметр фіксується на рівні 64. Час виконання практично однаковий для паралельних архітектур при зміні значення параметра ОП. Таким чином, у цьому дослідженні вважається, що значення ОП дорівнює 64, оскільки кілька досліджень використовують це стандартизоване значення. Проте можемо застосовувати два різні підходи в залежності від розміру елементів, які потрібно відсортувати: якщо розмір масиву перевищує 64 елементи, то використовується алгоритм MergeSort; у протилежному випадку на кроці сортування використовується InsertionSort.

Для використання алгоритму бульбашкового сортування в паралельному середовищі [14], часто застосовується його модифікація, відома як метод парно-непарної перестановки (the odd-even transposition method). Ця модифікація передбачає введення двох різних правил для виконання ітерацій алгоритму: залежно від парності або непарності номера ітерації сортування, обираються елементи з парними або непарними індексами відповідно. Порівняння значень, що відбуваються, здійснюються завжди з їх правими сусідніми елементами. Таким чином, на непарних ітераціях порівнюються пари $(a_1a_2)(a_3a_4),..,(a_{n-1}a_n)$ (при парному n), а на парних ітераціях обробляються елементи $(a_2a_3)(a_4a_5),..,(a_{n-2}a_{n-1})$.

Після проведення n ітерацій сортування над початковим набором даних, він виявляється у відсортованому стані.

З аналізу алгоритмів сортування видно, що кожен з них має свої унікальні характеристики та властивості. Деякі алгоритми, такі як QuickSort, відомі своєю швидкодією та ефективністю на практиці, але можуть мати складність у реалізації. Інші, такі як BubbleSort або InsertionSort, можуть бути простими для

реалізації, але мають меншу ефективність на великих об'ємах даних. При виборі алгоритму сортування важливо враховувати конкретні потреби завдання, розмір вхідних даних та характеристики цільової системи.

1.3 Аналіз методів сортування для великих масивів даних

Дослідження проведені в роботі підтвердило, що для сортування великих наборів даних часто використовують комбінацію зовнішнього сортування та алгоритмів прямого та швидкого сортування, які були описані у розділі 1.2. Оскільки дані для сортування зберігаються на зовнішніх пристроях і мають великий обсяг, який не можна повністю завантажити в оперативну пам'ять для сортування, потрібно застосовувати різноманітні стратегії [15]. У таких випадках дані розбиваються на частини, які можна обробити в оперативній пам'яті, сортуються та записуються в тимчасові файли. Після цього відбувається фаза злиття, під час якої всі ці тимчасові файли об'єднуються в один великий файл.

Найпоширенішими методами зовнішнього сортування є:

- сортування злиттям (просте та природне злиття);
- покращене сортування (багатофазове та каскадне сортування). Ключовим поняттям при зовнішньому сортуванні є ідея серії.

Серія (впорядкована підмножина) - це послідовність елементів, впорядкована за ключем.

Кількість елементів у серії називається довжиною серії. Серія, яка складається з одного елемента, завжди впорядкована. Остання серія може мати меншу довжину, ніж решта серій у файлі. Максимальна кількість серій у файлі дорівнює N, де N - кількість всіх не впорядкованих елементів. Мінімальна кількість серій дорівнює одній (всі елементи вже впорядковані).

Більшість методів зовнішнього сортування базуються на процедурах злиття та розподілу. Злиття - це процес об'єднання двох або більше впорядкованих серій у один впорядкований файл, вибираючи елементи, доступні у даний момент. Розподіл - це процес розділення впорядкованих серій на два або більше

допоміжних файли.

Крім того, важливим є поняття фази, що означає операцію з одноразовою обробкою всіх елементів послідовності. В залежності від виконання фаз алгоритми можуть бути класифіковані як двофазне сортування - коли фази розподілу та злиття виконуються окремо, та однофазне сортування - коли фази розподілу та злиття об'єднуються в одну.

Щоб покращити продуктивність алгоритмів зовнішнього сортування, одним з ключових методів є застосування паралелізму та його принципів:

- для підвищення продуктивності послідовного зчитування та запису даних доцільно використовувати паралельну роботу кількох дисків, що сприятиме зниженню витрат на розробку IT-інфраструктури;
- застосування сучасних багатоядерних процесорів та оптимізації програмного забезпечення дозволить підвищити швидкість операцій сортування завдяки ефективному використанню потоків;
- використання асинхронного введення-виведення при розробці програмного забезпечення сприяє підвищенню швидкості завдяки одночасному сортуванню даних під час запису в одному процесі та читанню у іншому;
- завдяки використанню декількох машин, які пов'язані між собою високошвидкісною мережею, сортування даних може виконуватися паралельно, що сприяє підвищенню продуктивності.

Розглянемо додаткові методи [16]: для поліпшення продуктивності сортування:

- зменшення кількості операцій пошуку на диску можливе за рахунок використання більшого обсягу оперативної пам'яті;
- використання сучасного обладнання, такого як швидкі зовнішні накопичувачі та тверді диски, дозволяє зменшити час сортування, хоча може підвищити загальні витрати на IT;
- оптимальний вибір та використання сумісного обладнання для створення IT-інфраструктури допомагає скоротити час сортування. Слід враховувати такі параметри, як швидкість, кількість ядер процесора, пропускна

здатність та інші;

– забезпечення балансу між економічною ефективністю та швидкістю можна досягти через використання кластерів та хмарових сервісів, де низька ціна забезпечується завдяки великій кількості вузлів кластера;

– оптимізація кількості вхідних та вихідних операцій, а також тимчасових файлів, сприяє зменшенню часу сортування даних;

– використання стандартизованого підходу до паралелізму на всіх етапах сортування;

– максимальне програмне розпаралелення під час сортування даних;

– оптимізація програмного коду за допомогою використання правильних типів даних та інших методів.

1.4 Основи методології проектування пристроїв сортування на основі просторово-часових графів.

Граф $G = (N, A)$ - це сукупність двох множин (об'єктів): $N = \{n_1, n_2 \dots n_n\}$ - скінчена непуста множина вершин (вузлів); $A = \{a_1, a_2 \dots a\}$ - скінчена множина пар, що з'єднані між собою і утворють ребро (дугу) графу G [17].

На рисунку 1.2 подана класифікація обчислювальних графів.

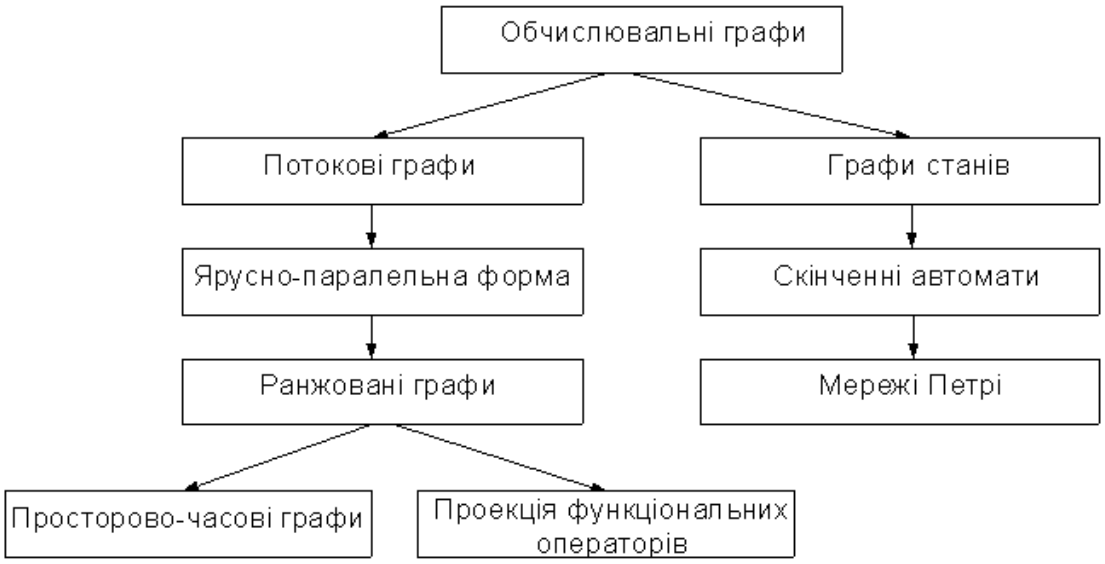


Рисунок 1.2 - Класифікація обчислювальних графів

Обчислювальні графи поділяються на потокові та графи станів. Потокові графи використовуються для проектування операційних пристроїв, тоді як графи станів використовуються для керуючих пристроїв. Недоліками потокових графів є обмеженість до бінарних операцій та відсутність розрізнення номерів входів до вершин, ускладнюючи створення ієрархічних вершин з багатьма входами та виходами. Графи станів, зазвичай, представляють собою скінченні абстрактні автомати Мура та Мілі.

Для максимізації паралелізму алгоритму важливо представити його у формі, що відображає його обчислювальні та структурні характеристики. Характеристики обчислень визначаються всіма функціональними операціями (вершинами) алгоритму і їх обчислювальними затримками. Структурні характеристики визначаються зв'язками між функціональними блоками та їх взаємозв'язками з даними.

Одним з можливих способів подання алгоритму є графічне представлення. На рисунку 1.3 показаний граф певного алгоритму [18]. Кожна вершина графа відображає функціональний оператор, а кожна дуга представляє зв'язок між функціональними операторами. Вершина, яка відповідає функціональному оператору з'єднується з вершиною, яка відповідає функціональному оператору , тільки в разі, коли результат, отриманий після виконання оператора є одним із аргументів для оператора .

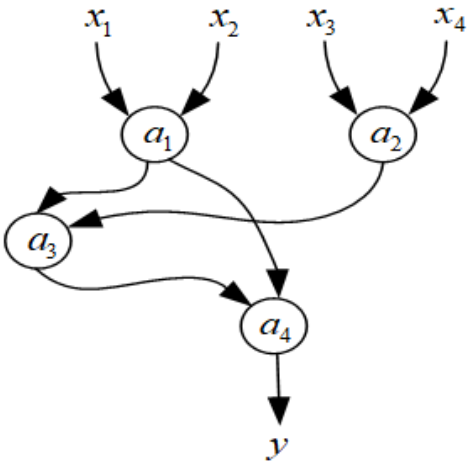


Рисунок 1.3 – Граф алгоритму

Граф алгоритму у вигляді ярусно-паралельної структури дозволяє виявляти і управляти паралелізмом, що дозволяє знаходити компромісні рішення між використанням обчислювальних ресурсів та часом виконання.

Ярусно-паралельна форма графа [19] - це структура, в якій вершини розподілені на яруси таким чином, що всі вершини в межах одного ярусу не мають прямих зв'язків між собою. У такій формі графу вершини в кожному ярусі залежать від результатів попереднього ярусу, але не залежать від вершин наступних ярусів. У межах кожного ярусу функціональні оператори не мають прямих зв'язків між собою. Ярусно-паралельна форма визначає ступінь паралелізму графа, що визначається максимальною кількістю вершин на одному ярусі, а також мінімально можливий час обчислення алгоритму, що визначається кількістю ярусів. Такий граф також називають потоковим. На рисунку 1.4 показано потоковий граф для раніше розглянутого алгоритму.

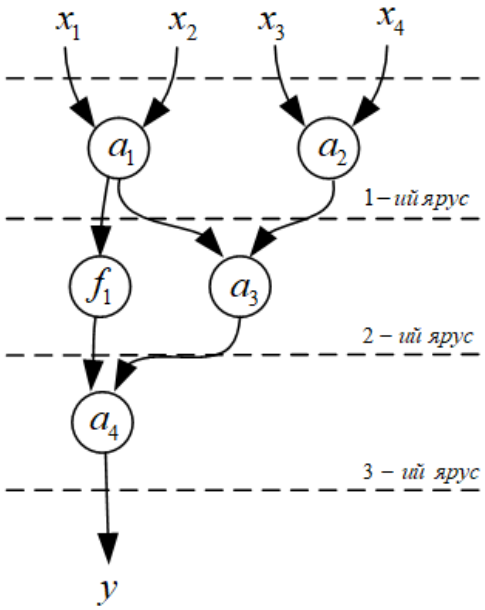


Рисунок 1.4 – Потоковий граф алгоритму

У представленому потоковому графі можна виділити два основних типи функціональних операторів: операційні (позначені символом (a_1, a_2, a_3, a_4)) для виконання конкретних операцій та перепускні (позначені символом (f_1)) для передачі інформації з входу на вихід. Ці графи дозволяють уявити алгоритми, які

не залежать від початкових даних, такі як алгоритми без галузей або такі, що залишаються незмінними під час зсуву.

Створення рекурсивних систем може бути важким через потенційні неузгодженості між алгоритмом і структурою системи. Для цього типу проектів застосовують просторово-часові графи (ПЧГ) для формалізації.

Просторово-часові графи мають вершини, дуги і вузли, які відрізняються залежно від типу графу: послідовний, паралельний або рекурсивний [20]. Ці графи оцінюються за двома параметрами: пропускна здатність та час затримки. На рисунку 1.5 показано послідовний просторово-часовий граф, який утворено з потокового графу алгоритму з рисунку 1.4, шляхом стиснення вершин на один рівень.

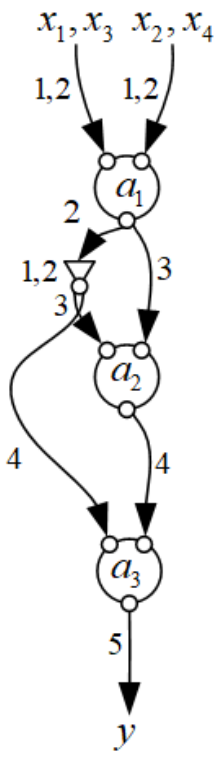


Рисунок 1.5 – Послідовний просторово-часовий граф алгоритму

Порівняно з потоковим графом, що має 4-и операційні вершини та час виконання 4-и такти з пропускною здатністю 1 такт, послідовний просторово-часовий граф складається з 3-х вершин, має час виконання 5-ть тактів та пропускну здатність 2 такти.

Цей граф іноді називають конвеєрним. через лінійну структуру вершин та послідовне виконання операцій. За такої структури кількість зовнішніх входів дорівнює двом, що обумовлено використанням операцій з двома входами та одним виходом. Однак можливе використання лише одного зовнішнього входу, що може сповільнити роботу системи [21].

На рисунку 1.6 показано паралельний просторово-часовий граф, що базується на потоковому графі алгоритму зображеному на рисунку 1.4. Під паралельним просторово-часовим графом ми розуміємо зжаття вершин потокового графу до кількості вершин, які розташовані на одному рівні і визначають його ширину [22].

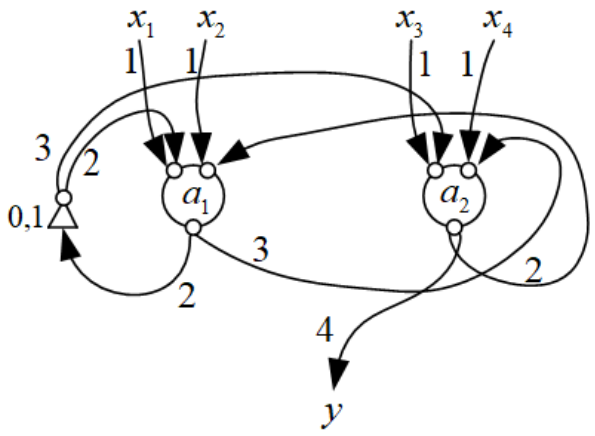


Рисунок 1.6 – Паралельний просторово-часовий граф алгоритму

В даному випадку перевага має побудова паралельного просторово-часового графу, що складається всього з 2-х вершин і має однаковий час виконання та пропускну здатність, обидва рівні 4-м тактам.

На рисунку 1.7 зображено рекурсивний просторово-часовий граф, який будується на основі потокового графу алгоритму з рисунку 1.3. Рекурсивний просторово-часовий граф об'єднує усі вершини потокового графу в єдину вершину, що послідовно виконує кожну операцію.

В даному графі є лише одна вершина, яка виконує 4-и операції послідовно у часі. Важливо зазначити, що для кожного вхідного та вихідного вузла має бути по 4-и дуги. Проте, у нашому випадку кількість дуг до вхідних вузлів менша у 2 рази,

оскільки кожен вузол працює двічі у різні моменти часу. Рекурсивний просторово-часовий граф складається з однієї вершини і має однаковий час виконання та пропускну здатність, обидва рівні 5-и тактам.

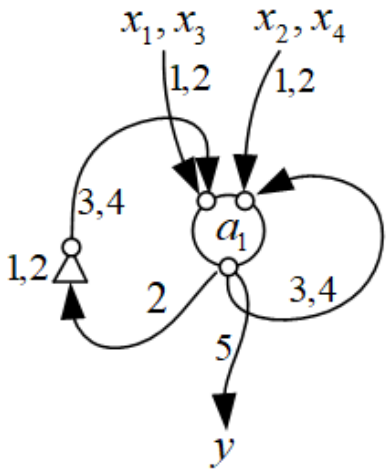


Рисунок 1.7 - Рекурсивний просторово-часовий граф алгоритму

1.5 Застосування програмованих логічних інтегральних систем для виконання операцій сортування даних

Сортування є ключовою проблемою інформатики, досліджуваною протягом багатьох десятиліть [23]. Це необхідна операція в багатьох галузях, таких як обробка зображень, мультимедіа, наукові обчислення та мережеві технології. На протязі років було запропоновано багато алгоритмів сортування, кожен з яких має свою ефективність в залежності від розміру проблеми. Хоча всі вони стають менш ефективними зі збільшенням обсягу даних, деякі працюють швидше, ніж інші.

Виконання програмного сортування на загальному комп'ютері стає неефективним через алгоритмічні відмінності у роботі та сортуванні в реальному часі [24]. Крім того, досягнення високої швидкості у програмному алгоритмі складне. Тому включення спеціалізованого апаратного блоку для сортування всередині арифметико-логічного пристрою (АЛП) стає доцільним. Ця проблема виникла на початковому етапі розвитку обчислювальної техніки. Це може пояснити велику кількість досліджень, які були проведені в цій області. Існує

кілька причин, що роблять сортування настільки важливою задачею.

По-перше, сортування є важливою складовою багатьох інших алгоритмічних проблем. Наприклад, воно використовується в бінарному пошуку та інших операціях, що потребують упорядкування даних.

По-друге, згідно деяких оцінок, приблизно 25% усіх циклів процесора витрачаються на сортування. Це свідчить про те, що оптимізація сортування може виявитися важливою для загальної продуктивності обчислювальних систем.

Крім того, різноманітні підходи до сортування можуть призвести до розробки корисних алгоритмів [25], які можна використовувати для вирішення різноманітних інших проблем. Таким чином, вивчення та оптимізація сортування може мати широкий вплив на різні галузі інформатики.

Із швидким розвитком області проектування надвисокопродуктивних інформаційно-вимірювальних систем такий модуль сортування можна реалізувати за допомогою ПЛІС на одному кристалі. ПЛІС є типом реконфігурованої вентильної матриці, що дозволяє розробнику змінювати проект реалізації з паралелізмом на рівні вентилів. Зберігання великої кількості даних здійснюється на пристрої зберігання, а відсортовані результати можуть бути відновлені з нього. Це означає, що сортування вимагає послідовної передачі даних до та з сортувальника. ПЛІС також має велику кількість пам'яті, до якої можна отримати доступ за допомогою обмеженої кількості контактів. Це дозволяє досягти кращої продуктивності та швидкості завдяки паралелізму. Сучасні ПЛІС відрізняються тим, що вони забезпечують можливість часткової реконфігурації протягом роботи. Це означає, що системи, що базуються на ПЛІС, можуть бути змінені таким чином, щоб вони відповідали конкретним конфігураціям апаратури на льоту [26].

Порівняно з ПЛІС попереднього покоління, нові покоління ПЛІС відзначаються значним зниженням споживаної потужності при збільшенні продуктивності. Наприклад, використання ПЛІС з родини Artix-7 порівняно з моделями Spartan-6 дозволяє зменшити енергоспоживання вдвічі та підвищити продуктивність на 30%. Так само, перехід від моделей Virtex-6 до Kintex-7 також

призводить до зменшення енергоспоживання вдвічі [27].

Програмовані логічні інтегральні схеми - це кремнієві пристрої, які передбачено виготовлені та можуть бути програмно налаштовані для реалізації різноманітних цифрових схем чи систем [28]. У порівнянні з інтегральними схемами, які призначені для застосування (ASIC), ПЛІС забезпечують дешевше та швидше рішення для невеликих і середніх обсягів виробництва, адже налаштування їх займає менше часу та вони доступні у відносно низьких цінових діапазонах - від кількох сотень до кількох тисяч доларів.

На рисунку 1.8 представлено структуру класичної ПЛІС, яка містить логічні блоки Це свідчить про те, що основна функціональність полягає у виконанні логічних операцій та обробці сигналів. Блоки введення/виведення також присутні, проте їх кількість може бути меншою порівняно з логічними блоками, оскільки ця частина відповідає за зовнішні з'єднання з іншими пристроями та системами [29].

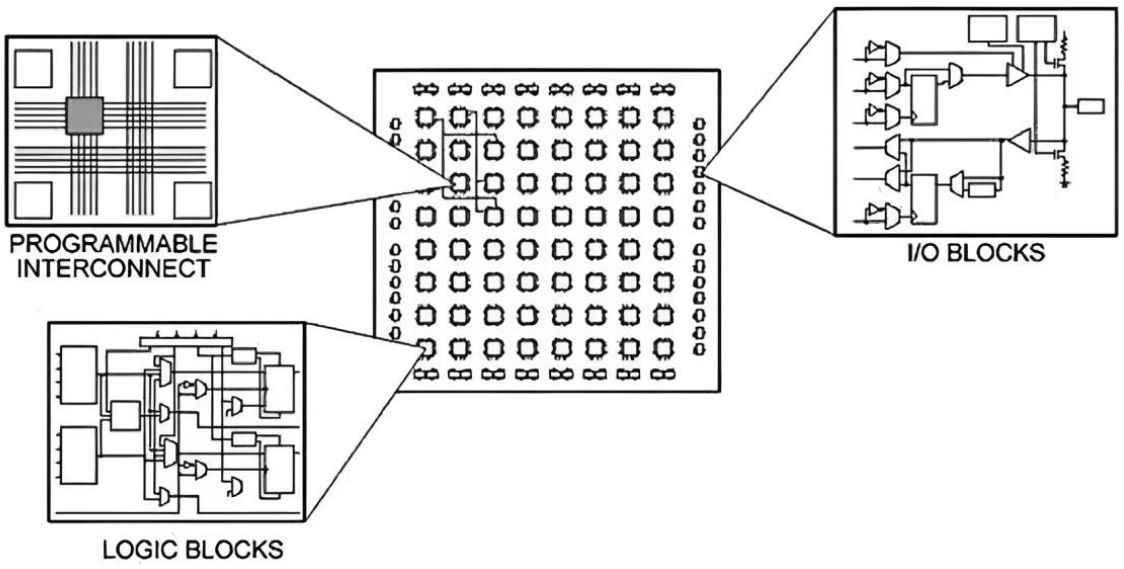


Рисунок 1.8 – Схема класичної ПЛІС

Крім того, важливою частиною ПЛІС є програмовані взаємозв'язки, які складаються з пучків металевих ліній, які працюють у вертикальному та горизонтальному напрямках. Ці взаємозв'язки дозволяють з'єднувати логічні блоки та блоки введення/виведення відповідно до цифрової схеми, що програмується для розгортання в ПЛІС. Перемикачі, які знаходяться на

перехрестях вертикальних та горизонтальних ліній, відповідають за з'єднання цих ліній за програмованим способом, що реалізується за допомогою транзисторів MOSFET [29].

Найбільшою перевагою ПЛІС є їх гнучкість, але ця сама характеристика робить їх більшими, менш швидкими та більш енергоємними в порівнянні з спеціалізованими інтегральними схемами . Такі недоліки в основному зумовлені програмованим маршрутизаційним з'єднанням ПЛІС, яке займає значну частину їх загальної площі. Однак, незважаючи на це, програмовані логічно-інтегральні схеми є переконливою альтернативою для реалізації цифрових систем [30], завдяки швидкому виходу на ринок та доступній ціні. Зазвичай вони складаються з програмованих логічних блоків, програмованої маршрутизації та блоків вводу/виводу, які забезпечують зв'язок з логічними блоками та створюють з'єднання поза мікросхемою.

Деякі алгоритми сортування показують кращу ефективність при обробці невеликої кількості даних, тоді як інші виявляються більш ефективними при великих обсягах. Бульбашкове сортування, сортування вибором і сортування вставкою є базовими алгоритмами, які, як правило, працюють швидше за складні алгоритми, такі як сортування злиттям або швидке сортування, коли маємо справу з невеликою кількістю даних.

Найпоширеніші реалізації сортування є алгоритмів бульбашкового сортування, сортування вибором та сортування вставкою з використанням мови опису апаратних засобів Verilog HDL. Для кожної з цих реалізацій щоб краще зорієнтувантися використовують RTL-діаграму, а також проводять аналіз часових діаграм у найскладніших випадках (наприклад, для бульбашкового сортування - коли дані вже відсортовані в зворотному порядку).

Застосування комп'ютерно-автоматизованих систем проектування (САПР) дозволяє встановлювати потрібну структуру цифрового пристрою у вигляді принципової електричної схеми або програми на спеціалізованих мовах опису апаратури, таких як Verilog, VHDL, AHDL та інші.

Згідно зі статистикою провідних фірм, таких як Aldec Inc. та Xilinx Inc. з

США, обсяг виробництва програмованих логічних інтегральних мікросхем з 2005 по 2020 рік зріс більш ніж у 20 разів і продовжує зростати [31]. На рисунку 1.9 представлено діаграму розподілу обсягів апоживання ПЛІС в окремих галузях.

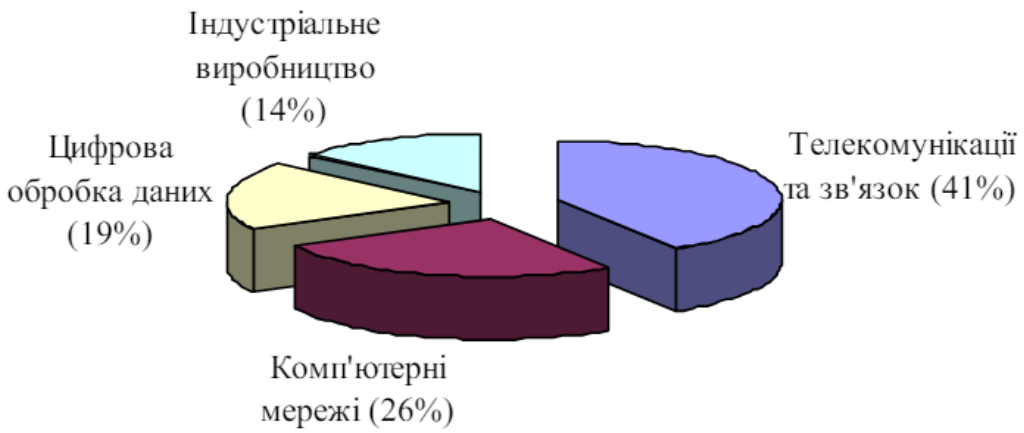


Рисунок 1.9 - Розподіл обсягів споживання ПЛІС в окремих галузях

Раніше ПЛІС використовувалися в обмеженому обсязі через їхню обмежену швидкодію та невелику кількість еквівалентних логічних вентилів. Однак з появою сучасних швидкодіючих ПЛІС високої інтеграції, які працюють на високих тактових частотах, їх вплив на світовий ринок значно виріс.

Сфера застосування ПЛІС постійно розширюється, охоплюючи такі галузі, як телекомунікації та зв'язок, комп'ютерні мережі, цифрова обробка даних і промислове виробництво. Серед великих споживачів ПЛІС можна відзначити такі відомі компанії, як Alcatel, Aser, Asus, IBM, Lockheed, Hewlett Packard, Fujitsu, Hitachi, Silicon Graphics, Texas Instruments, Motorola, Rockwell, Kodak і багато інших.

Використання програмованих логічних інтегральних схем (ПЛІС) для сортування даних відкриває широкі перспективи і потенційні можливості. ПЛІС забезпечують гнучкість та масштабованість у реалізації різних алгоритмів сортування, що дозволяє досягти високої продуктивності та ефективності у обробці даних. Також використання ПЛІС для сортування даних особливо актуально в сучасних системах реального часу та потоковій обробці інформації.

Швидкодія та низька затримка, що характеризують ПЛІС, роблять їх ідеальними кандидатами для реалізації алгоритмів сортування в таких вимогливих сценаріях.

Проте, успішне використання ПЛІС для сортування даних вимагає не лише технічної експертизи, але й уваги до оптимізації та адаптації до конкретних вимог та обмежень задачі [32]. Ефективне використання ресурсів ПЛІС, оптимальне розподілення завдань та мінімізація витрат енергії є ключовими аспектами, які потрібно враховувати при розробці та впровадженні сортувальних систем на базі ПЛІС. Загалом, використання ПЛІС для сортування даних є обіцяним напрямком розвитку, який може принести значний внесок у підвищення продуктивності та ефективності обробки інформації в різних областях застосування.

1.6 Висновки

Аналіз методів сортування на базі ПЛІС відкриває широкі можливості для оптимізації обробки даних у різних сферах. Різноманітні методи, такі як швидкі сортувальні алгоритми, апаратне сортування та використання спеціалізованих апаратних прискорювачів, дозволяють досягати високої ефективності та продуктивності при обробці великих обсягів даних. Наприклад, апаратне сортування може використовувати прискорювачі на базі FPGA для реалізації оптимізованих алгоритмів сортування, що дозволяє досягти значного прискорення порівняно з програмними методами. Ці системи відомі своєю швидкодією, низькою затримкою та високою ефективністю у виконанні складних алгоритмів сортування. Тому й використання ПЛІС для сортування даних є особливо важливим в галузях, де швидка обробка великих обсягів інформації є критичною, таких як фінансовий сектор, логістика та медицина. ПЛІС дозволяють реалізувати спеціалізовані алгоритми, що оптимізовані під конкретні потреби та вимоги користувачів. Проте, важливо враховувати потенційні виклики, пов'язані з використанням таких систем, такі як необхідність ефективного використання ресурсів, забезпечення безпеки та надійності обробки даних. Крім того, постійне вдосконалення та адаптація до змінних потреб користувачів є ключовими

аспектами успішного впровадження таких систем.

Також у даному розділі описано метод проектування спеціалізованих пристроїв, що розпочинається з графічного представлення алгоритму через обчислювальний граф. Після цього виконується побудова потокового графа для виявлення паралелізму та пошуку компромісів у просторово-часових характеристиках, що дає можливість переведення графа алгоритму в ярусно-паралельну форму. Далі здійснюється розробка різних видів просторово-часових графів, які дозволяють створювати конвеєрні, паралельні та рекурсивні спеціалізовані пристрої та оцінювати їх технічні параметри.

Загалом, спеціалізовані системи сортування на базі ПЛІС є потужним інструментом для оптимізації обробки даних у сучасному світі. З розвитком технологій важливо продовжувати дослідження та вдосконалення цих систем для забезпечення їхньої ефективності та відповідності вимогам ринку [33].

Таким чином необхідним є вирішення задачі створення спеціалізованої апаратної системи, яка здатна ефективно сортувати великі обсяги даних за допомогою програмованих логічних інтегральних схем (ПЛІС). Для цього необхідно:

- дослідницький підхід для систем сортування;
- дослідити існуючі методи сортування даних і їхні варіації;
- дослідити відомі методи оптимізації сортування на основі ПЛІС;
- дослідити методологічний вибір спеціалізованої системи сортування на базі ПЛІС;
- розглянути інтеграцію з існуючими системами;
- провести тестування та оцінку ефективності спеціалізованої системи сортування на базі ПЛІС;
- необхідно розробити покращену спеціалізовану системи сортування на базі ПЛІС;
- провести аналіз вартості та вигоди створеної системи.

2 МОДЕЛЮВАННЯ ПРОЦЕСІВ СОРТУВАННЯ ДАНИХ ЗА ДОПОМОГОЮ СПЕЦІАЛІЗОВАНИХ СИСТЕМ НА БАЗІ ПЛІС

2.1 Аналіз апаратних алгоритмів сортування

ПЛІС використовуються в різноманітних застосуваннях завдяки своїй гнучкості, що дозволяє розробникам вільно налаштовувати їхні обчислювальні потоки та системи надання даних під конкретні вимоги. Однак, традиційна розробка програм для ПЛІС передбачала складні RTL-дизайни, і вартість програмування та переносимість програми не були пріоритетними. Це обмежувало доступ до ПЛІС для більшості розробників додатків, оскільки потрібно високий рівень спеціалізованих знань з апаратної інженерії. Виникнення технік високорівневого синтезу (HLS) вирішило цю проблему, зокрема, Xilinx Vitis та Intel FPGA SDK for OpenCL, які надають універсальні середовища розробки, що приховують деталі реалізації ПЛІС від користувача. Таким чином, розробники додатків можуть використовувати ПЛІС без глибоких знань апаратної інженерії. Intel FPGA SDK for OpenCL надає механізм управління периферійними пристроями на ПЛІС з рівня OpenCL, що спрощує розробку високопродуктивних додатків HPC. Таким чином, наразі Intel FPGA SDK for OpenCL є найкращим рішенням для розробників додатків, які працюють з ПЛІС [34].

Однак, у той час як ПЛІС-реалізації на рівні реєстрового трансферу (RTL) дозволяють розробникам визначати архітектуру своїх застосунків з великою точністю, моделі програмування на основі OpenCL обмежені у своїй виразності. Крім того, моделі програмування на основі OpenCL часто стикаються з проблемами розміщення та маршрутизації, які обмежують максимальну частоту. Тому для розробки додатків з високою ефективності на ПЛІС з низькою вартістю програмування необхідно оптимізувати загальнопризначені ядра обчислень для практичного розвитку додатків на ПЛІС та зробити їх доступними користувачам. Наприклад, були запропоновано реалізацію алгоритму множення матриць Кеннона на основі OpenCL [35], оптимізованого для ПЛІС Intel Stratix 10, вважаючи, що він може служити блоком для алгоритмів, що базуються на

базовому функціоналі множення матриць. З цієї підстави було розроблено бібліотеку сортування, яка може бути використана з моделлю програмування OpenCL для ПЛІС [36]. Сортування є основною арифметичною операцією, яка використовується у розробці різноманітних застосунків, і для того, щоб врахувати його широкий спектр використання, бібліотеки надаються у формі std::sort() для C++ та thrust::sort() для CUDA. Також розробили апаратний двигун на рівні RTL для виконання процесу сортування і викликали його як бібліотеку з коду ядра OpenCL [37]. Двигун побудований за допомогою комбінації наступних трьох апаратних алгоритмів сортування: мережа сортування, дерево злиття високої пропускної здатності та віртуальне дерево злиття. Ці алгоритми розроблені для роботи на високих частотах так, щоб двигун, викликаний з коду ядра OpenCL, не став критичним шляхом та не погіршував загальну продуктивність застосунка.

Апаратні алгоритми сортування - це моделі, які втілюють процес сортування даних апаратним шляхом. Особливо сортувальна мережа та сортування злиттям є ефективними для FPGA з точки зору паралелізму, легкості керування та робочої частоти, і проведено численні дослідження, що використовують їх [20]. Сортувальна мережа - це модель, призначена для сортування послідовностей чисел, і складається з ліній зв'язку та блоку порівняння та обміну (CAE). На рисунку 2.1 представлена її робота.

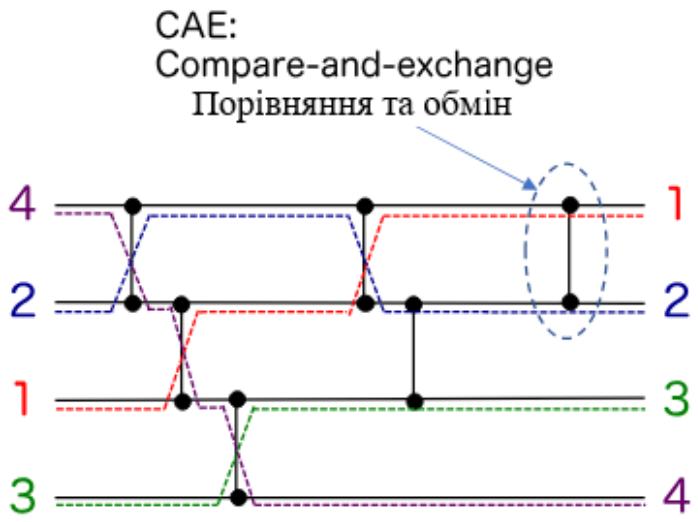


Рисунок 2.1 - Основні апаратні алгоритми мережі сортування

Лінії зв’язку переносять дані, а блок CAE приймає два вхідні значення, порівнює їх та виводить менше з них на одну лінію зв’язку та більше на іншу. Ця модель дозволяє сортувати дані паралельно, без складної логіки керування.

Сортування злиттям (англ. merge sorter tree) - це апаратна модель для злиття кількох відсортованих записів даних і складається з набору блоків порівняння та вибору (CAS), розміщених у бінарному дереві, як це зображено на рисунку 2.2.

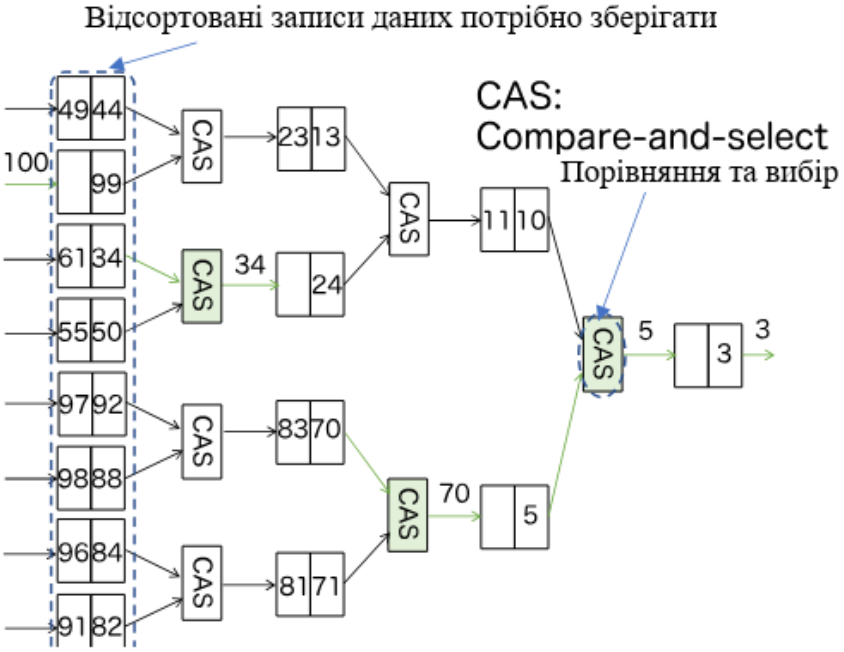


Рисунок 2.2 - Основні апаратні алгоритми сортування злиттям

Блок CAS - це комбінаційний ланцюг, який порівнює ключі двох вхідних записів даних та вибирає один із них згідно з результатом порівняння. На кожному такті ключі порівнюються у всіх блоках CAS, і записи даних, які виводяться згідно з результатами порівняння, зберігаються у буферах першим увійшов першим вийшов (FIFO), і, в кінцевому підсумку, відсортовані записи даних видаляються з кореня дерева злиття. FIFO, розташовані на вхідному порту дерева, завжди мають містити відсортовані записи даних, і тому часто для їх створення використовується мережа сортування.

Цей сортувальний механізм, розроблений на базі FPGA і складається з

трьох апаратних алгоритмів сортування: сортувальна мережа, сортування злиттям з високою пропускною здатністю та сортуванням злиття [38].

2.2 Аналіз досліджувальних алгоритмів й систем розробки методлології сортування даних на базі ПЛІС

Одним із ключових завдань, що виникає в контексті обробки даних, є сортування, що є поширеним і добре вивченим в комп'ютерних науках. У зусиллях досягнення ефективних методів сортування на базі ПЛІС, науковці та інженери вивчають різні підходи та алгоритми, спрямовані на оптимізацію продуктивності та ресурсоємності. У цьому контексті різноманітні дослідницькі роботи присвячені розробці та вдосконаленню методів сортування на базі ПЛІС. Деякі з цих робіт вивчають реалізації алгоритмів сортування, в той час як інші досліджують оптимальні стратегії використання апаратних ресурсів для підвищення швидкодії та продуктивності. У цьому контексті, науковці проводять експерименти та порівняльний аналіз різних підходів, щоб зрозуміти переваги та недоліки кожного методу [39]. Детальне дослідження цих аспектів може допомогти в розробці більш ефективних та оптимальних систем сортування на базі ПЛІС.

Зурек та співавтори [40] запропонували два різних алгоритми апаратної реалізації: паралельний швидкий злиття та гібридний алгоритм (паралельний бітонічний алгоритм на GPU + послідовне злиття MergeSort на CPU) з використанням фреймворків OpenMP та CUDA. Вони порівняли дві нові реалізації з різною кількістю елементів. Отриманий результат показує, що алгоритми сортування на багатоядерних процесорах є найбільш масштабованими та ефективними. Алгоритми сортування на GPU, порівняно з одиночним ядром, працюють до чотири рази швидше, ніж оптимізований алгоритм швидкого сортування. Реалізований гібридний алгоритм (частково виконується на CPU та GPU) ефективніший, ніж алгоритми, що виконуються лише на GPU (незважаючи на затримки передачі), але трохи повільніший за найбільш ефективний

паралельний швидкий алгоритм злиття на CPU. Вони показали, що гібридний алгоритм сповільнюється порівняно з найбільш ефективним паралельним швидким алгоритмом злиття на CPU.

Автори Абірамі та інші [41] представили ефективну апаратну реалізацію алгоритму сортування MergeSort за допомогою проектування цифрових схем. Вони оцінили ефективність, надійність та складність алгоритму MergeSort з використанням цифрової схеми. Абірамі використовував лише 4 входи та порівнював ефективність алгоритму MergeSort із алгоритмами сортування бульбашкою та вибору. Недолікою цієї роботи є обмежене використання лише алгоритму MergeSort на FPGA.

Пасетто та співавтори [32] розглянули кілька паралельних реалізацій п'яти алгоритмів сортування (MapSort, MergeSort, Tsigas-Zhang’s Parallel QuickSort, Alternative QuickSort та STLSort), використовуючи значну кількість елементів. Вони провели аналіз продуктивності цих алгоритмів з використанням двох різних методик (пряме сортування та сортування ключ/вказівник) на двох різних машинах: Westmere (Intel Xeon 5670, частота: 2,93 ГГц, 24 ГБ пам'яті) з 6 ядрами i7 та Nehalem (Intel Xeon 5550, 2,67 ГГц, 6 ГБ пам'яті) з 4 ядрами i7. Потім вони здійснили порівняння кількох алгоритмів з точки зору афінності до процесора, пропускної здатності, аналізу мікроархітектури та масштабованості.

На основі результатів дослідження автори рекомендують використовувати алгоритми MergeSort і MapSort, якщо обмеження пам'яті не є проблемою. Обидва ці алгоритми вважаються оптимальними, якщо кількість елементів не перевищує 100 000, оскільки вони вимагають проміжного розміру масиву. Данелутто та його колеги [35] представили реалізацію шаблону за допомогою openMP, фреймворку Intel TBB та паралельного програмування Fastflow на багатоядерних платформах. Вони розробили високорівневий інструмент для швидкого прототипування паралельних алгоритмів "Поділ і панування". Отримані результати підтвердили, що прототип паралельних алгоритмів дозволяє зменшити час виконання та потребує мінімальних зусиль у програмуванні порівняно з ручною оптимізацією для паралельного виконання.

Грозеа та його колеги розробили методику прискорення існуючих алгоритмів порівняння [41], таких як MergeSort, Бітонічне сортування та паралельне сортування вставками, щоб вони могли працювати на швидкості, типовій для мережевого зв'язку Ethernet 1 Гбіт/с. Вони використовували паралельні архітектури, такі як ПЛІС, багатоядерні системи на базі процесорів і GPU. Отримані результати показали, що ПЛІС є найбільш гнучкою платформою, але менш доступною. GPU, незважаючи на велику потужність, менше гнучкий, складніший у налагодженні та вимагає передачі даних для зменшення затримок. Іноді ЦП може бути занадто повільним, навіть з урахуванням багатьох ядер та процесорів, але він є найбільш простим у використанні.

Результати, отримані в різних дослідженнях щодо паралельного виконання алгоритмів сортування на різних архітектурах (ЦП, ГПУ та ПЛІС), свідчать, що прискорення залежить від кількості процесорів. Ці дослідження показують, що апаратна реалізація на ПЛІС забезпечує кращу продуктивність за рахунок зменшення часу та енергії. У цьому контексті, ми можемо максимально паралелізувати роботу, проте не досягаємо високих значень прискорення через обмежену можливість інструменту HLS видобути необхідну паралелізацію, особливо коли потрібно сортувати лише невелику кількість елементів.

Таким чином, інструмент HLS дозволяє підвищити продуктивність апаратного прискорювача, скорочуючи час розробки, а його головною перевагою є швидке дослідження простору проектування для пошуку оптимального рішення. Керуючись цими принципами оптимізації HLS, можна забезпечити апаратний IP, який балансує продуктивність та ефективністьДля прискорення різних застосувань, таких як система прийняття рішень у реальному часі та планування тривимірних маршрутів, на рисунку 2.3 наведено загальну структуру алгоритму сортування.

Також важливо зазначити, що деякі алгоритми не є оптимальними для виконання на ПЛІС. У цьому контексті, вибрано платформу ПЛІС та інструмент HLS для покращення продуктивності та вибору оптимального алгоритму сортування [42].

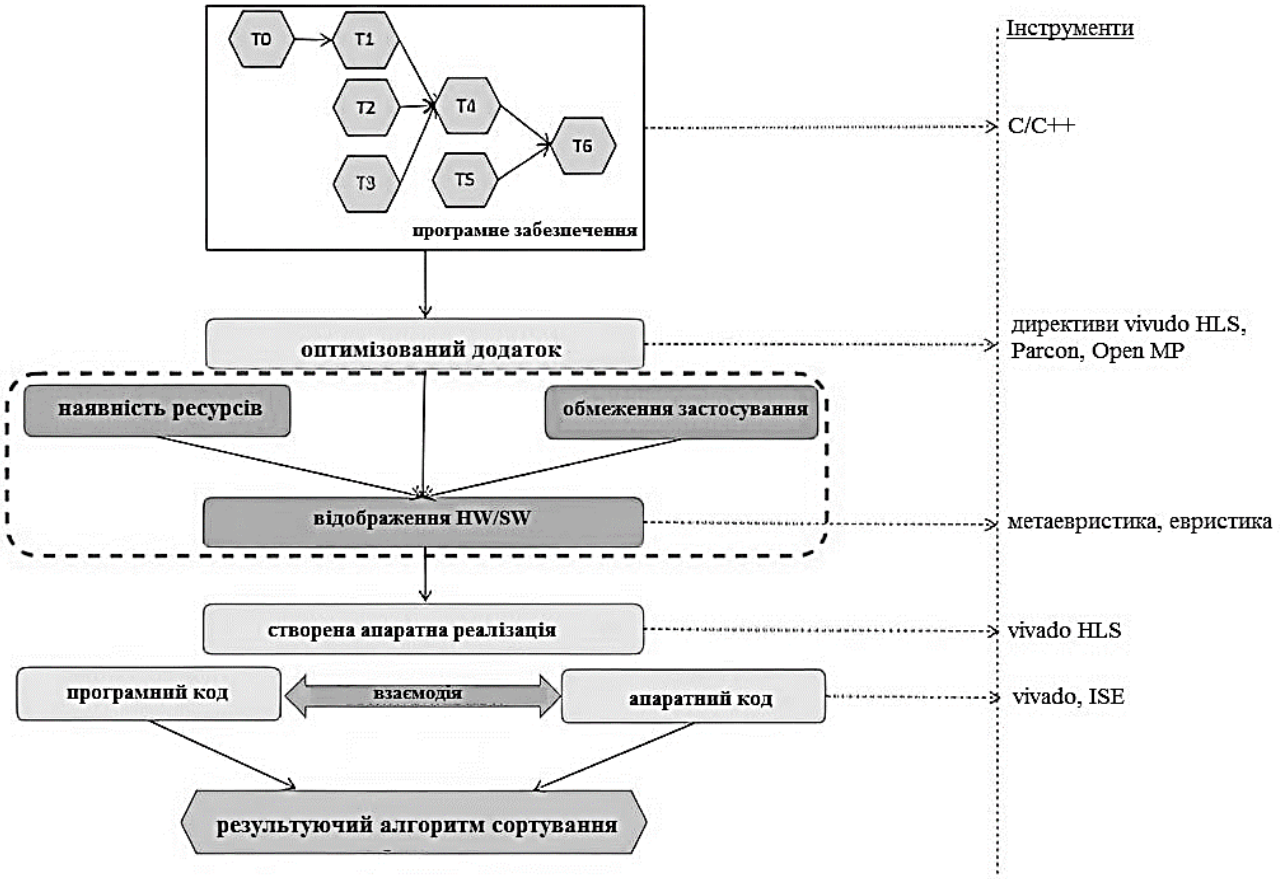


Рисунок 2.3 - Загальний потік проектування

Спочатку ми маємо набір завдань, написаних мовами програмування C/C++, які мають виконати на гетерогенній платформі CPU/FPGA [43]. Далі – оптимізація цієї програми за допомогою директив оптимізації високого рівня синтезу (HLS) для ефективної апаратної реалізації. Крім того, застосовується інший метод оптимізації, який дозволяє виконати ту саму функцію паралельно з різними вхідними даними. Потім оптимізовану програму розділяють на програмні та апаратні завдання за допомогою різних метаевристичних або модифікованих рішень HEFT (Heterogeneous Earliest-Finish Time) (MHEFT), з урахуванням обмежень застосування та доступних ресурсів. Апаратне завдання створюється за допомогою інструменту Vivado HLS на базі мови програмування C/C++. Нарешті, програмні та апаратні завдання взаємодіють через шину для їх виконання на FPGA. Ця взаємодія здійснюється за допомогою інструментів ISE або Vivado з метою отримання ефективного результату реалізації.

2.3 Способи оптимізації алгоритмів

Важливість оптимізації алгоритмів сортування виявляється у потребі забезпечити швидку та ефективну обробку даних у сучасних обчислювальних системах, особливо в умовах обмежених обчислювальних ресурсів. Це стає актуальною проблемою в контексті обробки великих обсягів даних у реальному часі та оптимізації продуктивності спеціалізованих пристроїв. Зокрема, конкретні методи оптимізації, такі як розгортання циклів, піпелінізація інструкцій та використання оптимізованих введення/виведення, виявляються ключовими засобами для досягнення швидкодії та ефективності сортування.

Для забезпечення ефективної апаратної реалізації можна використовувати наступні оптимізаційні методи у коді на мові С для кожного алгоритму сортування [44]:

− розгортання циклу: елементи таблиці зберігаються у пам'яті BRAM, яка має подвійні фізичні порти. Ці порти можуть бути налаштовані як два порти запису, два порти читання або один порт для кожної операції. Можна використовувати цю можливість, розгортаючи цикли у дизайні з коефіцієнтом розгортання = 2. Наприклад, у циклі виконуються лише операції запису елементів. Отже, два порти для масиву можуть бути налаштовані як порти запису, і ми можемо розгорнути цикл з коефіцієнтом розгортання = 2;

− піпелінізація ітерацій циклу: це метод оптимізації обчислювальних процесів, що полягає в розбитті завдання на послідовні етапи, кожен з яких виконується в окремому циклі обробки. Це дозволяє кожному етапу розпочинати роботу незалежно від попередніх та збільшує швидкість виконання завдання за рахунок більш ефективного використання обчислювальних ресурсів. Ітерації циклу піпеліновані у виконанні з різницею в один такт між ними, застосовуючи піпелінізацію циклу з інтервалом ітерації (II) = 1. Щоб відповісти цій умові, інструмент планує виконання циклу;

− інтерфейс введення/виведення: вхідні/вихідні порти налаштовані для використання протоколу AXI-Stream для передачі даних з мінімальною кількістю

сигналів зв'язку (DATA, VALID та READY). Також застосовується протокол AXI-Lite для конфігурації виконання, наприклад, для визначення поточного стану системи (старт, готовність, зайнятість).

2.4 Формулювання вимог і вибір принципів побудови компонентів інформаційних технологій паралельного сортування

На сучасному етапі розвитку інформаційних технологій великі обсяги даних стають все поширенішими. Обробка таких масивів даних часто потребує використання операцій сортування та пошуку, спрямованих на прискорення доступу до необхідної інформації. Ключовим вимогам до засобів сортування даних є досягнення високої швидкодії. Ця проблема особливо актуальна під час сортування великих масивів даних і в режимі реального часу [45].

Отже, для вирішення поставлених завдань у магістерській роботі необхідно сформулювати вимоги до компонентів інформаційних технологій паралельного сортування. Однією з головних вимог є забезпечення реального часу. Для досягнення цієї мети, виконання умови, що належить до програмно-апаратного сортування та пошуку даних, є необхідним:

$$t_{сор/пош} \leq t_{пм}\ , \tag{2.1}$$

де $t_{сор/пош}$ – час сортування даних; $t_{пм}$ – час надходження масиву даних, що в свою чергу визначено так:

$$t_{пм} = \frac{N_n}{F_d k n_k}, \tag{2.2}$$

де N – відповідає розміру масиву даних; n – розрядність даних; F_d – частоті надходження даних;

k – кількість каналів; n_k – розрядність каналів надходження даних.

Продуктивність програмно-апаратних засобів сортування та пошуку

даних має відповідати наступним критеріям:

$$\Pi_{\text{КЗ}} = \frac{\lambda R_{\text{сор/пош}}}{t_{\text{пм}}}, \quad (2.3)$$

де λ – коефіцієнт враховує індивідуальні характеристики архітектури комп'ютерних пристроїв; $R_{сор/пош}$ – кількість операцій, які потрібні для виконання конкретного алгоритму сортування даних.

При апаратній реалізації паралельно-потокових алгоритмів сортування та пошуку даних забезпечення режиму реального часу відбувається, коли виконується така умова:

$$P_d \le P_{A3}, \quad (2.4)$$

де P_d – інтенсивність даних, що надходять;

P_{A3} – інтенсивність сортування даних за допомогою апаратних засобів.

Швидкість надходження даних та швидкість їх сортування апаратними засобами визначаються так:

$$P_d = F_d k n_k, \; P_{A3} = F_d m_t n_t, \quad (2.5)$$

де m_t – число каналів сортування даних;; n_t – розрядність каналів сортування даних.

Крім того, апаратні засоби інформаційних технологій паралельного сортування даних повинні мати високий коефіцієнт використання обладнання, який розраховується таким чином:

$$E = \frac{R_{сор/\text{пош}}}{t_{сор/\text{пош}} W_{сор/\text{пош}}}, \quad (2.6)$$

де $W_{сор/пош}$ – представляє собою витрати на обладнання для реалізації апаратних засобів сортування даних.

Ефективність використання обладнання визначається зв'язком між продуктивністю апаратних засобів сортування даних та витратами на обладнання, оцінюючи компоненти за їхньою продуктивністю.

Як вже зазначалося, для підвищення швидкості сортування великих масивів даних використовується розпаралелювання процесу сортування та використання багатопроцесорних багатоядерних систем з великим обсягом пам'яті [46]. Серед таких засобів можна виділити графічні процесори (GPU – Graphics Processing Unit), які належать до класу процесорів SIMD (Single Instruction Multiple Data). Однією з ключових особливостей процесорів класу SIMD є можливість застосовувати одну операцію для обробки групи незалежних даних. Розробка програмної реалізації операцій пошуку та сортування масивів даних на багатопроцесорних багатоядерних системах з великим обсягом пам'яті потребує вдосконалення і створення нових паралельних алгоритмів. Паралельні методи та алгоритми сортування масивів даних, які призначені для реалізації на багатопроцесорних багатоядерних системах, повинні:

− структура алгоритмів повинна бути чіткою, з детермінованим переміщенням даних;

− алгоритми мають базуватися на однотипних операціях порівняння та перестановки даних з регулярними та локальними зв’язками;

− орієнтація на реалізацію на множині взаємозв’язаних процесорних ядер;

− використання конвеєризації та просторового паралелізму;

− при розробці паралельних алгоритмів сортування даних важливо враховувати багато факторів. Перш за все, алгоритми повинні бути рекурсивними, локально залежними та придатними для багатопроцесорних багатоядерних систем;

− для створення ефективних засобів паралельного сортування масивів даних пропонується інтегрований підхід, що включає в себе;

− дослідження та розробку методів та алгоритмів паралельного пошуку та

сортування даних;

− пошук нових алгоритмічних рішень, орієнтованих на багатопроцесорні багатоядерні системи, для мінімізації часу сортування даних;

− використання засобів автоматизації паралельного програмування;

− для забезпечення високої ефективності сортування даних важливо враховувати наступні принципи;

− адаптація алгоритмів до структури багатопроцесорних багатоядерних систем;

− використання конвеєрів та просторового паралелізму;

− модульність програмно-апаратних засобів.

2.5 Висновки

У розділі було досліджено й розглянуто агоритми апаратного сортування на базі ПЛІС, способи оптимізації алгоритмів , зокрема використання ефективних структур даних, оптимізацію роботи з пам'яттю, використання паралельних обчислень та попереднє упорядкування даних. Ці оптимізації спрямовані на підвищення швидкодії та ефективності апаратних реалізацій сортування. Проведено аналіз досліджувальних супутних алгоритмів й систем розробки методлології сортування даних на базі ПЛІС.

Проведено аналіз ефективності алгоритмів сортування, найчастіше пов’язаних із програмним моделюванням. Досліджено метологічний вибір системи сортування даних на базі ПЛІС.

Розглянуто проблеми сортування з використанням ПЛІС)та їх апаратної реалізації [47]. Досліджено алгоритми сортування та їх характеристики, враховуючи велику кількість факторів, таких як швидкодія, ресурсомісткість та ефективність використання апаратного обладнання. Зокрема, вивчено методи оптимізації алгоритмів сортування для забезпечення максимальної продуктивності та ефективності роботи апаратних реалізацій. Такий підхід дозволяє виявити переваги та недоліки різних підходів до сортування та

визначити оптимальні стратегії для їх впровадження в реальних системах.

У висновках підкреслено, що зазначені аспекти взаємодіють для створення стратегічного фундаменту для подальшого розвитку апаратних сортувальних систем. Цей стратегічний фундамент включає в себе не лише оптимізацію алгоритмів сортування для платформи ПЛІС, але й врахування різноманітних технічних, архітектурних та програмних аспектів. Такий комплексний підхід дозволить забезпечити високу продуктивність, ефективність та надійність апаратних реалізацій сортування, що стане основою для подальшого успішного розвитку та впровадження у різноманітних областях, таких як обробка даних, мультимедіа, наукові обчислення та інші.

Було встановлено, що використання алгоритмів машинного навчання має свої обмеження у контексті створення спеціалізованої системи сортування на базі ПЛІС. Однією з основних проблем є швидкість обробки даних та обсяг вхідних даних. Підвищення швидкості обробки є критичним для ефективної роботи системи сортування, особливо в випадку обробки великих обсягів даних. Збільшення обсягу даних може призвести до значного збільшення часу, необхідного для сортування, що може ускладнити використання системи у реальному часі та вимагати додаткових оптимізацій алгоритмів та апаратури.

Отже, створення спеціалізованої системи сортування на базі ПЛІС потребує уваги до питань як швидкості обробки даних, так і оптимізації ресурсів для ефективної роботи з великими обсягами інформації.

3 РОЗРОБЛЕННЯ УДОСКОНАЛЕНОГО СПЕЦІАЛІЗОВАНОГО МЕТОДУ СОРТУВАННЯ ДАНИХ

3.1 Основи удосконаленого підходу до сортування даних методом "бульбашки"

Як вже було зазначено у розділі 2, алгоритми сортування, орієнтовані на реалізацію на базі ПЛІС, потребують зменшення кількості виводів інтерфейсу та розрядності з’єднань між функціональними операторами для порівняння та перестановки даних [47]. Для досягнення ефективної апаратної реалізації використовували наступні оптимізаційні кроки для кожного алгоритму сортування:

− розгортання циклу: елементи таблиці зберігаються у пам'яті BRAM, яка має подвійні фізичні порти. Ці подвійні порти можуть бути налаштовані як пари для запису, пари для читання або один порт для кожної операції. Ми отримали вигоду від цієї оптимізації, розгортаючи петлі в дизайні з коефіцієнтом 2. Наприклад, якщо у циклі виконуються лише операції запису, то два порти для масиву можуть бути налаштовані як порти для запису, і, таким чином, ми можемо розгортати цикл з коефіцієнтом 2;

− конвеєрні ітерації циклу: Для зменшення часу виконання ітерації циклу конвеєрні, в нашому проекті, різниця між ітераціями циклу становить лише один тактовий цикл. Це досягається за допомогою конвеєрної обробки циклу з інтервалом ітерації (II) рівним 1;

− інтерфейс вводу/виводу: Порти вводу/виводу налаштовані на використання протоколу AXI-Stream для передачі даних з мінімальними сигналами зв'язку (DATA, VALID і READY). Крім того, для конфігурації проектування, такої як визначення поточного стану системи (запуск, готовність, зайнятість), використовується протокол AXI-Lite.

На рисунку 3.1 показана загальна структура великої інтегральної схеми, що складається з вхідного інтерфейсу, операційного автомату, вихідного інтерфейсу та пристрою керування.

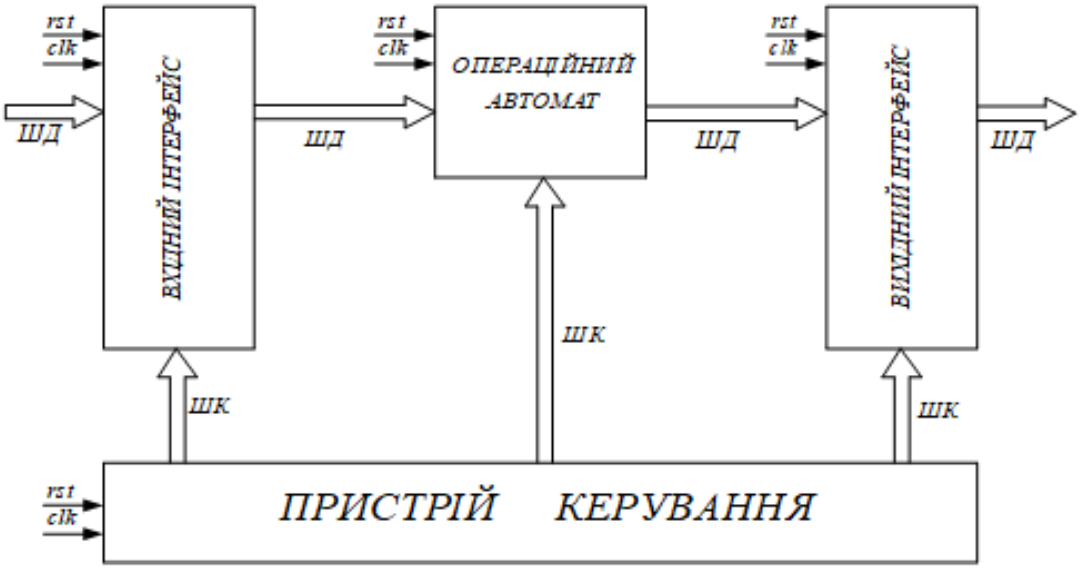


Рисунок 3.1 – Загальна структура системи

Вхідний інтерфейс є стандартним і складається з вхідних регістрів, в які послідовно записуються числа з шини даних (ШД), керованих сигналами керування від пристрою керування через шину керування (ШК). Операційний автомат виконує прості операції над операндами, що надходять з вхідного інтерфейсу, та повертає результати обробки. Виконання цих операцій автоматично здійснюється за допомогою відповідних сигналів керування, які створює пристрій керування. Вихідний інтерфейс призначений для взаємодії з шиною Wishbone [48]. Процес управління реалізований за допомогою автомата Мура.

Для побудови графу алгоритму сортування чисел вибрано метод “бульбашки”. Вхідні числа є 8-розрядними цілими числами в діапазоні $0 \le X < 1$.

Для створення графа алгоритму сортування чисел був обраний метод "бульбашки". Вхідні числа представлені 8-розрядними цілими числами у певному діапазоні $0 \le X < 1$.

На рисунку 3.2 показаний граф алгоритму, який відображає процес сортування 8 вхідних чисел.

На виході цього графа отримується нова послідовність з 8 чисел, в якій вхідні числа переставлені у потрібному порядку.

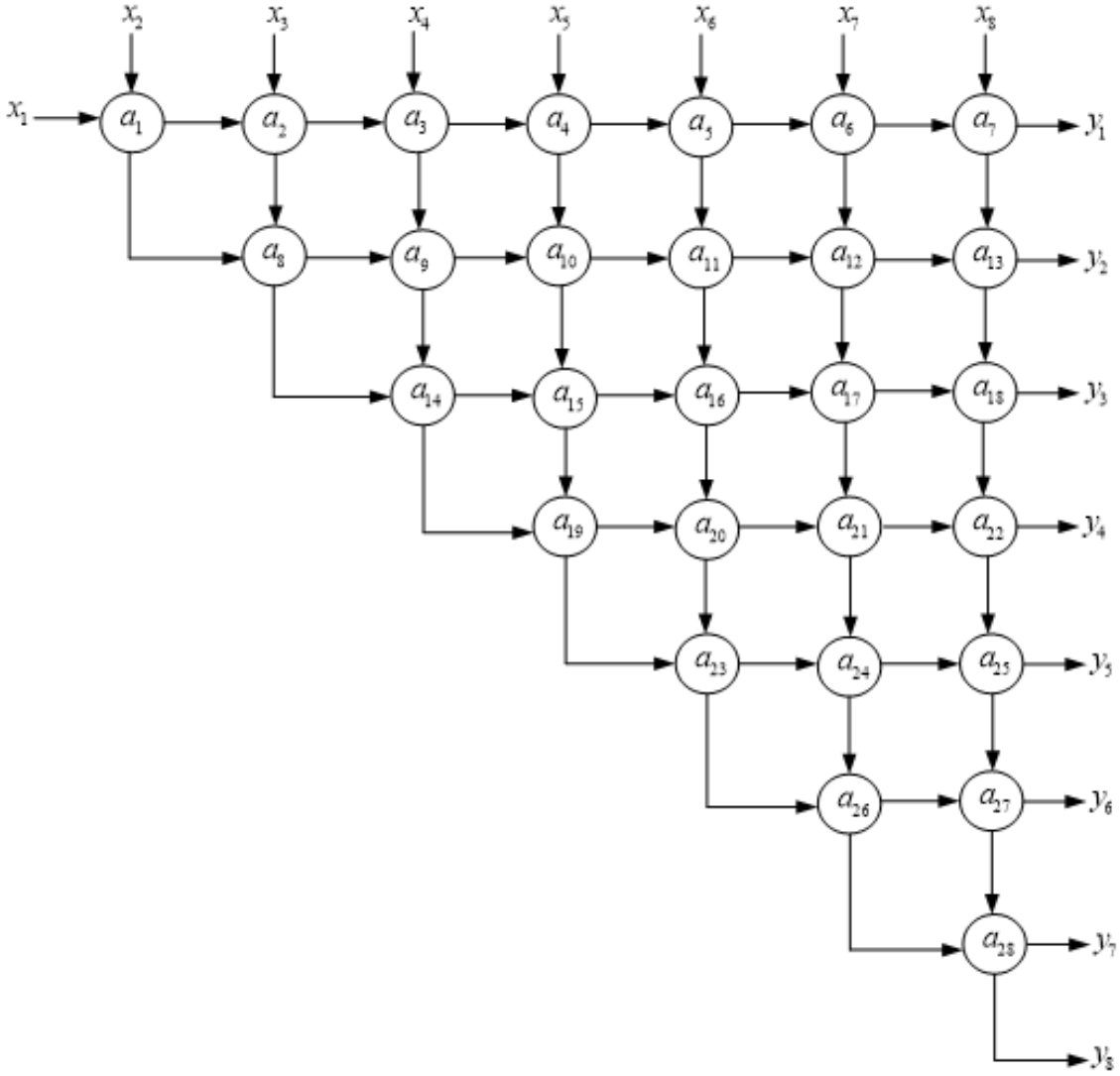


Рисунок 3.2 - Граф алгоритму сортування 8-ми чисел методом “бульбашки”

Для алгоритмів сортування методом "бульбашки" кількість операцій (вершин) у графі залежить від кількості вхідних чисел [49] і обчислюється за формулою $n(n-1)/2$, де n - кількість вхідних значень.

У розглянутому вище алгоритмі граф складається з 28 вершин $(a_1, \dots, a_{28})$, кожна з яких виконує операцію "порівняти і переставити". Після виконання цієї операції кожною вершиною графу на виходах ми отримуємо відсортовану послідовність чисел у порядку спадання.

На рисунку 3.3 показана структурна схема операції "порівняти і переставити" для описаного алгоритму сортування чисел методом "бульбашки". Слід зазначити, що ця операція є базовою для багатьох алгоритмів сортування .

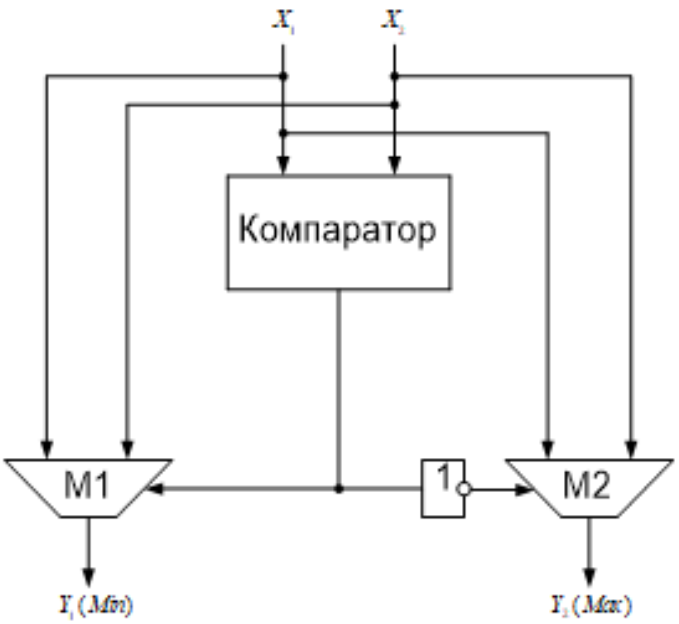


Рисунок 3.3 - Структурна схема операції “порівняти й переставити”

Структурна схема, яку було представлено, складається з компаратора [50], що приймає на вході два числа для порівняння, та двох мультиплексорів (M1 та M2), керованих вихідним сигналом компаратора, які одночасно видають на свої виходи мінімальне y1(Min) та максимальне y2(Max) число. Якщо припустити, що для реалізації компаратора потрібно 6 логічних елементів, а для реалізації двохвходового мультиплексора - 3 логічні елементи (за основу взято базис І-НЕ), то сумарна кількість вентилів для виконання однієї операції "порівняти й переставити" буде складати 12 логічних елементів. Загальна кількість вентилів ($W_{\text{вент.}}$) для всього алгоритму може бути визначена за допомогою формули:

$$W_{\text{вент.}} = N_{on} \times N_{\text{вент.}on} \tag{3.1}$$

де N_{on} - кількість операцій алгоритму, а $N_{\text{вент.}on}$ - кількість вентилів для реалізації однієї операції. Враховуючи це, загальна кількість вентилів для розглянутого алгоритму складатиме 336 логічних елементів.

У порівнянні з іншими алгоритмами сортування, реалізація бульбашкового сортування є досить простою. Та у найгіршому випадку, коли числа вже відсортовані у зворотному порядку, бульбашкове сортування має квадратичну

[52] складність - $O(n^2)$, де n - кількість сортованих елементів. Блок-схема для апаратної реалізації цього алгоритму сортування показана на рисунку 3.4.

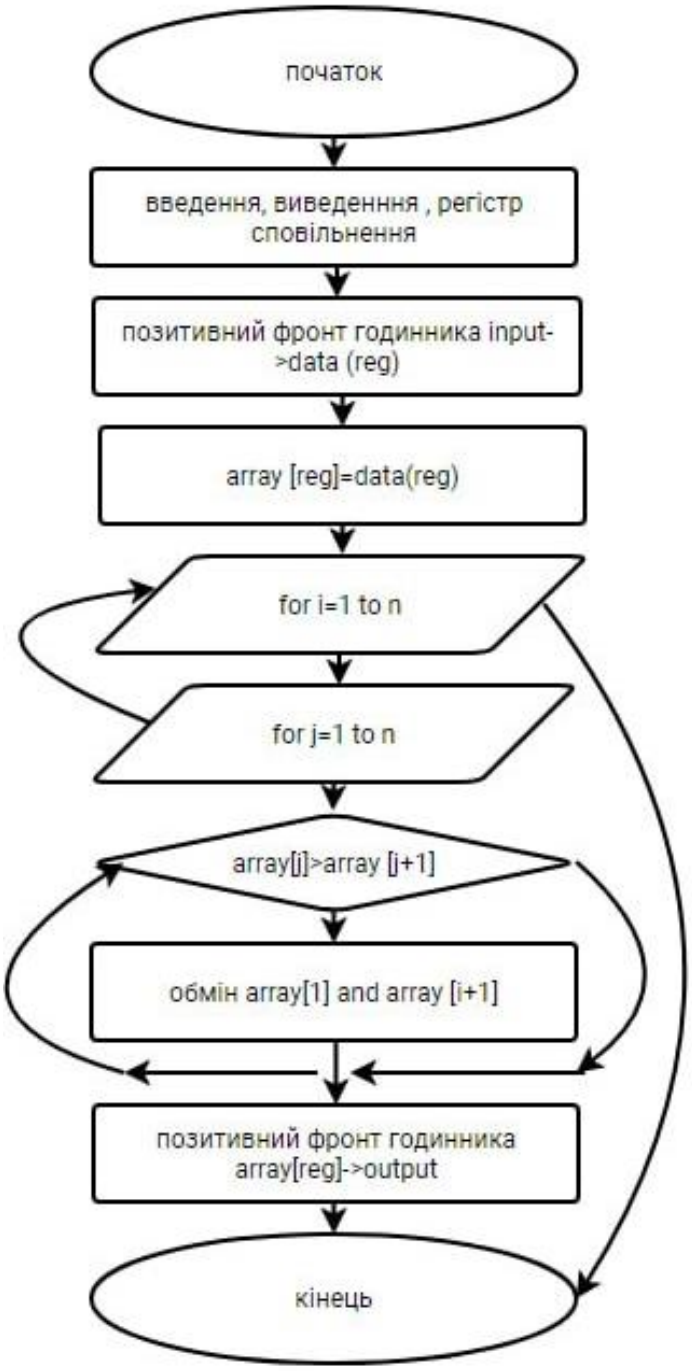


Рисунок 3.4 - Блок-схема для сортування бульбашкою

У даному алгоритмі сортування кожна пара сусідніх елементів порівнюється між собою. Після цього ця пара проходить кілька разів через весь список, що потрібно відсортувати, і якщо вони знаходяться в неправильному

порядку [51] (один елемент менший за попередній), то вони міняються місцями. Після кожної ітерації кількість порівнянь зменшується на одиницю, аж до того моменту, коли залишається лише один елемент для порівняння. Якщо ми маємо в загальному n елементів, то нам потрібно повторити процес заміни $n-1$ раз.

3.2 Побудова потокового та рекурсивного графу удосконаленого алгоритму

Розглянемо тепер теореми, які мають важливе значення в комп'ютерних науках та інженерії, особливо у вирішенні задач сортування та аналізу даних. Вони є фундаментальними для розуміння властивостей графів порівняння та використання їх у різних областях комп'ютерних наук.

Теорема 1. Якщо G - це граф порівняння, то G ациклічний. Таким чином, G є напрямним ациклічним графом [53].

Доведення. Припустимо, що G - це граф порівняння порядку n. Ми повинні показати, що G є ациклічним. Навпаки, припустимо, що G містить цикл, назвемо його C. Тоді для $k \leq n$ як довжина C, нехай $C = (1,\ldots,k,1)$, де i (для $1 \leq i \leq k$) є вершиною в G, і кожні дві сусідні вершини в C є напрямленою дугою в C. Нехай також ai представляє відповідне значення для кожної вершини в G. Тоді за визначенням графа порівняння маємо $a1 < \ldots ak < a1$, що є очевидним суперечливим. Отже, G є ациклічним, тобто G є напрямним ациклічним графом.

З цього легко випливають наступні наслідки:

Наслідок 1. Якщо G - це граф порівняння, то в G існує топологічне сортування.

Доведення. Припустимо, що G - це граф порівняння. Згідно з Теоремою 1, G є напрямним ациклічним графом. Тоді, з визначення випливає, що у G існує топологічне сортування.

Наслідок 2. Якщо G - це відповідний граф масиву A з будь-якою допустимою досяжністю r, то G ациклічний і має топологічне сортування.

Доведення. Оскільки кожен відповідний граф є графом порівняння зі

множиною значень, що представляють масив A, граф G обов'язково має топологічне сортування відповідно до Теореми 1 і Наслідку 1.

Хоча, ми можемо будувати алгоритми, що працюють з простими математичними визначеннями графів, для їх практичної реалізації вельми важливо створити корисні структури даних для представлення графів [54]. Використаємо представлення графів у вигляді списків суміжності, де кожна вершина має список суміжних вершин. Ця структура даних дуже ефективна за використання пам'яті та легко доступна. Фактично, буде вибиратись певні орієнтації цього списку, щоб найкраще відповідати потребам у процесах [55].

Використовують також пошук в глибину (DFS). Це означає проходити вглиб від головної вершини, доки можемо проходити тільки до вершини, на якій вже були, після чого починається відстеження назад до батьківських вершин, щоб продовжити інший обхід. Якщо повертатися до самого кореня, виявиться частина графа, і повторюється цей процес для іншої непроходженої вершини, . Таким чином, обходимо весь граф. Час роботи DFS на графі G є $O(n + m)$, де n і m - це порядок і розмір G, відповідно. Під час виконання DFS ми відстежуємо батьківські вершини та часи початку і завершення, де час початку - це дискретний час, коли проходимо вершину, а час завершення - це дискретний час, коли закінчується проходженняусіх нащадків. Наступна теорема - головна причина, чому DFS має важливе значення для наших цілей [56].

Розглянемо теорему, що стверджує ациклічність графів порівняння, що мають велике значення у комп'ютерних науках та інженерії. Графи порівняння використовуються для моделювання алгоритмів сортування та аналізу даних. Теорема доводить, що такі графи завжди є напрямними ациклічними, що робить їх особливо корисними у вирішенні різноманітних задач.

Також наведені наслідки від цієї теореми, які підтверджують, що графи порівняння, а також їх відповідні графи масивів, є ациклічними та мають топологічні сортування. Ці висновки мають важливе значення для теорії алгоритмів, а також для практичного застосування у вирішенні різноманітних задач сортування та оптимізації обчислень.

Для виявлення паралелізму та керування ним [57], необхідно представити граф алгоритму у вигляді ярусно-паралельної форми, яка застосовується до алгоритмів, що не залежать від вхідних даних. Цей тип графів також відомий як потоковий граф. На рисунку 3.5 зображено потоковий граф для обраного алгоритму.

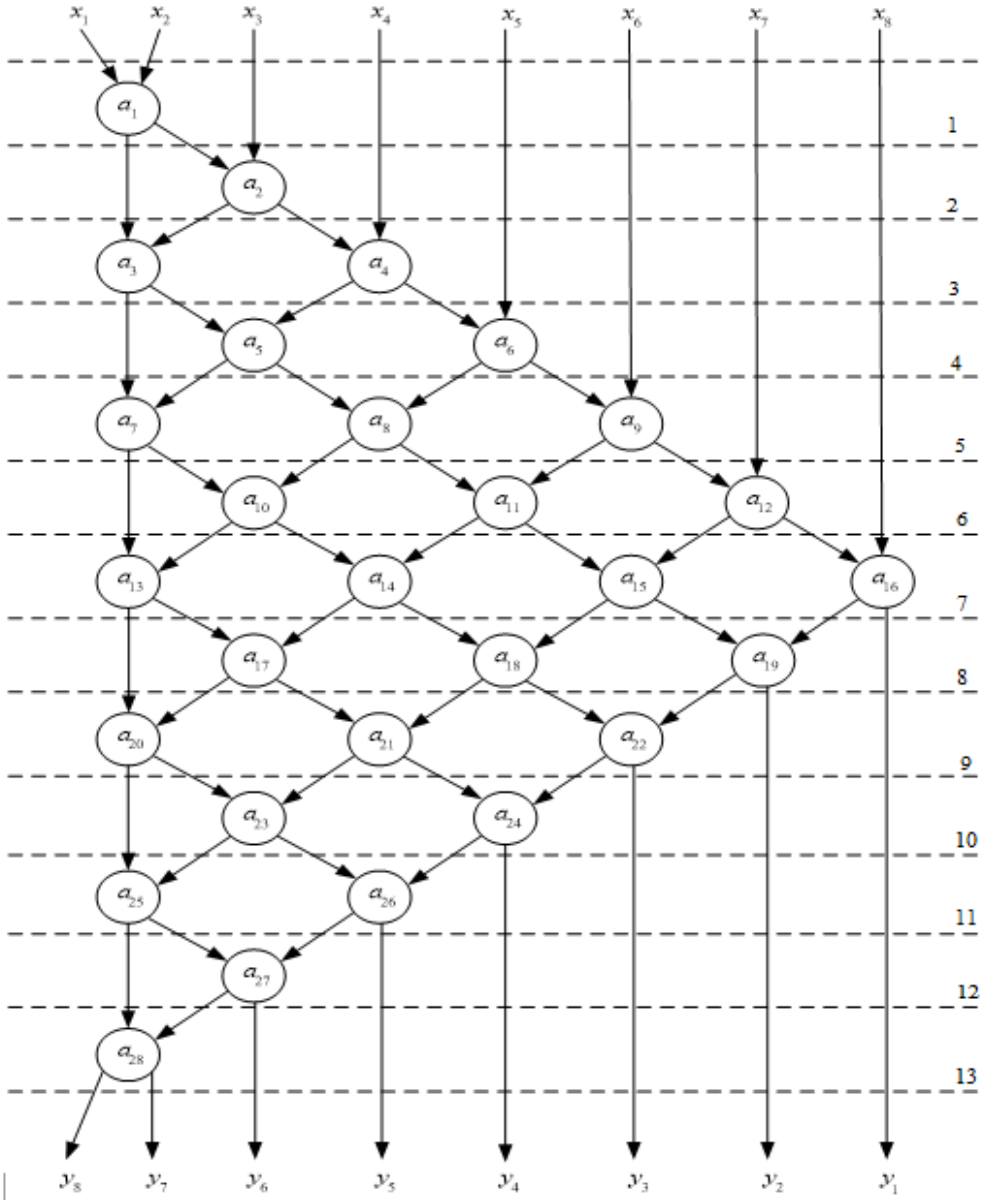


Рисунок 3.5 - Потоковий граф алгоритму сортування 8-ми чисел методом “бульбашки”

У потоковому графі алгоритму сортування методом "бульбашки" кожен ярус містить вершини, які відповідають за порівняння та обмін пар сусідніх

елементів [58]. Число вершин на кожному ярусі залежить від кількості елементів у вхідному масиві даних. Цей граф показує послідовність операцій, які виконуються над даними для їх відсортування.

Ширина або степінь паралелізму потокового графу алгоритму – максимальна кількість вершин на одному ярусі, визначаються за формулою $N/2$. В нашому випадку – це 4 вершини. Висота потокового графу – це кількість ярусів, для алгоритму сортування методом “бульбашки” визначаються за формулою $(2N-3)$. В нашому варіанті – це 13 ярусів. Пропускна здатність для даного графу складає 1 такт а часова затримка 13 тактів.

Тобто, ширина паралелізму визначає, скільки операцій можна виконати одночасно на одному ярусі, що впливає на швидкодію графу. Висота потокового графу визначається кількістю ярусів і вказує на кількість ітерацій, які потрібно виконати для завершення сортування. Таким чином, потоковий граф дозволяє краще розуміти роботу алгоритму сортування та може бути використаний для оптимізації його апаратної реалізації.

Для створення рекурсивного просторово-часового графу використовується методика просторово-часових графів [59]. Цей підхід полягає у зведенні потокового графу алгоритму до більш компактної форми, згорнувши вершини графу до однієї вершини, яка відображатиме послідовне виконання всіх операцій в часі.Таким чином, отримуємо рекурсивний просторово-часовий граф пристрою сортування за методом "бульбашки". Його структуру зображена на рисунку 3.6.

Рекурсивний просторово-часовий граф складається з однієї вершини, яка має два вхідні та два вихідні вузли. Цей граф представляє послідовне виконання операцій сортування методом "бульбашки". На дугах графу вказані числові такти [60], що відображають часові моменти, коли основні дані та проміжні результати подаються на вхідні вузли та надходять з вихідних вузлів відповідно.

Додатково, у графі присутні трикутні елементи, які відображають елементи затримки. Вони використовуються для затримки послідовності проміжних результатів на потрібну кількість тактів.

Часова затримка та пропускна здатність для рекурсивного просторово-

часового графу алгоритму сортування методом “бульбашки” співпадають і складають 29 тактів. Це важливі параметри, які визначають продуктивність та ефективність реалізації даного алгоритму.

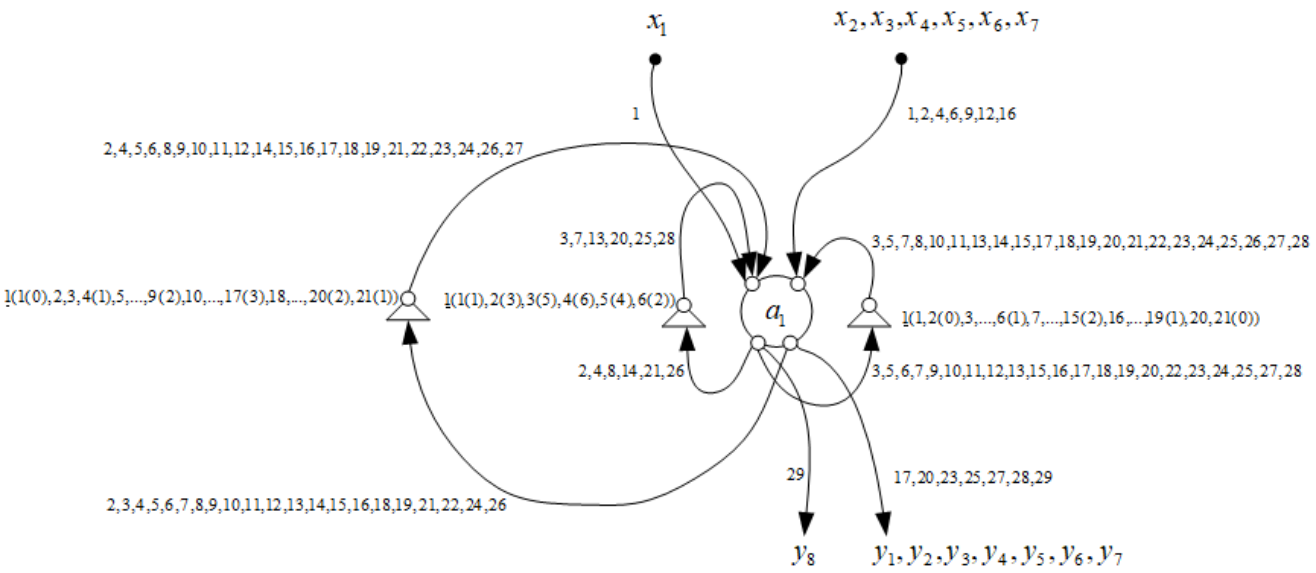


Рисунок 3.6 - Рекурсивний просторово-часовий граф алгоритму сортування 8-ми чисел методом “бульбашки”

Отже, цей рекурсивний граф відображає послідовне виконання операцій сортування, згорнутих у відповідні вершини, які представлені однією вершиною в просторі і часі. Такий підхід дозволяє краще зрозуміти структуру алгоритму і визначити можливі шляхи оптимізації та покращення його продуктивності.

Можливість застосування конвеєризації до алгоритму сортування підвищує продуктивність системи. Конвеєрний пристрій сортування, який зображений на рисунку 3.7 розбиває весь процес обчислення на послідовність закінчених кроків.

Кожен етап процедури порівняння і переставлення виконується на своїй сходинці конвеєра, при цьому всі сходинки конвеєра працюють паралельно. Результати, отримані на i-тій сходинці, передаються на подальшу обробку на наступну $(\mathrm{i}+1)$ сходинку конвеєра. Перенос інформації між сходинками відбувається через регістри (буферну пам'ять), розташовані між ними [61]. Результат виконаної операції зберігається в регістрах, і сходинка може

приступати до обробки наступної порції даних, використовуючи дані з регістрів.

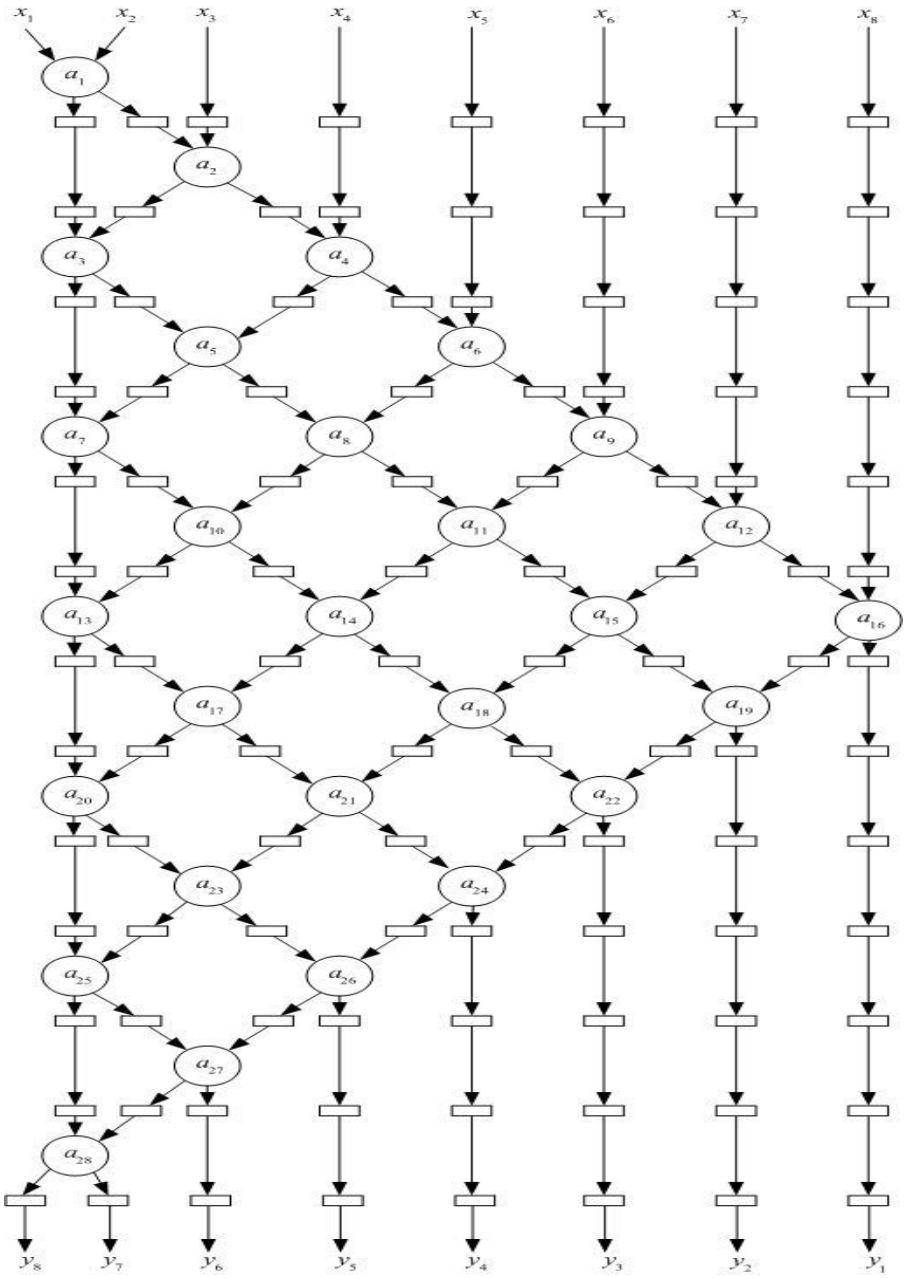


Рисунок 3.7 - Потоковий граф алгоритму сортування 8-ми чисел методом “бульбашки”

Синхронізація роботи конвеєра забезпечується тактовими імпульсами, період яких τ визначається найповільнішою сходинкою конвеєра τ_1 і затримкою в елементі буферної пам'яті $\tau_1: \tau = \max(\tau_1, \tau_2, \dots, \tau_k) + \tau_i$. У конвеєрному пристрої сортування з 13 сходинками відсортовані дані можуть подаватися на вхід з інтервалом в 13 разів меншим, ніж в звичайному пристрої сортування, і

результати появляються на виході з тим самим темпом [62].

Структура конвеєрного пристрою складається з 13 сходинок і нараховує 104 тригери. При цьому пропускна здатність і часова затримка складають один часовий інтервал.

3.3 Проектування вихідного інтерфейсу для HBIC

В даній системі HBIC використовується шина Wishbone з типом з'єднання "точка-точка" та циклом одноразового читання. Шина Wishbone має чотири можливі типи з'єднань: "точка-точка", "поток даних", "розділена шина" і "перехресний комутатор" [63].

З'єднання типу "точка-точка" є найпростішим способом комутації між двома IP-ядрами для шини Wishbone. На рисунку 3.8 показано, як інтерфейс ведучого пристрою комутується з інтерфейсом веденого пристрою.

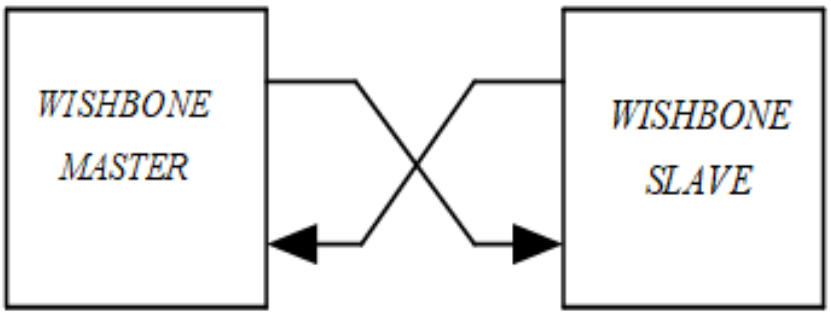


Рисунок 3.8 - З’єднання типу точка-точка

Шина Wishbone використовується для побудови вихідного інтерфейсу загальної схеми HBIC. Ця шина має тип з'єднання точка-точка та використовує цикл одноразового читання. Загалом, для шини Wishbone можливі чотири типи з'єднань: точка-точка, поток даних, розділена шина та перехресний комутатор.

Сигнали на шині Wishbone дозволяють MASTER і SLAVE взаємодіяти в різних режимах: одноразове читання/запис, блокове читання/запис, цикл зчитування-модифікації-запису. Асинхронний обмін даними гнучко керує процесом та вирівнює швидкості циклів.

На шині Wishbone можна використовувати користувацькі сигнали-теги, що розширює її функціональність. Шина підтримує різні типи з'єднань, цикли читання/запису та механізм асинхронного обміну, забезпечуючи вирівнювання швидкостей циклів. Організація всіх сигналів така, що інтерфейси MASTER і SLAVE [65], що зображено на рисунку 3.9 можуть напряму комутуватися один з іншим у формі нескладного з’єднання типу точка-точка, що спрощує реалізацію з’єднання за правилами шини Wishbone.

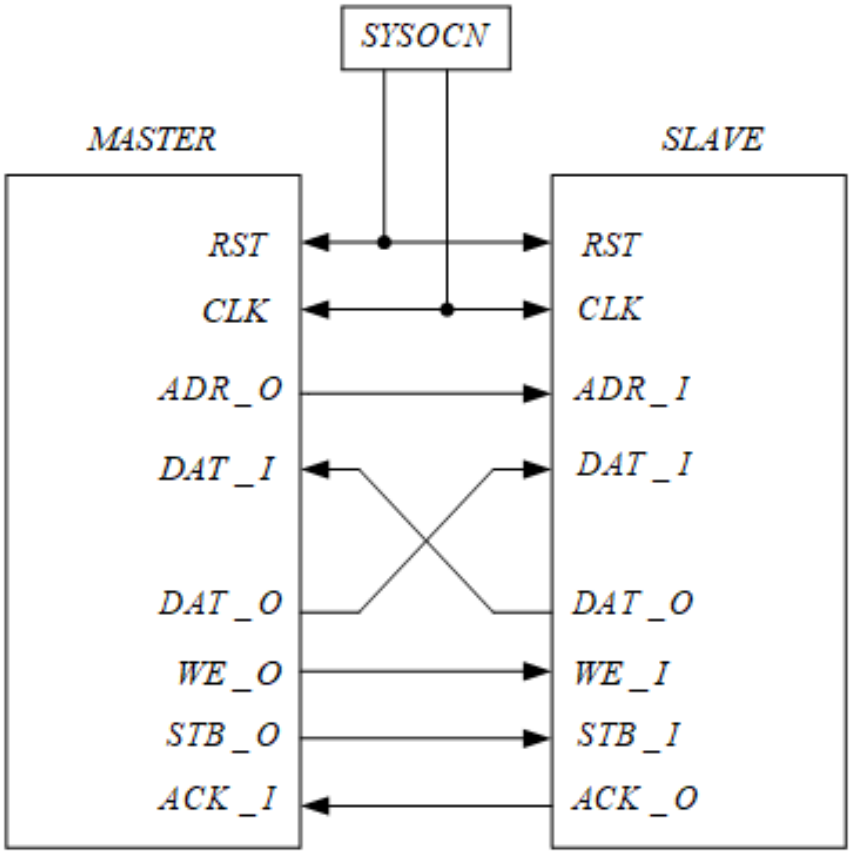


Рисунок 3.9 - Сигнали шини wishbone при з’єднанні точка-точка

Найпоширенішим типом циклу на шині Wishbone є одноразовий цикл читання/запису (single read/write cycle). Він використовується для передачі одного операнду даних через шину. На рисунку 3.10 показано, як відбувається передача даних під час циклу одиночного читання (single read) [66].

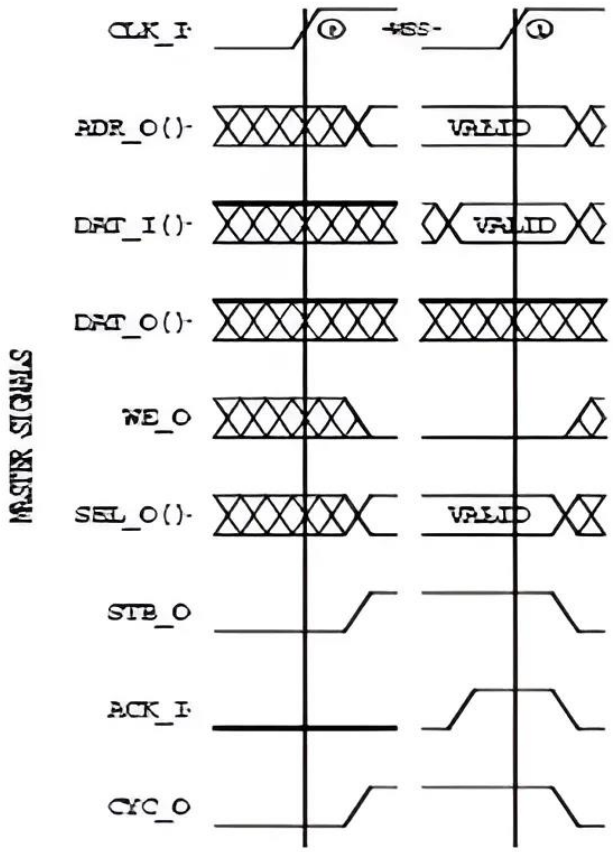


Рисунок 3.10 – Цикл одиночного читання

У циклі одиночного читання майстер формує сигнали на вихідних лініях після першого такту, включаючи адресу, прапорець запису, вибірку та цикл. Потім SLAVE розпізнає початок нового циклу, перевіряючи вхідні адреси, та генерує відповідні дані на власних вихідних лініях. Після того, як дані готові, SLAVE повідомляє майстеру за допомогою сигналу підтвердження. Майстер очікує підтвердження від SLAVE та після його отримання зчитує дані та відправляє вихідний стрімбу відповідно до підтвердження.

3.3 Висновки

У даному розділі досліджується структурна схема HBIC (надвелика інтегральна схема). Проведена побудова загальної схеми системи, яка містить опис основних вузлів. Зокрема, виконано побудову графа алгоритму сортування 8

чисел методом "бульбашки". На основі цього графа алгоритму створено потоковий граф для виявлення паралелізму операцій та створено конвеєрну структуру пристрою.

Розроблено рекурсивний просторово-часовий граф (ПЧГ). Виконано опис протоколу передачі даних для вихідного інтерфейсу, який базується на шині wishbone.

Подано структуру графу керуючого автомату, побудованого за принципом функціонування автомата Мура, та проаналізовано стани роботи кожної вершини цього графу.

Також проведено детальний аналіз станів роботи кожної вершини графу керуючого автомату. Це дозволяє краще зрозуміти логіку функціонування автомата та визначити його ефективність управління операціями пристрою. Крім того, надано огляд та пояснення структури операційного блоку, який відповідає за виконання операцій порівняння та перестановки двох значень у рекурсивному пристрої сортування.

Необхідно також відзначити, що у процесі роботи над структурною схемою НВІС було приділено увагу ефективності використання ресурсів, зокрема, оптимізації кількості використовуваних ресурсів та мінімізації затримок у пристрої. Це допомагає підвищити продуктивність та швидкодію НВІС у реальних умовах експлуатації.

В цілому, описані у розділі деталі надають повну картину про структуру та принципи роботи удосконаленого сортуванняметоду сортування , що допомагає зрозуміти принципи його функціонування та визначити його ефективність у вирішенні конкретних завдань сортування.

4 ПРОЕКТУВАННЯ СПЕЦІАЛІЗОВАНОЇ СИСТЕМИ СОРТУВАННЯ ДАНИХ ТА ЇЇ КОМПОНЕНТІВ

4.1 Архітектура системи сортування

На рисунку 4.1 зображено базову структуру спеціалізованої системи сортування двійкових даних. Вона складається з вхідного інтерфейсу, операційного автомату, вихідного інтерфейсу та пристрою керування [1] .

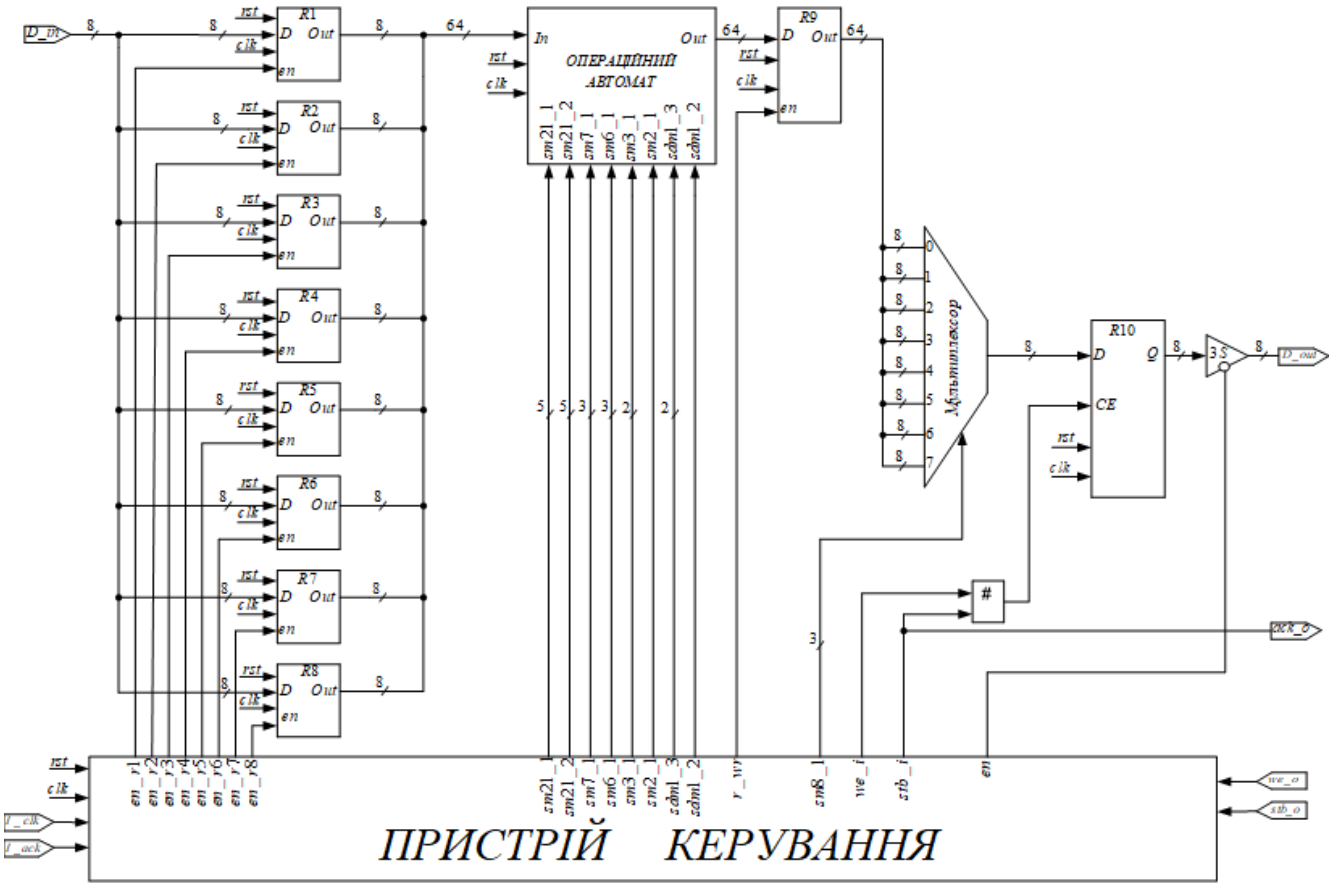


Рисунок 4.1 – Структура спеціалізованої системи сортування

Інформаційний тракт утворюється з вхідного інтерфейсу, рекурсивного пристрою сортування та вихідного інтерфейсу. На вхід 8-ми розрядної шини даних D_in, після установки сигналу I_ack (підтвердження надходження вхідних даних) у стан активності та подачі сигналу I_clk (тактування вхідних даних) по черзі надходять вісім однобайтових чисел [67]. Ці числа потрапляють до восьмох

8-ми розрядних регістрів (R1, R2, R3, R4, R5, R6, R7 та R8), де вони послідовно записуються на основі сигналів дозволу запису (en_r1, en_r2, en_r3, en_r4, en_r5, en_r6, en_r7 та en_r8), що генеруються керуючим пристроєм. Передача вхідних даних здійснюється по байтах. Часову діаграму протоколу передачі вхідних даних зображено на рисунку 4.2.

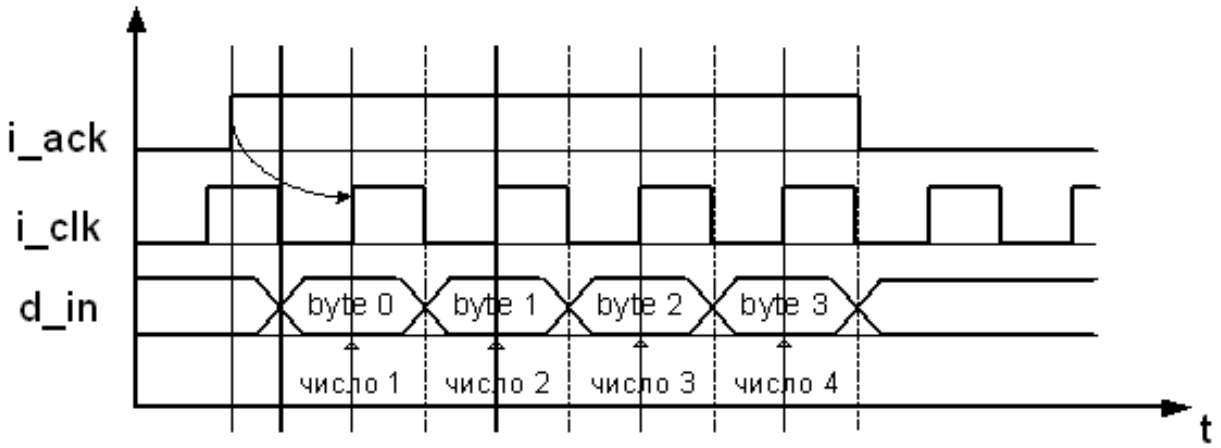


Рисунок 4.2 - Часова діаграма протоколу передачі вхідних даних

На цій діаграмі вхідні числа передаються групами через 8-бітову шину даних (d_in). Після цього дані з регістрів об'єднуються у 64-бітовий пакет та подаються на вхід операційного автомату для обробки. Операційний автомат виконує функції рекурсивного пристрою сортування, який складається із шести мультиплексорів (двох 21-входових, та по одному 7-, 6-, 3- та 2-входових), двох демультиплексорів та блоку порівняння і переставлення даних. Після цього дані записуються до 64-бітового регістру (R9) при активності сигналу r_wr. Дані з регістру подаються на мультиплексор, який має вісім 8-розрядних входів та три бітових входи керування [68].

Після цього дані проходять через 8-бітовий регістр, який реалізує 8-бітовий wishbone (slave) вихідний порт з типом з’єднання точка-точка та циклом одиночного читання. Даний регістр складається з D-тригера з асинхронним скидом та двох входової схеми виключного АБО.

Оскільки використовується цикл одиночного читання, то на виході з 8-

бітового регістра дані поступають на трьох становий елемент, який на основі сигналу дозволу en, який генерує керуючий пристрій, по черзі видає дані на вихідну шину даних D_out. Пристрій керування реалізований за допомогою автомату Мура.

Процес роботи ЕОМ, такої як мікропроцесор, включає послідовність тактових інтервалів, під час яких операційний автомат виконує елементарні операції над операндами і надає результати обробки. Операційний автомат керується відповідними сигналами керування, які генеруються пристроєм керування на основі коду операції та службових сигналів стану операційного автомату [69].

Функціональність операційного автомату забезпечується рекурсивним пристроєм сортування, побудованим на основі рекурсивного просторово-часового графу алгоритму сортування методом "бульбашки". У апаратному виконанні вершинам графу відповідають арифметико-логічні пристрої або операційні блоки, а вхідним та вихідним вузлам - мультиплексори та демультиплексори відповідно. Для забезпечення необхідної затримки використовуються регістри, що відповідають елементам затримки рекурсивного графу.

Операційний автомат отримує на вхід вісім однобайтових цілих чисел, які обробляються послідовно за допомогою операційного блоку порівняння та перестановки. На виході операційного автомату отримується відсортована в порядку спадання послідовність восьми однобайтових значень.

Електрична функціональна схема операційного автомату, зображена на рисунку 4.3 включає шість мультиплексорів (М1, М2, М3, М4, М5, М6), два демультиплексори (Дм1, Дм2), один операційний блок порівняння та перестановки, а також чотирнадцять регістрів для затримки вхідних та вихідних даних.

На структурній схемі операційного автомату мультиплексори та демультиплексори керують подачею та видачею даних та проміжних результатів до та від операційного блоку, який виконує операцію порівняння та перестановки двох значень 28 разів послідовно в часі. Важливо зазначити, що на даній схемі не

показані сигнали керування мультиплексорами та демультиплексорами, які генеруються пристроєм керування [70].

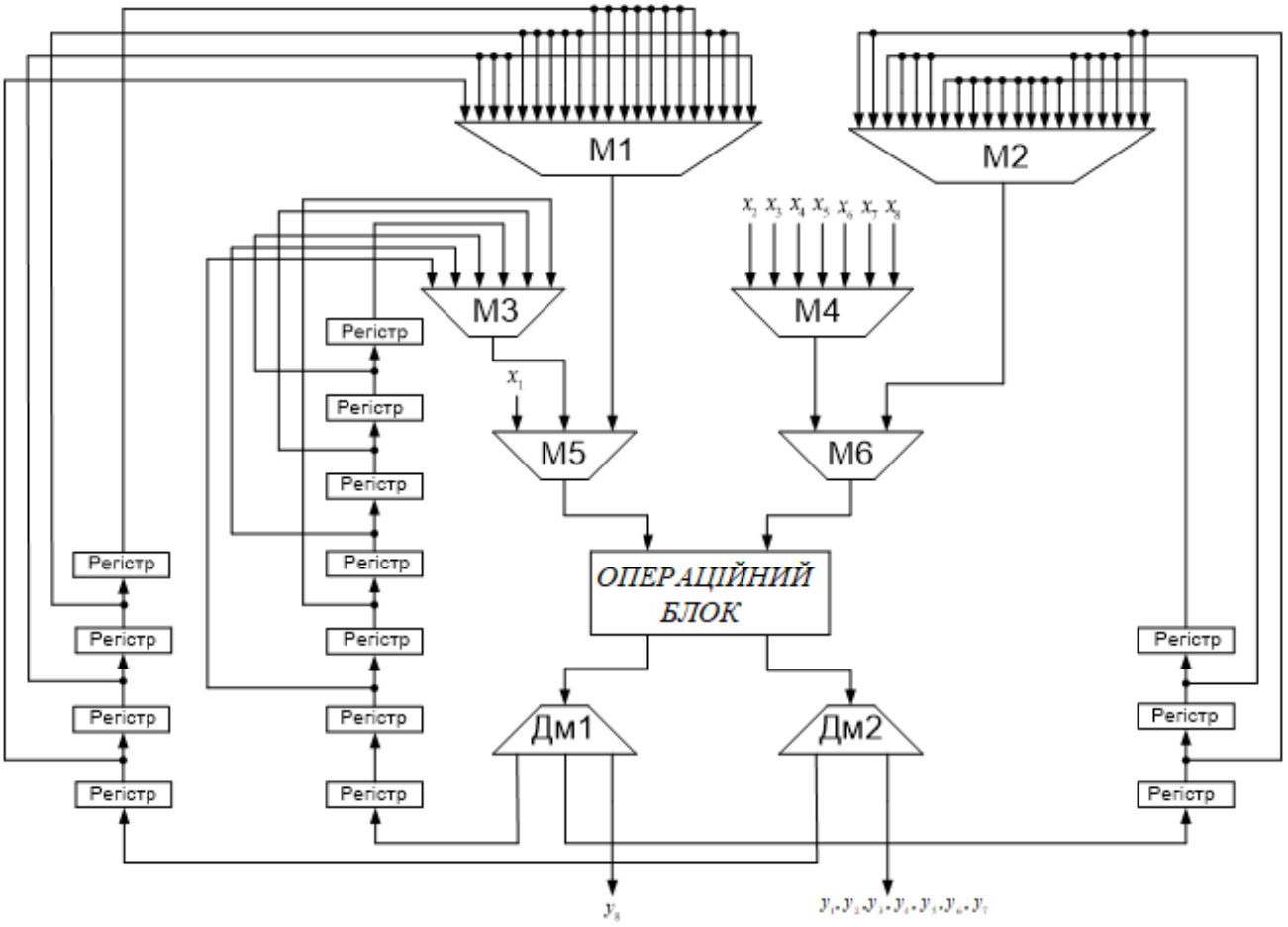


Рисунок. 4.3 - Схема електрична функціональна операційного автомату

Регістри, що забезпечують необхідну затримку вхідних та вихідних даних, працюють у режимі постійного переміщення даних. Крім того, для одночасної видачі кінцевого результату роботи необхідно затримати певні значення на другому виході демультиплексора Дм2 протягом певної кількості тактів, щоб всі вісім значень були доступні на виході одночасно.

4.2 Тестування системи сортування

Основною теоретичною базою для розробки пристроїв керування є теорія абстрактних автоматів Мілі та Мура [71]. У цій теорії ключовим поняттям є

"стан". Стан визначається як унікальний набір дій, які виконує пристрій протягом певного проміжку часу, так званого синхро-такту. Сам абстрактний автомат призначений забезпечити послідовність переходів між станами в часі для виконання певного алгоритму.

Для забезпечення вірного порядку переходів між станами в часі використовується механізм комірок пам'яті. Ці комірки пам'яті фіксують поточний стан пристрою, а також дозволяють визначати, до якого наступного стану потрібно перейти на основі функцій переходів. Урахування вхідних даних також впливає на визначення наступного стану пристрою, роблячи його більш адаптивним і гнучким у відповіді на зміни у вхідних умовах.

У класі синхронних автоматів найбільш поширеними є автомати Мілі та Мура. Особливість автоматів Мілі полягає в тому, що їх вихідні сигнали в певний момент часу не залежать від поточних станів автомата або вхідних сигналів в цей самий момент часу [72]. У свою чергу, вихідні сигнали автомата Мура однозначно визначаються його поточним станом в той же момент часу і не залежать від значень вхідних сигналів. Є ще один тип автоматів, які поєднують властивості Мілі та Мура, відомий як С-автомат. Він одночасно реалізує дві функції виходів, кожна з яких характерна для Мілі і Мура.

Граф керуючого автомату відповідає за генерацію сигналів, що керують вхідним інтерфейсом, операційним автоматом та вихідним інтерфейсом. Початковий стан INIT характеризується активним сигналом початкового скиду rst = ‘1’, при цьому всі інші сигнали перебувають у неактивному стані. При переході до вершини S0 сигнал rst переходить у стан '0'. При переході з вершини S0 в вершину S1 активується сигнал I_ack (підтвердження надходження вхідних даних). Починаючи з вершини S1 і закінчуючи вершиною S15, генерується сигнал I_clk (тактування вхідних даних). В вершинах S1, S3, S5, S7, S9, S11, S13 та S15 також активуються сигнали дозволу запису до вхідних регістрів (R1,...,R8).

Після переходу з вершини S16 до вершини S43 працює операційний автомат, керуючий автомат генерує сигнали керування для мультиплексорів та демультиплексорів операційного автомату протягом 28 тактів, оскільки

реалізується рекурсивний пристрій сортування. У вершині S43 генерується сигнал дозволу запису до вихідного регістру (R9). Починаючи з вершини S44 і закінчуючи вершиною S60, умовою переходу є два сигнали: we_o (дозвіл видачі адреси) та stb_o (сигнал стробування). У вершинах S45, S47, S49, S51, S53, S55, S57 та S59 генеруються сигнали: mux_sel (керуючий сигнал 8-ми входового мультиплексора) та en (дозвіл видачі вихідних даних) [73].

На рисунку 4.4. наведено блок-схему модуля сортування, що складається з восьми однобайтових входів (D_in1,..., D_in8), де надходять вхідні дані, та восьми однобайтових виходів (D_out1,..., D_out8), де формується результат сортування.

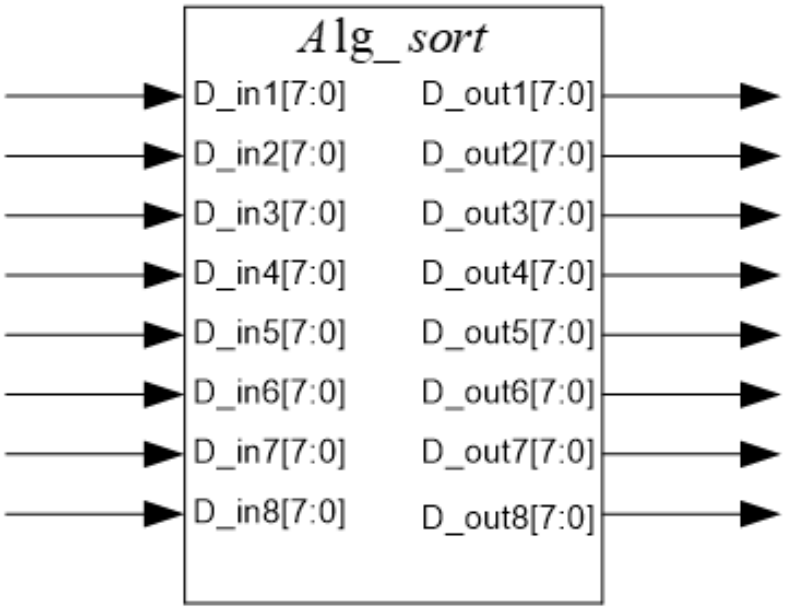


Рисунок 4.4 - Блок-схема пристрою сортування

Нижче подано лістинг vhdl-коду інтерфейсу пристрою сортування 8-ми чисел методом “бульбашки”, який є описом вхідного та вихідного інтерфейсу для модуля сортування:

entity alg_sort is port

(D_in1, D_in2, D_in3, D_in4, D_in5, D_in6, D_in7, D_in8: in std_logic_vector(7 downto 0);

D_out1, D_out2, D_out3, D_out4, D_out5, D_out6, D_out7, D_out8: out

```
std_logic_vector(7 downto 0) );
        end alg_sort;
```

Де entity alg_sort is - рядок визначає початок опису сутності (entity) у VHDL. Сутність - це аналог модуля чи компонента в мові опису апаратури. port(...) - ця частина визначає порти, тобто вхідні та вихідні сигнали сутності. У даному випадку, оголошуються вхідні сигнали D_in1 до D_in8 та вихідні сигнали D_out1 до D_out8, кожен з яких є вектором з 8 бітами (std_logic_vector(7 downto 0)). В свою чергу end alg_sort цей рядок позначає завершення визначення сутності.

На рисунку 4.5 зображена діаграма функціональної симуляції пристрою сортування 8-ми чисел методом “бульбашки”.

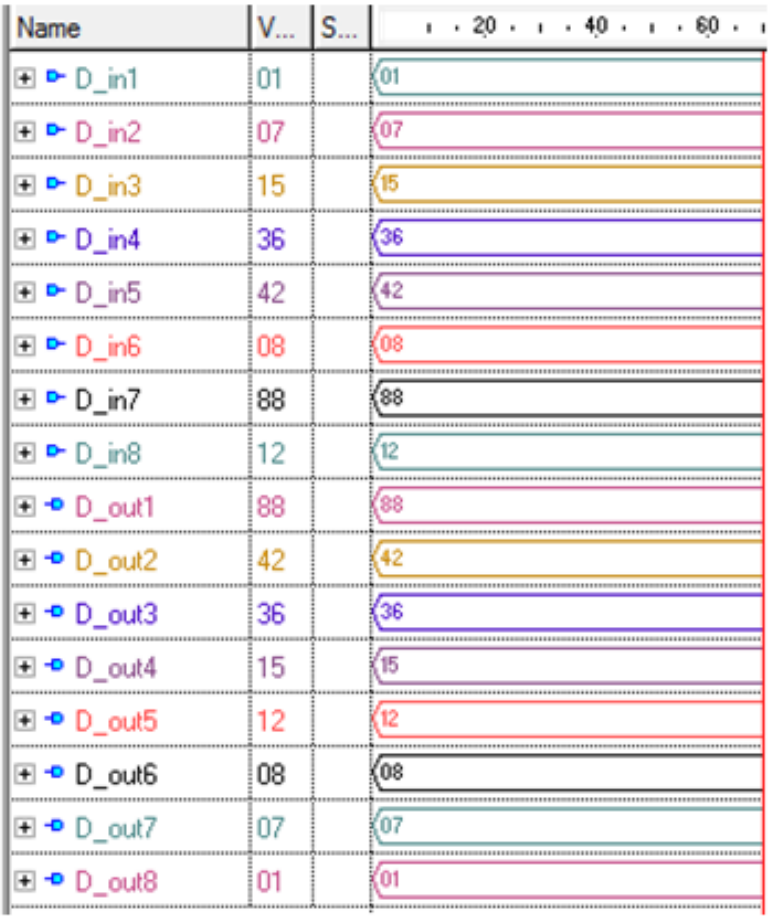


Рисунок 4.5 - Діаграма функціональної симуляції пристрою сортування

Діаграма ілюструє процес сортування вхідних однобайтових чисел методом

"бульбашки". Пристрій отримує на вході вісім однобайтових чисел, які потім сортуються у порядку спадання. Кожне вхідне число порівнюється з наступним, і якщо воно менше за наступне, вони обмінюються місцями. Цей процес повторюється до тих пір, поки всі числа не будуть відсортовані у порядку спадання. На виході ми отримуємо відсортовані значення вхідних даних, що показані на відповідних вихідних сигналах D_out1 до D_out8.

На рисунку 4.6 подано блок-схему конвеєризованого модуля пристрою сортування зображено на розширену структуру з включенням додаткових портів [74] для синхронізації (clk) та скиду (rst). Ці порти дозволяють забезпечити синхронну роботу пристрою за тактовим сигналом (clk) та можливість скидання внутрішнього стану до початкового (rst).

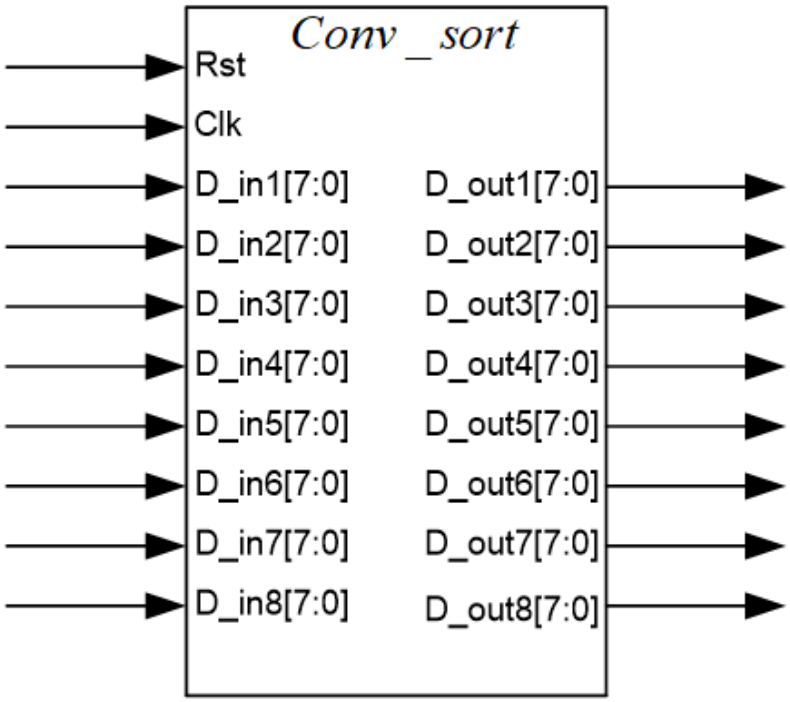


Рисунок 4.6 - Блок-схема конвеєрного пристрою сортування

На блок-схемі також присутні основні компоненти модуля сортування, які були згадані раніше, такі як вхідні порти (D_in1 до D_in8) та вихідні порти (D_out1 до D_out8), які призначені для передачі вхідних та відсортованих даних відповідно.

Нижче подано vhdl-код інтерфейсу, що описує сутність (entity) conv_sort, яка представляє інтерфейс модуля конвеєризованого пристрою сортування 8-ми чисел методом "бульбашки" у мові VHDL:

```
entity conv_sort is port
( Rst, Clk: in std_logic; D_in1, D_in2, D_in3, D_in4, D_in5, D_in6, D_in7, D_in8: in
std_logic_vector(7 downto 0); D_out1, D_out2, D_out3, D_out4, D_out5, D_out6,
D_out7, D_out8: out std_logic_vector(7 downto 0) );
 end conv_sort;
```

На діаграмі функціональної симуляції конвеєрного пристрою сортування, що на рисунку 4.7, представлено послідовність подій та залежності між ними. Формування результату, тобто відсортованих даних, відбувається на 14-му такті за зростаючим фронтом синхроімпульсу [75]. Це стало можливим завдяки присутності в операційному автоматі 13-ти конвеєрних регістрів, які забезпечують послідовну обробку даних у різних етапах операційного процесу.

Рисунок 4.7 - Діаграма функціональної симуляції конвеєрного пристрою сортування

На блок-схемі, з рисунку 4.8, рекурсивного модуля пристрою сортування 8-ми чисел за методом "бульбашки" видно, що процес сортування відбувається шляхом послідовного порівняння чисел і їх перестановки у відповідності до порядку. Кожне порівняння та перестановка виконуються за допомогою логічних операцій та управління керуючим автоматом.

У цій блок-схемі також відображені рекурсивні виклики, коли необхідно відсортувати меншу підмножину чисел. Це рекурсивне викликається досягається за умови, що поки є невідсортовані елементи. Кожен такий виклик обробляється окремо, допоки всі елементи не будуть впорядковані.

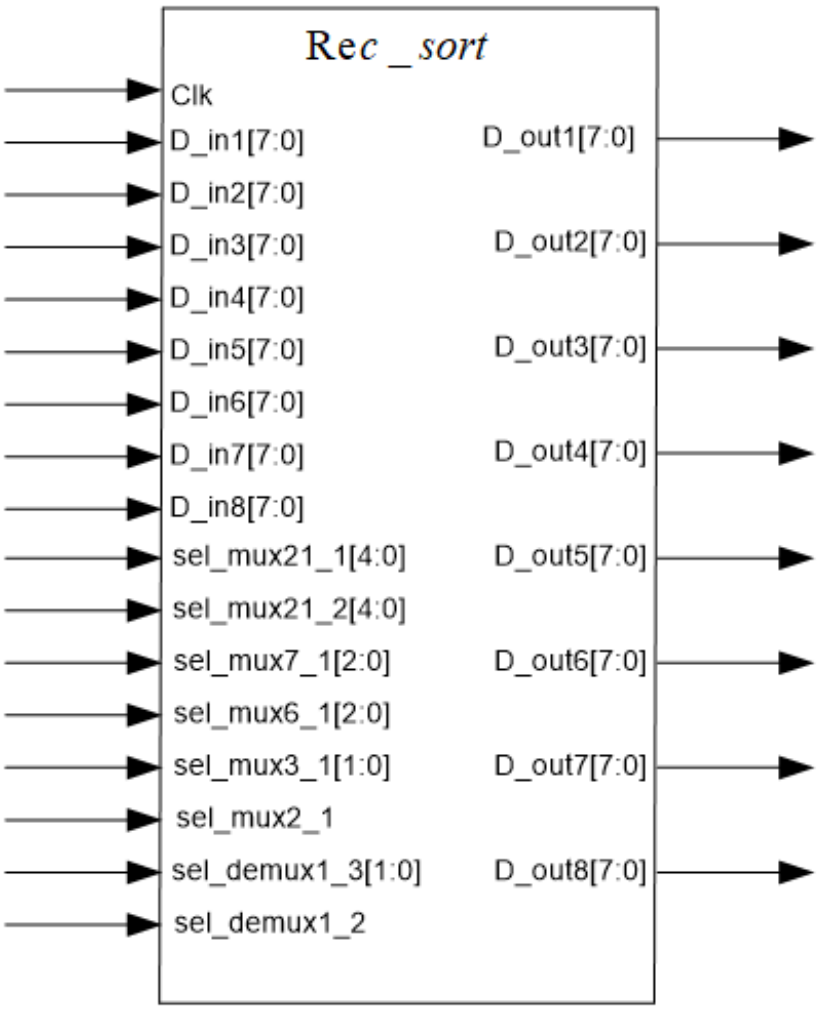


Рисунок 4.8 - Блок-схема рекурсивного пристрою сортування

Наведений нижче код опис інтерфейсу для модуля рекурсивного пристрою сортування 8-ми чисел методом "бульбашки" на мові VHDL. Окрім вхідних та

вихідних сигналів даних, у цьому інтерфейсі також присутні сигнали керування, які використовуються для управління мультиплексорами та демультиплексорами операційного автомату.

```
entity rec_sort is port
 ( Clk: in std_logic; D_in1, D_in2, D_in3, D_in4, D_in5, D_in6, D_in7, D_in8: in
std_logic_vector(7 downto 0);
sel_mux21_1, sel_mux21_2, sel_mux7_1, sel_mux6_1, sel_mux3_1: in
std_logic_vector(4 downto 0); sel_mux2_1: in std_logic; sel_demux1_3: in
std_logic_vector(1 downto 0);
sel_demux1_2: in std_logic; D_out1, D_out2, D_out3, D_out4, D_out5, D_out6,
D_out7, D_out8: out std_logic_vector(7 downto 0) );
end rec_sort;
```

Цей код описує, як дані надходять до модуля сортування, як вони обробляються та які дані видаються на виході. Крім того, він визначає, які сигнали використовуються для управління внутрішніми блоками модуля для ефективного сортування даних.

На діаграмі, що показана на рисунку 4.9 також видно, як кожен з вказаних 6-ти мультиплексорів (sel_mux21_1, sel_mux21_2, sel_mux7_1, sel_mux6_1, sel_mux3_1, sel_mux2_1) та 2-х демультиплексорів (sel_demux1_3, sel_demux1_2) отримує свої вхідні сигнали керування. Ці сигнали дозволяють вибирати потрібні дані для обробки та передачі через пристрій.

Кожний мультиплексор та демультиплексор працює з відповідними сигналами керування, що забезпечує ефективну роботу пристрою сортування.

Зазначений рекурсивний пристрій сортування виконує 28 операцій порівняння та переставлення послідовно в часі. Це вказує на те, що пристрій працює в режимі, де кожна операція виконується після завершення попередньої, і всі вони проходять через цикл рекурсивного сортування. Така послідовність операцій дозволяє досягти відсортованого результату на виході пристрою.

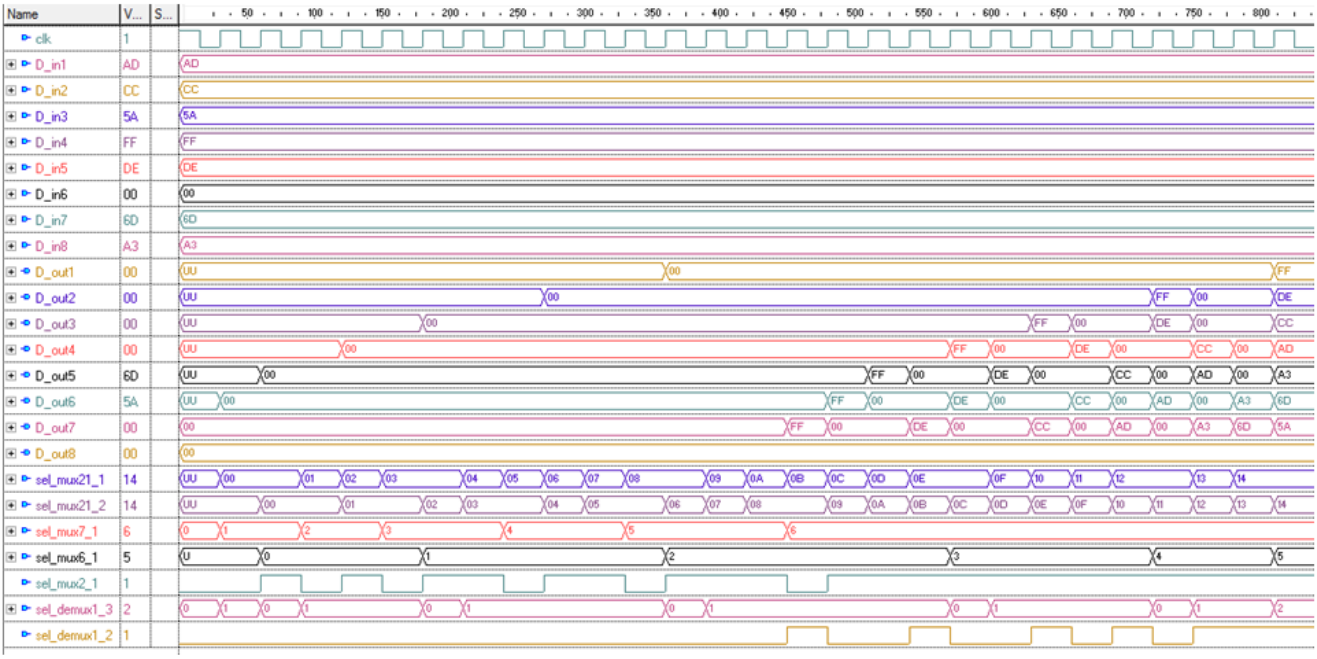


Рисунок 4.9 - Діаграма симуляції рекурсивного пристрою сортування

На діаграмі, що показана на рисунку 4.9 також видно, як кожен з вказаних 6-ти мультиплексорів (sel_mux21_1, sel_mux21_2, sel_mux7_1, sel_mux6_1, sel_mux3_1, sel_mux2_1) та 2-х демультиплексорів (sel_demux1_3, sel_demux1_2) отримує свої вхідні сигнали керування. Ці сигнали дозволяють вибирати потрібні дані для обробки та передачі через пристрій. Кожний мультиплексор та демультиплексор працює з відповідними сигналами керування, що забезпечує ефективну роботу пристрою сортування.

Зазначений рекурсивний пристрій сортування виконує 28 операцій порівняння та переставлення послідовно в часі. Це вказує на те, що пристрій працює в режимі, де кожна операція виконується після завершення попередньої, і всі вони проходять через цикл рекурсивного сортування. Така послідовність операцій дозволяє досягти відсортованого результату на виході пристрою [76].

На рисунку 4.10 зображено блок-схему модуля керуючого автомату пристрою сортування 8-ми чисел за методом "бульбашки" можна побачити логіку роботи та управління в процесі сортування. Робота розпочинається з початкового стану INIT, де сигнал скиду (rst) має значення '1'. Після збросу сигналу скиду (rst) до нуля, система переходить у початковий стан S0. Звідси відбувається

послідовний перехід через інші стани S1, S2, ..., S43, кожен з яких відповідає певній фазі сортування. На кожному кроці роботи керуючий автомат видає відповідні сигнали керування, такі як дозвіл запису до вхідних регістрів, тактування вхідних даних, видача адреси, сигнали стробування, керуючі сигнали мультиплексорів та демультиплексорів і т.д. Після завершення 28-ми тактів операцій порівняння та перестановок у стані S43, сигнал дозволу запису до вихідного регістру (R9) дозволяє зберегти відсортований результат. Після збереження результату система може повернутися в початковий стан INIT або перейти в інший відповідний режим для подальших операцій.

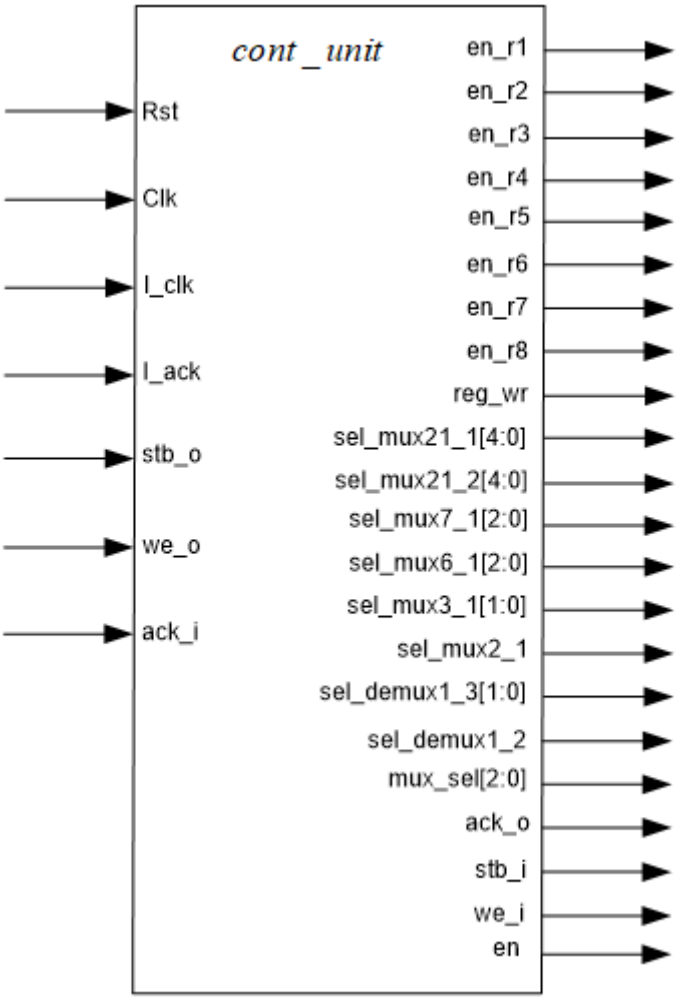


Рисунок 4.10 - Блок-схема керуючого автомата

Код визначає інтерфейс (порти) для контролера (керуючого автомата) модуля сортування:

entity cont_unit is port

(clk, rst, I_ack, I_clk, ack_o, we_o, stb_o, we_i, stb_i, en_r1, en_r2, en_r3, en_r4, en_r5, en_r6, en_r7, en_r8, en, reg_wr, sel_mux2_1, sel_demux1_2: in std_logic; ack_i: out std_logic;

sel_mux3_1, sel_mux6_1, sel_mux7_1, sel_mux21_1, sel_mux21_2, sel_demux1_3, mux_sel: out std_logic_vector (4 downto 0));

end cont_unit;

Основні вхідні сигнали включають тактовий сигнал clk, сигнал скидання rst, а також сигнали I_ack та I_clk, які вказують на готовність вхідних даних та тактовий сигнал для вхідних даних відповідно. Після отримання сигналу підтвердження прийому вхідних даних (рисунок 4.11) I_ack, керуючий автомат активує строби для дозволу запису в вхідні регістри. Ці строби, позначені як en_r1 до en_r8, дозволяють зберегти вхідні дані в відповідних регістрах.

Крім того, активується сигнал reg_wr, що дозволяє записати дані у вихідний регістр.

Керуючий автомат також визначає відповідні сигнали керування для мультиплексорів та демультиплексорів операційного автомата, щоб забезпечити правильну роботу операційного блоку під час сортування. Ці сигнали, позначені як sel_mux21_1, sel_mux21_2, sel_mux7_1, sel_mux6_1, sel_mux3_1, sel_mux2_1, sel_demux1_3 та sel_demux1_2, визначають, які дані будуть передаватися до операційного автомата та з нього.

Крім того, керуючий автомат формує сигнал mux_sel, який керує мультиплексором для вибору необхідного сигналу на виході. Ці сигнали дозволяють операційному автомату правильно виконувати операції порівняння та переставлення під час сортування.

Нарешті, пристрій керування генерує вихідні сигнали для взаємодії з вихідним інтерфейсом на основі шини wishbone. Ці сигнали дозволяють передавати результати сортування та взаємодіяти з іншими частинами системи.

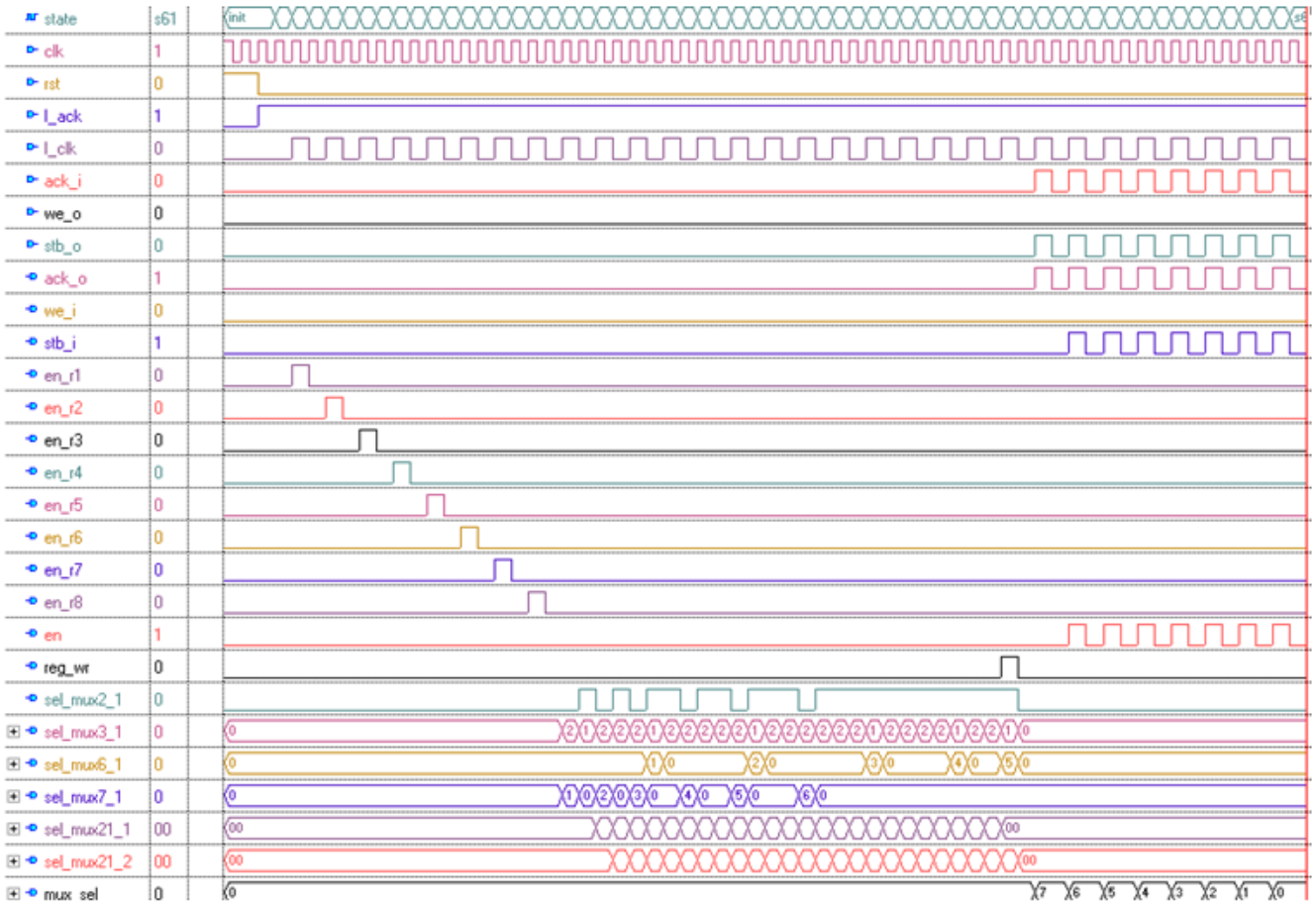


Рисунок 4.11 - Діаграма функціональної симуляції керуючого автомата

На рисунку 4.12 представлено блок-схему завершеної структури апаратного пристрою сортування, яка включає в себе всі компоненти, необхідні для виконання операції сортування вхідних даних.

У верхній частині схеми зображено модуль керуючого автомата, який відповідає за управління всіма іншими блоками пристрою. Керуючий автомат приймає вхідні дані, ініціалізує операційний автомат та координує всі інші операції.

Нижче розміщені два блоки операційного автомата, які відповідають за виконання фактичного сортування. Кожен операційний автомат отримує вхідні дані та виконує порівняння та переставлення, використовуючи внутрішню логіку [77].

Посередині схеми розташовані модулі вхідного та вихідного інтерфейсів, які забезпечують зв'язок пристрою з зовнішнім середовищем. Ці модулі

відповідають за прийом вхідних даних та передачу результатів сортування через внутрішню шину даних.

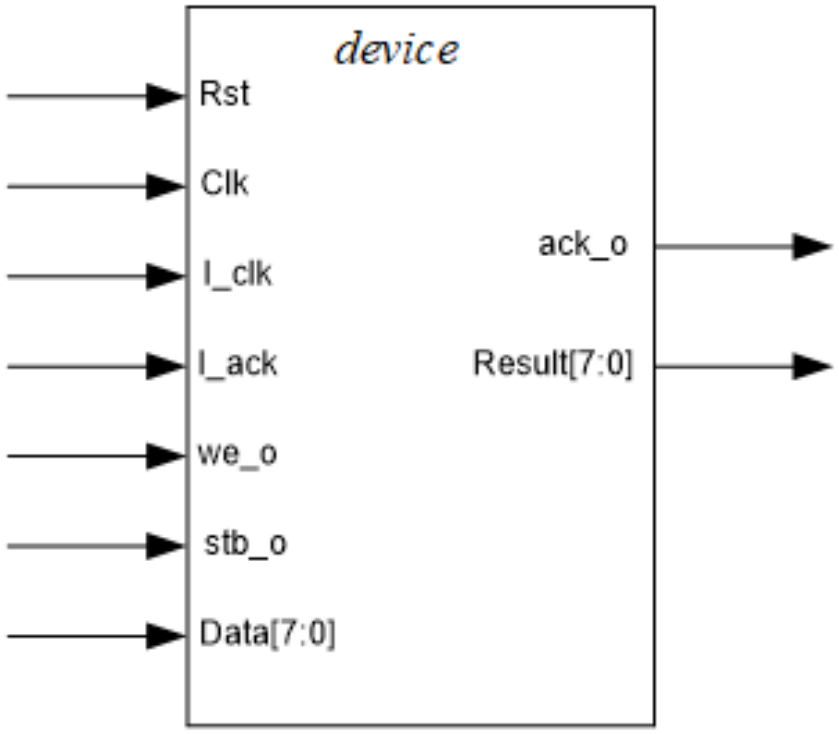


Рисунок 4.12 - Блок-схема завершеного пристрою

Внизу схеми розміщений блок зовнішньої системної шини, який забезпечує зв'язок пристрою з іншими компонентами системи. Цей блок відповідає за обмін даними та командами між апаратним пристроєм сортування та іншими модулями чи пристроями у системі.

В цілому, ця блок-схема представляє повну апаратну реалізацію алгоритму сортування, яка включає в себе всі необхідні компоненти для ефективної та швидкої обробки великого обсягу даних. Цей код описує інтерфейс для завершеної структури апаратного пристрою сортування:

```
entity device is port
( clk, rst, I_clk, I_ack, we_i, stb_i: in std_logic; ack_o: out std_logic;
Data: in std_logic_vector(7 downto 0); Result: out std_logic_vector(7 downto 0);
end device;
```

На рисунку 4.13 на діаграмі можна спостерігати, як дані восьми розрядів записуються у відповідні вхідні регістри, коли сигнали дозволу регістрів (en_r1, en_r2, en_r3, en_r4, en_r5, en_r6, en_r7, en_r8) активні. Ці дані подаються на подальшу обробку в операційний автомат за допомогою рекурсивного пристрою сортування.

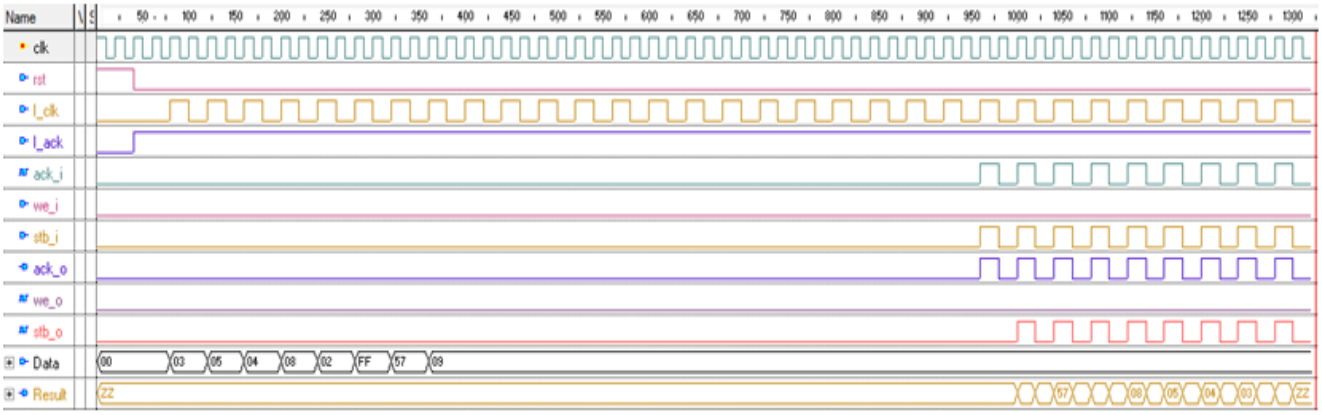


Рисунок 4.13 - Діаграма функціональної симуляції роботи завершеного структури HBIC

Після обробки результати зберігаються у 64-розрядному регістрі на виході операційного автомата та подаються на вхід 8-входового мультиплексора. Відтак, вони формують 64-розрядний пакет даних, який подається на вхід 8-розрядного регістра. Цей регістр включає в себе тригер з асинхронним записом та два входових елементи додавання по модулю 2. Ця структура дозволяє взаємодіяти з іншими пристроями за допомогою шини wishbone, забезпечуючи цикл одноразового читання/запису.

Для забезпечення правильної взаємодії між пристроями, майстер і раб формують сигнали дозволу надходження вхідних та вихідних даних, сигнали стробування та сигнали дозволу запису. Це дозволяє послідовно видачі результатів роботи на вихідний інтерфейс.

На даній діаграмі також демонструється передача даних через систему шини wishbone, що дозволяє взаємодіяти з іншими пристроями на основі циклу одноразового читання/запису. За допомогою сигналів дозволу надходження

вхідних та вихідних даних, а також сигналів стробування та дозволу запису, пристрій може послідовно видачі результатів роботи на вихідний інтерфейс. Така організація взаємодії дозволяє ефективно і надійно обмінюватися даними між різними компонентами системи.

4.3 Оцінка ефективності та швидкодії системи сортування

Після завершення моделювання на VHDL основних модулів базової структури НВІС пристрою сортування, наступним кроком є їх синтез на програмованій логічній інтегральній схемі Spartan3 від Xilinx. Цей процес зазвичай виконується за допомогою САПР Xilinx WebPack, яка доступна безкоштовно [78]. Процес проектування складається з кількох етапів, включаючи введення проекту, розміщення його на кристалі та верифікацію.

Під час розміщення проекту в кристалі використовується програмний засіб розміщення і трасування. Цей засіб автоматично виконує процес розміщення проекту на кристалі. Він отримує вихідну інформацію про проект у вигляді переліку зв'язків та прив'язує абстрактні логічні елементи до реальних фізичних ресурсів архітектури пристрою. Потім він визначає найкраще місце для розміщення цих елементів, враховуючи інформацію про міжз'єднання та бажану швидкодію.

Останнім етапом є трасування, яке виконує з'єднання між фізичними блоками, щоб забезпечити належну роботу пристрою. Цей етап завершує процес проектування та готує пристрій до наступних етапів тестування та виробництва.

Керуюча оболонка пакета WebPack, а саме Навігатор проекту, надає зручний інтерфейс для роботи з проектом і управління всіма процесами проектування та програмування ПЛІС. Запуск всіх необхідних програмних модулів пакету можна виконати безпосередньо з середовища.

Навігатора проекту. Основне Вікно Навігатора (рисунок 4.14) проекту складається з чотирьох вбудованих вікон: вхідних модулів, вихідних модулів (файлів) проекту, необхідних процедур (процесів) для вибраного вихідного

модуля проекту, консольних повідомлень програмних модулів і редактора текстових HDL-описів проекту.

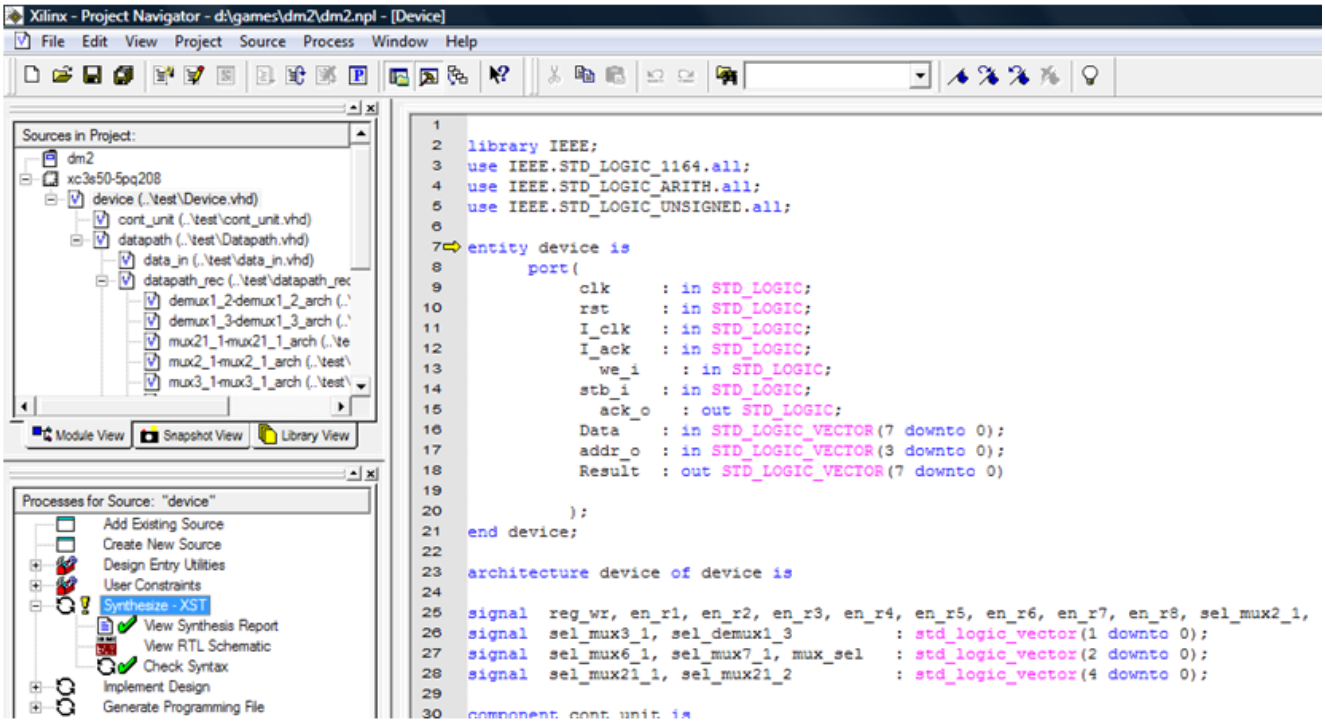


Рисунок 4.14 - Вікно навігатора проекту системи Xilinx

Для створення нового проекту використовується команда File/New Project у головному меню. Після вибору цього пункту меню відкривається діалогове вікно (рисунок 4.15), де користувач вказує назву проекту (у даному випадку проект називається "device"), сімейство ПЛІС, тип кристалу і інструменти для синтезу пристрою. Після введення цих даних у вікні вихідних модулів проекту з'явиться піктограма основного модуля з вказанням типу кристалу і інструментів синтезу.

Сімейство Spartan3 FPGA відзначається впровадженням нової концепції розробки FPGA-платформ, що дозволяє використовувати ПЛІС як основний компонент цифрових пристроїв.

Одна мікросхема серії Spartan3 може містити до 8 мільйонів системних вентилів, що робить їх ідеальним вибором для створення складних цифрових систем. В порівнянні з аналогічними інтегральними схемами, використання Spartan3 значно скорочує час розробки.

Це сімейство підходить для реалізації широкого спектру високопродуктивних систем низького та високого рівня інтеграції, таких як пристрої передачі даних та обробки сигналів. Мікросхеми Spartan3 також знаходять застосування у сфері телекомунікаційних систем, мережевих технологій, бездротового зв'язку та навіть у комп'ютерних системах для космічних апаратів [79].

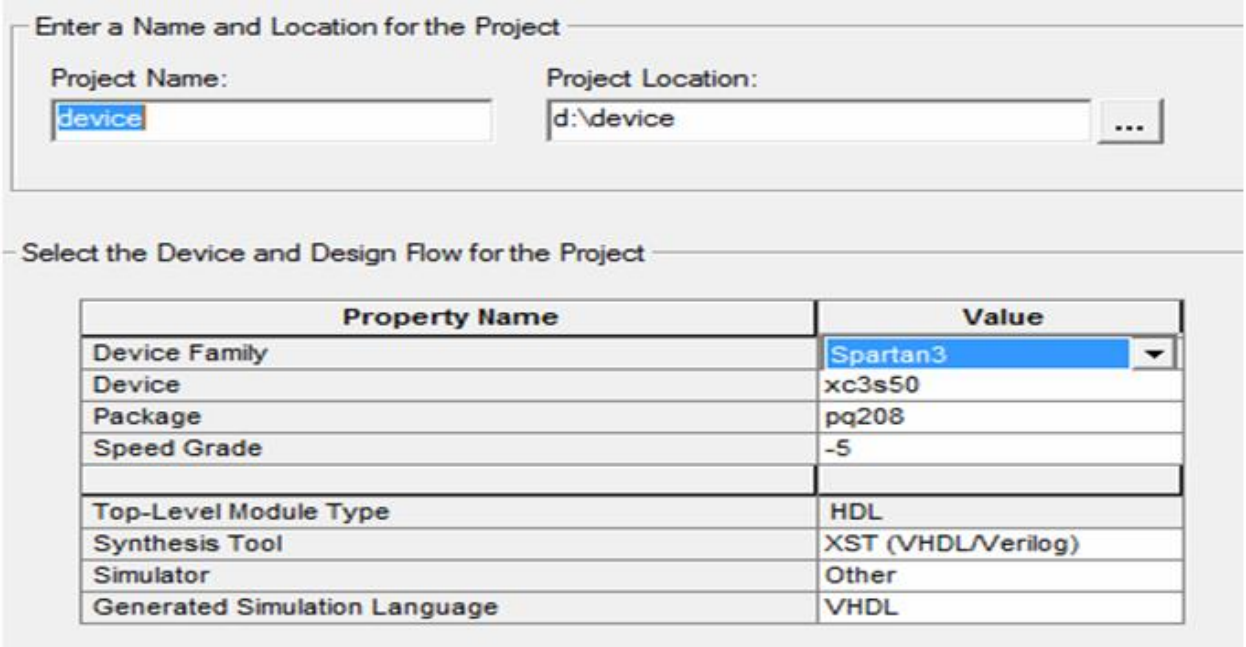


Рисунок 4.15 - Панель створення нового проекту

Створене на основі попереднього досвіду сімейство Spartan3 є кроком вперед у виробництві програмованої логіки, встановлюючи нові стандарти.

Це поєднання різноманітних системних властивостей, ієрархії швидкісних і гнучких трасуючих ресурсів з передовою технологією виготовлення на базі кремнію, дозволяє реалізувати широкий спектр швидкодіючих та великої логічної ємності цифрових пристроїв при значному скороченні часу розробки [80].

Після вибору кристалу можна завантажити модулі асоціативної пам’яті, розроблені на мові VHDL, використовуючи команду Project/Add Source. У діалоговому вікні обираються необхідні файли, після чого в вікні вихідних модулів проекту з'являться вибрані модулі з встановленою ієрархією.

Для початку синтезу відбувається запуск процесу Synthesize у вікні

необхідних процедур. Цей процес включає аналіз ієрархії та перевірку синтаксису. Звіт про виконання синтезу містить результати попередніх обчислень часових параметрів проекту.

Результати синтезу розробленого проекту включають опис базової структури НВІС пристрою сортування, яка складається з інформаційного тракту та пристрою керування. На рисунку 4.16 показано функціональну схему інформаційного тракту запропонованої НВІС структури.

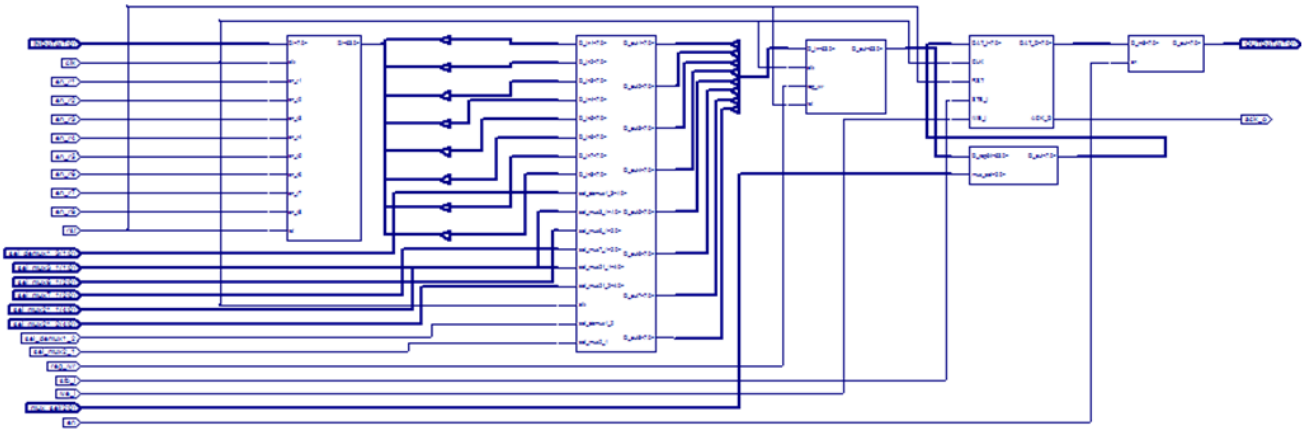


Рисунок 4.16 - Функційна схема інформаційного тракту проектованої структури НВІС

Функційна схема базової структури НВІС пристрою сортування була створена на основі vhdl-коду за допомогою САПР Xilinx WebPack. Ця схема була імплементована на програмованій логічній інтегральній схемі (ПЛІС) від Xilinx. Для цього використовувався кристал XC3S50 з сімейства Spartan3.

Варто розглянути деякі ключові показники, що були отримані після імплементації цієї структури [80]:

– швидкодія (Clock Frequency): Результати показують, що максимальна частота тактового сигналу, яку може підтримувати ця структура, складає 80.567 МГц. Це означає, що період тактового сигналу становить 12.412 наносекунди;

– затрати обладнання (Hardware Utilization);

– кількість Slices: вказано, що було використано 397 з 768 доступних Slice ресурсів на цій ПЛІС. Slice - це основний логічний блок у ПЛІС, який містить

логічні елементи, регістри та інші ресурси;

− відсоток використання: У відсотковому вираженні це означає, що приблизно 51% доступного обладнання на ПЛІС було використано для цієї базової структури НВІС.

Головними результатами дипломного проекту є визначення швидкодії та витрат обладнання ключових модулів розробленої НВІС. У таблиці 4.1 наведено результати швидкодії та використання ресурсів на програмованих логічних інтегральних схемах (ПЛІС) від Xilinx (сімейство Spartan3).

Розглянутих конвеєрний і рекурсивний пристрої сортування 8-ми чисел методом "бульбашки", які мають восьмирозрядний формат. Крім того, у таблиці наведені результати для завершеної структури НВІС, що включає в себе ці основні вузли та єфективно поєднує їх функціонал для досягнення високої продуктивності.

Таблиця 4.1 - Порівняння затрат обладнання та швидкодії на ПЛІС фірми Xilinx

	Конвеєрний пристрій	Рекурсивний пристрій	Завершена структура НВІС
Швидкодія (МГц)	615.764	260.960	80.567
Затрати обладнання (Slices)	609 out of 768 79%	238 out of 768 30%	397 out of 768 51%

Повністю завершена структура НВІС продемонструвала функціонування на частоті 80,567 МГц. Кількість використаних ресурсів на логіку та зв'язки становить 397, що складає 51% від загальної доступної кількості.

Ці дані свідчать про ефективність розробленої НВІС у відношенні до використання ресурсів обладнання та її здатності до роботи на високих частотах, що є важливими критеріями у проектуванні цифрових систем.

4.4 Висновки

У розділі представлена розроблена спеціалізована система сортування на базі програмованих логічних інтегральних схем (ПЛІС), включаючи детальний огляд її архітектури та визначення основних вимог.

Основною метою цієї системи є підвищення швидкодії та ефективності сортування великих обсягів даних за допомогою спеціалізованих алгоритмів, реалізованих на апаратному рівні. Використання програмованих логічних схем дозволяє прискорити процес сортування та зменшити витрати ресурсів порівняно з традиційними алгоритмами, що виконуються на загального призначення процесорах.

Вивчаючи різні алгоритми сортування, такі як швидке сортування, сортування злиттям, сортування бульбашкою та інші, було проведено аналіз їх ефективності та придатності для конкретного завдання. В результаті цього аналізу було визначено, що алгоритм сортування бульбашкою найбільш підходить для даної задачі. Для реалізації сортування було використано як конвеєрний, так і рекурсивний пристрої сортування.

Конвеєрний пристрій сортування використовується для розподілу завдання сортування на послідовні етапи, кожен з яких обробляє окрему частину вхідних даних. Рекурсивний пристрій сортування використовує метод рекурсії для поділу задачі на менші підзадачі та їх послідовне вирішення. Обидва пристрої забезпечують ефективне та швидке виконання сортування, кожен з них має свої переваги та особливості, що дозволяють використовувати їх у різних сценаріях та застосуваннях.

Та можна відзначити, що конвеєрний пристрій сортування виявив високу швидкодію в порівнянні з рекурсивним. Це відбувається завдяки розділенню завдання сортування на послідовні етапи, що можуть виконуватись паралельно, що сприяє прискоренню процесу.

З іншого боку, рекурсивний пристрій сортування, хоча може мати меншу швидкодію, виявився менш вимогливим до ресурсів обладнання. Це означає, що

він може працювати ефективно навіть на обмежених обчислювальних ресурсах, що робить його привабливим для деяких застосувань, де обмежені ресурси грають критичну роль.

Результати дослідження показали, що розроблена система сортування на базі ПЛІС має високу ефективність. З використанням різних алгоритмів сортування, включаючи конвеєрний та рекурсивний підходи, система забезпечує швидкодію та високу ефективність

ВИСНОВКИ

Під час виконання даної кваліфікаційної роботи було проведено ретельне дослідження та розв'язання ряду визначених завдань, що дозволило зробити наступні висновки:

- проведено аналіз відомих методів сортування даних на базі програмованих логічних інтегральних схем (ПЛІС) та виявлено їх переваги та обмеження;
- досліджено існуючі системи сортування даних на базі ПЛІС та проведено їх порівняльний аналіз, виявлено переваги та недоліки кожної з систем;
- проаналізовано використання спеціальних інтегральних схем (ПЛІС) для реалізації спеціалізованих систем сортування даних та визначено їхню ефективність у порівнянні з альтернативними методами;
- досліджено апаратні алгоритми сортування, які будуються на основі графів;
- розроблено модель базової структури НВІС рекурсивного та конвеєрного пристрої сортування 8-ми чисел методом “бульбашки”;
- проведена симуляція роботи основних вузлів базової структури НВІС на функціональному рівні;
- проведено тестування, оцінку ефективності та затрати обладнання розробленої системи сортування даних на базі ПЛІС, що дозволило підтвердити її працездатність та високу швидкодію.

В контексті вирішення завдання розробки спеціалізованої системи сортування даних на базі ПЛІС також був використаний дослідницький підхід. Це дозволило систематично вивчити проблематику сортування даних на цій платформі та визначити шляхи вдосконалення. Аналіз існуючих методів сортування на базі ПЛІС підтвердив їхню ефективність і допоміг визначити оптимальні шляхи їх впровадження в розроблювану систему.

Дослідження методологічних підходів до розробки системи сортування даних дозволило визначити найкращі практики та забезпечити оптимальний вибір методу. Метод сортування був ретельно проаналізований, а оновлена система сортування

була успішно реалізована з урахуванням сучасних технологічних вимог і можливостей платформи ПЛІС.

Було виконано моделювання роботи основних елементів базової структури НВІС на рівні функціональності, а також здійснено синтез та тестування пристрою на ПЛІС від компанії Xilinx. Після цього було проведено порівняння затрат обладнання та швидкодії.

У дипломному проекті основна увага при побудові графу пристрою керування зосереджена на абстрактному автоматі Мура. У додатку А зображено граф-схему алгоритму, який складається з 62-х різних станів.

Було досліджено й розроблено апаратні алгоритми сортування даних на основі графів, такі як рекурсивний та конвеєрні пристрої. Такі алгоритми можуть бути використані у різноманітних областях, включаючи аналіз соціальних мереж, оптимізацію мережевих протоколів та розробку швидкодіючих систем обробки даних. Враховуючи постійний розвиток апаратних технологій, використання апаратних алгоритмів сортування на основі графів може стати ще більш привабливим напрямком досліджень у майбутньому.

За темою кваліфікаційної роботи магістра опублікована теза [1].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Грига В. , Горбач Г. Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, моделювання та управління. *Спеціалізована система сортування двійкових даних.* зб. матеріалів проблемно-наукової міжгалузевої конференції, Надвірна, 2023. с. 146-159.

2. Lyashenko V., Mohammad A., Kobylin O. Experiments with Fusion of Images with Use of Wavelet Transformation. *Problems of the Text Information Analysis*, 2015. vol. 2, no. 1. pp. 16 – 37.

3. Грига В. М. Побудова та аналіз рекурсивних просторово-часових графів / В. М. Грига *Вісник "Комп'ютерні системи та мережі":* тези доп. всеукр. наук.-практ. конф Львів: Національний університет "Львівська політехніка", м.Львів, 2007. № 603. С. 31–35.

4. Steinley, D. Kimeans clustering: a halfacentury synthesis, *British Journal of Mathematical and Statistical Psychology*, 2006. vol.59, no. 1. pp.1-34.

5. Huang Z., Ngan K. A fuzzy k-modes algorithm for clustering categorical data. *Transactions on Fuzzy Systems*, 1999. vol. 7, no. 4. pp. 446-452.

6. Грига В. М. Оцінка варіантів синтезу паралельного алгоритму сортування методом Бетчера. *Матеріали міжнародної проблемно-наукової міжгалузевої конференції "Інформаційні проблеми комп'ютерних систем, юриспруденції, енергетики, економіки, моделювання та управління*": тези доп. всеукр. наук.-практ. конф. Бучач, Бучацький інститут менеджменту і аудиту, 2011. С. 27-31.

7. Adopting Model-Based Design for FPGA, ASIC, and SoC Development. URL: https://la.mathworks.com/videos/adopting-model-based-design-for-fpga-asic-and-soc-development-1559811099219.html (дата звереннння 17.01.2024).

8. Грига В. М. Просторово-часове перетворення паралельних алгоритмів сортування. *Вісник "Комп'ютерні системи та мережі*": тези доп. всеукр. наук.-практ. конф. Львів, 2011. № 717. С. 31–35.

9. Грига В. М. Просторово-часове перетворення алгоритму сортування

чисел методом Бетчера . *Науково-технічний журнал "Радіоелектронні і комп'ютерні системи"*: тези доп. всеукр. наук.-практ. конф. Харків: Національний аерокосмічний університет ім. М.Є. Жуковського "Харківський авіаційний інститут", 2012. №6(58). С. 84–87.

10. Kobayashi R., Miura K., Fujita N., Boku T. and Amagasa, T. A Sorting Library for FPGA Implementation in OpenCL Programming, *Proc. 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, 2022. vol. 10, no. 6. pp. 146-158.

11. Kobayashi R., Miura K., Fujita N., Boku T. An open-source FPGA library for data sorting. *Journal of Information Processing*, 2022. vol. 30, no. 1. pp. 766-777.

12. Saitoh, M., Elsayed E.A., Chu T.V., Mashimo S. and Kis, K. A High-Performance and Cost-Effective Hardware Merge Sorter without Feedback Datapath, *Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM*), 2018. vol 2, no. 1. pp.97–124

13. Saitoh, M. and Kise, K. Very Massive Hardware Merge Sorter, *International Conference on Field-Programmable Technology (FPT),* 2018. vol. 12, pp.86–93

14. Abirami, R. Sudha, and G. Suresh Kumar. Comparative study based on analysis of coronavirus disease (COVID-19) detection and prediction using machine learning models." *SN Computer Science* , 2022. vol. 1, no. 1. pp. 1-8.

15. Pasetto, D. & Akhriev, A. A comparative study of parallel sort algorithms. *ACM International conference companion on Object oriented programming systems languages and applications companion*, 2019. vol. 5, no. 1. pp.203–204.

16. Danelutto K., Marco A. Algorithmic skeletons and parallel design patterns in mainstream parallel programming. *International Journal of Parallel Programming* , 2021. vol. 4, no. 1. pp.177-198.

17. Цмоць І. Удосконалення паралельного сортування масивів чисел методом злиття. *Комп'ютерна інженерія*, 2020. Т. 30. № 4. С. 134 – 142.

18. Video Analytics Hardware, Software, and Services Revenue to Reach Billion by 2022. URL: https://www.embedded-vision.com/industry-analysis/marketanalysis/

video-analytics-hardware-software-and-service-revenue-reach-3-bil. (дата звернення: 25.01.2024).

19. A Sorting Library for FPGA Implementation in OpenCL Programming. URL:https://videos.uni-paderborn.de/category/video/a-sorting-library-for-fpga-implementation-in-opencl-programming-/31b5059ea3881a3f2ff6317af8441cfd/3. (дата звернення: 23.02.2023).

20. Ерметов Ю.О. Проектування обчислювальних структур на основі просторово-часових графів. *Вісник Хмельницького національного університету:* Хмельницький: Хмельницький національний університет, 2006. № 4. С. 172–177.

21. Programmable embedded platforms for remote and compute intensive image processing applications. URL: http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/K009583/1 (дата звернення: 02.04.2024).

22. Noris Freire, L. Nunes De Castro, B. Mauricio, and N. Freire, FPGA recommender systems: a systematic review, *Knowl. Inf. Syst*., 2021. vol. 63, pp. 1–20.

23. Грига В. М. Методи побудови рекурсивних пристроїв сортування на основі просторово-часових графів. *Науково-технічний журнал "Радіоелектронні і комп'ютерні системи* ": тези доп. всеукр. наук.-практ. Конф. Харків, 2009. №6. С. 209–212.

24. Lengnick-Hall and S. Moritz, The impact of on the FPGA function, *J. Labor Res*.,2020. vol. 24, no. 3. pp. 365–379.

25. Emma Parry, Handbook of Strategic e-Business. *Berlin, Heidelberg: Springer Berlin Heidelberg*. 2019. vol. 28, no. 1. pp. 98– 131.

26. Allden F. and L. Harris, Building a positive experience: Towards networked model, *J. Bus. Strategy*, 2019. vol. 34, no. 5. pp. 36–47.

27. Collins J, and Han J., Exploring applicant pool quantity and quality: the effects of early sort practice strategies, *Corporate advertising, and firm reputation*, 2021. vol. 2, no. 5. pp. 56-64.

28. Bryman and E. Bell, Business research methods. *Oxford University Press*, USA, 2017. vol. 1, no. 3, pp. 1-22.

29. Грига В. М. Побудова та аналіз рекурсивних просторово-часових графів. *Вісник "Комп'ютерні системи та мережі"*. Львів: Національний університет "Львівська політехніка", 2007. № 603. С. 31–35.

30. Kumar, P. A journal of management research, and undefined, *Impact of online practice on recruitment performance*. , 2021. vol. 67, no. 5. pp. 722-735.

31. Sobocka-Szczapa, Recruitment of employees—assumptions of the risk model, *Risks*, 2021. vol. 9, no. 3. pp. 200-236.

32. Bartram, Internet Recruitment and Selection: Kissing Frogs to find Princes, *Int. J. Sel. Assess*., vol. 8, no. 4, pp. 261–274, 2020.

33. Ramzan, International journal of social and management studies (IJOSMAS) Impact of Web-Based Recruitment: *A Study among HR Professionals*, 2019. vol. 02, no. 04, pp. 123– 140.

34. Marcelino R., Cardoso, J. Unbalanced FIFO sorting for FPGA-based systems, *International Conference on Electronics*, 2009. vol. 12, no. 01. pp. 431-434.

35. Dokey and M. Abunar, Effectiveness of FPGA in Attracting Talented 144 Employees : a Study on Sorting, *Palarch's J. Archaeol. Egypt/Egyptology*, 2021.vol. 18, no. 12. pp. 144–154.

36. S´erot J., Berry F., Bourrasset C. High-level dataflow programming for real-time image processing on smart cameras. *Journal of Real-Te Image Processing (JRTIP)*, 2019. vol. 12, no. 4. pp. 635-647.

37. Melanthiou, F. Pavlou, and E. Constantinou, The Use of Sorting Network Sites, *J. Transnatl. Manag*., 2021. vol. 20, no. 1. pp. 31–49.

38. Swider, R. D. Zimmerman, V. Tech, and M. R. Barrick, Searching for the Right Fit: Development of Applicant. *Person-Organization Fit Perceptions During the Recruitment Process*, 2017. vol. 36, no. 7. pp. 114-115.

39. Smythe, A. Grotlüschen, and K. Buddeberg, The automated literacies of erecruitment and online services, *Stud. Educ. Adults*, 2019. vol. 53, no. 1. pp. 4–22.

40. Parmenter A. Attracting and Sorting data, *Financ. Controll. CFO's Toolkit*, 2019. vol. 17, no. 2. pp. 91-124.

41. Chapman and J. Webster, The use of technologies screening, and selection

processes for job candidates, International Journal of Selection and Assessment Blackwell Publishing Ltd, 2018. vol. 28, no. 11. pp. 209-221.

42. Kinnunen M. Feeling the Right Personality. Consultants' *AffectiveDecision Making in Interviews With Employee Candidates Jaana Parviainen,* 2019. vol. 6, no. 3. pp. 182-234.

43. Faliagka K., Ramtas M., Rigou S., Sirmakessis M. Towards an Influence Indication Score, *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Bioinformatics),* 2020.vol. 14, pp. 252261.

44. Malki E., Atlam Z., Malki A., Atlam N., Graduate Students and Companies Web Based E-Recruitment Sys-tem, *J. Comput. Commun*., 2021.vol. 9, pp. 71–84.

45. Akila, S. Vasantha, and P. G. Thirumagal, Effectiveness of e-recruitment for man power selection process, *J. Crit. Rev*., 2022. vol. 7, no. 5. pp. 306–310.

46. Алгоритми та паралельні структури сортування даних методом вставки *Науковий вісник НЛТУ України: Збірник науково-технічних праць.* Львів, 2016. №26.(1). С. 340–350.

47. Sultana and N. Sultana, Analyzing the Effectiveness of FPGA: A Case Study of Bangladesh, *Asian Bus*. Rev., 2019. vol. 7, no. 2. pp. 79– 84.

48. Sayel Sabha, Impact of Online recruitment on reruitment, *Int. J. Educ. Res*., 2022. vol. 6, no. 4. pp. 47–52.

49. Student Opinions of the Efficacy of Select Moethods of External sortings. *Gale Academic OneFile.* (дата звернення: 19.02.2024).

50. Грига В. М. Просторово-часове перетворення потокових графів алгоритму. *Науковий журнал "Технічні науки":* тези доп. всеукр. наук.-практ. конф. Хмельницький, 2010. №4. С.113–117.

51. Michael Armstrong, No TitleArmstrong's Handbook of Human *Resource Practice. Kogan Page, ,* 2021. vol. 6, no. 1. pp. 81-107.

52. Blommaert, M. Coenders, and F. van Tubergen, Discrimination of Arabic-named applicants in the Netherlands: An internet-based field experiment examining different phases in online sorting , *Soc. Force*s, 2022. vol. 92, no. 3. pp. 957–982.

53. Geetha and D. Bhanu Sree Reddy, sorting through artificial intelligence: A

conceptual study, *Int. J. Mech. Eng. Technol*., 2019. vol. 9, no. 7. pp. 64–71.

54. Kaylor D, IT Technology & A with Jonathan Kestenbaum of Talent Tech Lab, *IT Tech Insid*., 2019. vol. 1000, no. 1. pp. 1-7.

55. Rochelle Dilip and R. Rohini, A Study on Employees Perception Towards Artificial Intelligence in Competency Mapping and at Wildcraft Pvt. Ltd., *Int. J. Res. Eng. Sci. Manag*., 2021.vol. 3, no. 2. pp. 491–496.

56. R. Oksanen, New technology-based sorting methods, *Masterarbeit*, no. May, 2020. vol. 30, no. 5. pp. 35-40.

57. Danelutto, M. De Matteis, T., Mencagli, & M. Torquati, A divide-and-conquer parallel pattern implementation for multicores. *ACM International Workshop on Software Engineering for Parallel Systems*., 2016. pp. 10-19.

58. Cong, J., Liu B., Neuendorffer S., Noguera J., Vissers K., & Zhang Z. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2011. pp. 473-491.

59. Mueller, Rene, Jens Teubner, and Gustavo Alonso. "Data processing on FPGAs." *Proceedings of the VLDB Endowment 2*.1, 2009. vol. 7, no. 2. pp. 79– 84.

60. M. Rene, T. Jens, A. Gustavo, Data processing on FPGAs. *Proceedings of the VLDB Endowment.*, 2009. vol. 2, no.1. pp.910-921.

61. M. Rene, T. Jens, A. Gustavo, Sorting networks on FPGAs. *The VLDB Journal*, 2012. vol. 21, no. 1. pp. 103-123.

62. Xu J., Han J., Xiong K., Nie F. Robust and Sparse Fuzzy K-Means Clustering. *In IJCAI* 2016, vol. 23, no. 1. pp. 224-230.

63. Little R. J., Rubin D. B. Statistical analysis with missing data. *Information Technology and Management Science*. 2019. vol. 793, no. 1. pp. 890-960.

64. Jain K., Murty N., Flynn J. Data clustering: a review. *ACM computing surveys (CSUR)*, 1999. vol. 31, no. 3. pp.264-323.

65. Perret B., Chierchia G., Cousty J., Guimarães S. J. F., Kenmochi Y. Higra Hierarchical graph analysis. *SoftwareX*, 2019. vol. 73, no. 1. pp. 219-232.

66. Khachumov M., Distances metrics and cluster analysis. *Scientific and Technical Information Processing*, 2012, vol. 39, no. 6. pp. 310-316.

67. Штовба, С. Побудова функцій належності нечітких множин за кластеризацією експериментальних даних. *Інформаційні технології та комп'ютерна інженерія*, 2006. № 4. С. 92 – 95.

68. Bodyanskiy Y., Tyshchenko K., Mashtalir, S. Fuzzy Clustering High-Dimensional Data Using Information Weighting. *In International Conference on Artificial Intelligence and Soft Computing*, 2019. vol. 13, no. 5. pp. 385- 395.

69. Mashtalir S., Mashtalir V., Stolbovyi M. Video shot boundary detection via sequential clustering. *International Journal "Information Theories and Applications*, 2017. vol. 24, no. 1. pp. 50-59.

70. Mashtalir, S., Mashtalir, V., Stolbovyi, M. Representative Based Clustering of Long Multivariate Sequences with Different Lengths. *In 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*, 2018. vol. 39, no. 6. pp. 445-548.

71. Bodyanskiy, Y., Kobylin, I., Rashkevych, Y., Vynokurova, O., Peleshko, D. Hybrid fuzzy-clustering algorithm of unevenly and asynchronously spaced time series in computer engineering. *International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineerin*, 2018. vol. 14, no. 5. pp. 930-935.

72. Akhter N. Sorting Algorithms. A Comparative Study. *International Journal of Computer Science and Information Security*, 2016. vol. 14, no.12, pp. 930-936.

73. Agrawal N. Analysis and Review of Sorting Algorithms. *International Journal of Scientific Engineering and Research*, 2019. vol. 2, no. 3. pp. 81-84.

74. Xiang W. Analysis of the time complexity of quick sort algorithm. *International Conference on Information Management, Innovation Management and Industrial Engineering*, 2011. vol. 34, no. 9. pp. 400-410.

75. Lobo J. and Kuwelkar S. Performance analysis of merge sort algorithms. *Proceedings of the International Conference on Electronics and Sustainable Communication Systems*, 2020. vol. 4, no. 3. pp. 110-115.

76. Aliyu A., Zirra B. A Comparative Analysis of Sorting Algorithms on Integer and Character Array. *The International Journal Of Engineering And Science*

(IJES), 2013. vol. 2, no. 7. pp. 25-30.

77. Chauhan Y., Duggal A. Different Sorting Algorithms. *Comparison based upon the Time Complexity*, 2020. vol. 7, no. 3. pp.89-99.

78. Kobayashi R., Kise K. Fast and customizable sorting accelerator for heterogeneous many-core systems. *Embedded Multicore/Many-core Systems-on-Chip*, 2015. vol. 2, no. 1. pp. 49–56.

79. Usui T., Van Chu, Kise K. A cost-effective and scalable merge sorter tree on FPGAs. *Fourth Int'l Symp. on Computing and Networking (CANDAR)*, 2016. vol. 1, no. 1. pp. 47–56.

80. Farmahini-Farahani A., Duwe H., Schulte M., Compton K. Modular design of high-throughput, low-latency sorting units. *Transactions on Computers*, 2012. vol. 62, no. 7. pp. 1389–1402.

81. Saitoh M. and Kise K., Very massive hardware merge sorter. *Int'l Conference on Field-Programmable Technology (FPT)*, 2018. vol. 4, no. 2. pp. 86–93.

ДОДАТОК А

ГРАФ СХЕМА АЛГОРИТМУ СОРТУВАННЯ

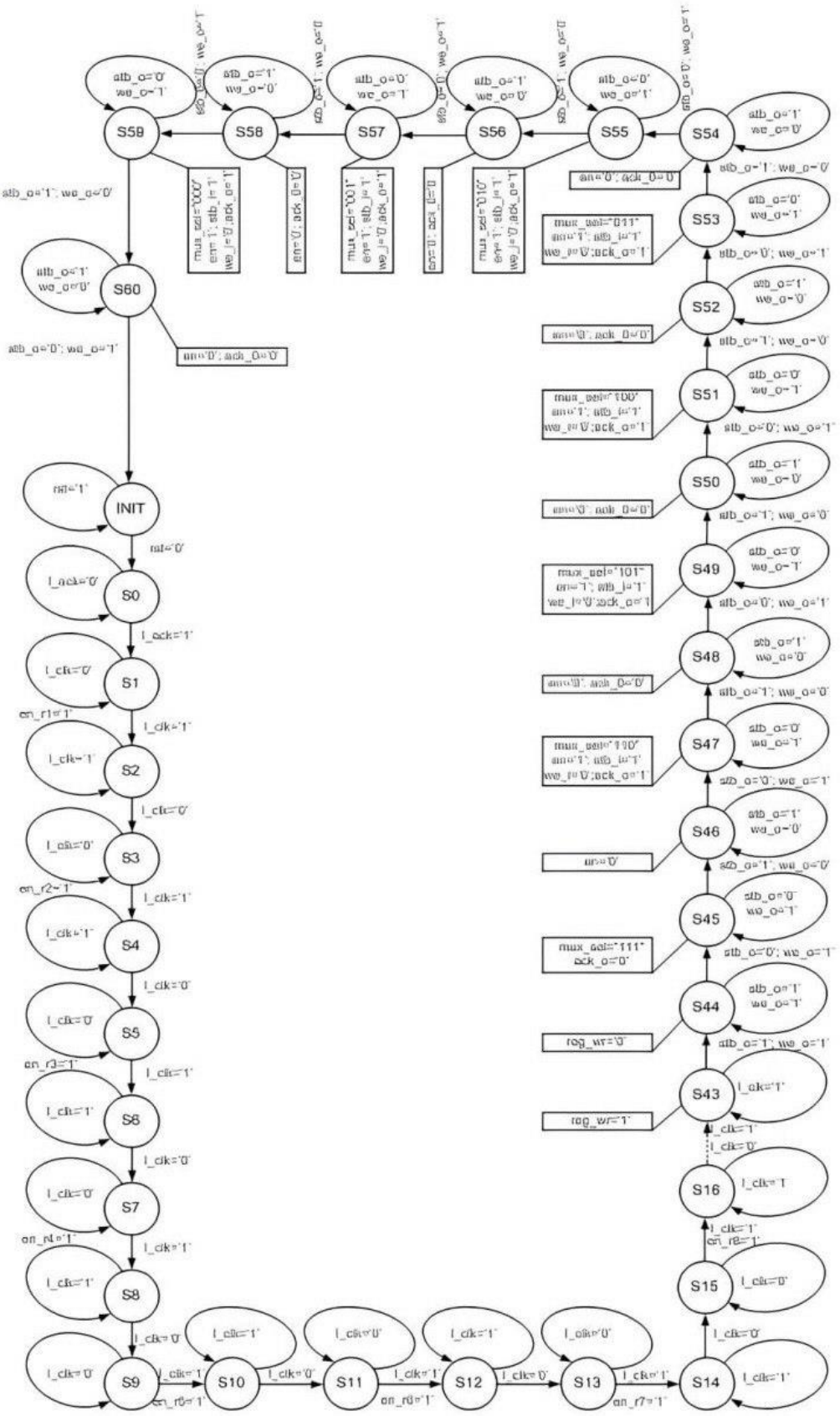


Рисунок А.1 - Граф-схема алгоритму сортування

ДОДАТОК Б

VHDL-КОД КОНВЕЄРНОГО ПРИСТРОЮ СОРТУВАННЯ

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity conv_sort is
            port(
    clk          : in STD_LOGIC;
    rst          : in STD_LOGIC;
D_in1        : in STD_LOGIC_VECTOR(7 downto 0);     D_in2         : in STD_LOGIC_VECTOR(7 downto 0);
D_in3        : in STD_LOGIC_VECTOR(7 downto 0);
D_in4        : in STD_LOGIC_VECTOR(7 downto 0);
D_in5        : in STD_LOGIC_VECTOR(7 downto 0) D_in6         : in STD_LOGIC_VECTOR(7 downto 0);
D_in7        : in STD_LOGIC_VECTOR(7 downto 0);
D_in8        : in STD_LOGIC_VECTOR(7 downto 0);
D_out1       : out STD_LOGIC_VECTOR(7 downto 0);
D_out2       : out STD_LOGIC_VECTOR(7 downto 0);
D_out3       : out STD_LOGIC_VECTOR(7 downto 0);
D_out4       : out STD_LOGIC_VECTOR(7 downto 0);
D_out5       : out STD_LOGIC_VECTOR(7 downto 0);
 D_out6      : out STD_LOGIC_VECTOR(7 downto 0);
 D_out7      : out STD_LOGIC_VECTOR(7 downto 0);
D_out8       : out STD_LOGIC_VECTOR(7 downto 0)
                    );
end conv_sort;

architecture conv_sort of conv_sort is

signal
s12_in,s13_in,s131_in,s23_in,s24_in,s35_in,s37_in,s371_in,s45_in,s46_in,s57_in,s58_in,s68_in,s69_in,s710_in,s713_in,s7
131_in,s810_in,s811_in,s911_in,s912_in,s1013_in,s1014_in,s1114_in,s1115_in,s1215_in,s1216_in,s1317_in,s1320_in,s13
201_in,s1417_in,s1418_in,s1518_in,s1519_in,s1619_in,s1720_in,s1721_in,s1821_in,s1822_in,s1922_in,s2023_in,s2025_i
n,s20251_in,s2123_in,s2124_in,s2224_in,s2325_in,s2326_in,s2426_in,s2527_in,s2528_in,s25281_in,s2627_in,s2728_in:
STD_LOGIC_VECTOR(7 downto 0);
signal
s12_out,s13_out,s131_out,s23_out,s24_out,s35_out,s37_out,s371_out,s45_out,s46_out,s57_out,s58_out,s68_out,s69_out,s
710_out,s713_out,s7131_out,s810_out,s811_out,s911_out,s912_out,s1013_out,s1014_out,s1114_out,s1115_out,s1215_out
,s1216_out,s1317_out,s1320_out,s13201_out,s1417_out,s1418_out,s1518_out,s1519_out,s1619_out,s1720_out,s1721_out,
s1821_out,s1822_out,s1922_out,s2023_out,s2025_out,s20251_out,s2123_out,s2124_out,s2224_out,s2325_out,s2326_out,s
2426_out,s2527_out,s2528_out,s25281_out,s2627_out,s2728_out: STD_LOGIC_VECTOR(7 downto 0);
signal x31_in, x31_out : STD_LOGIC_VECTOR(7 downto 0);
signal x41_in, x41_out, x42_in, x42_out : STD_LOGIC_VECTOR(7 downto 0);
signal x51_in, x51_out, x52_in, x52_out, x53_in, x53_out : STD_LOGIC_VECTOR(7 downto 0);
signal x61_in, x61_out, x62_in, x62_out, x63_in, x63_out, x64_in, x64_out : STD_LOGIC_VECTOR(7 downto 0);
signal x71_in, x71_out, x72_in, x72_out, x73_in, x73_out, x74_in, x74_out, x75_in, x75_out : STD_LOGIC_VECTOR(7
downto 0);
signal x81_in, x81_out, x82_in, x82_out, x83_in, x83_out, x84_in, x84_out, x85_in, x85_out, x86_in, x86_out :
STD_LOGIC_VECTOR(7 downto 0);
signal y11_in, y11_out, y12_in, y12_out, y13_in, y13_out, y14_in, y14_out, y15_in, y15_out, y16_in, y16_out, y17_in,
y17_out : STD_LOGIC_VECTOR(7 downto 0);
signal y21_in, y21_out, y22_in, y22_out, y23_in, y23_out, y24_in, y24_out, y25_in, y25_out, y26_in, y26_out :
STD_LOGIC_VECTOR(7 downto 0);
signal y31_in, y31_out, y32_in, y32_out, y33_in, y33_out, y34_in, y34_out, y35_in, y35_out : STD_LOGIC_VECTOR(7
downto 0);
signal y41_in, y41_out, y42_in, y42_out, y43_in, y43_out,y44_in, y44_out : STD_LOGIC_VECTOR(7 downto 0);
signal y51_in, y51_out, y52_in, y52_out, y53_in, y53_out : STD_LOGIC_VECTOR(7 downto 0);
signal y61_in, y61_out, y62_in, y62_out : STD_LOGIC_VECTOR(7 downto 0);
signal y71_in, y71_out : STD_LOGIC_VECTOR(7 downto 0);
signal y81_in, y81_out : STD_LOGIC_VECTOR(7 downto 0);
```

```
component operation_block is
            port(
A_in : in STD_LOGIC_VECTOR(7 downto 0);
B_in : in STD_LOGIC_VECTOR(7 downto 0);
Greater : out STD_LOGIC_VECTOR(7 downto 0);
Less : out STD_LOGIC_VECTOR(7 downto 0)                    );
end component operation_block;

begin

D1: operation_block
   port map(
A_in    => D_in1,
 B_in    => D_in2,
Greater => s12_in,
Less    => s13_in);
D2: operation_block
   port map(
A_in    => s12_out,
B_in    => x31_out,
Greater => s24_in,
Less    => s23_in);
D3: operation_block
   port map(
A_in    => s131_out,
  B_in    => s23_out,
Greater => s35_in,
Less    => s37_in);
D4: operation_block
   port map(
A_in    => s24_out,
B_in    => x42_out,
Greater => s46_in,
Less    => s45_in);
D5: operation_block
   port map(
A_in    => s35_out,
B_in    => s45_out,
Greater => s58_in,
Less    => s57_in);
D6: operation_block
   port map(
A_in    => s46_out,
B_in    => x53_out,
Greater => s69_in,
Less    => s68_in);
D7: operation_block
   port map(
A_in    => s371_out,
B_in    => s57_out,
Greater => s710_in,
Less    => s713_in);
D8: operation_block
   port map(
A_in    => s58_out,
B_in    => s68_out,
Greater => s811_in,
Less    => s810_in);
D9: operation_block
   port map(
A_in    => s69_out,
B_in    => x64_out,
```

```
Greater => s912_in,
Less    => s911_in);
D10: operation_block
   port map(
A_in    => s710_out,
B_in    => s810_out,
Greater => s1014_in,
Less    => s1013_in);
D11: operation_block
   port map(
A_in    => s811_out,
B_in    => s911_out,
Greater => s1115_in,
Less    => s1114_in);
D12: operation_block
   port map(
A_in    => s912_out,
   B_in    => x75_out,
Greater => s1216_in,
Less    => s1215_in);
D13: operation_block
   port map(
A_in    => s7131_out,
B_in    => s1013_out,
Greater => s1317_in,
Less    => s1320_in);
D14: operation_block
   port map(
A_in    => s1014_out,
 B_in    => s1114_out,
Greater => s1418_in,
Less    => s1417_in);
D15: operation_block
   port map(
A_in    => s1115_out,
B_in    => s1215_out,
Greater => s1519_in,
Less    => s1518_in);
D16: operation_block
   port map(
A_in    => s1216_out,
 B_in    => x86_out,
Greater => y11_in,
Less    => s1619_in);
D17: operation_block
   port map(
A_in    => s1317_out,
 B_in    => s1417_out,
Greater => s1721_in,
Less    => s1720_in);
D18: operation_block
   port map(
A_in    => s1418_out,
B_in    => s1518_out,
Greater => s1822_in,
Less    => s1821_in);
D19: operation_block
   port map(
A_in    => s1519_out,
B_in    => s1619_out,
Greater => y21_in,
Less    => s1922_in);
D20: operation_block
```

```
   port map(
A_in    => s13201_out,
B_in    => s1720_out,
Greater => s2023_in,
Less    => s2025_in);
D21: operation_block
   port map(
A_in    => s1721_out,
 B_in    => s1821_out,
Greater => s2124_in,
Less    => s2123_in);
D22: operation_block
   port map(
A_in    => s1822_out,
B_in    => s1922_out,
Greater => y31_in,
Less    => s2224_in);
D23: operation_block
   port map(
A_in    => s2023_out,
B_in    => s2123_out,
Greater => s2326_in,
Less    => s2325_in);
D24: operation_block
   port map(
A_in    => s2124_out,
B_in    => s2224_out,
Greater => y41_in,
Less    => s2426_in);
D25: operation_block
   port map(
A_in    => s2025_out,
B_in    => s2325_out,
Greater => s2527_in,
Less    => s2528_in);
D26: operation_block
   port map(
A_in    => s2326_out,
B_in    => s2426_out,
Greater => y51_in,
Less    => s2627_in);
D27: operation_block
   port map(
A_in    => s2527_out,
B_in    => s2627_out,
Greater => y61_in,
Less    => s2728_in);
D28: operation_block
   port map(
A_in    => s25281_out,
B_in    => s2728_out,
Greater => y71_in,
Less    => y81_in);
----reg file 1-----------------------------------------------------
process (clk, rst)
begin
   if rst='1' then
                x31_out <= (others => '0');
                x41_out <= (others => '0');
                x51_out <= (others => '0');
                x61_out <= (others => '0');
                x71_out <= (others => '0');
                x81_out <= (others => '0');
```

```
    elsif (clk'event and clk='1') then
                x31_out <= x31_in;
                x41_out <= x41_in;
                x51_out <= x51_in;
                x61_out <= x61_in;
                x71_out <= x71_in;
                x81_out <= x81_in;
      end if;
end process;
x31_in <= D_in3;
x41_in <= D_in4;
x51_in <= D_in5;
x61_in <= D_in6;
x71_in <= D_in7;
x81_in <= D_in8;
----reg file 2-----------------------------------------------------
process (clk, rst)
begin
    if rst='1' then
                s12_out <= (others => '0');
               s13_out  <= (others => '0');
                x42_out <= (others => '0');
                x52_out <= (others => '0');
                x62_out <= (others => '0');
                x72_out <= (others => '0');
                x82_out <= (others => '0');
    elsif (clk'event and clk='1') then
                s12_out <= s12_in;
                s13_out <= s13_in;
                x42_out <= x41_out;
                x52_out <= x51_out;
                x62_out <= x61_out;
                x72_out <= x71_out;
                x82_out <= x81_out;
    end if;
end process;
----reg file 3-----------------------------------------------------
process (clk, rst)
begin
    if rst='1' then
                s23_out  <= (others => '0');
                s24_out   <= (others => '0');
                s131_out <= (others => '0');
                x53_out  <= (others => '0');
                x63_out  <= (others => '0');
                x73_out  <= (others => '0');
                x83_out  <= (others => '0');
    elsif (clk'event and clk='1') then
                s23_out  <= s23_in;
                s24_out  <= s24_in;
                s131_out <= s13_in;
                x53_out  <= x52_out;
                x63_out  <= x62_out;
                x73_out  <= x72_out;
                x83_out  <= x82_out;
    end if;
end process;
----reg file 4---------------------------------
process (clk, rst)
begin
    if rst='1' then
                s37_out  <= (others => '0');
                s35_out  <= (others => '0');
```

```
                s45_out  <= (others => '0');
                s46_out  <= (others => '0');
                x64_out  <= (others => '0');
                x74_out  <= (others => '0');
                x84_out  <= (others => '0');
    elsif (clk'event and clk='1') then
                s37_out <= s37_in;
                s35_out  <= s35_in;
                s45_out  <= s45_in;
                s46_out  <= s46_in;
                x64_out  <= x63_out;
                x74_out  <= x73_out;
                x84_out  <= x83_out;
    end if;
end process;
----reg file 5-------------------------------------------------
process (clk, rst)
begin
    if rst='1' then
                 s371_out <= (others => '0');
                s57_out   <= (others => '0');
                s58_out  <= (others => '0');
                s68_out   <= (others => '0');
                s69_out   <= (others => '0');
                x75_out   <= (others => '0');
                x85_out   <= (others => '0');
    elsif (clk'event and clk='1') then
                s371_out <= s37_in;
                s57_out   <= s57_in;
                s58_out   <= s58_in;
                s68_out   <= s68_in;
                s69_out   <= s69_in;
                x75_out  <= x74_out;
                x85_out  <= x84_out;
    end if;
end process;
-----reg file 6-----------------------------------------------
process (clk, rst)
begin
    if rst='1' then
                s713_out <= (others => '0');
                s710_out <= (others => '0');
                s810_out <= (others => '0');
                s811_out <= (others => '0');
                s911_out <= (others => '0');
                s912_out <= (others => '0');
                x86_out  <= (others => '0');
    elsif (clk'event and clk='1') then
                s713_out <= s713_in;
                s710_out <= s710_in;
                s810_out <= s810_in;
                s811_out <= s811_in;
                s911_out <= s911_in;
                s912_out <= s912_in;
                x86_out  <= x85_out;
    end if;
end process;
----reg file 7-------------------------------------------------------
process (clk, rst)
begin
    if rst='1' then
          s7131_out <= (others => '0');
          s1013_out <= (others => '0');
```

```
            s1014_out <= (others => '0');
            s1114_out <= (others => '0');
            s1115_out <= (others => '0');
            s1215_out <= (others => '0');
            s1216_out  <= (others => '0');
            y11_out     <= (others => '0');
    elsif (clk'event and clk='1') then
            s7131_out <= s713_in;
            s1013_out <= s1013_in;
            s1014_out <= s1014_in;
            s1114_out <= s1114_in;
            s1115_out <= s1115_in;
            s1215_out <= s1215_in;
             s1216_out <= s1216_in;
            y11_out   <= y11_in;
    end if;
end process;
----reg file 8-----------------------------------------------------
process (clk, rst)
begin
    if rst='1' then
            s1320_out <= (others => '0');
            s1317_out <= (others => '0');
            s1417_out <= (others => '0');
            s1418_out <= (others => '0');
            s1518_out <= (others=> '0');
            s1519_out <= ( others => '0');
            s1619_out <= (others => '0');
            y12_out    <= (others => '0');
            y21_out    <= (others > '0');
    elsif (clk'event and clk='1') then
            s1320_out <= s1320_in;
            s1317_out <= s1317_in;
            s1417_out <= s1417_in;
            s1418_out <= s1418_in;
            s1518_out <= s1518_in;
            s1519_out <= s1519_in;
            s1619_out <= s1619_in;
            y12_out   <= y11_out;
            y21_out   <= y21_in;
    end if;
end process;
-----reg file 9-----------------------------------------------------
 process (clk, rst)
begin
    if rst='1' then
            s13201_out <= (others => '0');
            s1720_out <= (others => '0');
            s1721_out <= (others => '0');
            s1821_out <= (others => '0');
            s1822_out <= (others => '0');
            s1922_out <= (others => '0');
            y13_out    <= (others => '0');
            y22_out    <= (others => '0');
            y31_out    <= (others => '0');
    elsif (clk'event and clk='1') then
            s13201_out <= s1320_in;
            s1720_out <= s1720_in;
            s1721_out <= s1721_in;
            s1821_out <= s1821_in;
            s1822_out <= s1822_in;
            s1922_out <= s1922_in;
            y13_out   <= y12_out;
```

```
            y22_out   <= y21_out;
            y31_out   <= y31_in;
    end if;
end process;
----reg file 10-------------------------------------------
process (clk, rst)
begin
    if rst='1' then
                        s2025_out <= (others => '0');
                        s2023_out <= (others => '0');
                         s2123_out <= (others => '0');
                         s2124_out <= (others => '0');
                        s2224_out <= (others => '0');
                        y14_out   <= (others => '0');
                        y23_out   <= (others => '0');
                        y32_out   <= (others => '0');
                        y41_out   <= (others => '0');
    elsif (clk'event and clk='1') then
                         s2025_out <= s2025_in;
                          s2023_out <= s2023_in;
                         s2123_out <= s2123_in;
                         s2124_out <= s2124_in;
                        s2224_out <= s2224_in;
                         y14_out   <= y13_out;
                         y23_out   <= y22_out;
                         y32_out   <= y31_out;
                        y41_out   <= y41_in;
    end if;
end process;
-----reg file 11-----------------------------------------------------
process (clk, rst)
begin
    if rst='1' then
                        s20251_out <= (others => '0');
                        s2325_out <= (others => '0');
                        s2326_out <= (others => '0');
                        s2426_out <= (others => '0');
                        y15_out   <= (others => '0');
                        y24_out   <= (others => '0');
                        y33_out   <= (others => '0');
                        y42_out   <= (others => '0');
                        y51_out   <= (others => '0');
    elsif (clk'event and clk='1') then
                         s20251_out <= s2025_in;
                         s2325_out <= s2325_in;
                        s2326_out <= s2326_in;
                        s2426_out <= s2426_in;
                        y15_out   <= y14_out;
                        y24_out   <= y23_out;
                        y33_out   <= y32_out;
                        y42_out   <= y41_out;
                        y51_out   <= y51_in;
    end if;
end process;
---reg file 12-------------------------------------------------------
process (clk, rst)
begin
    if rst='1' then
                        s2528_out <= (others => '0');
                        s2527_out <= (others => '0');
                        s2627_out <= (others => '0');
                        y16_out   <= (others => '0');
                        y25_out   <= (others => '0');
```

```
                    y34_out   <= (others => '0');
                    y43_out   <= (others => '0');
                    y52_out   <= (others => '0');
                    y61_out   <= (others => '0');
    elsif (clk'event and clk='1') then
                    s2528_out <= s2528_in;
                    s2527_out <= s2527_in;
                    s2627_out <= s2627_in;
                    y16_out   <= y15_out;
                    y25_out   <= y24_out;
                    y34_out   <= y33_out;
                    y43_out   <= y42_out;
                    y52_out   <= y51_out;
                    y61_out   <= y61_in;
    end if;
end process;
----reg file 13-----------------------------------------
process (clk, rst)
begin
    if rst='1' then
                    s25281_out <= (others => '0');
                    s2728_out <= (others => '0');
                     y17_out   <= (others => '0');
                    y26_out   <= (others => '0');
                    y35_out   <= (others => '0');
                    y44_out   <= (others => '0');
                    y53_out   <= (others => '0');
                    y62_out   <= (others => '0');
                    y71_out   <= (others => '0');
                    y81_out   <= (others => '0');
    elsif (clk'event and clk='1') then
                    s25281_out <= s2528_in;
                    s2728_out <= s2728_in;
                    y17_out   <= y16_out;
                    y26_out   <= y25_out;
                    y35_out   <= y34_out;
                    y44_out   <= y43_out;
                    y53_out   <= y52_out;
                    y62_out   <= y61_out;
                    y71_out   <= y71_in;
                    y81_out   <= y81_in;
    end if;
end process;

D_out1 <= y17_out;
D_out2 <= y26_out;
D_out3 <= y35_out;
D_out4 <= y44_out;
D_out5 <= y53_out;
D_out6 <= y62_out;
D_out7 <= y71_out;
D_out8 <= y81_out;

end conv_sort;
```

ДОДАТОК В

VHDL-КОД РЕКУРСИВНОГО ПРИСТРОЮ СОРТУВАННЯ

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity datapath_rec is
        port(
D_in1           : in std_logic_vector (7 downto 0);
D_in2           : in std_logic_vector (7 downto 0);
D_in3           : in std_logic_vector (7 downto 0);
D_in4           : in std_logic_vector (7 downto 0);
D_in5           : in std_logic_vector (7 downto 0);
D_in6           : in std_logic_vector (7 downto 0);
D_in7           : in std_logic_vector (7 downto 0);
D_in8           : in std_logic_vector (7 downto 0);
sel_mux2_1      : in std_logic;
sel_mux3_1      : in std_logic_vector(1 downto 0);
sel_mux6_1      : in std_logic_vector (2 downto 0);
sel_mux7_1      : in std_logic_vector (2 downto 0);
sel_mux21_1     : in std_logic_vector (4 downto 0);
sel_mux21_2     : in std_logic_vector (4 downto 0);
 sel_demux1_2   : in std_logic;
 sel_demux1_3   : in std_logic_vector (1 downto 0);
clk             : in std_logic;
D_out1          : out std_logic_vector (7 downto 0);
D_out2          : out std_logic_vector (7 downto 0);
D_out3          : out std_logic_vector (7 downto 0);
D_out4          : out std_logic_vector (7 downto 0);
D_out5          : out std_logic_vector (7 downto 0);
D_out6          : out std_logic_vector (7 downto 0);
D_out7          : out std_logic_vector (7 downto 0);
D_out8          : out std_logic_vector (7 downto 0)
            );
end datapath_rec;

architecture datapath_rec of datapath_rec is

signal OM1,OM2,IDL,IDG,Z1,Z2,Z3,Z4: STD_LOGIC_VECTOR(7 downto 0);
signal DIN1,DIN2,DIN3,DIN4,DIN5,DIN6,DIN7,DIN8,DIN9,DIN10 : STD_LOGIC_VECTOR(7 downto 0);
signal
DOUT,DOUT1,DOUT2,DOUT3,DOUT4,DOUT5,DOUT6,DOUT7,DOUT8,DOUT9,DOUT10,DOUT11,DOUT12,DOU
T13,DOUT14,DOUT15,DOUT16,DOUT17,DOUT18,DOUT19,DOUT20,DOUT21,DOUT22,DOUT23,DOUT24,DOUT2
5,DOUT26,R1 : STD_LOGIC_VECTOR(7 downto 0);

component mux6_1 is
port (
                I0 : in std_logic_vector (7 downto 0);
                 I1 : in std_logic_vector (7 downto 0);
                 I2 : in std_logic_vector (7 downto 0);
                 I3 : in std_logic_vector (7 downto 0);
                 I4 : in std_logic_vector (7 downto 0);
                 I5 : in std_logic_vector (7 downto 0);
                 S : in std_logic_vector (2 downto 0);
                 R : out std_logic_vector (7 downto 0)
        );
end component mux6_1;
component mux7_1 is
port (
```

```
                I0 : in std_logic_vector (7 downto 0);
                I1 : in std_logic_vector (7 downto 0);
                I2 : in std_logic_vector (7 downto 0);
                I3 : in std_logic_vector (7 downto 0);
                I4 : in std_logic_vector (7 downto 0);
                I5 : in std_logic_vector (7 downto 0);
                I6 : in std_logic_vector (7 downto 0);
                S : in std_logic_vector (2 downto 0);
                R : out std_logic_vector (7 downto 0)
        );
end component mux7_1;
component mux21_1 is
        port (
                I0  : in std_logic_vector (7 downto 0);
                I1  : in std_logic_vector (7 downto 0);
                I2  : in std_logic_vector (7 downto 0);
                I3  : in std_logic_vector (7 downto 0);
                I4  : in std_logic_vector (7 downto 0);
                I5  : in std_logic_vector (7 downto 0);
                I6  : in std_logic_vector (7 downto 0);
                I7  : in std_logic_vector (7 downto 0);
                I8  : in std_logic_vector (7 downto 0);
                I9  : in std_logic_vector (7 downto 0);
                I10 : in std_logic_vector (7 downto 0);
                I11 : in std_logic_vector (7 downto 0);
                I12 : in std_logic_vector (7 downto 0);
                I13 : in std_logic_vector (7 downto 0);
                I14 : in std_logic_vector (7 downto 0);
                I15 : in std_logic_vector (7 downto 0);
                I16 : in std_logic_vector (7 downto 0);
                I17 : in std_logic_vector (7 downto 0);
                I18 : in std_logic_vector (7 downto 0);
                I19 : in std_logic_vector (7 downto 0);
                I20 : in std_logic_vector (7 downto 0);
                S : in std_logic_vector (4 downto 0);
                R : out std_logic_vector (7 downto 0)
        );
end component mux21_1;
component mux3_1 is
port (
                I0 : in std_logic_vector (7 downto 0);
                I1 : in std_logic_vector (7 downto 0);
                I2 : in std_logic_vector (7 downto 0);
                S : in std_logic_vector (1 downto 0);
                R : out std_logic_vector (7 downto 0)
        );
end component mux3_1;
component mux2_1 is
port (
                I0 : in std_logic_vector (7 downto 0);
                I1 : in std_logic_vector (7 downto 0);
                S : in std_logic;
                R : out std_logic_vector (7 downto 0)
        );
end component mux2_1;
component demux1_2 is
port (
        O0 : out STD_LOGIC_VECTOR (7 downto 0);
        O1 : out STD_LOGIC_VECTOR (7 downto 0);
        S : in STD_LOGIC;
        I : in STD_LOGIC_VECTOR (7 downto 0)
        );
end component demux1_2;
```

```
component demux1_3 is
port (
        O0 : out STD_LOGIC_VECTOR (7 downto 0);
        O1 : out STD_LOGIC_VECTOR (7 downto 0);
        O2 : out STD_LOGIC_VECTOR (7 downto 0);
        S : in STD_LOGIC_VECTOR (1 downto 0);
        I : in STD_LOGIC_VECTOR (7 downto 0)
        );
end component demux1_3;

component operation_block is
        port(
         A_in : in STD_LOGIC_VECTOR(7 downto 0);
         B_in : in STD_LOGIC_VECTOR(7 downto 0);
         Greater : out STD_LOGIC_VECTOR(7 downto 0);
         Less : out STD_LOGIC_VECTOR(7 downto 0)
                );
end component operation_block;

component regs is
port(
      clk : in STD_LOGIC;
DIN1 : in STD_LOGIC_VECTOR(7 downto 0);
DIN2 : in STD_LOGIC_VECTOR(7 downto 0);
DIN3 : in STD_LOGIC_VECTOR(7 downto 0);
DIN5 : in STD_LOGIC_VECTOR(7 downto 0);
DIN6 : in STD_LOGIC_VECTOR(7 downto 0);
DIN7 : in STD_LOGIC_VECTOR(7 downto 0);
DIN8 : in STD_LOGIC_VECTOR(7 downto 0);
DIN9 : in STD_LOGIC_VECTOR(7 downto 0);
DIN10 : in STD_LOGIC_VECTOR(7 downto 0);
R1   : in STD_LOGIC_VECTOR(7 downto 0);
DOUT1 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT2 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT3 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT4 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT5 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT6 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT7 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT8 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT9 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT10 : out STD_LOGIC_VECTOR(7 downto0); DOUT11 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT12 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT13 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT14 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT15 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT16 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT17 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT18 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT19 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT20 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT21 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT22 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT23 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT24 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT25 : out STD_LOGIC_VECTOR(7 downto 0);
DOUT26 : out STD_LOGIC_VECTOR(7 downto 0)
                );
end component regs;

begin

D1 : mux21_1
```

```
        port map(
           I0  => DOUT1,
           I1  => DOUT2,
           I2  => DOUT2,
           I3  => DOUT2,
           I4  => DOUT3,
           I5  => DOUT3,
           I6  => DOUT3,
           I7  => DOUT3,
           I8  => DOUT3,
           I9  => DOUT4,
           I10 => DOUT4,
           I11 => DOUT4,
           I12 => DOUT4,
           I13 => DOUT4,
           I14 => DOUT4,
           I15 => DOUT4,
           I16 => DOUT4,
           I17 => DOUT3,
           I18 => DOUT3,
           I19 => DOUT3,
           I20 => DOUT2,
           S  => sel_mux21_1,
           R  => Z2
                 );
D2 : mux21_1
        port map(
           I0  => DOUT11,
           I1  => DOUT11,
           I2  => DOUT12,
           I3  => DOUT12,
           I4  => DOUT12,
           I5  => DOUT12,
           I6  => DOUT13,
           I7  => DOUT13,
           I8  => DOUT13,
           I9  => DOUT13,
           I10 => DOUT13,
           I11 => DOUT13,
          I12 => DOUT13,
           I13 => DOUT13,
           I14 => DOUT13,
           I15 => DOUT12,
           I16 => DOUT12,
           I17 => DOUT12,
           I18 => DOUT12,
           I19 => DOUT11,
           I20 => DOUT11,
           S  => sel_mux21_2,
           R  => Z4
                 );
D3 : mux7_1
        port map(
           I0 => D_in2,
           I1 => D_in3,
           I2 => D_in4,
           I3 => D_in5,
           I4 => D_in6,
          I5 => D_in7,
           I6 => D_in8,
           S  => sel_mux7_1,
           R  => Z3
                 );
```

```
D4 : mux6_1
        port map(
           I0 => DOUT5,
           I1 => DOUT6,
           I2 => DOUT7,
           I3 => DOUT8,
           I4 => DOUT9,
           I5 => DOUT10,
           S  => sel_mux6_1,
           R  => Z1
                 );
D5 : mux3_1
        port map(
           I0 => D_in1,
           I1 => Z1,
           I2 => Z2,
           S  => sel_mux3_1,
           R  => OM1
                 );
D6 : mux2_1
        port map(
           I0 => Z3,
           I1 => Z4,
           S  => sel_mux2_1,
           R  => OM2
                 );
D7 : operation_block
     port map(
           A_in   => OM1,
           B_in  => OM2,
           Greater => IDG,
           Less   =>   IDL
                  );

D8 : demux1_3
        port map(
           O0 => DIN2,
           O1 => DIN3,
           O2 => D_out8,
           S  => sel_demux1_3,
           I  => IDL
                 );
D9 : demux1_2
        port map(
           O0 => DIN1,
           O1 => R1,
           S  => sel_demux1_2,
           I  => IDG
                 );
D10 : regs
        port map(
                 clk => clk,
                 DIN1 => DIN1,
                 DOUT1 => DOUT1,
                 DIN2 => DIN2,
                 DOUT2 => DOUT2,
                 DIN3 => DIN3,
                 DIN5 => D_in3,
                 DIN6 => D_in4,
                 DIN7 => D_in5,
                 DIN8 => D_in6,
                 DIN9 => D_in7,
                 DIN10 => D_in8,
```

```
                R1 => R1,
                DOUT3 => DOUT3,
                DOUT4 => DOUT4,
                DOUT5 => DOUT5,
                DOUT6 => DOUT6,
                DOUT7 => DOUT7,
                DOUT8 => DOUT8,
                DOUT9 => DOUT9,
                DOUT10 => DOUT10,
                DOUT11 => DOUT11,
                DOUT12 => DOUT12,
                DOUT13 => DOUT13,
                DOUT14 => DOUT14,
                DOUT15 => DOUT15,
                DOUT16 => DOUT16,
                DOUT17 => DOUT17,
                DOUT18 => DOUT18,
                DOUT19 => DOUT19,
                DOUT20 => DOUT20,
                DOUT21 => DOUT21,
                DOUT22 => DOUT22,
                DOUT23 => DOUT23,
                DOUT24 => DOUT24,
                DOUT25 => DOUT25,
                DOUT26 => DOUT26
        );
D_out1 <= DOUT20;
D_out2 <= DOUT19;
D_out3 <= DOUT18;
D_out4 <= DOUT17;
D_out5 <= DOUT16;
D_out6 <= DOUT15;
D_out7 <= DOUT14;

end datapath_rec;
```

ДОДАТОК Г

VHDL-КОД ЗАВЕРШЕНОГО ПРИСТРОЮ

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity device is
      port(
clk      : in STD_LOGIC;
 rst      : in STD_LOGIC;
 I_clk  : in STD_LOGIC;
 I_ack  : in STD_LOGIC;
we_i    : in STD_LOGIC;
stb_i     : in STD_LOGIC;
ack_o   : out STD_LOGIC;
Data     : in STD_LOGIC_VECTOR(7 downto 0);
Result  : out STD_LOGIC_VECTOR(7 downto 0)
         );
end device;
architecture device of device is
signal  reg_wr, en_r1, en_r2, en_r3, en_r4, en_r5, en_r6, en_r7, en_r8, sel_mux2_1, sel_demux1_2,  oe, en, we_o, stb_o,
ack_i : std_logic;
signal  sel_mux3_1, sel_demux1_3          : std_logic_vector(1 downto 0);
signal  sel_mux6_1, sel_mux7_1, mux_sel   : std_logic_vector(2 downto 0);
signal  sel_mux21_1, sel_mux21_2          : std_logic_vector(4 downto 0);
component cont_unit is
         port(
  clk                : in std_logic;
  rst                : in std_logic;
  I_ack              : in std_logic;
  I_clk              : in std_logic;
  ack_i              : in std_logic;
 ack_o               : out std_logic;
 we_o                : in std_logic;
 stb_o               : in std_logic;
 we_i                : out std_logic;
 stb_i               : out std_logic;
en_r1                : out std_logic;
en_r2                : out std_logic;
en_r3                : out std_logic;
en_r4                : out std_logic;
en_r5                : out std_logic;
en_r6                : out std_logic;
en_r7                : out std_logic;
en_r8                : out std_logic;
en                   : out std_logic;
reg_wr               : out std_logic;
sel_mux2_1     : out std_logic;
sel_demux1_2 : out std_logic;
sel_mux3_1     : out std_logic_vector(1 downto 0);
sel_mux6_1     : out std_logic_vector (2 downto 0);
 sel_mux7_1    : out std_logic_vector (2 downto 0);
sel_mux21_1   : std_logic_vector (4 downto 0);
sel_mux21_2   : out std_logic_vector (4 downto 0);
sel_demux1_3 : out std_logic_vector (1 downto 0);
 mux_sel      : out std_logic_vector (2 downto 0)
    end component cont_unit;
component datapath is
         port(
INDATA            : in std_logic_vector (7 downto 0);
```

```
clk                 : in std_logic;
 rst                : in std_logic;
 en_r1              : in std_logic;
 en_r2              : in std_logic;
  en_r3             : in std_logic;
  en_r4             : in std_logic;
en_r5               : in std_logic;
en_r6               : in std_logic;
 en_r7              : in std_logic;
 en_r8              : in std_logic;
  reg_wr            : in std_logic;
sel_mux2_1          : in std_logic;
sel_mux3_1          : in std_logic_vector (1 downto 0);
sel_mux6_1          : in std_logic_vector (2 downto 0);
sel_mux7_1          : in std_logic_vector (2 downto 0);
sel_mux21_1         : in std_logic_vector (4 downto 0);
sel_mux21_2         : in std_logic_vector (4 downto 0);
 sel_demux1_2       : in std_logic;
sel_demux1_3        : in std_logic_vector (1 downto 0);
mux_sel             : in std_logic_vector (2 downto 0);
en                  : in std_logic;
 we_i               : in std_logic;
stb_i               : in std_logic;
ack_o               : out std_logic;
OUTDATA             : out std_logic_vector (7 downto 0)     );
   end component datapath;
begin
    D7 : cont_unit
    port map(
rst             =>  rst,
clk             =>  clk,
I_ack           =>  I_ack,
I_clk           =>  I_clk,
ack_i           =>  ack_i,
ack_o           =>  ack_o,
we_i            =>  we_o,
stb_i           =>  stb_o,
we_o            =>  we_i,
en              =>  en,
reg_wr          =>  reg_wr,
stb_o           =>  stb_i,
mux_sel         =>  mux_sel,
sel_mux21_1     =>  sel_mux21_1,
sel_mux21_2     =>  sel_mux21_2,
sel_mux7_1      =>  sel_mux7_1,
sel_mux6_1      =>  sel_mux6_1,
 sel_mux3_1     =>  sel_mux3_1,
 sel_mux2_1     =>  sel_mux2_1,
 sel_demux1_2   =>  sel_demux1_2,
 sel_demux1_3   =>  sel_demux1_3,
 en_r1          =>  en_r1,
       en_r2            =>  en_r2,
       en_r3            =>  en_r3,
       en_r4            =>  en_r4,
       en_r5            =>  en_r5,
       en_r6            =>  en_r6,
       en_r7            =>  en_r7,
       en_r8            =>  en_r8);
   D8 : datapath
    port map(
 clk            =>  clk,
 rst            =>  rst,
 INDATA         =>  Data,
```

```
 mux_sel       =>   mux_sel,
sel_mux21_1  =>   sel_mux21_1,
sel_mux21_2  =>   sel_mux21_2,
sel_mux7_1    =>   sel_mux7_1,
sel_mux6_1    =>   sel_mux6_1,
sel_mux3_1    =>   sel_mux3_1,
sel_mux2_1    =>   sel_mux2_1,
sel_demux1_2 =>   sel_demux1_2,
sel_demux1_3 =>   sel_demux1_3,
en_r1             =>   en_r1,
en_r2             =>   en_r2,
en_r3             =>   en_r3,
en_r4             =>   en_r4,
en_r5             =>   en_r5,
en_r6             =>   en_r6,
en_r7             =>   en_r7,
en_r8             =>   en_r8,
reg_wr           =>   reg_wr,
we_i              =>   we_i,
stb_i             =>   stb_i,
ack_o            =>   ack_i,
en                 =>   en,
OUTDATA     =>  Result     );
end device;
```

ДОДАТОК Д

(обов’язковий)

ПРЕЗЕНТАЦІЯ ДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ

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

Спеціалізована система сортування даних на базі ПЛІС

Виконала студентка групи КІ2м-22-2 Горбач Г.С.
Науковий керівник-к.т.н., доцент Грига В.М.

Хмельницький 2024

МЕТА І ЗАДАЧІ ДОСЛІДЖЕННЯ

- *Метою* кваліфікаційної роботи є дослідження й проєктування спеціалізованої системи сортування даних на базі ПЛІС
- Об’єктом дослідження є процес проектування спеціалізованої системи сортування даних на основі просторово-часових графів.
- *Предметом* дослідження є методи та алгоритми сортування масивів двійкових даних.

МЕТА І ЗАДАЧІ ДОСЛІДЖЕННЯ

Поставлена мета досягається розв'язанням таких основних зададач:

- дослідити відомі методи сортування даних;
- дослідити відомі методи сортування даних, їх переваги та недоліки;
- дослідити бази ПЛІС як засіб для вирішення задачі;
- розробити спеціалізовану систему сортування даних на базі ПЛІС;
- протестувати та оцінити ефективність системи сортування даних на базі ПЛІС, написаною на мові програмування VHDL.

НАУКОВА НОВИЗНА ОТРИМАНИХ РЕЗУЛЬТАТІВ

- набув подальшого розвитку рекурсивний метод сортування масивів двійкових даних на базі просторово-часових графів, який на відміну від відомих дозволяє ефективно оптимізувати кількість операцій сортування, що забезпечує низьку апаратну складність спеціалізованої системи;
- набула подальшого розвитку інформаційна система сортування даних на базі ПЛІС, такий підхід відкриває нові можливості для швидкого та ефективного сортування великих обсягів даних, що робить його важливим внеском у сучасну область обчислювальних технологій;
- удосконалено структуру спеціалізованої системи сортування двійкових даних на основі графів, яка завдяки використанню удосконалених методів, функціональних моделей паралельно-потокового сортування та урахуванню інтенсивності надходження даних, розмірів масивів даних і можливостей реалізації дозволяє здійснювати сортування даних у реальному часі з високою ефективністю використання обладнання.

ПРАКТИЧНЕ ЗНАЧЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Дослідження спрямоване на створення ефективних засобів спеціалізованих систем сортування, які базуються на ПЛІС. Ці засоби призначені для оптимізації процесу сортування великих обсягів даних у різних сферах, що забезпечує оптимальне використання ресурсів і прискорює обробку даних.

АКТУАЛЬНІСТЬ

Сортування наразі важливе для організації даних у більш зручний і структурований спосіб. Це процес, що допомагає систематизувати дані шляхом розташування їх у певному логічному порядку. Під час сортування елементи однотипної послідовності даних, такі як масив, список або файл, перегруповуються таким чином, щоб вони відповідали певному критерію порядку.

Алгоритм сортування - це конкретний метод, за допомогою якого елементи розташовуються в певному порядку. Цей порядок може бути числовим (за зростанням або спаданням значень), лексикографічним (за алфавітом) або будь-яким іншим, який задає користувач.

Мета сортування полягає в полегшенні подальшої роботи з даними, такої як пошук певного елемента, оновлення значень, виключення або включення нових елементів у структуру даних.

Алгоритми сортування

Кожен алгоритм сортування має свої особливості:

Сортування бульбашкою: Цей алгоритм порівнює сусідні елементи і переміщує їх, якщо вони знаходяться у неправильному порядку. Він продовжує робити це до тих пір, поки всі елементи не будуть впорядковані.

Сортування вставками: Цей алгоритм працює, перебираючи елементи списку один за одним і вставляючи їх у відповідні місця вже впорядкованої частини списку. Це дозволяє ефективно впорядковувати малі списки.

Сортування вибором: Цей алгоритм вибирає найменший (або найбільший) елемент зі списку і переміщує його на відповідне місце. Це продовжується до тих пір, поки весь список не буде впорядкований.

Швидке сортування: Цей алгоритм використовує стратегію "розділити і володіти", рекурсивно розбиваючи список на менші частини, сортуючи їх і потім об'єднуючи назад у вірному порядку.

Сортування злиттям: Цей алгоритм також використовує стратегію "розділити і володіти", розбиваючи список на половини, сортуючи їх і потім об'єднуючи дві впорядковані половини в один список.

Класифікація методів сортування

ОБГРУНТУВАННЯ ВИБРАНОЇ МЕТОДИКИ СОРТУВАННЯ

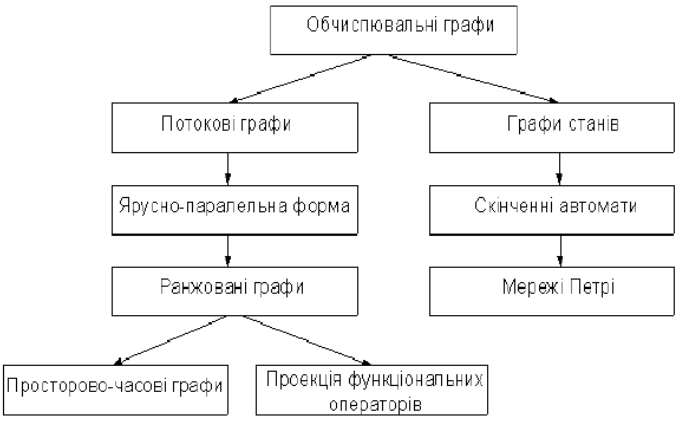
Загальноприйнято проектування апаратури комп'ютерних систем проводити окремо для пристроїв керування та для операційних пристроїв. Якщо для проектування пристроїв керування в більшості випадків підходять методи теорії абстрактних автоматів, то для операційних пристроїв ці методи не завжди підходять,оскільки робота їх вимагає врахування просторового та часового аспектів функціонування. *З цією метою було запропоновано* **графові моделі**, а саме моделі потокового графа, а саме у ярусно-паралельній формі (ЯПФ) . Наприклад, на основі ЯПФ базуються методи побудови конвеєрних пристроїв , синтезу топологій комп'ютерних систем та інші. Вони орієнтовані на максимальне застосування паралелізації чи конвеєризації, тобто орієнтовані на максимальний обсяг апаратури.

КАСИФІКАЦІЯ ОБЧИСЛЮВАЛЬНИХ ГРАФІВ

Граф $G = (N, A)$ - це сукупність двох множин (об'єктів): $N = \{n_1, n_2 \dots n_n\}$ - скінчена непуста множина вершин (вузлів); $A = \{a_1, a_2 \dots a\}$ - скінчена множина пар, що з'єднані між собою і утворють ребро графу G.

Обчислювальні графи поділяються на потокові та графи станів.

Потокові графи використовуються для проектування операційних пристроїв, тоді як графи станів використовуються для керуючих пристроїв. У потокових графів є обмеженість до бінарних операцій та відсутність розрізнення номерів входів до вершин. Графи станів, зазвичай, представляють собою скінченні абстрактні автомати Мура та Мілі.



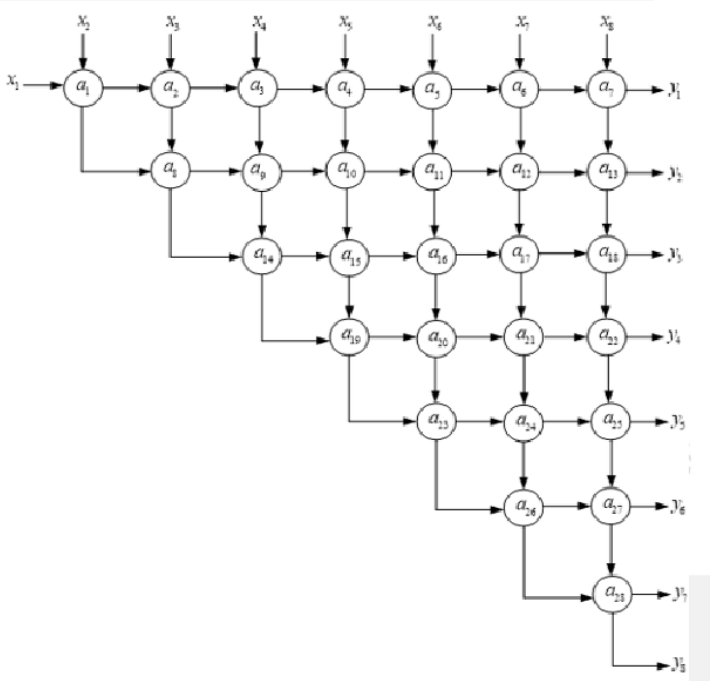
ПРОЕКТУВАННЯ ПРИСТРОЇВ СОРТУВАННЯ НА ОСНОВІ ПРОСТОРОВО-ЧАСОВИХ ГРАФІВ

Проектування спеціалізованої системи сортування згідно даної методики розпочинається з графічного способу представлення алгоритму за допомогою обчислювального графу. Далі отримавши граф алгоритму, потрібно побудувати потоковий граф, щоб виявити паралелізм забезпечуючи тим самим можливість знаходження компромісних просторово-часових співвідношень для зображення графа алгоритму в ярусно-паралельній формі. Наступним кроком є побудова різних типів просторово-часових графів, які дозволяють проектувати конвеєрні, паралельні та рекурсивні спеціалізовані пристрої та робити оцінку побудованим структурам згідно описаних технічних параметрів.

Ітерація внутрішнього циклу обраного методу сортування «бульбашкою» виконується за час O(n), всього ми виконуємо O(n) ітерацій; таким чином, складність «бульбашкового» сортування - $O(n^2)$.

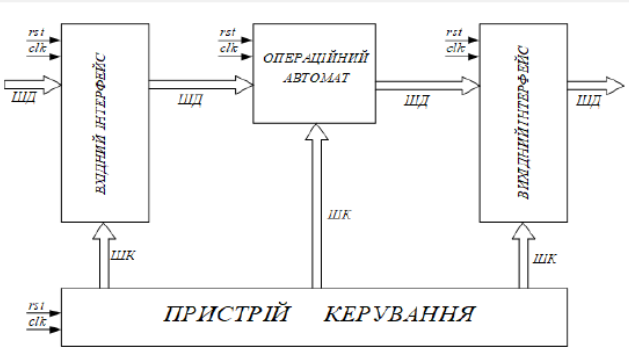
Послідовний алгоритм сортування чисел методом «бульбашки» можна описати апаратним способом – за допомогою графу алгоритму в якому вершини виконують операцію порівняння, а дуги служать для передачі даних з однієї вершини до іншої.

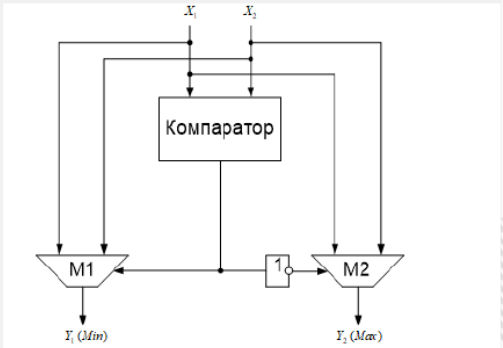
Для алгоритмів сортування цього методу кількість операцій (вершин) залежить від кількості вхідних чисел та визначається за формулою $n(n-1)/2$,



Структура удосконаленої системи сортування двійкових даних

На рисунку зображено загальну структуру спеціалізованої системи сортування двійкових даних, яка складається з вхідного інтерфейсу, операційного автомату, вихідного інтерфейсу та пристрою керування.

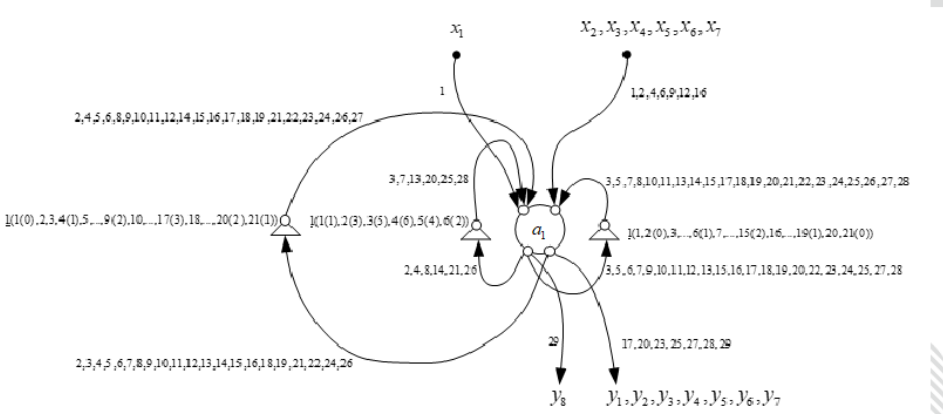




Структурна схема операції “порівняти й переставити”

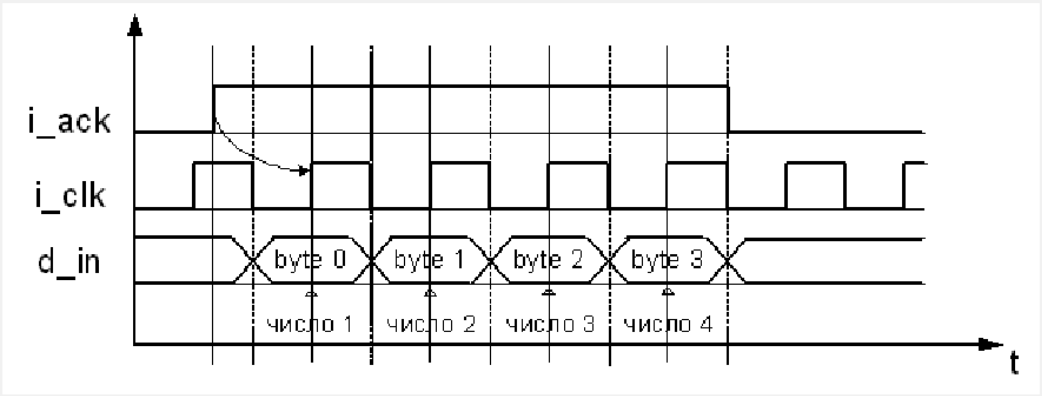
Структура рекурсивного просторово-часового графу алгоритму сортування 8-ми чисел методом

Складається з однієї вершини, яка має два вхідні та два вихідні вузли. Біля дуг даного графу зображені числові такти під час яких на вхідні вузли поступають основні дані та проміжні результати, які надходять з вихідних вузлів.



Трикутними елементами позначаються елементи затримки, які затримують послідовність проміжних результатів на потрібну кількість тактів. Часова затримка та пропускна здатність для рекурсивного просторово-часового графу алгоритму сортування методом “бульбашки” співпадають і складають 29 тактів. Пристрій керування реалізований за допомогою автомату Мура.

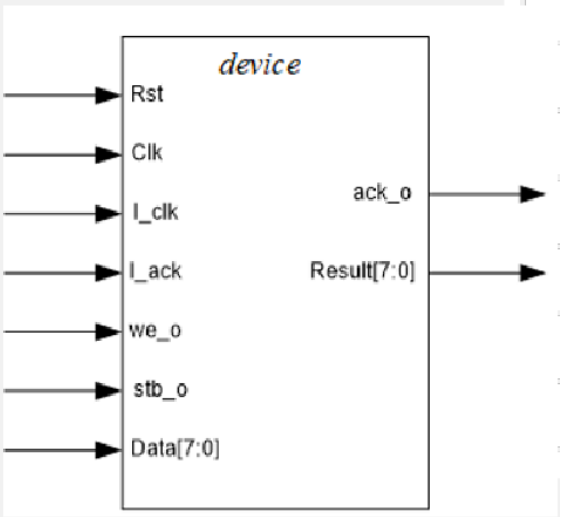
Часова діаграма протоколу передачі вхідних даних



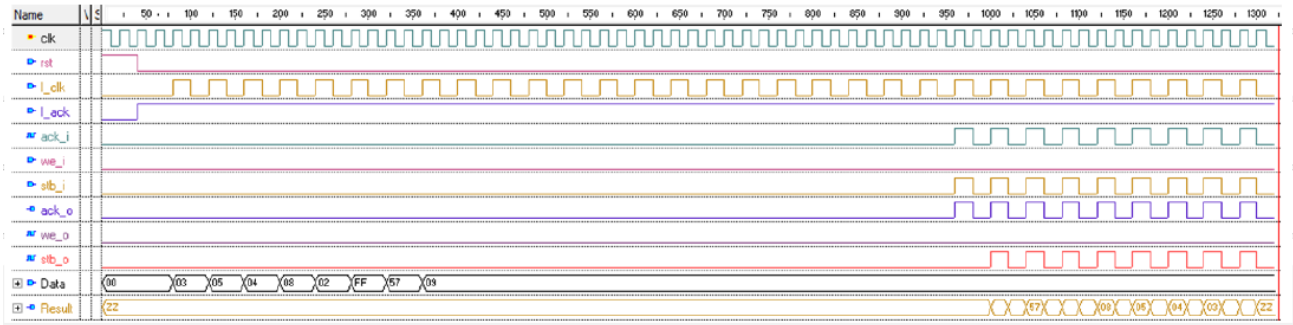
На даній діаграмі вхідні числа передаються пакетами по 8-ми бітовій шині даних (d_in). На виході регістрів дані об’єднуються в 64-ох розрядний пакет та поступають на вхід операційного автомату для обробки.

Блок-схема

На рисунку представлено блок-схему спеціалізованої системи сортування, яка включає в себе всі компоненти, необхідні для виконання операції сортування вхідних даних.
У верхній частині схеми зображено модуль керуючого автомата.Керуючий автомат приймає вхідні дані, координує всі інші операції.
Нижче розміщені два блоки операційного автомата, які відповідають за виконання фактичного сортування. Кожен операційний автомат отримує вхідні дані та виконує порівняння та переставлення, використовуючи внутрішню логіку.
Посередині схеми розташовані модулі вхідного та вихідного інтерфейсів. Внизу схеми розміщений блок зовнішньої системної шини



Діаграма функціональної симуляції спеціалізованої системи сортування двійкових даних. На даній діаграмі зображені вхідні та вихідні сигнали, які генерує пристрій керування та сигнали інформаційного тракту проектованої системи.



Ключові показники, що були отримані після імплементації цієї системи

- швидкодія (Clock Frequency): Результати показують, що максимальна частота тактового сигналу, яку може підтримувати ця структура, складає 80.567 МГц. Це означає, що період тактового сигналу становить 12.412 наносекунди;
- затрати обладнання (Hardware Utilization);
- кількість Slices: вказано, що було використано 397 з 768 доступних Slice ресурсів на цій ПЛІС. Slice - це основний логічний блок у ПЛІС, який містить логічні елементи, регістри та інші ресурси;
- відсоток використання: У відсотковому вираженні це означає, що приблизно 51% доступного обладнання на ПЛІС було використано для цієї базової структури НВІС.

Висновки

Результати дослідження показали, що розроблена спеціалізована система сортування двійкових даних на базі рекурсивного просторово-часового графу має високу ефективність та швидкодію. Описано компоненти спеціалізованої системи сортування двійкових даних на мові VHDL. Проведено моделювання структури рекурсивного пристрою сортування та синтез на ПЛІС фірми Xilinx. Визначено, що рекурсивна спеціалізована система має накращі характеристики по затратах обладнання у порівнянні з відомими реалізаціями пристроїв сортування.

Дякую за увагу!

ДОДАТОК Е

(обов'язковий)

КОПІЯ ПУБЛІКАЦІЇ

УДК 004.942

В.М.Грига[1], Г.С. Горбач[2], Т.І.Ткачук[3], В.М. Вінтоняк[1], А.В. Іванишин[1], Є.Л. Процак[4]

[1]Прикарпатський національний університет імені Василя Стефаника
gr.volodymyr2018@gmail.com, vitalii.vintoniak.23a@pnu.edu.ua
aivanushun920@gmail.com
[2]Хмельницький національний університет
horbach_h@gmail.com
[3]Національний університет "Львівська політехніка"
xaunder@gmail.com
[4]ВСП "Надвірнянський фаховий коледж НТУ"
protsakyevhen@ukr.net

СПЕЦІАЛІЗОВАНА СИСТЕМА СОРТУВАННЯ ДВІЙКОВИХ ДАНИХ

Вступ. Впродовж останніх років у розвитку комп'ютерної техніки проявилася стійка тенденція все ширшого застосування спеціалізованих комп'ютерних систем, оскільки вони, як відомо, більш ефективно розв'язують одну конкретну задачу чи клас задач по відношенню до універсальних комп'ютерних систем [1]. Крім того, спеціалізовані комп'ютерні системи не мають "зайвої" апаратури, яка завжди є в універсальних системах для забезпечення розв'язку довільної задачі, а тому вони мають вищу надійність [2,3]. Ще одним фактором інтенсивного розвитку спеціалізованих комп'ютерних систем є можливість їх реалізації на одному кристалі НВІС [1-4]. Щодо структур спеціалізованих комп'ютерних систем, то можна зауважити надзвичайно велике їх різноманіття. Попри те, у них усіх можна виявити одну особливість - застосування спеціалізованих пристроїв, які забезпечують реалізацію окремих операцій, наприклад, арифметичних, порівняння і переставлення даних, обміну інформації тощо, за один чи декілька тактів, що, таким чином, підвищує продуктивність систем в цілому [2-4]. Серед них можна виділити апаратні пристрої сортування [2], які застосовуються в процесі розв'язку задач обробки даних, побудови комутуючих мереж та ін [5]. Ці пристрої можуть бути побудовані так, що операція виконується за один такт і при цьому вони мають найбільший обсяг апаратури внаслідок її максимального паралелелізму, або за певну кількість тактів, при якій обсяг апаратури є меншим внаслідок багатократного використання одних і тих самих елементів. Важливим моментом у цьому випадку є вибір оптимального співвідношення між часом роботи таких пристроїв (кількістю тактів) та обсягом їх апаратури при заданій продуктивності спеціалізованої комп'ютерної системи [1-3, 6].

Сортування даних – це задача, яка найбільш часто використовується в комп'ютерних системах обробки даних [7]. Існує багато способів сортуванняя чисел, зокрема обмінне сортування, сортування вставками, сортування вибором, сортування злиттям, сортування підрахунком та інші [7-12]. Дані способи реалізуються за допомогою послідовних та паралельних алгоритмів сортування. Послідовні методи проходять по багатотактній структурі пристрою сортування, тобто результат отримується через певну кількість циклів. Паралельні методи сортування проходять по однотактовій структурі пристрою сортування, тобто відбувається подача всіх вхідних даних одночасно. Кожний з даних методів має свої переваги та недоліки [7,8,10,12].

Метою роботи є аналіз послідовних та паралельних алгоритмів сортування двійкових даних та проектування спеціалізованої системи сортування двійкових даних з використанням мови апаратного опису VHDL, виконанні симуляції на функціональному рівні та проведення тестування і синтезу пристрою на ПЛІС.

Розділ 1. Класифікація методів сортування.

Сортування є одною із типових проблем обробки даних і зазвичай розуміється як завдання розміщення елементів неврегульованого набору значень $(S = a_1, a_2, ..., a_n)$ в порядку монотонного зростання або спадання: $S \sim S' = (a'_1, a'_2, ..., a'_n) : a'_1 \le a'_2 \le ... \le a'_n$ [7].

Рис. 1. Класифікація методів сортування.

Обчислювальна трудомісткість процедури впорядковування є достатньо високою. Так, для ряду відомих простих методів (бульбашкове сортування, сортування включенням і ін.) кількість необхідних операцій визначається квадратичною залежністю числа впорядковуваних даних [7,8]:

$$T \sim n^2$$

Для ефективніших алгоритмів (сортування злиттям, сортування Шелла, швидке сортування) трудомісткість визначається величиною:

$$T_1 \sim n\log_2 n.$$

Даний вираз дає також нижню оцінку необхідної кількості операцій для впорядковування набору з n значень; алгоритми з меншою трудомісткістю можуть бути отримані тільки для шаблонних варіантів завдання.

Прискорення сортування може бути отримано, використанням декількох ($p>1$) процесорів. Початковий тестовий набір у цьому випадку розділяється між процесорами; в ході сортування дані пересилаються між процесорами і порівнюються між собою. Результуючий (впорядкований) набір, як правило, також розподіляється між процесорами; при цьому для систематизації такого розділення для процесорів вводиться та або інша система послідовної нумерації і зазвичай потрібно, щоб при завершенні сортування значення, що розташовуються на процесорах з меншими номерами, не перевищували значень процесорів з великими номерами [9].

Розділ 2. Послідовні алгоритми сортування двійкових даних.

2.1. Послідовний алгоритм бульбашкового сортування

Послідовний алгоритм бульбашкового сортування (the bubble sort algorithm) порівнює і обмінює сусідні елементи в послідовності, яку потрібно відсортувати [7,10,12]. Основний принцип методу полягає у виштовхуванні маленьких значень на вершину списку в той час, як великі значення

опускаються вниз. Алгоритм бульбашкового сортування виконує декілька проходів по списку. Для послідовності(a_1a_2 ..., a_n) алгоритм виконує n_1 базових операцій "порівняння - обміну" для послідовних пар елементів (a_1a_2), (a_2a_3), ..., ($a_{n1}a_n$).

Якщо порядок сусідніх елементів неправильний, то вони міняються позиціями.

Рис.2. Візуальне представлення алгоритму сортування бульбашкою.

Кожний прохід починається з початку списку. Спершу порівнюються перший і другий елементи, потім другий і третій, потім третій і четвертий і так далі; елементи з неправильним порядком в парі переставляються. При виявленні на першому проході найбільшого елементу списку він буде переставлятися зі всіма наступними елементами, поки не дійде до кінця списку. Тому при другому проході немає необхідності проводити порівняння з останнім елементом. При другому проході другий по величині елемент списку переміститься на другу позицію з кінця. При продовженні процесу на кожному проході як мінімум одне з наступних по величині значень переміщається на свою позицію. При цьому менші значення також збираються вгорі. Якщо при виконанні проходу не сталося жодної перестановки елементів, то всі вони стоять в потрібному порядку, і виконання алгоритму можна припинити. Варто помітити, що при кожному проході ближче до своєї позиції просувається відразу декілька елементів, хоча гарантовано займає кінцеве положення лише один.

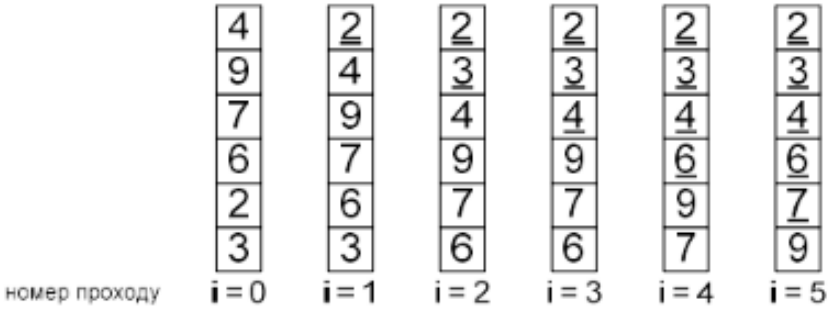


Рис. 3. Проміжні набори сортування.

Як можна побачити, послідовність буде відсортована після n ітерації. У алгоритму сортування бульбашкою є декілька різновидностей. В нашому випадку ми розглядаємо простий метод сортування "бульбашкою", в літературі також відомі модифікований метод сортування "бульбашкою" та метод парно-непарної перестановки. Ефективність бульбашкового сортування може бути покращена, якщо завершувати алгоритм у разі відсутності змін у сортованій послідовності даних в ході ітерацій сортування.

Ітерація внутрішнього циклу "бульбашкового" сортування виконується за час O(n), всього ми виконуємо O(n) ітерацій; таким чином, складність "бульбашкового" сортування - $O(n^2)$.

Послідовний алгоритм сортування чисел методом "бульбашки" можна описати апаратним способом – за допомогою графу алгоритму в якому вершини виконують операцію порівняння а дуги служать для передачі даних з однієї вершини до іншої.

Для алгоритмів сортування методом "бульбашки" кількість операцій (вершин) залежить від кількості вхідних чисел та визначається за формулою $n(n-1)/2$, де n - кількість вхідних значень [2,6,10,12]

На рис. 4. зображено граф алгоритму для сортування 5-ти вхідних цілих чисел [2,6,10,12].

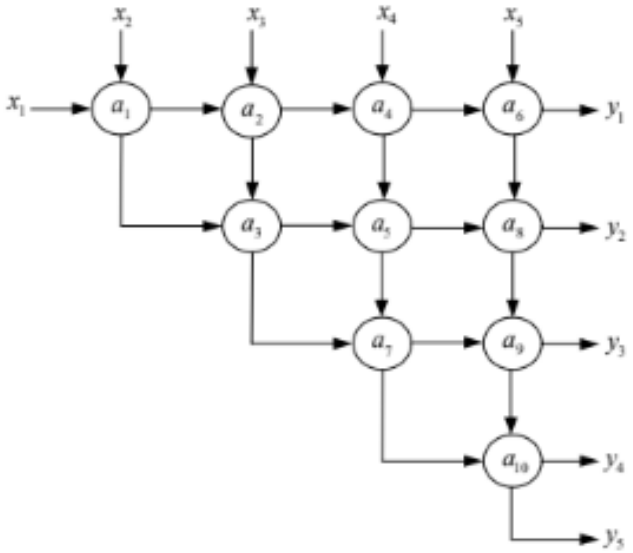


Рис. 4. Граф алгоритму сортування чисел методом "бульбашки".

На виході графу отримується нова послідовність 5-ти вихідних чисел, яка складається з вхідних чисел переставлених в потрібному порядку.

Даний граф виконує 10 операцій порівняння і переставлення. В результаті виконання кожної операції алгоритму на виходах ми отримаємо відсортовану послідовність чисел в порядку спадання.

2.2. Послідовний алгоритм Шелла

Сортування Шелла було запропонований Дональдом Л. Шеллом [7-9]. Загальна ідея сортування Шелла (the Shell sort) полягає в порівнянні на початкових стадіях сортування пар значень, що розташовуються достатньо далеко один від одного в впорядковуваному наборі даних. Така модифікація методу сортування дозволяє швидко переставляти далекі неврегульовані пари значень (сортування таких пар зазвичай вимагає великої кількості перестановок, якщо використовується порівняння тільки сусідніх елементів). Незвичність цього методу полягає в тому, що вона розглядає весь список як сукупність перемішаних підсписків. На першому кроці ці підсписки представляють собою просто пари елементів. На другому кроці вони формуються уже в групи по чотири елементи. При повторенні процесу число елементів у кожному підсписку збільшується, а кількість підсписків, відповідно, зменшується.

Загальна схема методу полягає в наступному. На першому кроці алгоритму відбувається впорядковування елементів n/2 пара (a_i, $a_{n/2+i}$) для $1 \le i \le n/2$. Далі, на другому кроці упорядковуються елементи в n/4 групах з чотирьох елементів (ai, an/4+i, an/2+i, a3n/4+i) для $1 \le i \le n/4$. На третьому кроці упорядковуються елементи вже в n/4 групах з восьми елементів і т.д. На останньому кроці

упорядковуються елементи відразу у всьому масиві ($a_1, a_2 ..., a_n$). Як можна відмітити, загальна кількість ітерацій алгоритму Шелла рівна $\log_2 n$ (рис. 5).

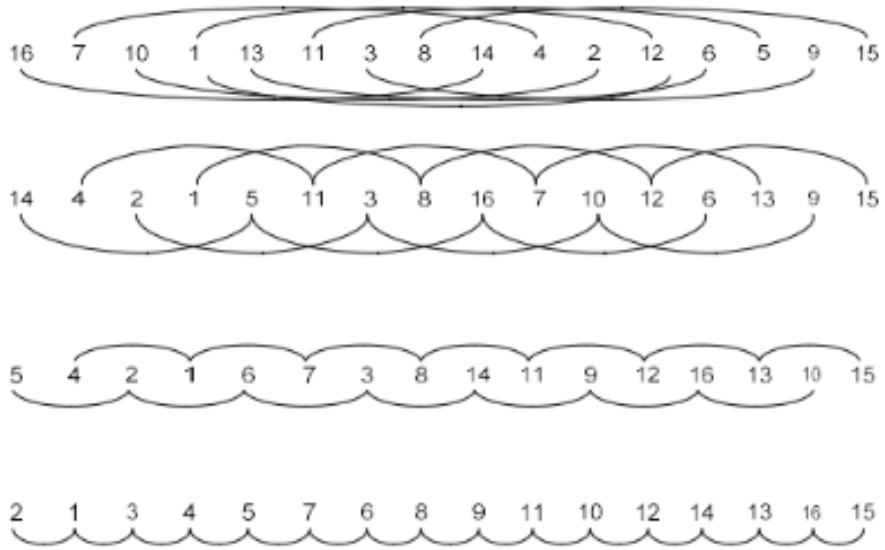


Рис. 5. Чотири проходи сортування Шелла.

Перший підсписок цього разу містить перший, п'ятий, дев'ятий і тринадцятий елементи. Другий підсписок складається з другого, шостого, десятого і чотирнадцятого елементів. Далі показані два підсписки, що складаються з елементів з непарними і парними номерами відповідно, і знову повертаємося до одного списку.

Сортування підсписків виконується шляхом однократного використання сортування вставками.

Паралельний алгоритм сортування Шелла може бути отриманий як узагальнення методу паралельного бульбашкового сортування. Основна відмінність полягає в тому, що на перших ітераціях алгоритму Шелла відбувається порівняння пар елементів, які в початковому наборі даних знаходяться далеко один від одного.

Розділ 3. Паралельні алгоритми сортування двійкових даних.

3.1. Паралельний алгоритм Шелла

Для алгоритму Шелла може бути запропонований паралельний аналог методу, якщо топологія комунікаційної мережі може бути ефективно представлена у вигляді N-мірного гіперкуба (тобто кількість процесорів рівна $p = 2^N$) [7,8]. Виконання сортування у такому разі може бути розділено на два послідовні етапи. На першому етапі (N ітерацій) здійснюється взаємодія процесорів, що є сусідніми в структурі гіперкуба (але ці процесори можуть виявитися далекими при лінійній нумерації; для встановлення відповідності двох систем нумерації процесорів може бути використаний, як і раніше, код Грея. Формування пар процесорів, що взаємодіють між собою при виконанні операції «порівняти і розділити», може бути забезпечене за допомогою наступного простого правила – на кожній ітерації i, $0 \le i \le N$ парними стають процесори, у яких відмінність в бітових представленнях їх номерів є тільки у позиції N_i.

Другий етап полягає в реалізації звичайних ітерацій паралельного алгоритму парно-непарної перестановки. Ітерації даного етапу виконуються до припинення фактичної зміни сортованого набору, і, тим самим, загальна кількість L таких ітерацій може бути різним – від 2 до p.

На рис. 6 показаний приклад сортування масиву з 16 елементів за допомогою розглянутого способу (процесори показані кругами, номери процесорів дані в бітовому уявленні). Потрібно відмітити, що дані виявляються впорядкованими вже після першого етапу і немає необхідності виконувати парно-непарну перестановку.

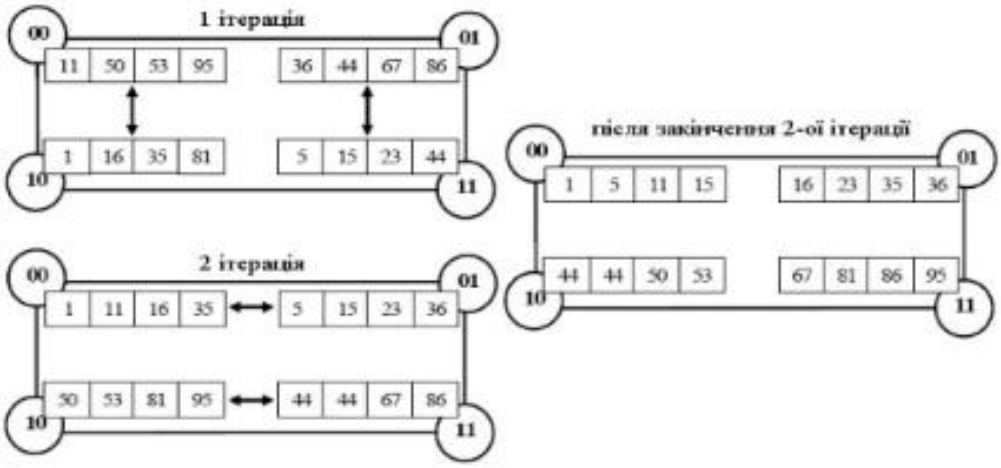


Рис. 6. Приклад роботи алгоритму Шелла для 4 процесорів.

З урахуванням представленого опису паралельного варіанту алгоритму Шелла базова підзадача для організації паралельних обчислень, як і раніше, може бути визначена на основі операції "порівняти і розділити". Як результат, кількість підзадач завжди співпадає з числом наявних процесорів (розмір блоків даних в підзадачах рівний n/p) і не виникає проблеми масштабування. Розподіл блоків упорядковуваного набору даних по процесорах повинен бути вибраний з урахуванням можливості ефективного виконання операцій "порівняти і розділити" при представленні топології мережі передачі даних у вигляді гіперкуба.

3.2. Паралельний алгоритм сортування методом "парно-непарної" перестановки.

Алгоритм сортування методом "парно-непарної" перестановки потребує для N вхідних значень - $N(N-1)/2$ операцій "порівняти і переставити" а часова складність даного алгоритму складає N операцій "порівняти і переставити" [2,6,10,12].

На рис. 7 зображено граф алгоритму сортування "парно-непарної" перестановки для 6-ти вхідних значень [2,6,10,12].

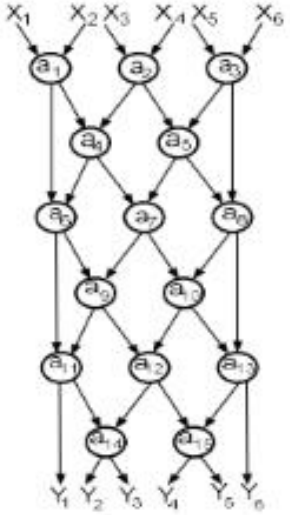


Рис. 7. Граф алгоритму сортування 6-ти значень методом "парно-непарної" перестановки.

Для виявлення паралелізму графу алгоритму сортування методом "парно-непарної" перестановки, його потрібно подати у вигляді ярусно-паралельної форми (ЯПФ), в якій всі вершини графу розділені на яруси таким чином, що в межах одного ярусу між вершинами не має зв'язків. ЯПФ визначає степінь паралелізму графу (максимальна кількість вершин на одному ярусі або ширина графу), а також мінімально-можливий час обчислення даного алгоритму (кількість ярусів або висота графу).

Ширина ПГ алгоритму сортування методом "парно-непарної" перестановки складає - $N/2$, а висота - N операцій "порівняти і переставити".

Апаратна реалізація описаного вище алгоритму сортування передбачає повне відображення його ПГ у структуру операційного пристрою, в якому вершинам графу (функціональним операторам) буде відповідати апаратний блок (операція) а дугам – лінії для передачі вхідних даних та проміжних і кінцевих результатів.

3.3. Паралельний алгоритм сортування методом Бетчера.

Паралельний алгоритм Бетчера базується на деяких парах несусідніх ключів, що дозволяє зменшити кількість операцій порівняння та час роботи алгоритму [2,6,10-12].

На рис. 8 зображено апаратну структуру пристрою сортування двійкових чисел методом Бетчера для 8-ми вхідних значень ($x_1,\ldots,x_8$).

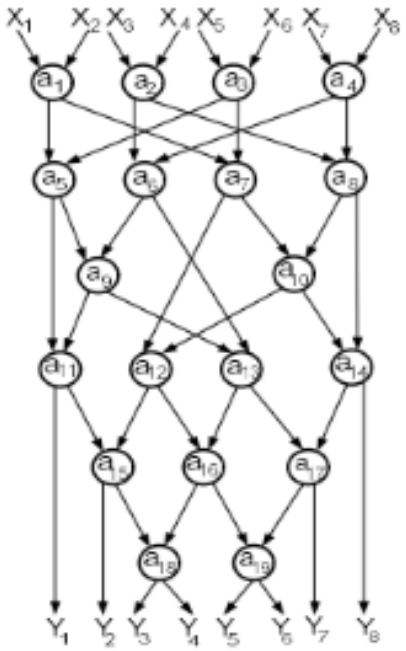


Рис. 8. Граф алгоритму сортування 8-ми значень методом Бетчера.

На виходах алгоритму ($y_1,\ldots,y_8$) утворюється відсортована послідовність масиву двійкових чисел у порядку спадання.

Обчислювальні затрати на реалізацію алгоритму сортування двійкових чисел методом Бетчера для N вхідних значень складають - $0,48N\ln^2 N$ операцій "порівняти і переставити".

Для алгоритму Бетчера загальна кількість вентилів буде визначатися за допомогою формули (1).

$$W_{вент.} = 0,48N\ln^2 N \times N_{вент.ел.}, \quad (1)$$

де, N - кількість вхідних даних, $N_{вент.ел.}$ - кількість вентилів для реалізації однієї операції "порівняти і переставити".

Швидкодія даного алгоритму сортування двійкових чисел методом Бетчера визначається найбільш довгим маршрутом розповсюдження сигналу і в нашому випадку складає - $(\frac{1}{2}[\log_2 N]([\log_2 N+1]))$ операцій "порівняти й переставити" (max/ min).

Розділ 4. Структура спеціалізованої системи сортування двійкових даних.

На рис. 9 зображено загальну структуру спеціалізованої системи сортування двійкових даних, яка складається з вхідного інтерфейсу, операційного автомату, вихідного інтерфейсу та пристрою керування.

Рис. 9. Загальна структура системи.

Вхідний інтерфейс проектується стандартним і складатиметься із вхідних регістрів, які по черзі будуть записуватися числа по шині даних (ШД) на основі сигналів керування, що генерує керуючий пристрій по шині керування (ШК). Операційний автомат виконує елементарні операції над операндами (впорядковує дані) , які приходять з вхідного інтерфейсу та видає результати обробки. Виконання даних елементарних операцій операційний автомат виконує на основі відповідних сигналів керування, які генерує пристрій керування. Вихідний інтерфейс реалізований для взаємодії з накристальною шиною wishbone. Пристрій керування реалізований на основі автомату Мура.

На рис. 10. зображена функціональна схема спеціалізованої системи сортування двійкових даних, яка складається з вхідного інтерфейсу, операційного автомату, вихідного інтерфейсу та пристрою керування.

Інформаційний тракт складається з вхідного інтерфейсу, рекурсивного пристрою сортування та вихідного інтерфейсу. На вхід 8-ми розрядної шини даних D_in, після встановлення сигналу I_ack (підтвердження надходження вхідних даних) в активний стан та подачі сигналу I_clk (тактування вхідних даних) по черзі поступають вісім однобайтових чисел, які надходять до вісьмох 8-ми розрядних регістрів (R1, R2, R3, R4, R5, R6, R7 та R8) і на основі сигналів дозволу запису (en_r1, en_r2, en_r3, en_r4, en_r5, en_r6, en_r7 та en_r8), які генерує керуючий пристрій послідовно записуються в дані регістри. Передача вхідних даних здійснюється побайтно. На рис.11 зображена часова діаграма протоколу передачі вхідних даних.

На даній діаграмі вхідні числа передаються пакетами по 8-ми бітовій шині даних (d_in). На виході регістрів дані об'єднуються в 64-ох розрядний пакет та поступають на вхід операційного автомату для обробки. Функції операційного автомату виконує рекурсивний пристрій сортування (рис. 12), який складається з шести мультиплексорів (М1, М2, М3, М4, М5, М6), які керуються сигналами s_m21_1, s_m_21_2, s_m7_1, s_m6_1, s_m3_1, s_m2_1, двох демультиплексорів (Дм1, Дм2), що керуються сигналами sdm1_3 та sdm1_2, одного операційного блоку, який виконує операцію порівняння та переставлення двох чисел (рис. 13) та чотирнадцяти регістрів, які затримують вхідні та вихідні дані на потрібну кількість тактів.

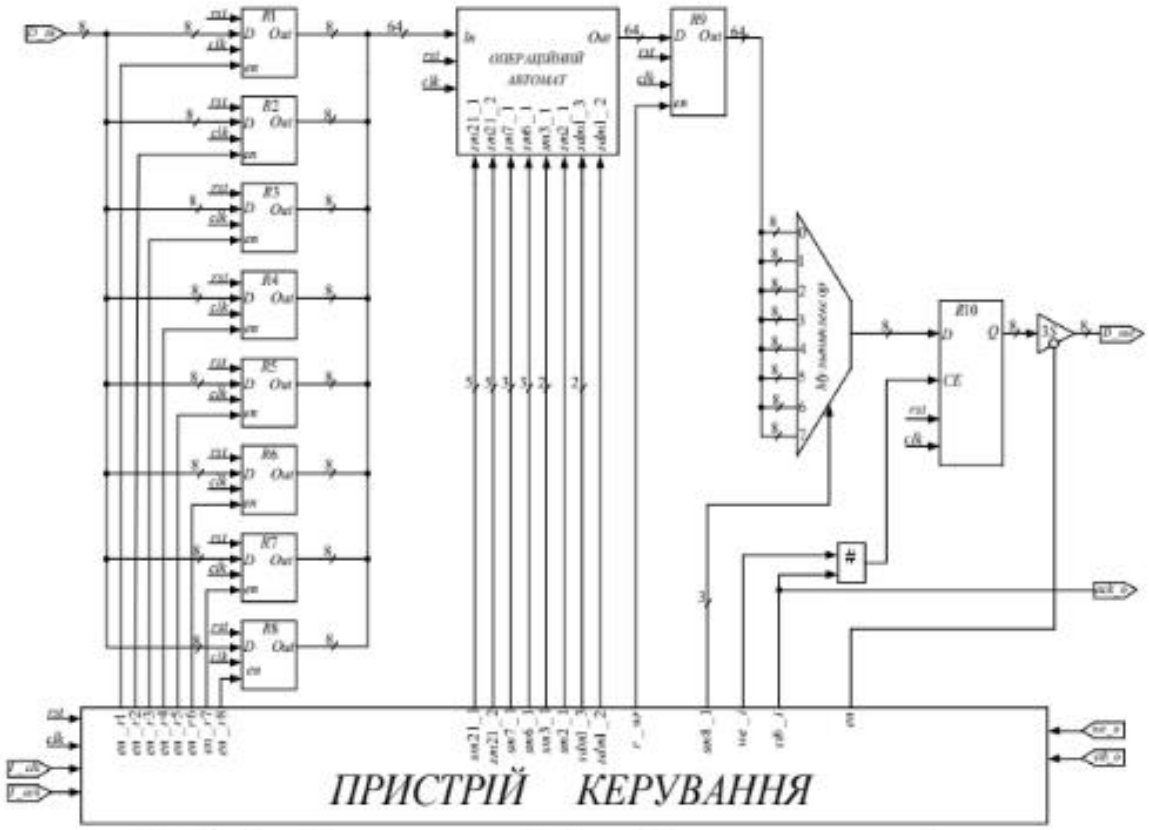


Рис. 10. Функціональна схема спеціалізованої системи сортування двійкових даних.

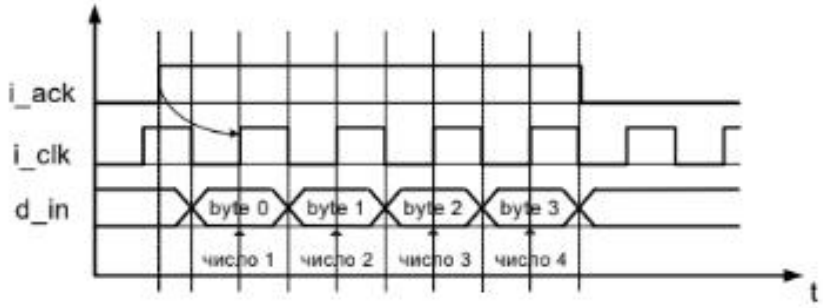


Рис. 11. Часова діаграма протоколу передачі вхідних даних.

На рис. 13 зображена структурна схема операції "порівняти й переставити" описуваного алгоритму сортування чисел методом "бульбашки", слід зауважити, що дана операція є базовою для великої кількості алгоритмів сортування [6,14].

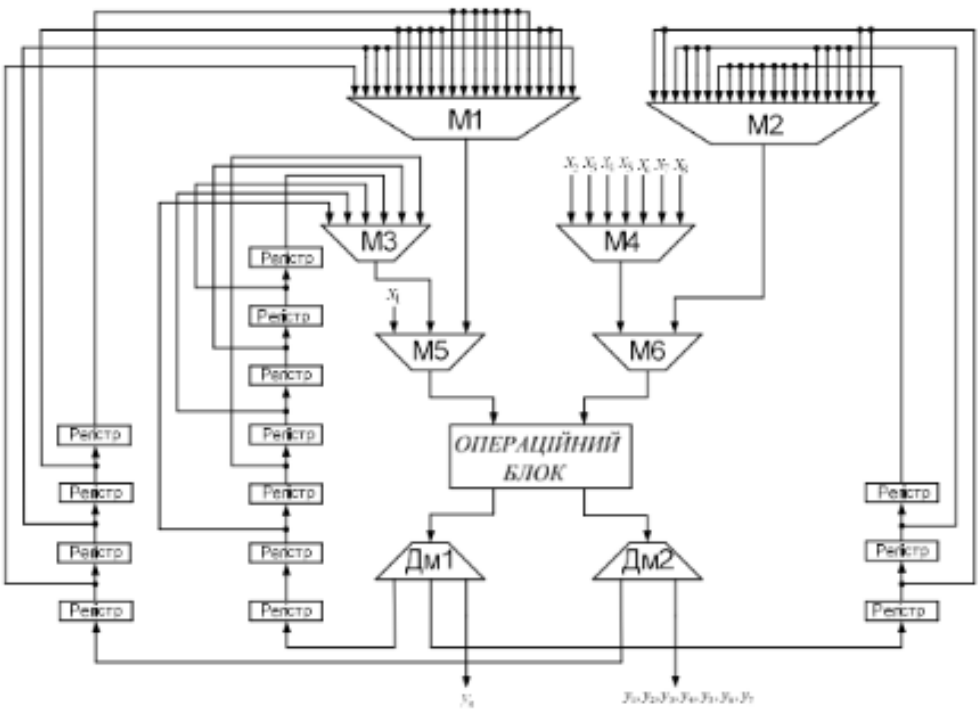


Рис. 12. Структура операційного автомату системи.

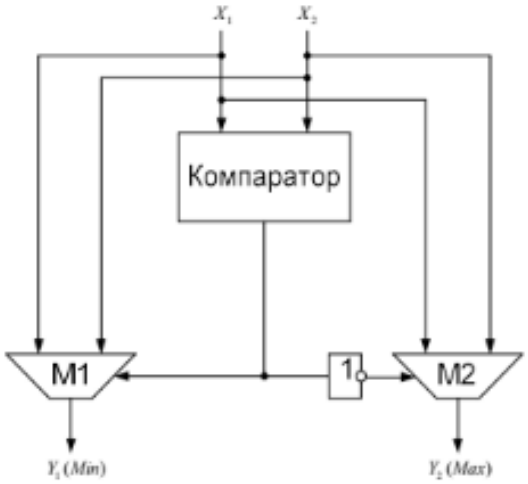


Рис. 13. Структурна схема операції "порівняти й переставити".

Представлена структурна схема складається з компаратора на входи якого подаються два числа для порівняння, та двох мультиплексорів ($M1$ та $M2$), які керуються вихідним сигналом компаратора та одночасно видають на свої виходи мінімальне (y_1) та максимальне (y_2) число.

Операційний автомат побудований на основі рекурсивного просторово-часового графу [6,13,14] алгоритму сортування 8-ми чисел методом "бульбашки", який зображено на рис. 14.

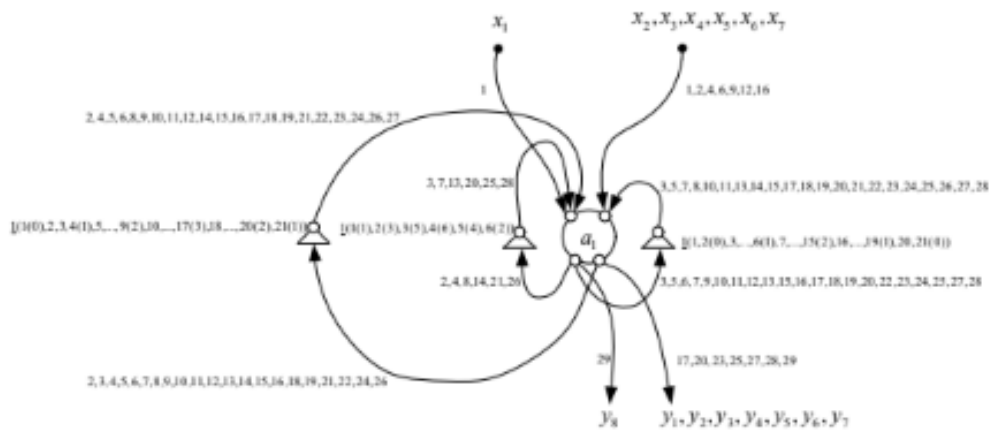


Рис. 14. Структура рекурсивного просторово-часового графу алгоритму сортування 8-ми чисел методом "бульбашки".

Для побудови рекурсивного просторово-часового графу застосовано методику просторово-часових графів [6]. Рекурсивний ПЧГ складається з однієї вершини, яка має два вхідні та два вихідні вузли. Біля дуг даного графу зображені числові такти під час яких на вхідні вузли поступають основні дані та проміжні результати, які надходять з вихідних вузлів [6,13-17]. Трикутними елементами позначаються елементи затримки, які затримують послідовність проміжних результатів на потрібну кількість тактів. Часова затримка та пропускна здатність для рекурсивного просторово-часового графу алгоритму сортування методом "бульбашки" співпадають і складають 29 тактів.

Для перетворення елементів даного графу в апаратуру, слід зазначити, що вершині графу в апаратному виконанні вілповідає арифметико-логічний пристрій або операційний блок, вхідним вузлам, які виконують функцію пропускання необхідних даних в певний момент часу на вхід операційного блоку, апаратурно відповідають мультиплексори а для вихідних вузлів, які виконують функцію видачі проміжних та кінцевих результатів, відповідають в аппаратному виконанні демультиплексори. Елементам затримки рекурсивного просторово-часового графу в аппаратному виконанні відповідають регістри, які забезпечують необхідну затримку вхідних та вихідних даних.

На вхід операційного автомату поступають вісім однобайтових цілих числа, які послідовно в часі обробляються одним операційним блоком порівняння та перестановки. Після обробки вхідних чисел операційним блоком на виході операційного автомату ми отримаємо відсортовану в порядку спадання послідовність восьми однобайтових значень.

На структурній схемі операційного автомату мультиплексори та демультиплексори забезпечують подачу та видачу даних та проміжних результатів на вхід одного операційного блоку, який послідовно в часі виконує 28 разів операцію порівняння та переставлення двох значень. Зрозуміло, що мультиплексори та демультиплексори мають сигнали керування, що не зображено на даній схемі, які генерує пристрій керування. Регістри, які забезпечують необхідну затримку вхідних та вихідних даних працюють в режимі постійного переміщення даних. Слід зауважити, що для одночасної видачі кінцевого результату роботи, потрібно затримати певні значення на другому виході демультиплексора Дм2 на певну кількість тактів, щоб всі вісім значень одночасно були на виході.

На виході рекурсивного пристрою сортування при активності сигналу r_wr дані записуються до 64-х розрядного регістру (R9). З регістру (R9) дані поступають на мультиплексор, який має вісім 8-ми розрядних входів та трьох розрядний вхід керування. Керування мультиплексором здійснюється за допомогою сигналу sm8_1, який генерує пристрій керування. З виходу мультиплексора дані поступають до 8-ми бітового регістру, який реалізує 8-ми бітовий wishbone (slave) вихідний порт при типі з'єднання точка-точка та циклі одиночного читання. Даний регістр складається з D-тригера з асинхронним скидом та двох входової схеми виключного АБО. Оскільки використовується цикл одиночного читання то на виході з 8-бітового регістра дані поступають на трьох становий елемент,

який на основі сигналу дозволу en, який генерує керуючий пристрій, по черзі видає дані на вихідну шину даних D_out. Пристрій керування реалізований за допомогою автомату Мура.

Розділ 5. Моделювання спеціалізованої системи сортування двійкових даних.

На рис. 15. зображена діаграма функціональної симуляції рекурсивного пристрою сортування 8-ми чисел методом "бульбашки".

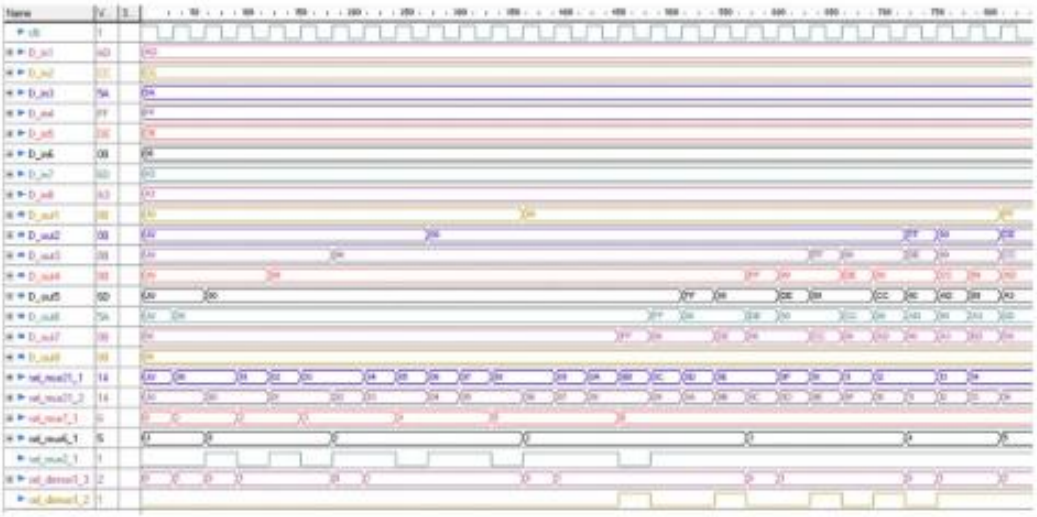


Рис. 15. Функціональна діаграма моделювання рекурсивного пристрою сортування.

На даній діаграмі бачимо формування сигналів керування 6-ма мультиплексорами (sel_mux21_1, sel_mux21_2, sel_mux7_1, sel_mux6_1, sel_mux3_1, sel_mux2_1) та сигналів керування двома демультиплексороми (sel_demux1_3, sel_demux1_2). Кінцевий результат формується на 28-му такті, оскільки рекурсивний пристрій сортування виконує послідовно в часі 28 операцій порівняння та переставлення.

На рис. 16 зображена діаграма функціональної симуляції спеціалізованої системи сортування двійкових даних. На даній діаграмі зображені вхідні та вихідні сигнали, які генерує пристрій керування та сигнали інформаційного тракту проектованої системи.

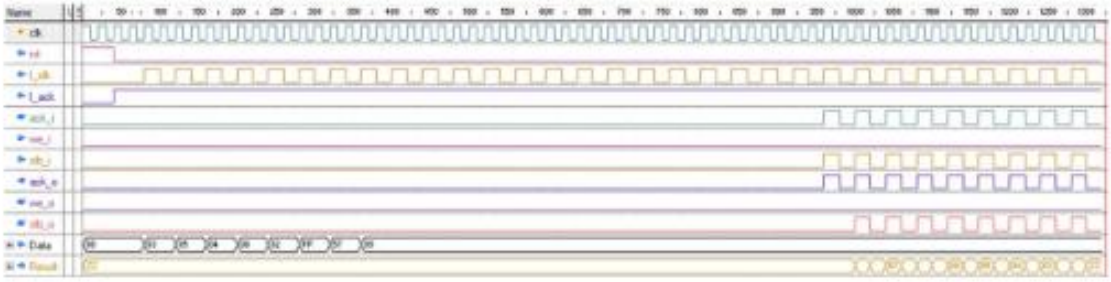


Рис. 16. Діаграма функціональної симуляції роботи спеціалізованої системи сортування двійкових даних.

На діаграмі можна бачити, записані до вхідних регістрів 8-ми розрядні дані (Data), коли сигнали дозволу регістрів (en_r1, en_r2, en_r3, en_r4, en_r5, en_r6, en_r7, en_r8) є стробуючими (знаходяться в стані лог.'1'). З виходів даних регістрів 8-ми розрядні дані з'єднуються в 64-ох розрядний пакет і подаються на операційний автомат для обробки рекурсивним пристроєм сортування. Після обробки результати записуються в 64-ох розрядний регістр на виході операційного автомата та потрапляють по

байтно на 8-ми входовий мультиплексор. З виходу мультиплексора дані поступають на 8-ми розрядний регістр, який складається з тригера з асинхронним записом та двох входового елемента додавання по модулю 2, для взаємодії за правилами шини wishbone для циклу одноразового читання/запису. В зв'язку з цим master і slave формують сигнали дозволу надходження вхідних та вихідних даних (ack_i, ack_o), сигнали стробування (stb_i, stb_i) та сигнали дозволу запису (we_i, we_o), для почергової видачі результатів роботи на 8-ми розрядну шину вихідного інтерфейсу.

На рис. 17 зображена функційна схема інформаційного тракту спеціалізованої системи сортування двійкових даних.

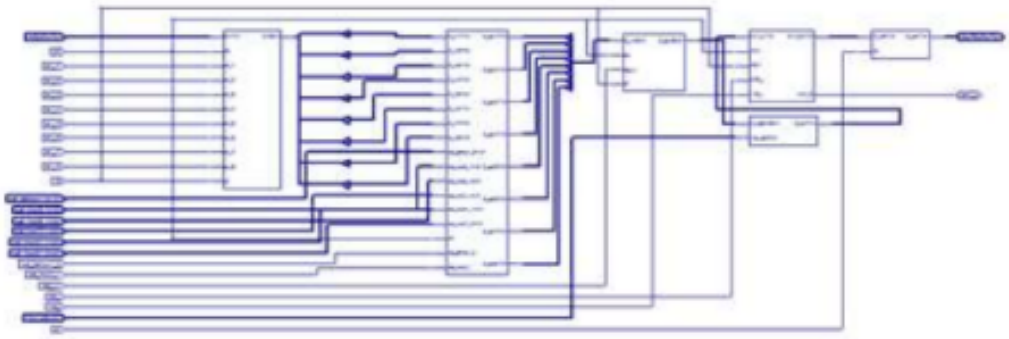


Рис. 17. Функційна схема інформаційного тракту спеціалізованої системи сортування двійкових даних.

Функційна схема спеціалізованої системи сортування двійкових даних згенерована на основі VHDL-коду в САПР Xilinx Vivado [18,19]. Імплементування даної схеми проводився на ПЛІС фірми Xilinx. В якості кристалу був вибраний кристал XC100T сімейства Artix7 і нижче подані результати швидкодії та затрат обладнання.

Швидкодія спеціалізованої системи сортування двійкових даних складає: 244МГц (період – 42,983нс).

Затрати обладнання спеціалізованої системи сортування двійкових даних складають: 6834 з 15850 наявних.

В табл. 1 приведені результати швидкодії та затрат обладнання на ПЛІС фірми Xilinx (сімейство Artix7, кристал XC7A100T) для конвеєрного і рекурсивного пристрою сортування 8-ми чисел методом "бульбашки", дані числа є 8-ми розрядними, а також наведені результати для завершеної структури НВІС.

Таблиця 1.

Порівняння затрат обладнання та швидкодії на ПЛІС фірми Xilinx.

Характеристики	Конвеєрний пристрій сортування	Рекурсивний пристрій сортування	Завершена структура спеціалізованої системи
Швидкодія (МГц)	398	145	244
Кількість таблиць відповідності, (LUT)	4045 з 15850 наявних	1249 з 15850 наявних	6834 з 15850 наявних

Порівнюючи приведені в табл.1. показники швидкодії та затрат обладнання для конвеєрного та рекурсивного пристроїв сортування бачимо, що конвеєрний пристрій має у 3 рази більшу швидкодію ніж рекурсивний пристрій, а по затратах обладнання рекурсивний пристрій має найменші витрати. Повністю завершена структура спеціалізованої системи функціонує на частоті 244 Мгц і на її реалізацію потрібно 6834 таблиць відповідності.

Висновки. Проведено огляд та дослідження паралельних та послідовних методів сортування двійкових даних. Визначено, що паралельні алгоритми сортування мають покращену швидкодію, проте значні апаратні затрати для побудови структури операційних пристроїв. Побудовано структуру

спеціалізованої системи сортування двійкових даних на базі рекурсивного просторово-часового графу. Описано компоненти спеціалізованої системи сортування двійкових даних на мові VHDL. Проведено моделювання структури рекурсивного пристрою сортування та синтез на ПЛІС фірми Xilinx. Визначено, що рекурсивна спеціалізована система має накращі характеристики по затратах обладнання у порівнянні з відомими реалізаціями пристроїв сортування.

Список використаних джерел:

1. Мельник А.О. Спеціалізовані комп'ютерні системи реального часу, - ДУ "ЛП" , Львів, 2002. - 60с.
2. Мельник А.О. Архітектура комп'ютера. Наукове видання. – Луцьк: Волинська обласна друкарня, 2008. – 470с.
3. Спеціалізовані комп'ютерні технології в інформатиці / за загальною редакцією Я.М. Николайчука. – Тернопіль: Видавництво Бескиди, 2017. – 919 с.
4. Valeriy Zadiraka, Yaroslav Nykolaichuk. Methods of effective protection of information flows.- Ternopil: Terno-graf, 2014. -308 p.
5. Мельник А.О. Пам'ять із впорядкованим доступом. Монографія. Львів: Видавництво Львівської політехніки, 2014. – 296 с.
6. Ерметов Ю.О. Проектування обчислювальних структур на основі просторово-часових графів // Вісн. ХНУ. – 2006. - №4. С. 172-177
7. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to algorithms: - third edition. Massachusetts Institute of Technology, 2009. – 1313 p.
8. Knuth, D.E. The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1. Addison-Wesley Professional, 912, 2014.
9. Srikanth Alaparthi, Kanupriya Gulati, Sunil P.Khatri (2009) Sorting Binary Numbers in Hardware – A Novel Algotiyhm and its Implementation // IEEE International Symposium on Circuits and System, Taipei, Taiwan. pp. 2225-2228
10. Грига В. М. Просторово-часове перетворення паралельних алгоритмів сортування / В. М. Грига // Вісник "Комп'ютерні системи та мережі". – Львів: Національний університет "Львівська політехніка", 2011. – № 717. – С. 31–35
11. Грига В. М. Просторово-часове перетворення алгоритму сортування чисел методом Бетчера / В. М. Грига // Науково-технічний журнал "Радіоелектронні і комп'ютерні системи" – Харків: Національний аерокосмічний університет ім. М.Є. Жуковського "Харківський авіаційний інститут", 2012. – №6(58) – С. 84–87
12. V. Hryha, B. Dzundza, S. Melnychuk, I. Manuliak, A. Terletsky, M. Deychakivski Design of various operating devices for sorting binary data // Eastern-European Journal of Enterprise Technologies – Kharkiv, Ukraine, 2023. – V 4/4 (124), P. 6-18.
13. Грига В. М. Побудова та аналіз рекурсивних просторово-часових графів / В. М. Грига // Вісник "Комп'ютерні системи та мережі". – Львів: Національний університет "Львівська політехніка", 2007. – № 603. – С. 31–35.
14. Грига В. М. Методи побудови рекурсивних пристроїв сортування на основі просторово-часових графів / В. М. Грига // Науково-технічний журнал "Радіоелектронні і комп'ютерні системи" – Харків: Національний аерокосмічний університет ім. М.Є. Жуковського "Харківський авіаційний інститут", 2009. – №6(40) – С. 209–212.
15. V. Gryga, Y. Nykolaichuk, N. Vozna, B. Krulikovskyi Synthesis of a microelectronic structure of a specialized processor for sorting an array of binary numbers // Perspective technologies and methods in MEMS design. Proceedings of XIIIth International Conference. MEMSTECH 2017. – Lviv-Svalyava, Ukraine, 2017. – P. 170-173.
16. Грига В. Методи та апаратні засоби сортування масивів двійкових чисел / В. М. Грига, Я. М. Николайчук // Матеріали Всеукраїнської конференції з міжнародною участю "Сучасні комп'ютерні інформаційні технології" (АСІТ'2017) – Тернопіль, Україна, 2017. – С. 287-290.
17. V. Gryga, Y. Nykolaichuk, N. Vozna, A. Voronych, B. Krulikovskyi Development and Research of Conveyor Structures of Binary Sorting Algorithms // Advanced Computer Information Technologies. International Conference. ACIT 2018. – Ceske Budejovice, Czech Republic, 2018. – P. 123-127.
18. V. Gryga, M. Karpinski, R. Kochan, A. Voronych, I. Kogut Design and research of operational and pipelined binary number sorting devices // 18th International Multidisciplinary Scientific Geoconference & Expo SGEM 2018. – Albena, Bulgaria, 2018. – P. 279-292.
19. V. Gryga, Y. Nykolaichuk, L. Nykolaichuk, N. Vozna, H. Klym Structuring of Algorithms for Data sorting and New Principles of Their Parallelization // Advanced Computer Information Technologies. International Conference. ACIT 2019. – Ceske Budejovice, Czech Republic, 2019. – P. 205-208.

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016260050

Дата перевірки:
17.05.2024 22:00:38 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
17.05.2024 22:06:56 EEST

ID користувача:
100005591

Назва документа: **Горбач_Спеціалізовані системи сортування даних на базі ПЛІС**

Кількість сторінок: **96** Кількість слів: **18149** Кількість символів: **138084** Розмір файлу: **2.47 MB** ID файлу: **1016048356**

3.95%
Схожість

Найбільша схожість: **0.91%** з Інтернет-джерелом (http://www.parallax.com/dl/docs/prod/indl/StampPLC.pdf)

3.65% Джерела з Інтернету	192	Сторінка 98
0.93% Джерела з Бібліотеки	48	Сторінка 100

0.06% Цитат

Цитати	2	Сторінка 101
Посилання	1	Сторінка 101

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	30

Fri May 17 20:48:24 EEST 2024, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 8%**

	Документ		Сумарний збіг по Базі Даних	
ID: 126534 Назва: МКР Спеціалізовані системи сортування даних на базі ПЛІС Додано в БД: 2024-05-17 Автора: Горбач Г.С. Керівники: Грига М.В. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	120358	988	1417 (1%)	23 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач: Горбач Галина Степанівна

Тема: Спеціалізована система сортування даних на базі ПЛІС

Спеціальність: 123 «Інформаційні системи та технології»

Обсяг кваліфікаційної роботи:

Кількість сторінок записки 80 с.

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи магістра є проектування і дослідження спеціалізованої системи сортування даних на базі ПЛІС.

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

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проведено огляд відомих методів сортування даних. Досліджено відомі рішення та засоби в цій сфері. У другому розділі запропоновано модель системи сортування даних на базі ПЛІС з використанням просторово-часових графів. Ця модель базується на ідеї використання ПЛІС для розподілу та оптимізації обчислювальних завдань, що дозволяє використовувати паралельність та високошвидкісні можливості цих схем. У третьому розділі запропоновано удосконалену систему сортування даних на базі ПЛІС. Ця система відрізняється від попередніх версій завдяки вдосконаленню алгоритмів сортування, оптимізації використання ресурсів ПЛІС, а також підвищенню швидкості та надійності обробки даних. У четвертому розділі запропоновано спеціалізовану систему сортування даних на базі ПЛІС, включаючи детальний огляд її архітектури та визначення основних вимог.

4. Позитивні сторони роботи: Завдяки ретельному аналізу та використанню спеціалізованих функцій ПЛІС, система сортування працює швидше та ефективніше, що дозволяє оптимізувати час обробки даних.

5. Негативні сторони роботи: В роботі присутні певні логічні помилки щодо опису спеціалізованої системи сортування даних на базі ПЛІС.

6. Оцінка графічного оформлення та пояснювальної записки роботи: –

7. Відгук про роботу в цілому: В загальному робота виконана на задовільному рівні.

8. Інші зауваження:

9. Оцінка кваліфікаційної роботи:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи вважаю, що робота заслуговує оцінки «задовільно» ( )

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Мартинюк Валерій Володимирович, доц. каф. АКІТ та Р

"22" 05 2024р.

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

Горбач Галини Степанівни
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-22-2

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

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

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

18 квітня 2024
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

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

Назва: Спеціалізована система сортування даних на базі ПЛІС

Автор: Горбач Галина Степанівна

Спеціальність: 123 – Комплютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Грига В.М. к.т.н., доцент

Після аналізу звіту подібності зроблено такий висновок:

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

Підтвердження:

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

1) запозичення розміщені в розділах є збіг зі звітом з науково-дослідної практики автора Горбач Галичи "Спеціалізована система сортування даних на базі ПЛІС", який було додано в репозитраій ХНУ 21 березня 2024 року;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
4) У якості запозичень у деяких місцях системою зафіксовано послідовності чотирирозрядних двійкових кодів та поєднання латинських символів українськими скороченнями та індексами у формулах. Такі модифікації не можуть бути розглянуті як переробка тексту, вони відносяться до використання спеціальних символів та форматування, яке є типовим для математичних та технічних виразів.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості Unicheck, складає 3.95% і адресується до 240 першоджерела, та системою Anti-Plagiarism складає 1% що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В.М. Грига

Гарант ОП _____ О.С. Савенко

Завідувач кафедри КПС _____ Т. О. Говорущенко