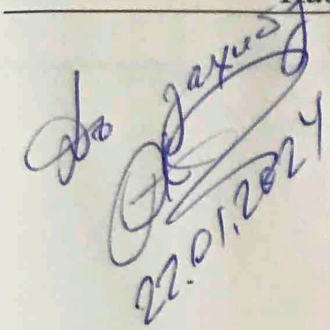
Міністерство освіти і науки України
Український державний університет науки і технологій

«Комп'ютерні технології і системи»
(назва факультету)

Кафедра «Електронні обчислювальні машини»
(повна назва кафедри)



Пояснювальна записка
до кваліфікаційної роботи
магістра
(ступінь вищої освіти)

на тему: Використання ПЛІС в багатопроцесорних системах

за освітньою програмою Комп'ютерна інженерія
зі спеціальності: 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

Виконав: студент групи: КС2226

(підпис студента) / Максим ВАНІН /
(Ім'я ПРІЗВИЩЕ)

Керівник: (підпис) /доцент Володимир ШАПОВАЛОВ /
(посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: (підпис) /доцент Володимир ШАПОВАЛОВ /
(посада, Ім'я ПРІЗВИЩЕ)

Консультанти:

(назва розділу) (підпис) / /
(посада, Ім'я ПРІЗВИЩЕ)

(назва розділу) (підпис) / /
(посада, Ім'я ПРІЗВИЩЕ)

(назва розділу) (підпис) / /
(посада, Ім'я ПРІЗВИЩЕ)

(назва розділу) (підпис) / /
(посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент (підпис)

Дніпро – 2024 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
(faculty)

Department Electronic computing machines
(department)

Explanatory Note
to Master's Thesis
master
(higher education degree)

on the topic: The use of FPGAs in multiprocessor systems

according to educational curriculum Computer engineering

in the Speciality: 123 Computer engineering
(speciality and its code)

Done by the student of the group: KS2226 / Maksim VANIN /
(name, surname)

Scientific Supervisor: / associate professor Volodymyr SHAPOVALOV /
(position, name, surname)

Normative controller : / associate professor Volodymyr SHAPOVALOV /
(position, name, surname)

Supervisors

(Chapter title heading)	/ / (position, name, surname)
(Chapter title heading)	/ / (position, name, surname)
(Chapter title heading)	/ / (position, name, surname)
(Chapter title heading)	/ / (position, name, surname)

Dnipro – 2024

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерні технології і системи
Кафедра: ЕОМ
Рівень вищої освіти: Другий (магістерський)
Освітня програма: Комп'ютерна інженерія
Спеціальність: 123 Комп'ютерна інженерія
(шифр та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ЕОМ
_____ Ігор ЖУКОВИЦЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)
Дата 16.11.2023

З А В Д А Н Н Я

на кваліфікаційну роботу магістра
(ступінь вищої освіти)

студенту Ванін Максим Віталійович
(Прізвище, Ім'я По батькові)

1. Тема роботи: Використання ПЛІС в багатопроцесорних системах

Керівник роботи: Шаповалов В. О, к.т.н., доцент
(Прізвище, Ім'я, По батькові, науковий ступінь, вчене звання)

затверджені наказом від "21" квітня 2023 р. № 332 ст

2. Строк подання студентом роботи: 17.01.2024 р.
3. Вихідні дані до роботи: Проаналізувати багатопроцесорні системи на ПЛІС та реалізувати операційну частину чотирипроцесорної системи на ПЛІС для виконання арифметичних операцій над комплексними числами.
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
4.1 Аналітична частина: Огляд специфіки і характеристики багатопроцесорних систем; Огляд напрямків використання ПЛІС в багатопроцесорних системах; Огляд ПЛІС і засобів автоматизованого проектування для багатопроцесорних систем.
4.2 Основна частина: Призначення, основні функції, структура і VHDL-опис операційної частини багатопроцесорної системи, яка розробляється; Побудова функціональної схеми багатопроцесорної системи; Моделювання і аналіз розробленої багатопроцесорної системи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Графічна частина не передбачена

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Завдання видав: (підпис консультанта, дата)	Завдання прийняв: (підпис студента, дата)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	16.10.2023 – 17.10.2023	1%
2	Характеристика і класифікація багатопроцесорних систем	18.10.2023 – 26.10.2023	10%
3	Огляд напрямків використання ПЛІС в багатопроцесорних системах.	27.10.2023 – 04.11.2023	10%
4	Огляд ПЛІС і засобів автоматизованого проектування для багатопроцесорних систем	05.11.2023 – 12.11.2023	9%
5	Призначення, основні функції і структура багатопроцесорної системи	13.11.2023 – 21.11.2023	21%
6	Побудова функціональної схеми і VHDL-опис багатопроцесорної системи	22.11.2023 – 29.12.2023	26%
7	Моделювання і аналіз розробленої системи	30.12.2023 – 09.01.2024	9%
8	Висновки	10.01.2024 – 15.01.2024	4%
9	Підготовка презентації та доповіді	16.01.2024 – 22.01.2024	10%
10	Подання кваліфікаційної роботи до кафедри		
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії		

Студент _____ (підпис) Максим ВАНІН (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ (підпис) Володимир ШАПОВАЛОВ (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Робота виконана на 73 аркушах, містить 4 додатки та посилання на список використаних літературних джерел з 11 найменувань. У роботі наведено 32 рисунків та 2 таблиць.

Метою кваліфікаційної роботи є аналіз багатопроцесорних систем на ПЛІС та реалізація операційної частини чотирипроцесорної системи на ПЛІС для виконання арифметичних операцій над комплексними числами.

Поставлена мета досягається розв’язанням таких основних задач:

1) аналіз багатопроцесорних систем;

2) аналіз типів ПЛІС для багатопроцесорних систем;

2) розробка структури та функціональної схеми багатопроцесорної системи;

3) реалізація мовою VHDL операційної частини багатопроцесорної системи на ПЛІС та її дослідження.

Робота над багатопроцесорною системою виконувалася шляхом створення VHDL-коду, синтезу схеми і моделювання в САПР з прив’язкою до конкретної ПЛІС. Під час роботи над проектом використовувався власний комп'ютер та програмне забезпечення від фірми AMD (Xilinx) Vivado Design Suite 2018.2.

Ключові слова: БАГАТОПРОЦЕСОРНІ СИСТЕМИ, ПЛІС, XILINX, MISD, VHDL, SMP, САПР.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

ПЛІС - Програмована логічна інтегральна схема

SISD - Single Instruction Stream/Single Data Stream

MISD - Multiple Instruction Stream/Single Data Stream

SIMD - Single Instruction Stream/Multiple Data Stream

MIMD - Multiple Instruction Stream/Multiple Data Stream

SMP - Симетрична багатопроцесорна система

AMP - Асиметрична багатопроцесорна система

DSP - Процесор цифрових сигналів

MPSoC - Багатопроцесорна система на кристалі

ЦП - Центральний процесор

САПР - Система автоматизованого проектування

HDL - Hardware Description Language

ВСТУП

Протягом багатьох років розробники апаратного забезпечення покладалися на збільшення тактової частоти системи як на спосіб підвищення продуктивності. Однак цей підхід більше не є життєздатним, оскільки такі проблеми, як розсіювання тепла і тепловідводи, стали занадто складними для подолання. У пошуках більш простих способів підвищення продуктивності багатопроцесорні системи стають все більш поширеним рішенням.

Багатопроцесорна система - це система з декількома процесорами, які можуть виконувати кілька процесів одночасно. З розвитком технологій з'явилася можливість інтегрувати цілі багатопроцесорні системи на одному кристалі. Такі системи називаються MPSoC (багатопроцесорні системи на кристалі). MPSoC сьогодні є дуже привабливим рішенням в області вбудованих систем, що дозволяє вбудованим системам виконувати завдання в режимі реального часу і в той же час долати значні обмеження по енергоспоживанню і займаному простору.

У цьому контексті програмовані логічні інтегральні схеми (ПЛІС) стають новою і перспективною платформою для реалізації багатопроцесорних систем без проблем, пов'язаних з прикладними інтегральними схемами (ASIC), що дозволяє швидко створювати прототипи і досліджувати нові архітектури. Однак проектування на мовах опису апаратури HDL займає багато часу, і альтернативою проектуванню на HDL є використання програмних процесорів у ПЛІС для побудови багатопроцесорних систем. Програмні процесори - це конфігуровані процесори, розроблені відповідно до дизайну ПЛІС. Сучасні ПЛІС можуть інтегрувати сотні, тисячі процесорів, що значно збільшує потужність паралельних обчислень.

1 БАГАТОПРОЦЕСОРНІ СИСТЕМИ

1.1 Поняття багатопроцесорних систем

Багатопроцесорні системи базуються на об'єднанні процесорів у спільній області оперативної пам'яті. Ця область називається спільною пам'яттю. Спільною пам'яттю керує спільна операційна система. Це дозволяє швидше обмінюватися інформацією між процесорами, ніж між комп'ютерами в багатокомп'ютерних системах (багатофункціональних), і таким чином, підвищувати загальну продуктивність системи. Іноді це також називають "справжнім" мультипроцесором.

У таких системах, як правило, кількість паралельних процесорів невелика і контролюється центральною операційною системою. Кожен процес обмінюється інформацією через спільну оперативну пам'ять. Це призводить до затримок через між процесорну конкуренцію. При створенні великих багатопроцесорних комп'ютерів (мейнфреймів, суперкомп'ютерів) докладаються великі зусилля для збільшення пропускної здатності оперативної пам'яті. В результаті витрати на апаратне забезпечення зростають в геометричній прогресії, але продуктивність системи не збільшується прямо пропорційно кількості процесорів, а складні способи зменшення міжпроцесорної конкуренції в оперативній пам'яті суперкомп'ютерів серії CRAY X-MP/Y-MP дозволили в чотирипроцесорних конфігураціях систем досягти коефіцієнтів прискорення до 3,5 разів і менше.

В рамках проекту Соломона був розроблений процесор, який міг працювати і виконувати операції над вмістом одного або декількох регістрів, але це було в 1962 році.

Історія розвитку багатопроцесорних обчислювальних систем почалася в 1970-х роках з появою першого суперкомп'ютера CRAY 1. Той самий принцип роботи був застосований у машині ILLIAC IV. Векторні процесори - це процесори, які оперували з векторами за допомогою однієї команди. Такі процесори були використані при розробці суперкомп'ютера SGau-1.

З'явилася нова класифікація комп'ютерів з векторними входами, решту стали називати скалярними. У міру їх розвитку було вирішено, що такі суперскалярні процесори будуть надавати завдання таким чином, щоб процесор сам вибирав, які операції виконувати паралельно. Цю ідею було реалізовано в процесорах Intel i960 та AMD 29050.

Одним з напрямків розвитку архітектури фон Неймана є розпаралелювання потоків. Ця технологія була успішно використана для одночасної багатопотокової обробки SMT, а також для багатопотокової обробки на рівні мікросхеми.

Різниця між цими підходами полягає в тому, що вони не мають однакової концепції потоку: DEC Alpha EV4 21064 був першим багатопотоковим процесором. У цьому процесорі інструкції були розділені на два потоки, і процесор сам вирішував, який з них виконувати, що призвело до збільшення продуктивності на третину.

Першими представниками архітектури SMC (ядро на кристалі) були процесори, призначені для використання в серверах, які являли собою прості тандеми з двох по суті незалежних ядер на одній платі. Крім економії місця, таке рішення може призвести до значної економії енергії, оскільки деякі компоненти є спільними для обох ядер.

Про розпаралелювання в комп'ютерних системах вчені говорять вже багато років. Однак, зі зрозумілих причин, жоден виробник не був готовий інвестувати в цю можливість, доки подолання пов'язаних з цим труднощів, таких як необхідність поділу додатків на потоки, компенсувалося постійним підвищенням продуктивності процесорів. Тому паралельні обчислення залишалися особливістю суперкомп'ютерів аж до появи багатоядерних процесорів.

Хоча ідея переходу на багатоядерні процесори не нова, вона з'явилася приблизно у 2005 році. Ще в 1960-х роках перевагу декількох процесорних ядер над одним процесорним ядром продемонстрував Сеймур Клей, який реалізував

цю ідею в своєму суперкомп'ютері CDC6600. Однак через консервативність дизайнерів цей підхід до проектування централізованого процесора не отримав подальшого розвитку.

Відродження багатоядерності відбулося завдяки інженерам компанії Digital Equipment: Наприкінці 90-х вони почали думати про багатоядерність мікропроцесорів. Саме тоді процесор був оновлений з Alpha 21164 (EV5) до Alpha 21264 (EV6). На той час дослідники змогли встановити дві важливі моделі, які застосовуються до процесорів. По-перше, було виявлено, що для досягнення лінійного зростання продуктивності одноядерних процесорів потрібно квадратичне збільшення кількості транзисторів. По-друге, складність конструкції також зростає нелінійно. Таким чином, сума продуктивності декількох ядер дає таку ж загальну продуктивність, як і одне ядро з невеликою кількістю транзисторів. Основне питання полягає в тому, як об'єднати потужність окремих ядер, що є фундаментальною проблемою багатоядерності: Проект Piranha є відповіддю на цю проблему, яка визначається тим, що кожне ядро має окремі кеші для інструкцій і даних, а процесори з'єднуються за допомогою перемикачів для спільної роботи, утворюючи 8-ядерний процесор.

1.2 Суперкомп’ютери

Сучасний суперкомп'ютер - це величезна машина, що виконує трильйони, а іноді й квадрилліони (число з 15 нулями) обчислювальних операцій.

Хоча певною мірою кожен суперкомп'ютер — не більше ніж послідовність кластерів, що складаються з вузлів материнських плат з процесорами та ядрами — вся ця «кремнієва орава» паралельно обробляє величезні масиви даних, призначені для вирішення вузькоспеціалізованих завдань.

Одні з найпотужніших обчислювальних машин Землі використовують у всьому: від дослідження кращих методів лікування вірусів до розкриття походження Всесвіту.

Фундаментальний показник – швидкість суперкомп'ютера – визначається кількістю арифметичних операцій з плаваючою комою, які пристрій може виконати за секунду. Його зазвичай називають "FLOP".

Однак досягнення в області обчислювальної потужності зробили свій внесок — горезвісний «флоп» тепер має приставку, що означає розряд числа операцій: терафлопси (трильйони) та петафлопси (квадрилліони). Просто щоб розуміти — iPhone 11 приблизно в шість разів швидше за найшвидший суперкомп'ютер у світі на момент 1993 року.

У таблиці 1.1 наведені перші рядки списку — кращі обчислювальні машини світу.

Таблиця 1.1 – Перші рядки списку суперкомп'ютерів

Позиція	Назва	Rmax Rpeak	Потужність	Розміщення, країна, рік
1	Frontier	1194.00 1679.82	22,7	Ок-Ріджська національна лабораторія США, 2022
2	Aurora	585.34 1059.33	24,7	Аргонський національна лабораторія США, 2023
3	Eagle	561.20 846.84		Microsoft США, 2023
4	Фугаку	442.010 537.212	29,9	Інститут фізико-хімічних досліджень Японія, 2020
5	LUMI	379.70 531.51	7,1	EuroHPC JU[en] Європейський Союз, CSC – IT Center for Science[en] Каяані, Фінляндія, 2022
6	Leonardo	238.70 304.47	7,4	EuroHPC JU[en] Європейський Союз, CINECA[en] Казалеккьо-ді-Рено, Італія, 2022
7	Summit	148.600 200.795	10,1	Ок-Ріджська національна лабораторія США, 2018
8	MareNostrum 5 ACC	138.20 265.57	2,6	EuroHPC JU[en] Європейський союз, BSC–CNS[en] Барселона, Іспанія, 2023
9	Eos NVIDIA DGX SuperPOD	121.40 188.65		Nvidia США, 2023
10	Sierra	94.640 125.712	7,4	Ліверморська національна лабораторія США, 2018

1.3 Класифікація багатопроцесорних систем

Існують різні класифікації архітектур комп'ютерних систем. Найбільш вдалою є класифікація М. Флінна.

Вона визначає характеристики архітектури комп'ютерної системи з точки зору потоку інструкцій (команд) і потоку даних. Такий підхід дозволяє віднести архітектуру комп'ютера до одного з чотирьох класів (див. табл. 1.2, рис. 1.1) [1].

Ця класифікація базується не на структурі, а на способі взаємодії інструкцій обчислювальної машини з даними.

Таблиця 1.2 – Класифікація архітектур комп'ютерних систем

Потік команд	Одиночний потік інструкцій Single Instruction	Множинний потік інструкцій Multiple Instruction
Одиночний потік даних Single Data	SISD	MISD
Множинний потік даних Multiple Data	SIMD	MIMD

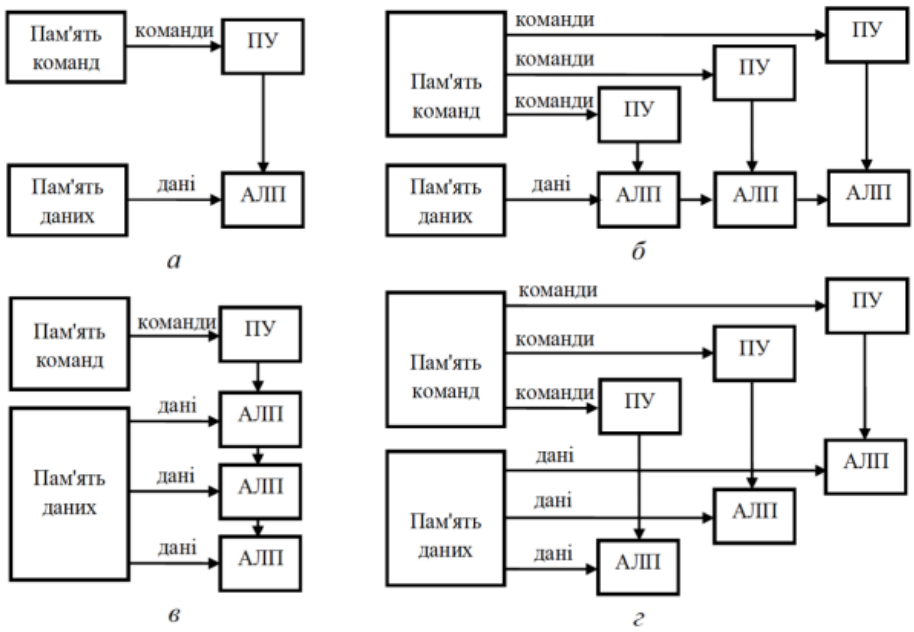


Рисунок 1.1 – Архітектура комп'ютерних систем за Флінном: а – SISD; б – MISD; в – SIMD; г – MIMD

SISD (Single Instruction Stream/Single Data Stream) - єдиний потік інструкцій і даних. Представником цього класу є класичний фоннеймановський комп'ютер. Інструкції обробляються послідовно, і кожна інструкція запускає процес у потоці даних. До цього класу комп'ютерів також належать конвеєрні комп'ютери; деякі фахівці вважають, що до SISD-систем належать також векторно-конвеєрні операційні системи[1].

MISD (Multiple Instruction Stream/Single Data Stream) - кілька потоків інструкцій і один потік даних. Ця архітектура передбачає кілька процесорів, які обробляють один і той же потік даних. Багато дослідників відносять конвеєрні системи до цього класу. Хоча прийнято вважати, що цей клас не використовується в даний час, він корисний для розробки радикально нових концепцій побудови обчислювальних систем [1].

SIMD (Single Instruction Stream/Multiple Data Stream) означає один потік інструкцій і кілька потоків даних. Така архітектура дозволяє виконувати одну арифметичну операцію одночасно над багатьма елементами даних вектора. Типовими прикладами цього класу є системи з процесорною матрицею, де один контролер керує кількома процесорними елементами. Всі процесорні елементи отримують однакові інструкції від контролера і виконують їх над своїми локальними даними. Векторні конвеєрні операційні системи також належать до цього класу, якщо кожен елемент вектора вважається окремим елементом даних[1].

MIMD (Multiple Instruction Stream/Multiple Data Stream) - кілька потоків інструкцій і даних. До цього класу відносяться системи, в яких декілька пристроїв обробки команд об'єднані в єдиний комплекс і кожен з них працює зі своїм потоком команд. Цей клас дуже широкий, оскільки включає всі типи багатопроцесорних систем[1].

1.4 Типи багатопроцесорних систем

1.4.1 Симетрична багатопроцесорна система

Симетрична багатопроцесорна обробка, звана SMP в короткій формі, є типом багатопроцесорної системної операції, яка використовує кілька процесорів однієї конфігурації в одній системі для виконання декількох завдань. Тут усі процесори мають однакову природу та розглядаються однаково(див. рис.1.2).

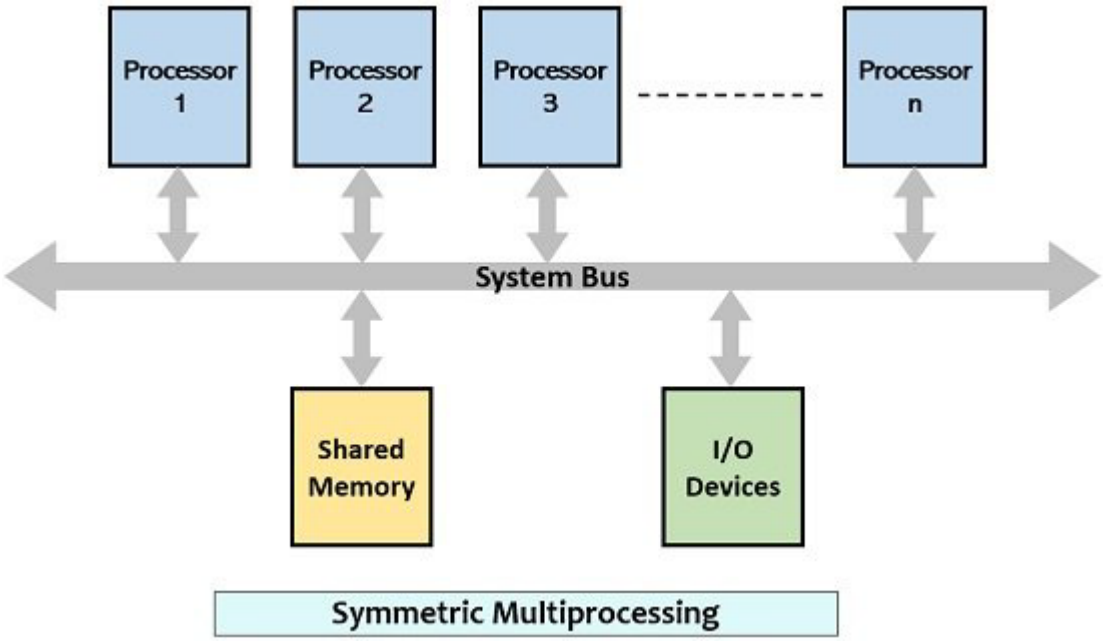


Рисунок 1.2 – Симетрична багатопроцесорна система

У цьому вся типі операційна система існує у пам'яті і доступна всім процесорам системи, але з одночасно. Це означає, що він заснований на методі загальної операційної системи, який називається плаваючим майстер-методом. Якщо для запуску коду ОС потрібно кілька процесорів, то той, який запросив першим, зможе виконати першим, і до цього часу він залишатиметься заблокованим (м'ютексом, що називається) для інших процесорів.

1.4.2 Асиметрична багатопроцесорна система

Асиметрична багатопроцесорна обробка, скорочено AMP, - це ще один тип багатопроцесорної техніки, що складається з декількох процесорів, один з яких

веде себе як ведучий, а інший як ведений. Головний процесор відповідає за виконання операційної системи(див. рис.1.3).

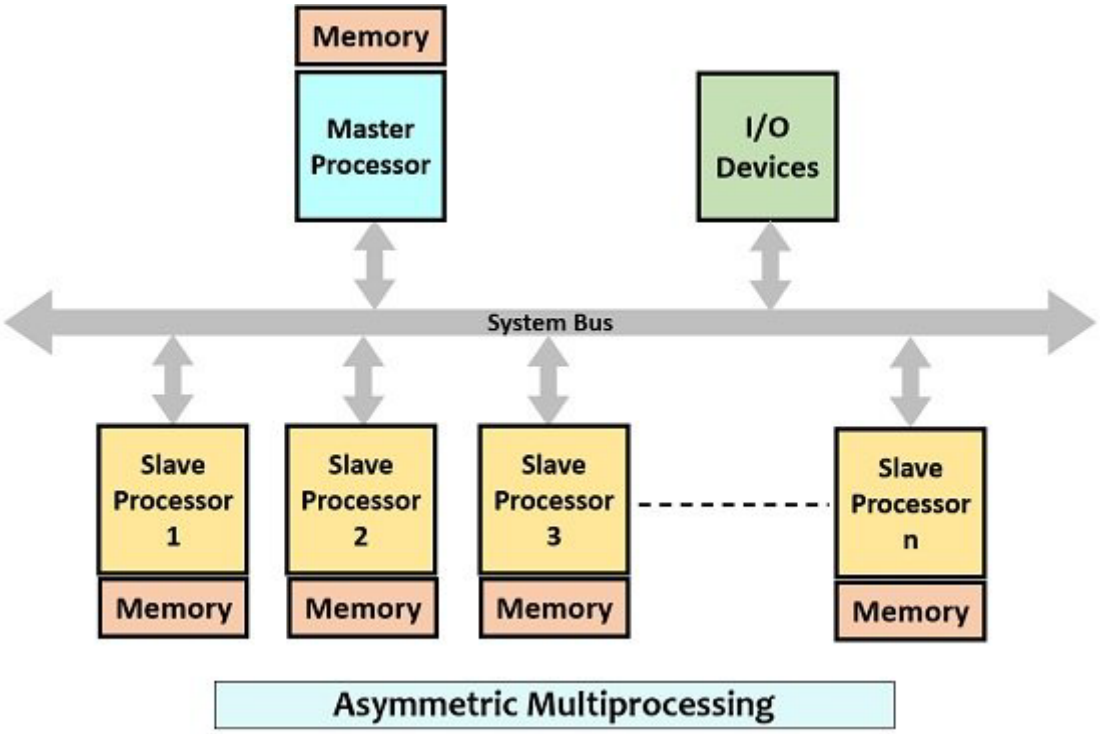


Рисунок 1.3 – Асиметрична багатопроцесорна система

Цей метод дозволяє одночасно виконувати кілька завдань на кількох процесорах. У його пам'яті зберігається структура даних, що містить деталі готових процесів. Тут, якщо головний процесор виходить з ладу, то один із провідних комутаторів перемикається для виконання операцій провідного процесора.

1.5 Проблема когерентності

Основною проблемою багатопроцесорних систем є проблема узгодженості. Кожного разу, коли передається елемент даних, має повертатися збережене значення останнього часового інтервалу. Це основна проблема узгодженості. Однак існує певна проблема затримки, оскільки миттєвість цієї операції зчитування ускладняються тим, що один процесор зчитує інформацію, а інший записує її. Більше того, у випадках, коли декілька процесорів намагаються отримати доступ до однієї і тієї ж ділянки пам'яті з протилежними цілями,

наприклад, для читання і запису, немає гарантії, що після операції читання повертаються правильні дані, які ще не були змінені. Для вирішення цієї проблеми використовуються певні типи пам'яті, а також розпаралелювання пересилань і паралельних операцій. Також має сенс фіксувати момент перезапису даних, щоб правильно синхронізувати обчислення в системі.

Для вирішення вищезазначених проблем необхідно організувати два процеси та гарантувати їх коректну взаємодію[2].

Для виконання процесу, який читає певну область БОС, необхідно переконатися, що відповідна комірка в даний момент не використовується іншим процесом для запису інформації.

Ця особливість пов'язана зі своєчасним визначенням стану процесора. Адже якщо операція завжди повертає лише попереднє значення, це свідчитиме про відсутність узгодженості.

Щоб виконати кілька послідовних операцій запису над коміркою з декількома послідовними інструкціями, необхідно виконати цю операцію таким чином, щоб інші процесори чекали завершення послідовних операцій запису.

Більш складним завданням є забезпечення строгої послідовності операцій запису. Якщо припустити, що послідовність запису не є послідовною, то процесори P1 і P2 будуть записувати один за одним, і в кінцевому підсумку дані, записані першим з цих процесорів, P1, не можна буде відстежити, і тільки дані P2 будуть доступні для читання. Якщо запис не виконується послідовно, виникає ситуація, коли P1 намагається отримати доступ до даних, над якими вже працював P2. Тому перший процесор буде записувати дані з набагато більшою затримкою через латентність. Якщо організація записів, зроблених двома процесорами, неправильна, якщо третій процесор спробує прочитати дані з тієї ж комірки, він може втратити частину даних, а може і не втратити, замість очікуваного результату читання значень, записаних процесорами P1 і P2 послідовно. Навіть якщо інформація, записана другим процесором, буде знайдена, дані, записані першим процесором, вже будуть втрачені. Щоб

гарантувати правильність цих операцій, використовується функція, яка називається серіалізацією. Це послідовне зчитування інформації процесорами.

Існує кілька способів підтримувати узгодженість. Одним з них є необхідність мати певні дозволи на виконання перед початком операції запису інформації. Цей спосіб є найпоширенішим і прийнятий усіма існуючими схемами. Права на запис, надані вищезгаданим протоколом, гарантують, що не існує іншої копії комірки пам'яті, куди буде виконуватися запис, таким чином гарантуючи, що в комірку буде записано саме те, що потрібно. Щоб пояснити, як забезпечується властивість консистентності, опишемо ситуацію, коли операція читання виконується другим процесором одразу після операції запису. Для того, щоб почати записувати дані у вибрану комірку пам'яті, необхідно отримати права доступу. І поки процесор має ці права, він намагатиметься прочитати дані до того, як операція читання завершиться успішно. Як тільки операція запису завершиться, можна починати зчитування даних. Якщо припустити, що інший процесор намагається записати дані в комірку, згідно з цим протоколом, процесор повинен зробити копію і попрацювати з нею перед записом. Процесор з правом доступу скасовує копію іншого процесора і тільки після цього може почати запис. Інший процесор починає працювати з оновленою копією після того, як перший процесор записав дані [2].

Наступний тип протоколу працює за принципом, що всі копії оновлюються тільки тоді, коли в цю комірку пам'яті виконується запис. Існує дві загальноприйняті назви цього протоколу: протокол оновлення та з трансляцією (write update/broadcast protocol). Щоб зменшити кількість вимог до пропускної здатності, виконується перевірка, чи є частина слова в пам'яті. Якщо в результаті пошуку не знайдено жодного збігу, немає необхідності оновлювати кеш.

Іншим способом вирішення проблеми узгодженості є кешування даних. Такі схеми кешування можуть бути реалізовані за допомогою програмного забезпечення, яке керує узгодженістю системи. Копіювання інформації на

груповій основі значно спростило б цей метод, але якщо цей варіант не розглядати, то такий спосіб реалізації можна реалізувати повністю програмно, не враховуючи можливості апаратного забезпечення. Недоліком цього методу є обмеженість програмних засобів підтримки. Тому цей метод підходить для додатків, де паралельна структура створюється на етапі програмної реалізації.

Архітектура машини Cray T3D називається масово-паралельною. Вимоги вищезгаданих типів машин взаємопов'язані. Наприклад, збільшення розміру комп'ютера дозволяє встановити більше процесорів, що, в свою чергу, збільшує довжину каналів передачі інформації. Величина довжини каналу обернено пропорційна тактовій частоті. Тому зі збільшенням довжини каналу тактова частота пропорційно зменшується.

Іншим варіантом при розробці апаратної частини пристрою є побудова системи на основі довідника, що містить інформацію про всі стани комірок пам'яті, готових до роботи з кешем. Цей довідник містить велику кількість інформації, в тому числі і про те, що певні блоки даних замінюються. Деякі типи протоколів містять у своєму кеші деяку загальну інформацію. При розробці апаратного забезпечення пам'ять і каталоги розміщуються на схемі поруч.

Перевагою такої структури є простота і зберігання всієї необхідної інформації в одному місці. Недоліком є обсяг пам'яті, який відповідає її розміру. Однак цей недолік виникає лише в системах з великою кількістю процесорів. Крім того, якщо ця кількість не перевищує кількох сотень, апаратна потужність дозволяє словниковій системі передавати інформацію в достатньому обсязі. Це стає проблемою лише тоді, коли кількість елементів системи перевищує кілька сотень.

2 ОГЛЯД НАПРЯМКІВ ВИКОРИСТАННЯ ПЛІС В БАГАТОПРОЦЕСОРНИХ СИСТЕМАХ

Сьогодні виробники стикаються з новими викликами при розробці виробничих ліній. Ці виробничі лінії або машини повинні відповідати вимогам продукції, яку вони виробляють. По-перше, життєвий цикл продукції стає коротшим, а по-друге, значно зростають вимоги до якості та надійності. Внаслідок цих ринкових тенденцій виробничі машини повинні дозволяти швидко та економічно ефективно вносити зміни та модифікації. Крім того, обсяг даних, які необхідно обробляти під час виробництва, і вимоги до швидкості обробки надзвичайно зростають.

Виробничі машини зазвичай складаються з відносно нерухомих механічних компонентів, таких як корпуси машин і двигуни, та електронних систем управління. Більшість систем керування складаються з електронної апаратної платформи, на якій розміщуються різні програми керування. На відміну від базового обладнання, програми керування зазвичай реалізовані в програмному забезпеченні і тому є дуже гнучкими.

Традиційно розробники мали три основні варіанти реалізації апаратних платформ для керування машинами: цифрові сигнальні процесори (DSP), ПЛІС та ASIC. ASIC, з іншого боку, пропонують найкращу продуктивність, оскільки конструкція може бути оптимально сконфігурована відповідно до вимог програми. Однак рішення на основі ASIC є найдорожчими, частково через те, що кількість виробничих машин відносно невелика порівняно з продуктами, які вони виробляють, а отже, кількість одиниць продукції невелика. Рішення на основі DSP, з іншого боку, є економічно ефективними, але пропонують лише субоптимальну швидкість обробки через виконання програмного забезпечення.

Між цими двома крайнощами ПЛІС пропонують сприятливий компроміс для багатьох застосувань: вони використовуються вже багато років і були добре прийняті інженерами і розробниками, але ще не набули широкого поширення в

мікроконтролерах для конкретних застосувань. Найважливішими причинами цього були ціна і брак знань.

Однак в останні роки розвиток технології ПЛІС досягнув прогресу з точки зору використовуваних ресурсів, кількості вентилів, швидкості обробки, енергоспоживання та роздрібної ціни. Тому сучасні ПЛІС стали цікавою альтернативою традиційним мікроконтролерам і ASIC у вищезгаданих застосуваннях.

Сучасні системи на основі ПЛІС поєднують в собі багато переваг DSP та ASIC. До них відносяться швидкі цикли розробки, висока гнучкість і можливість багаторазового використання, помірна вартість, легка модернізація (з використанням абстрактної мови опису апаратного забезпечення (HDL)) і функціональні розширення (коли ПЛІС не використовується). Крім того, сучасні ПЛІС дозволяють інтегрувати програмне ядро з процесором. Це означає, що ПЛІС можуть використовувати загальну функціональність процесора.

На додаток до нових розробок, рішення на основі ПЛІС також можуть бути з користю використані в існуючих системах, що розвиваються природним шляхом. Важливою характеристикою таких природних систем є те, що з часом початкова апаратна платформа доповнюється різними платами розширення. Як наслідок, такі системи важко підтримувати та розширювати, а також вони схильні до виходу з ладу компонентів. Інша проблема виникає через розпорошеність взаємодіючих функцій через необхідність передачі (великих обсягів) даних. Це ще більше ускладнює систему і знижує її гнучкість та надійність. Крім того, найстаріші компоненти обмежують подальшу модифікацію таких гетерогенних систем. Наприклад, обмежена функціональність вводу/виводу, непідтримувані типи шин та протоколи і т.д. Нарешті, гетерогенні системи покладаються на підтримку та прихильність великої кількості різних постачальників; ПЛІС пропонують можливість усунути деякі з вищезгаданих проблем проектування.

2.1 Програмовані логічні інтегральні схеми

Перші комерційні ПЛІС були випущені компанією XILINX в середині 80-х років минулого століття. На той час пристрої були маленькими, повільними і досить дорогими. Перший пристрій XILINX XC2064 мав 1200 вентилів. Сьогодні ПЛІС пропонують мільйони системних вентилів і працюють на швидкостях до 300 МГц. Ціни на сучасні ПЛІС починаються з менш ніж 10 доларів США, і багато виробників використовують ПЛІС навіть у кінцевих продуктах, таких як мобільні телефони, мережеві рішення та мультимедійні пристрої.

Навіть додатки для автоматизації виробництва можуть отримати вигоду від чудового співвідношення продуктивності та вартості сучасних ПЛІС. ПЛІС представляють тут особливий інтерес, оскільки вони вимагають значних зусиль при розробці невеликої кількості виробничих систем.

Використання ПЛІС у процесі проектування має три очевидні переваги. По-перше, більшість виробників ПЛІС підтримують процес проектування і розробки, надаючи потужні і прості у використанні електронні засоби проектування (EDA), відмінну документацію і персональну підтримку. По-друге, вони не вимагають високих виробничих витрат, характерних для ASIC. По-третє, вони можуть бути модифіковані та скориговані на будь-якій стадії процесу проектування та в польових умовах. Розширені системи розвивають цей останній пункт ще далі, оскільки вони дозволяють динамічно реконфігурувати апаратне забезпечення під час роботи. Такі системи, також відомі як обладнання, що динамічно реконфігурується, наразі перебувають на стадії дослідження. Цей підхід останнім часом викликає великий інтерес, оскільки дозволяє повторно використовувати цінні ресурси.

Розміри ПЛІС зросли настільки, що тепер можна без проблем впроваджувати процесорні ядра, які також можуть бути ядрами DSP. Тому завдання проектування полягає в тому, щоб вирішити, які частини системи, тобто функції, будуть реалізовані безпосередньо на апаратних вентилях або

оброблятися програмним забезпеченням, що працює в програмному ядрі ПЛІС, наприклад, генератором System on Programmable Chip (SOPC) від ALTERA. Сучасні інструменти EDA, такі як XILINX Embedded Development Kit (EDK), значно спрощують процес паралельної розробки апаратного та програмного забезпечення. Це досягається за рахунок автоматичного розпізнавання всіх апаратних компонентів і надання програмних методів доступу до них. Крім того, інструмент розробки виконує автоматичне керування адресами та ресурсами на льоту.

Крім того, розробники можуть використовувати ряд готових компонентів, також відомих як інтелектуальна власність (IB), таких як мережеві інтерфейси, відеоконтролери, контролери пам'яті і т.д., і таким чином допомагають знизити витрати.

Наразі основними виробниками ПЛІС є AMD (придбала XILINX) та Intel (придбала ALTERA).

2.2 Програмні ядра та багатопроцесорні системи

Як згадувалося вище, ПЛІС можуть містити програмне ядро. Процесор програмного ядра можна розглядати як еквівалент мікроконтролера або "комп'ютера на кристалі". Процесори програмного ядра поєднують центральний процесор, периферійні пристрої та пам'ять на одному кристалі. Вони також забезпечують доступ до інших частин мікросхеми, окрім власне ПЛІС, через вбудовані стандартні та користувацькі інтерфейси. Сучасні архітектури комп'ютерів зі скороченим набором інструкцій (RISC) включають:

1. Процесори NIOS та NIOS II від ALTERA,
2. Процесори LEON2 та LEON3 від Gaisler Research,
3. Процесор Microblaze від XILINX.

Деякі з доступних сьогодні ПЛІС можуть реалізовувати декілька процесорів одночасно. Як відомо, багатопроцесорні системи - це правильний спосіб підвищити продуктивність системи і сконцентрувати обчислювальні елементи в одній ПЛІС. Багатопроцесорні системи підтримуються інструментами EDA, тому виготовлення системи може бути завершено за кілька днів. Як правило, більшість виробників підтримують багатопроцесорні системи, надаючи спеціальні апаратні засоби, такі як м'ютекси для одночасного доступу до спільних ресурсів.

2.3 Багатопроцесорна архітектура на основі FPGA

Нова архітектура базується на програмованих пристроях FPGA, як показано на рисунку 2.1 на основі програмованих пристроїв. Важливою частиною нової системи є багатопроцесорна архітектура на базі NIOS II, яка має перевагу в тому, що забезпечує, серед іншого, масштабованість на вимогу та слабко пов'язану конструкцію системи. Обрана ПЛІС містить багатопроцесорну систему на кристалі (MPSoC) та логіку тригерів, попередньо завантажену в хост-комп'ютер. Для забезпечення належної синхронізації системи, тобто своєчасної координації рухів моторів, тригерних подій і даних зображення, MPSoC містить єдине процесорне ядро, яке керує критично важливими за часом операціями. Такий підхід значно зменшує взаємозалежності та обмеження між системою телепортації та хост-комп'ютером. Оскільки логіка тригера знаходиться всередині ПЛІС, скорочуються шляхи зв'язку, а також зменшуються вимоги та структури для розповсюдження сигналу на льоту.

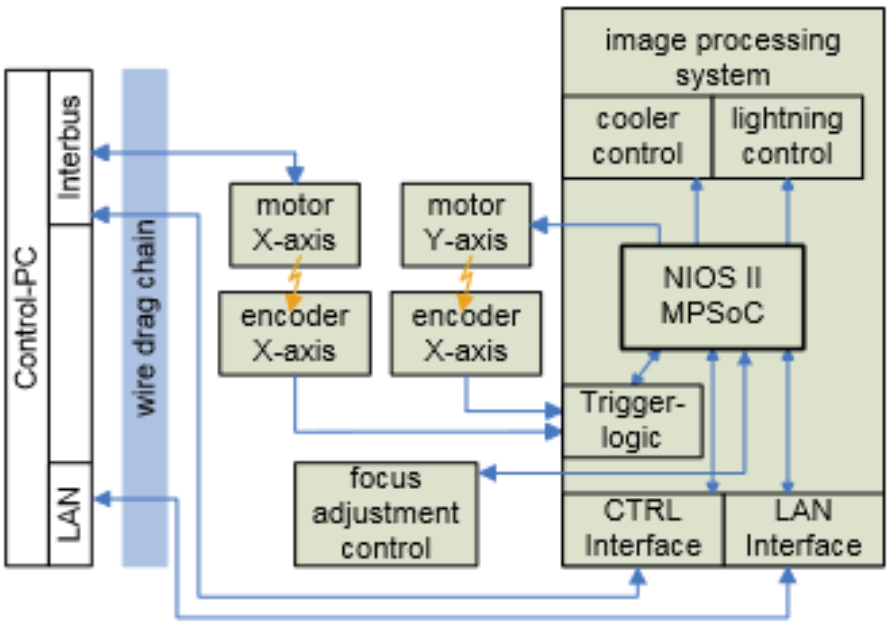


Рисунок 2.1 – Схема модифікованої системи

Якщо розробляється новий процес сповіщення, логіка запуску повинна бути відповідним чином адаптована. Нова логіка замінює стару логіку тригера, вже реалізовану в системі.

Поки зовнішній інтерфейс логіки тригера фіксований, зміна логіки тригера не має побічних ефектів на системній платі; нове завантаження файлу конфігурації обладнання в ПЛІС впливає тільки на внутрішні логічні елементи ПЛІС.

Структура MPSoC, запропонована в цьому документі, являє собою асиметричну архітектуру спільної пам'яті, де всі ядра мають доступ до пристроїв зовнішньої пам'яті, показаних на схемі. Це дозволяє здійснювати зв'язок між різними процесорними ядрами за допомогою глобально доступної конструкції пам'яті. Доступ до спільної пам'яті забезпечується простими операціями, які значно спрощують обмін даними. Це означає, що такий підхід не вимагає особливих накладних витрат, таких як черги повідомлень, обслуговування черг або обробка повідомлень.

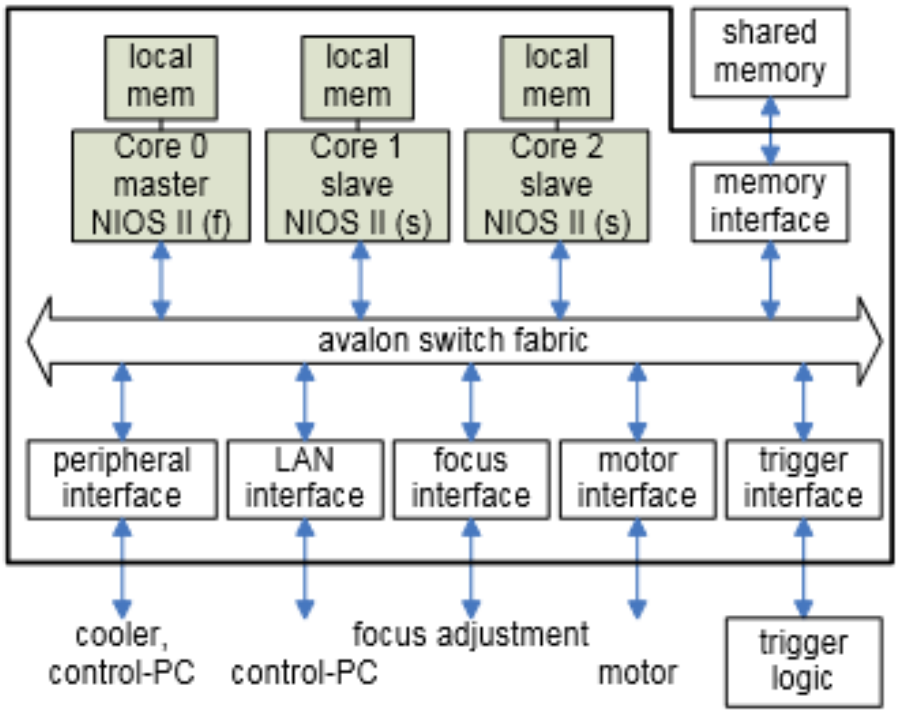


Рисунок 2.2 – Компоненти NIOSII-MPSoC

Спільну пам'ять також можна використовувати для динамічної зміни програмного середовища. Наприклад, залежно від типу використовуваної друкарської форми, ядро 0 може отримувати параметри або цілі розділи програмного забезпечення за допомогою інтерфейсу локальної мережі. Після надсилання повідомлення інший процесор, наприклад, Core 1, може отримати доступ до тих самих даних без необхідності подальшого копіювання чи обробки. Це підвищує гнучкість для різних застосувань, оскільки система може бути адаптована до різних типів пластин, які мають різні характеристики і тому потребують незначних змін під час процесу експонування.

Ядро 0 позначене як головне і відповідає за збереження інформації про стан і керування передачею даних з головним комп'ютером. Воно також контролює роботу двох підлеглих ядер 1 і 2. Перше підлегле ядро обробляє всі дані про рух, що генеруються підключеною тригерною логікою, тоді як друге підлегле ядро діє як периферійний контролер, що підтримує інтегровані функції

системи, такі як керування фокусуванням, кулерами та різними іншими допоміжними компонентами.

Сам MPSoC здебільшого побудований з використанням користувацьких елементів із системної бібліотеки ALTERA. Розробникам потрібно лише розробити системні інтерфейси, такі як керування фокусуванням, підключення двигунів та логіку тригерів. Однак це завдання також підтримується інструментами ALTERA. Це мінімізує зусилля розробників.

Система, показана на рисунку 2.2, є розширюваною системою; 2.2 - це розширювана платформа. Якщо додаткова функціональність перевищує доступні обчислювальні потужності, розробник може вирішити цю проблему, додавши ще один процесор. Таким чином, продуктивність системи може бути адаптована до будь-яких вимог. Такий підхід особливо корисний для операцій, критичних до часу.

3 ОГЛЯД ПЛІС І ЗАСОБІВ АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ ДЛЯ БАГАТОПРОЦЕСОРНИХ СИСТЕМ

3.1 ПЛІС від компанії AMD

AMD пропонує широкий асортимент багатовузлових рішень для задоволення широкого спектру вимог до додатків. Якщо ви розробляєте новітні високопродуктивні мережеві додатки, що вимагають максимальної ємності, пропускної здатності та продуктивності, або шукаєте недорогу, компактну, польову програмовану матрицю вентилів (ПЛІС) для виведення програмно-визначених технологій на новий рівень, багатовузлові рішення AMD відповідають широкому спектру вимог до додатків. Найбільш популярними сімействами ПЛІС в AMD є Spartan™ 7, Artix™ 7, Kintex™ 7 і Virtex™ 7:

1) Spartan™ 7 - Ці пристрої оснащені програмним процесором MicroBlaze™ з більш ніж 200 мікросхемами DMIP, що підтримують пам'ять DDR3 800 Мбіт/с, і виготовлені за 28-нм технологією. Крім того, пристрої Spartan 7 оснащені вбудованим аналого-цифровий перетворювач (АЦП), спеціальними функціями безпеки та підтримують роботу в діапазоні від -40°C до +125°C на всіх комерційних пристроях. Ці пристрої ідеально підходять для промислових, споживчих і автомобільних застосувань, таких як підключення "будь-який до будь-якого", об'єднання датчиків і вбудовані системи машинного зору.
2) Artix™ 7 - це оптимізовані за вартістю ПЛІС, які забезпечують високу продуктивність на ват, лінійну швидкість трансивера, DSP-обробку та інтеграцію AMS. Завдяки підтримці 066 модулів DDR1 3 Мбіт/с і програмному процесору MicroBlaze™, це сімейство пропонує найкраще співвідношення ціни і якості для різноманітних чутливих до витрат і енергоспоживання додатків, включаючи програмно-визначене радіо, камери машинного зору і недорогі бездротові транзитні мережі.

3) Kintex™ 7 пропонує відмінне співвідношення ціна/продуктивність/ват на 28 нм техпроцесі, високий коефіцієнт DSP, економічно вигідну упаковку і підтримку ключових стандартів, таких як PCIe® Gen3 і 10 Gigabit Ethernet. Сімейство Kintex 7 ідеально підходить для бездротового зв'язку 3G і 4G, пласких дисплеїв і рішень для відео по IP.
4) Virtex™ 7 оптимізовано для продуктивності та системної інтеграції за технологічним процесом 28 нм і пропонує виняткову потужність, продуктивність DSP та пропускну здатність вводу/виводу для ваших проектів. Це сімейство використовується в найрізноманітніших додатках, таких як мережі 10-100 Гбіт/с, портативні радари та прототипування ASIC.

3.2 ПЛІС від компанії Intel

ПЛІС від Intel пропонують широкий спектр конфігурованої оперативної пам'яті на кристалі, високошвидкісних приймачів, високошвидкісного вводу/виводу, логічних блоків і маршрутизації. У поєднанні з вбудованою інтелектуальною власністю та чудовими програмними інструментами, ПЛІС скорочують час розробки, енергоспоживання та вартість.

1) Intel Agilex® 9 - пристрій поєднують в собі ПЛІС з провідними в галузі високопродуктивними широкосмуговими перетворювачами даних і високоточними середньо частотними перетворювачами даних. Ці ПЛІС Direct RF SoC використовують 10-нм технологічний процес Intel SuperFin, чотирьох ядерні процесори ARM, приймально-передавальні інтерфейси з наднизькою затримкою, швидкість передачі даних до 58 Гбіт/с і високу щільність каналів, що дозволяє задовольнити жорсткі вимоги до розмірів, ваги та енергоспоживання.
2) Intel Stratix® 10 забезпечує інноваційні переваги в продуктивності, енергоефективності, інтеграції та системній інтеграції. Поєднуючи революційну архітектуру ПЛІС Intel® Hyperflex™ з запатентованою

технологією вбудованого багатокристального моста (EMIB), вдосконаленою інтерфейсною шиною (AIB) та розширеним портфоліо мікросхем, пристрої Intel® Stratix® 10 забезпечують майже вдвічі вищу продуктивність порівняно з ПЛІС попереднього покоління.

3) Intel Arria® 10 має на 40% нижче енергоспоживання, ніж ПЛІС попереднього покоління та ПЛІС SoC, і використовують відкритий код OpenCore1 для забезпечення продуктивності ядра того ж класу швидкості або вище, а також до 20% fMAX у порівнянні з конкурентами, що дає їм конкурентну перевагу. ПЛІС Intel® Arria® 10 та ПЛІС Soc забезпечують оптимальну продуктивність, енергоефективність та малий форм-фактор для широкого спектру застосувань, включаючи комунікації, центри обробки даних, військову промисловість, телерадіомовлення, автомобільну промисловість та інші сфери застосування ПЛІС середнього класу.
4) Intel Cyclone® 10 - сімейство пристроїв, що входить до складу ПЛІС Intel® Edge Centric, оптимізовано для забезпечення збалансованої потужності та пропускної здатності для недорогих додатків, тоді як сімейство пристроїв Intel® Cyclone 10 GX оптимізовано для додатків з більшою пропускною здатністю та продуктивністю.

3.3 Засоби проектування на ПЛІС

Системи автоматизованого проектування (САПР) - це автоматизовані системи, призначені для автоматизації технічного процесу проектування виробів, результатом якого є комплект конструкторських документів, достатній для виробництва і подальшої експлуатації проектованого об'єкта. Вони реалізуються на основі спеціалізованого програмного забезпечення, автоматизованих баз даних і широкого спектру периферійного обладнання.

Найбільш популярними САПР є:

- Vivado - це програмне забезпечення для розробки адаптивних SoC і ПЛІС від AMD, включаючи введення даних, синтез, розміщення і маршрутизацію, а також інструменти верифікації/моделювання. Vivado дозволяє розробникам скоротити час побудови та ітерації проектування, а також більш точно оцінити енергоспоживання адаптивних SoC і ПЛІС від AMD.
- Vitis™ - це інтегроване середовище розробки, яке можна використовувати для прискорення штучного інтелекту на платформах AMD. Інструментарій надає такі ресурси, як оптимізована інтелектуальна власність, інструменти, бібліотеки, моделі та ресурси, такі як зразки проектів і навчальні посібники, щоб допомогти користувачам у процесі розробки.
- Quartus - це програмне забезпечення для проектування, яке включає все необхідне для проектування на ПЛІС, SoC і CPLD від Intel. Це включає введення проекту, синтез та оптимізацію, перевірку та моделювання. Quartus значно розширює можливості пристроїв із багатомільйонними логічними елементами, надаючи розробникам ідеальну платформу для проектування наступного покоління.

3.4 Засоби високорівневого проектування на ПЛІС

3.4.1 Matlab

Matlab - це мова високого рівня та інтерактивне середовище для програмування, чисельних розрахунків, візуалізації результатів і технічних розрахунків Matlab має набір розширень прикладного програмного забезпечення Matlab можна використовувати для аналізу даних, розробки алгоритмів і створення моделей та додатків.

Matlab має багато розширень для різних галузей. Одним з таких розширень є пакет Xilinx System Generator for DSP, ключовий компонент платформи Xilinx для цифрової обробки сигналів (DSP).

System Generator for DSP - це провідний в галузі інструмент для розробки високопродуктивних систем цифрової обробки сигналів на базі ПЛІС і SoC Xilinx. System Generator for DSP дозволяє розробляти високопродуктивні паралельні системи на сучасних ПЛІС, моделювати системи і автоматично генерувати код з Simulink і MATLAB, RTL-код, IP-модулі і код MATLAB, інтегрувати апаратні компоненти в системи DSP.

System Generator for DSP входить до складу Vivado System Edition.

Основні можливості та переваги System Generator for DSP:

- DSP моделювання;
- Бітова і циклічна реалізація з плаваючою і фіксованою точкою;
- Автоматична генерація VHDL або Verilog коду і пакетної IP з Simulink;
- Апаратна ко-симуляція
- Аналіз часу і ресурсів

3.4.2 OpenCL

OpenCL дозволяє програмісту писати функції, які працюють паралельно на конкретному прискорювачі або наборі прискорювачів, розташованих на конкретній машині. Для вказівки конкретного прискорювача використовується поняття контексту. Це структура даних, яка визначає необхідний клас прискорювача (багатоядерний процесор, графічний процесор, ПЛІС) і концепція черги інструкцій структури даних, де виконуються операції на конкретному прискорювачі.

Програмна модель OpenCL базується на концепції ядра. Ядро - це функція, яка виконується паралельно. Ядро визначається програмістом як функція розширення мови С. Створення ядра поділяється на декілька фаз, кожна з яких відповідає виклику відповідної функції OpenCL на центральному процесорі. Спочатку один або декілька вихідних кодів ядра, заданих у вигляді набору рядків, повинні бути перетворені в програму на OpenCL, а потім отримана програма повинна бути скомпільована для потрібного пристрою. Після

компіляції програми на OpenCL з ядра можуть бути отримані функції, що відповідають функціональності вихідного коду, що відповідає функціональності вихідного коду.

Для запуску ядра на прискорювачі необхідно вказати його аргументи. Аргументом ядра може бути скалярне значення або адреса буфера в пам'яті прискорювача. Загалом, оскільки прискорювачі не мають прямого доступу до пам'яті процесора (наприклад, якщо прискорювачем є ПЛІС або графічний прискорювач), ядро працює з адресою прискорювача; OpenCL виділяє буфер пам'яті на прискорювачі і розподіляє його між централізованим процесором і пам'яттю прискорювача.

Необхідне ядро викликається за допомогою функцій з бібліотеки OpenCL на центральному процесорі. Прискорювач, на якому має бути запущено ядро, визначається чергою команд, з якої надсилається запит на запуск ядра.

4 ПОБУДОВА СТРУКТУРНОЇ ТА ФУНКЦІОНАЛЬНОЇ СХЕМИ БАГАТОПРОЦЕСОРНОЇ СИСТЕМИ

4.1 Структурна схема багатопроцесорної системи

В цій дипломній роботі треба реалізувати операційну частину багатопроцесорної системи на базі ПЛІС від компанії AMD (Xilinx). З типів багатопроцесорних систем було вибрана SMP з спільною пам’яттю. Зі способів взаємодії інструкцій обчислювальної машини з даними була вибрана MISD.

Проект був реалізований на базі ПЛІС сімейства Artix™ 7 із за програмного процесора MicroBlaze™, який дозволяє реалізувати декілька процесорів одночасно.

В проекті передбачається загальна структурна схема багатопроцесорної системи для проведення операцій над комплексними числами, яка наведена на рисунку 4.1.

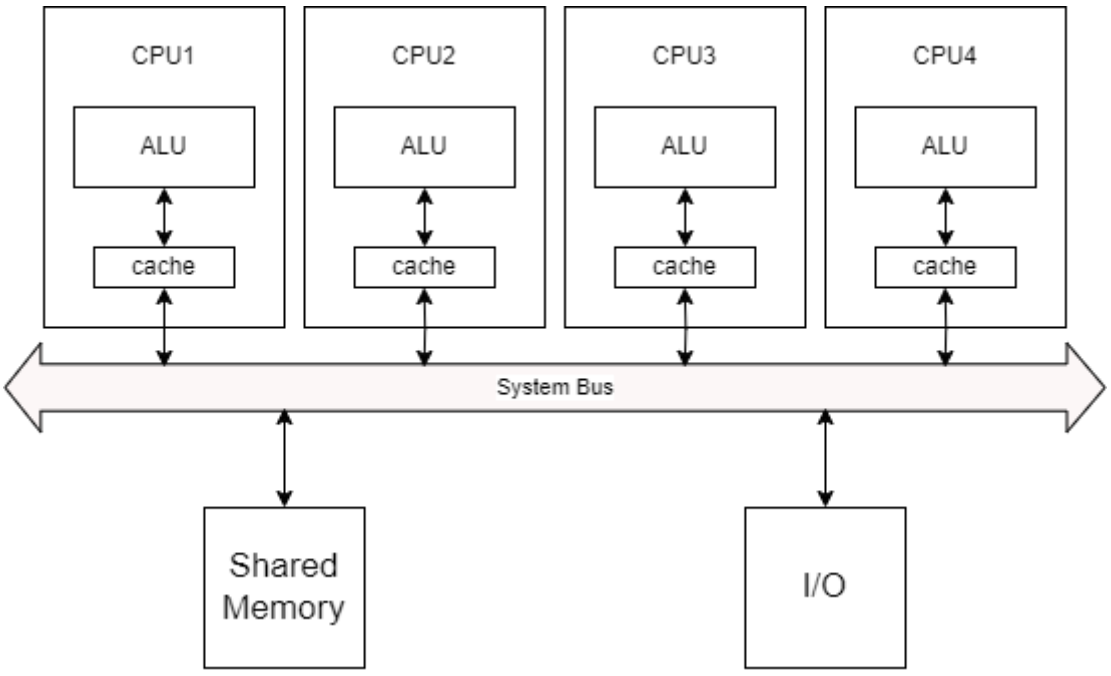


Рисунок 4.1 – Структурна схема багатопроцесорної системи

4.2 Функціональна схема багатопроцесорної системи та процесору

Передбачається, що багатопроцесорна система складається з чотирьох процесорів, блоку управління, портів вводу/виводу та блоку спільної пам'яті. Кожен процесор може виконувати окремі програми, що зчитуються з оперативної пам'яті. Для обчислення комплексних чисел процесори оснащені арифметичними блоками. Крім того, кожен процесор має елементи керування та кеш-пам'ять. Якщо різні процесори намагаються використовувати один і той самий ресурс одночасно, їхні сигнали будуть інтерферувати, тому блок керування доступом до ресурсів отримує від процесорів сигнали запиту до ресурсу і видає сигнали дозволу.

Функціональна схема багатопроцесорной системи зображена на рисунку 4.2.

Функціональна схема процесора зображена на рисунку 4.3.

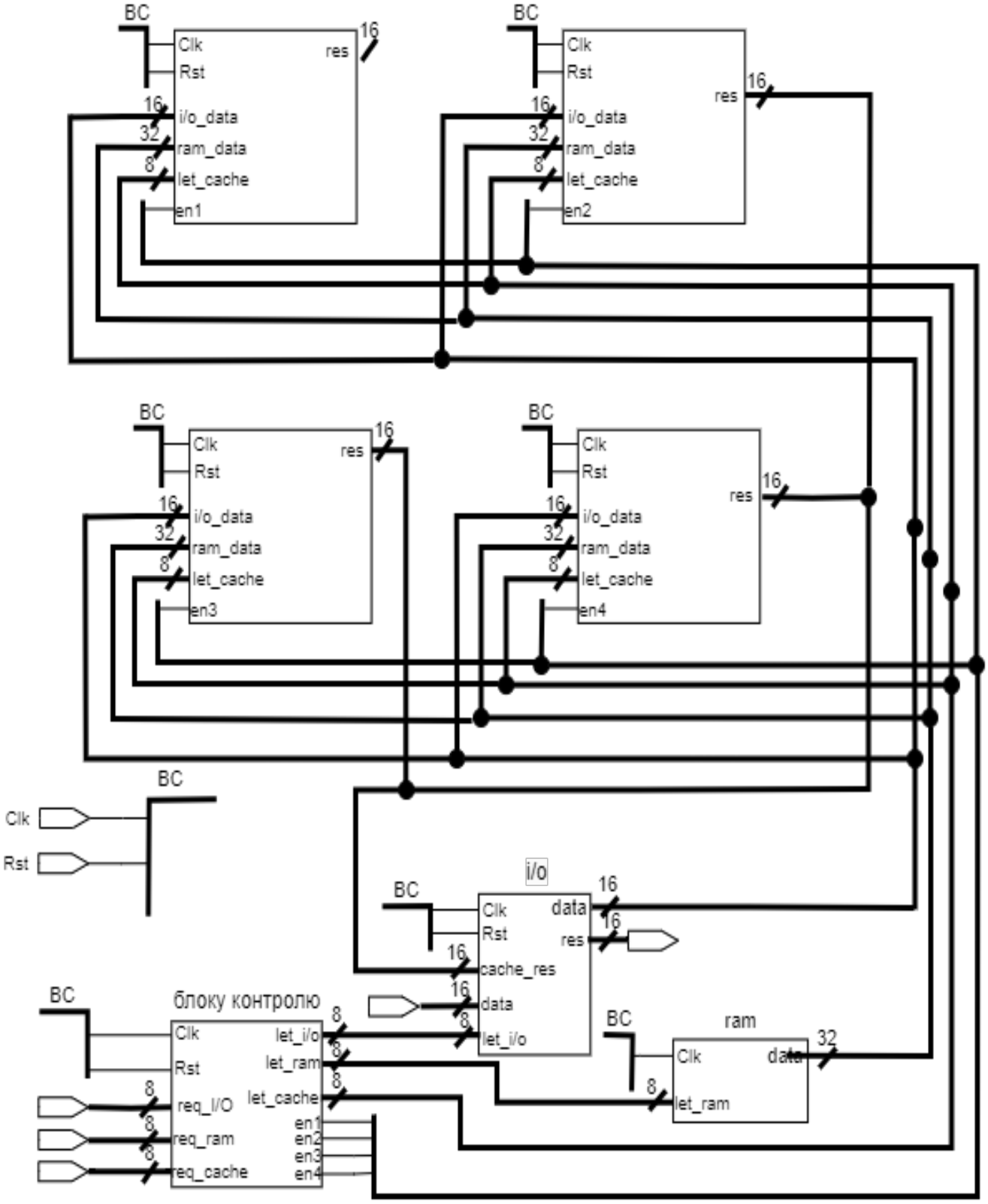


Рисунок 4.2 – Функціональна схема багатопроцесорной системи

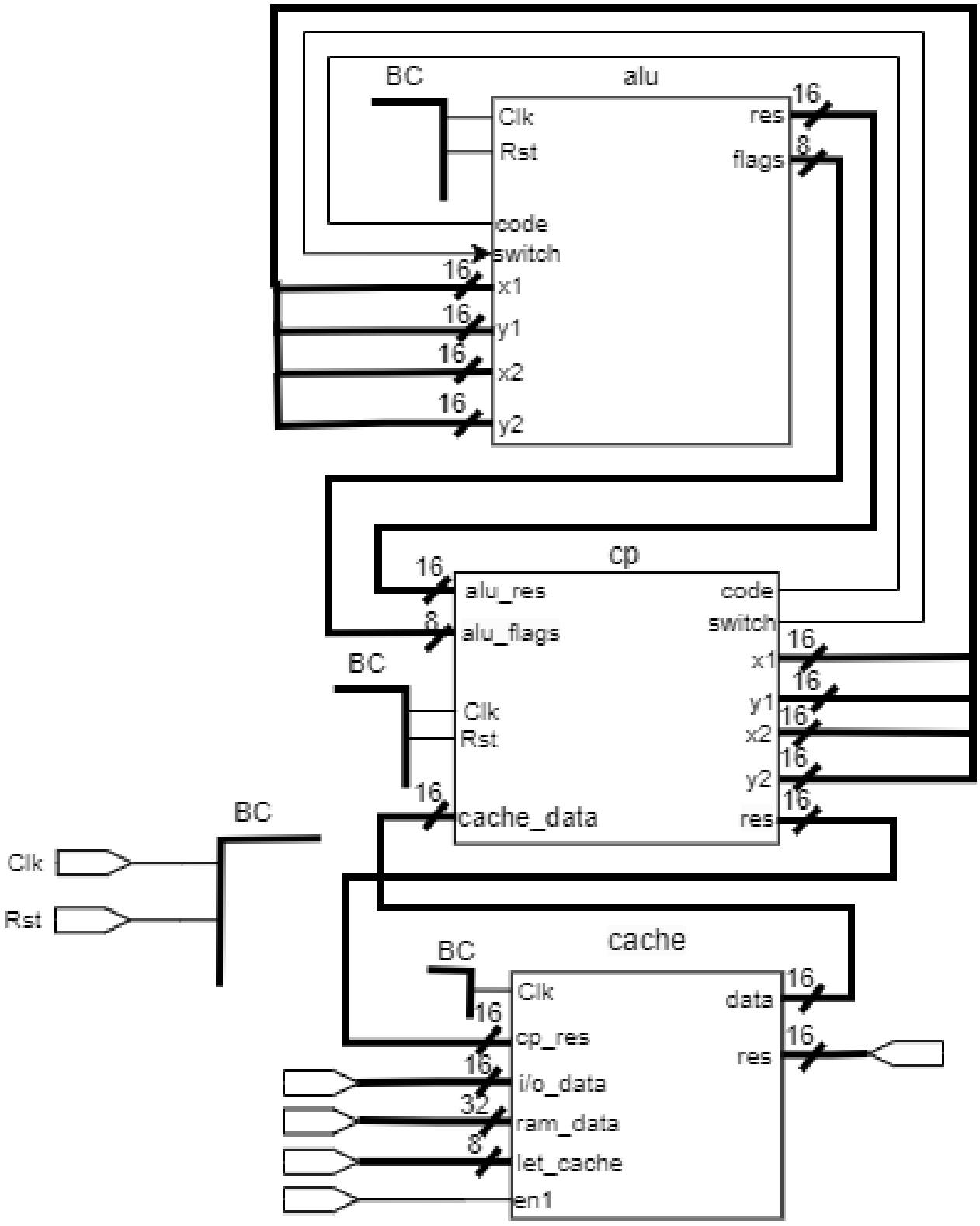


Рисунок 4.3 – Функціональна схема процесора

4.3 Умовні графічні позначення функціональних схем

Для створення функціональної схеми розділу 4.2 спочатку було визначено умовні графічні позначення для елементів, які використовуються для побудови функціональної схеми багатопроцесорної системи та процесора.

Умовне графічне позначення блоку контролю багатопроцесорної системи наведено на рисунку 4.4. На порти req_I/O, req_ram, req_cache, заводяться сигнали адресів після чого передаються до відповідних блоків. З порта let_I/O - блок ввода/вивода, let_ram-блок спільної пам’яті, let_cache - кеш пам'ять процесора. З портів en1-en4 подаються сигнали до процесора який має працювати.

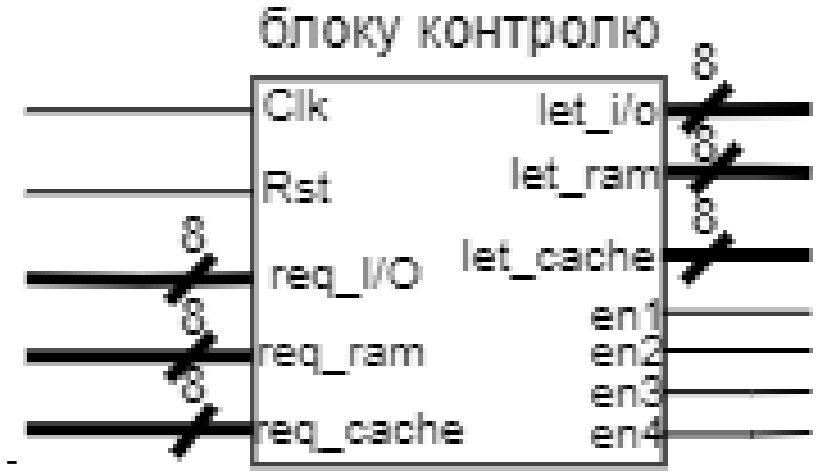


Рисунок 4.4 – Умовне графічне позначення блоку контролю

Умовне графічне позначення спільной пам’яті наведено на рисунку 4.5. На порт let_ram приходе сігнал з блоку контролю о запуске. З порта data подається код програми до процесора.

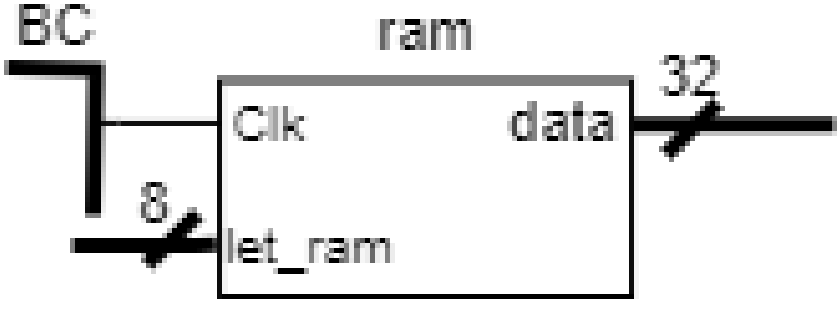


Рисунок 4.5 – Умовне графічне позначення спільной пам’яті

Умовне графічне позначення порту вводу-виводу наведено на рисунку 4.6. На порт let_i/o приходе сігнал з блоку контролю о запуске. В порт data вводяться вхідні операнди після цього передаються до вибраного процесора. На порт cache_res приходить результат обчислення після чого він видає на порт виводу res

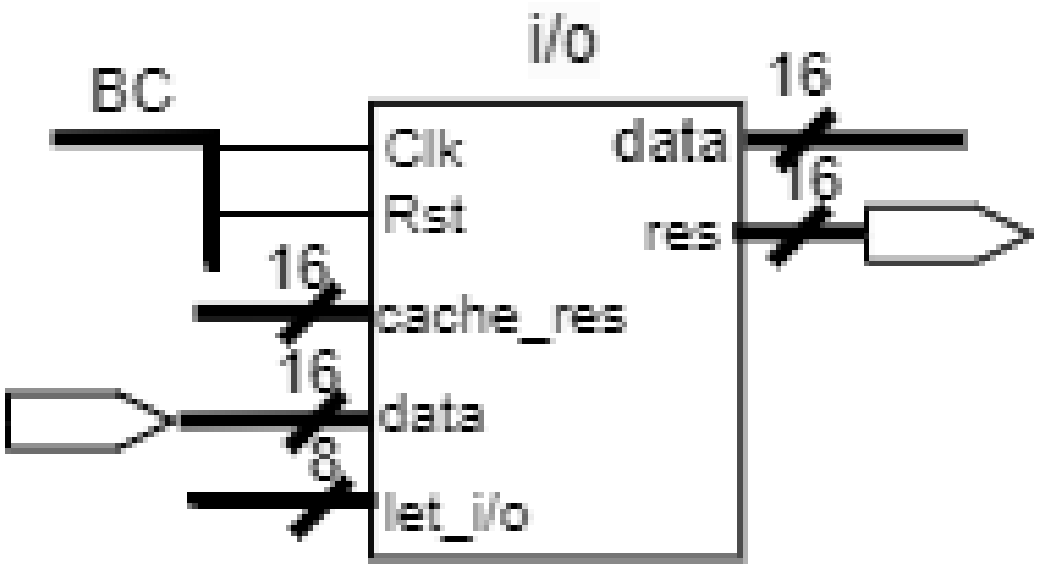


Рисунок 4.6 – Умовне графічне позначення пристрою вводу-виводу

Умовне графічне позначення кешу процесора на рисунку 4.7. На порти let_cache і en приходе сігнал з блоку контролю о запуске процесора і кеш памяті. В порт i/o_data приходять вхідні операнди В порт data данні передаються в контролер процесора. На порт cp_res приходить результат обчислення після чого відправляються до порта виводу через res.

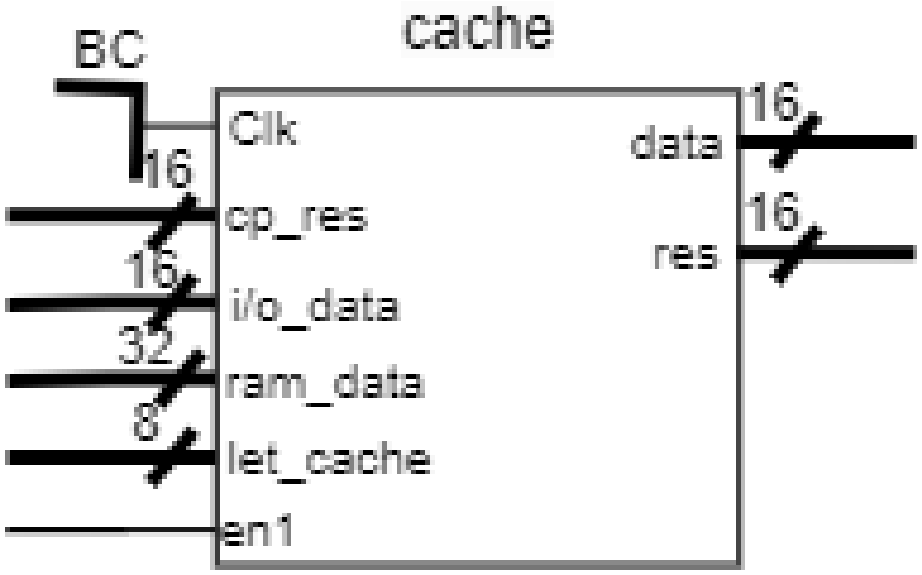


Рисунок 4.7 – Умовне графічне позначення кешу процесора

Умовне графічне позначення контрольного модуля процесора наведено на рисунку 4.8. На порти cache_data приходять вхідні данні. З портів x1, x2, y1, y2 данні відправляються до арифметико-логічного пристрою. На порт alu_res приходить результат обчислення після чого він відправляється до КЕШ памяті.

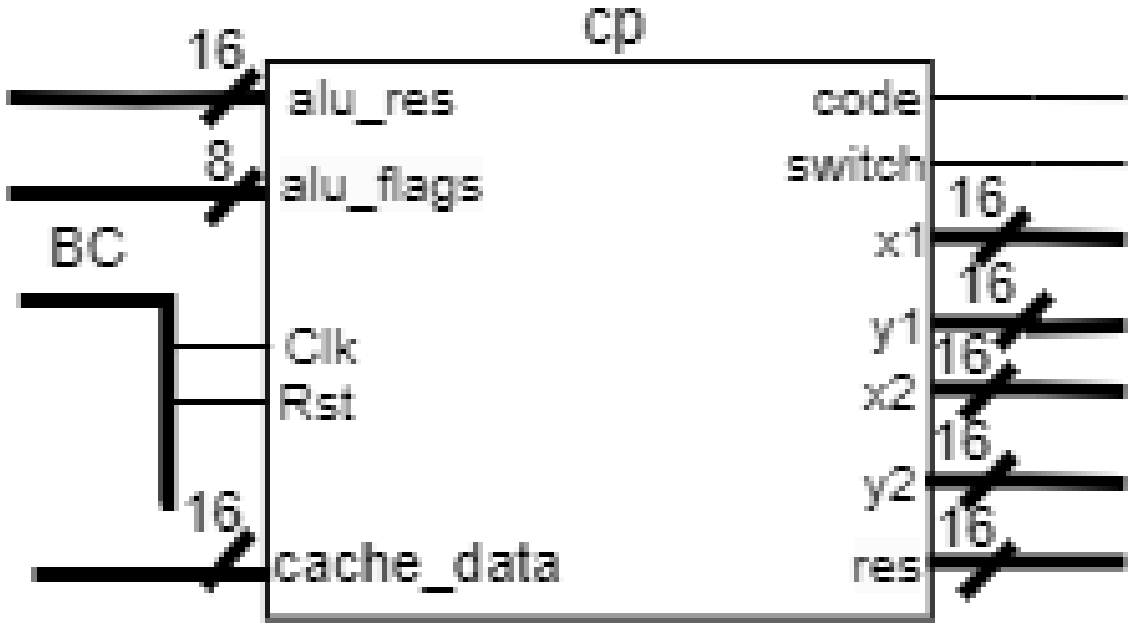


Рисунок 4.8 – Умовне графічне позначення контрольного модуля процесора

Умовне графічне позначення арифметико-логічного пристрою наведено на рисунку 4.9. На порти x1, x2, y1, y2 приходять данні. З порта res відправляється результат до контрольного модуля.

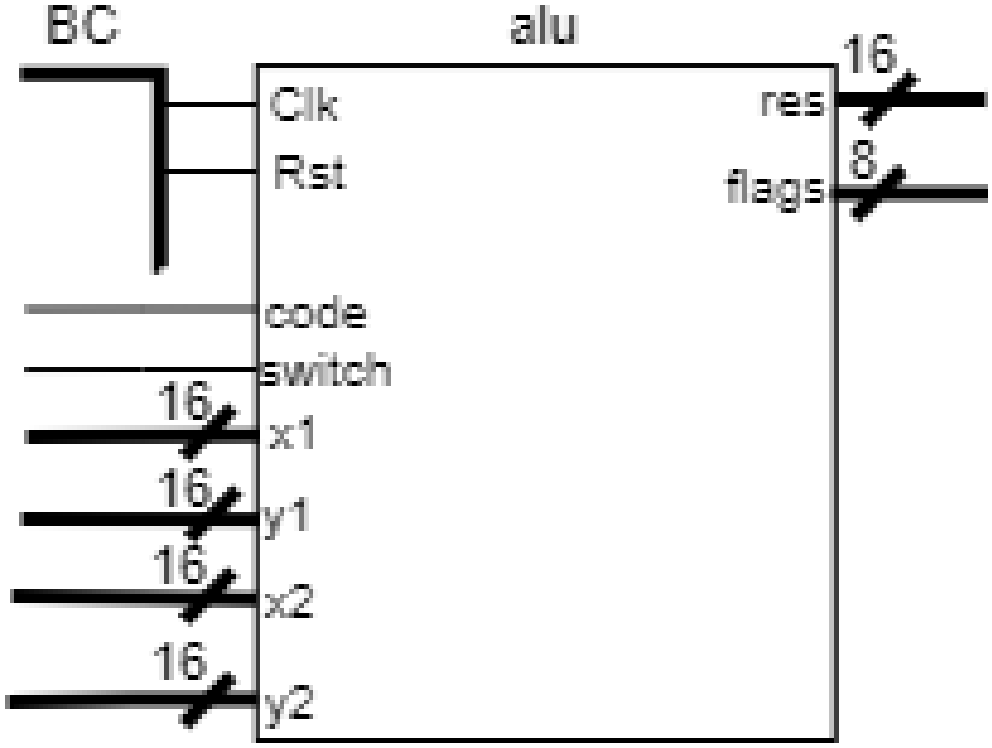


Рисунок 4.9 – Умовне графічне позначення арифметико-логічного пристрою

Умовне графічне позначення процесору зображено на рисунку 4.10.

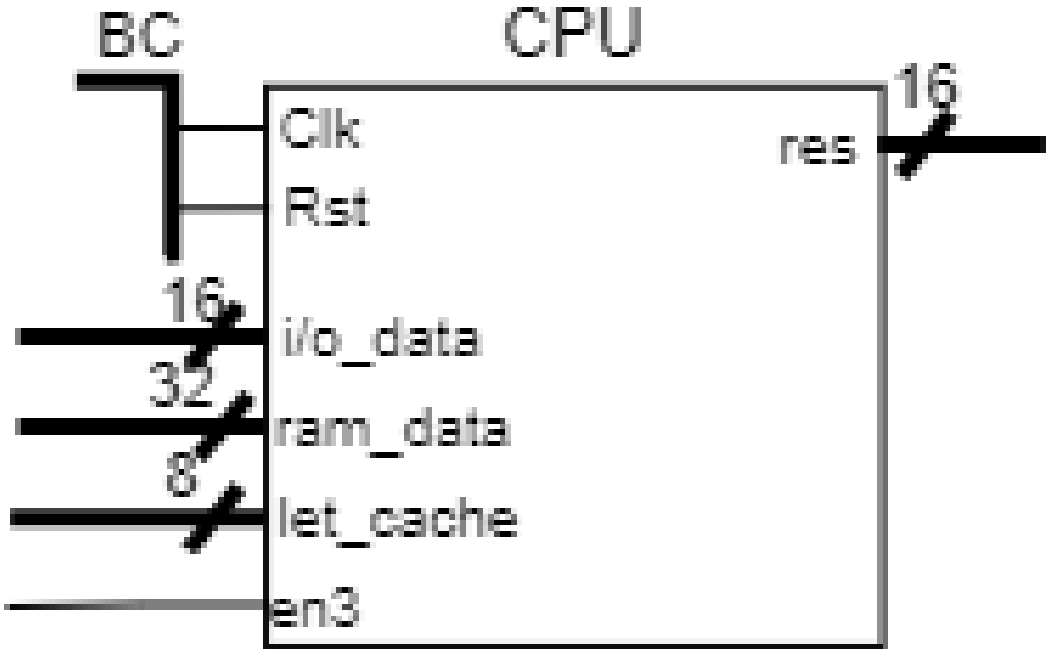


Рисунок 4.10 – Умовне графічне позначення процесору

5 РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ ПРОЦЕСОРА

5.1 Етапи проектування пристроїв на ПЛІС

Для реалізації проекту на ПЛІС від Xilinx необхідно виконати наступні кроки:

1) Створити проект у Vivado Design Suite 2018.2;

2) Вибрати сімейство та тип ПЛІС;

3) Описати пристрій на основі алгоритма або схеми мовою VHDL;

4) Виконати синтез пристрою;

5) Виконати моделювання ;

6) Визначити місце розміщення Place і Trace трасування розробленого проекту на ПЛІС ;

7) Виконати часове моделювання ;

8) Виконати проектування на ПЛІС ;

9) Завантажити проект на мікросхему задопомогою згенерованого bitstream файлу.

В зв’язку відсутністю ПЛІС були виконані перші п’ять кроків.

При виборі конкретної ПЛІС слід враховувати декілька факторів, зокрема наступні:

1) Гранична вартість розробки;

2) Вимоги до продуктивності

3) Складність проекту.

Пристрій, що розробляється, може бути представлений кількома способами:

1) VHDL опис;

2) схеми.

3) діаграми станів, пакети та бібліотеки розробника.

В процесі синтезу створюється користувацький список з'єднань з набором примітивів і компонентів з урахуванням вхідних даних проекту і реалізується відповідно до обраного сімейства ресурсів і конкретних кристалів.

Процес функціонального моделювання здійснюється без урахування фактично виміряної затримки проходження сигналу, яка використовується під час роботи алгоритму для перевірки придатності роботи ПЛІС від отримання вхідних даних до вихідного сигналу.

Завершальним етапом розробки пристрою є завантаження конфігурації (BitStream файл) в мікросхему. Зазвичай це робиться шляхом завантаження бітового файлу з комп'ютера через старий LPT-порт або сучасний USB- чи Ethernet-порт. Цей файл передається безпосередньо на ПЛІС через порт JTAG.

5.2 Етапи створення проекту в Vivado 2018.2

Проект було створено у Vivado 2018.2 фірми Xilinx.

Для того, щоб створити новий проект у САПР Vivado 2018.2 розробнику необхідно відкрити відповідну програму, створити новий проект нажавши «Create Project» - у розділи «Quick Start» (див. рис. 5.1). Після цього розробник вказує назву проекту та його розташування.

Рисунок 5.1 – Вікно створення проекта

Наступним етапом є вибір типа проектування, мови проектування і симуляції та створення внутрішніх файлів проекту (див. рис. 5.2).

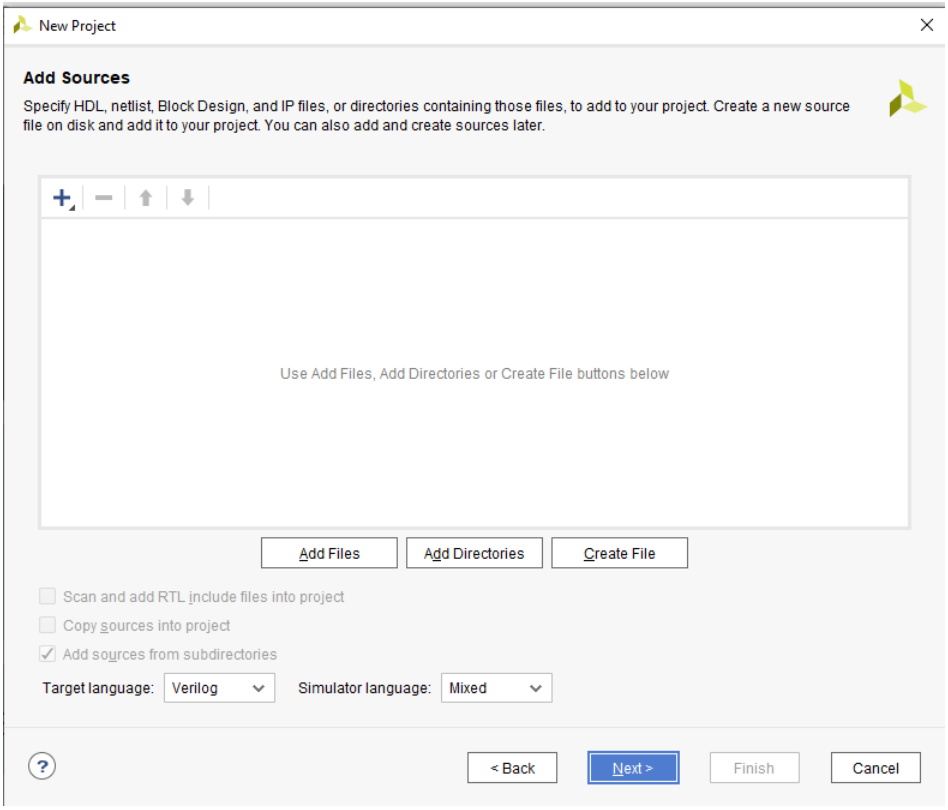


Рисунок 5.2 – Вікно настройки проекта

Далі потрібно вказати основні властивості ПЛІС(див. рис. 5.3).

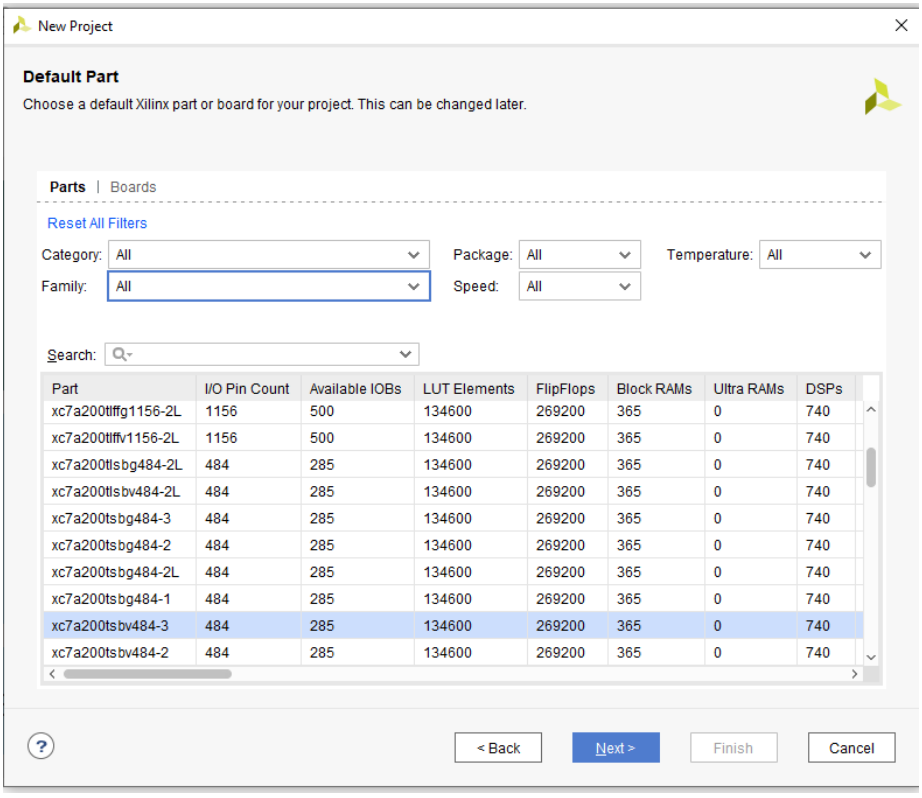


Рисунок 5.3 – Вікно вибора ПЛІС

Після налаштування відкривається стандартне вікно з повним інструментарієм (див. рис. 5.4).

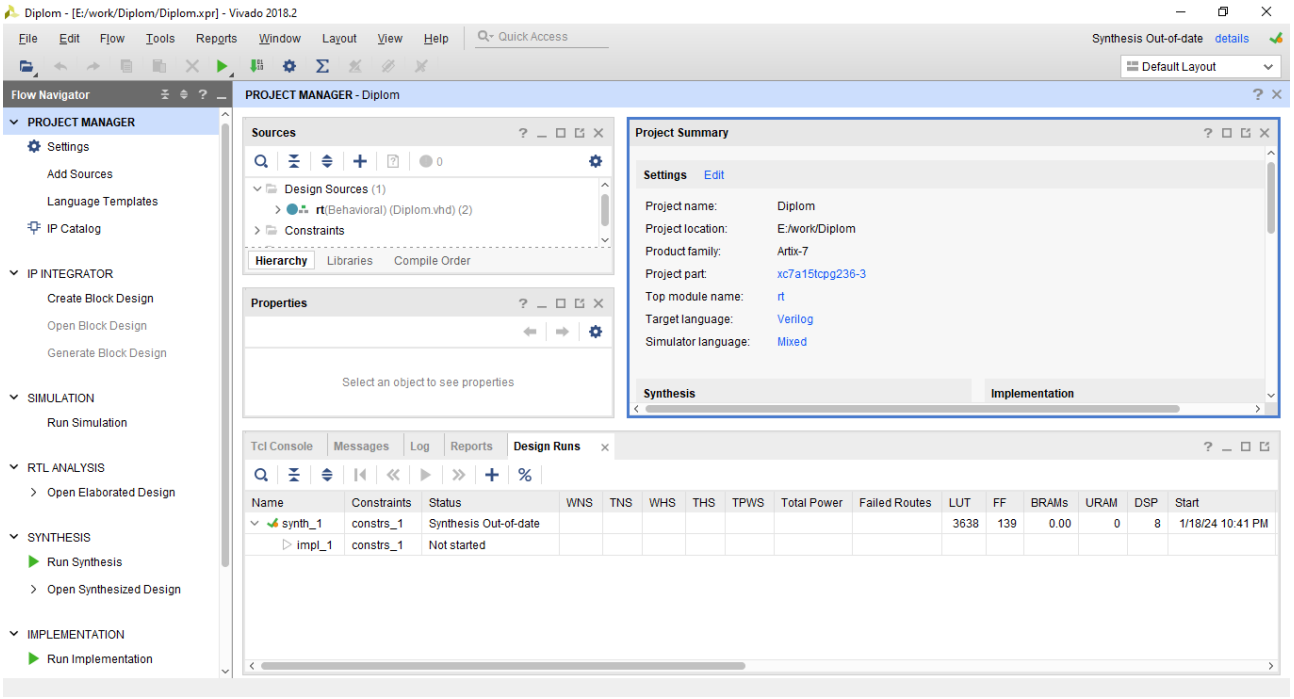


Рисунок 5.4 – Стандартне вікно Vivado 2018.2

Далі за допомогою мови VHDL описується пристрій.

Потім відбувається синтез. Після чого проводиться моделювання.

Для цього необхідно створити VHDL - Test Bench. Потім розробник вказує об'єкт, для якого повинен бути створений тестовий стенд. Тестовий стенд VHDL автоматично створюється за допомогою структурного стилю, до якого додаються вирази стимулів вхідного сигналу. Після синтезу та моделювання створюється файл бітового потоку, який завантажується за допомогою пункту Create Programming File (Створити файл програми) у вікні Operations (Операції).

5.3 Блоки цифрової обробки

ПЛІС є ефективними для додатків цифрової обробки сигналів (DSP), оскільки вони можуть реалізувати спеціальні, повністю паралельні алгоритми. Додатки DSP використовують багато двійкових помножувачів і накопичувачів, які найкраще реалізуються у виділених фрагментах DSP. Усі FPGA серії 7 мають багато спеціалізованих повністю призначених для користувача сегментів DSP з низьким енергоспоживанням, що поєднує високу швидкість із малими розмірами, зберігаючи при цьому гнучкість конструкції системи. Зрізи DSP підвищують швидкість і ефективність багатьох додатків за межами цифрової обробки сигналів, таких як широкі динамічні зсуви шини, генератори адрес пам'яті, мультиплексори широкої шини та регістри вводу/виводу, відображені в пам'яті. На рисунку 5.5 представлений стандартний блок DSP. Ресурси DSP оптимізовані та масштабовані для всіх сімейств серії 7, забезпечуючи загальну архітектуру, яка покращує ефективність впровадження, впровадження IP та міграцію дизайну.

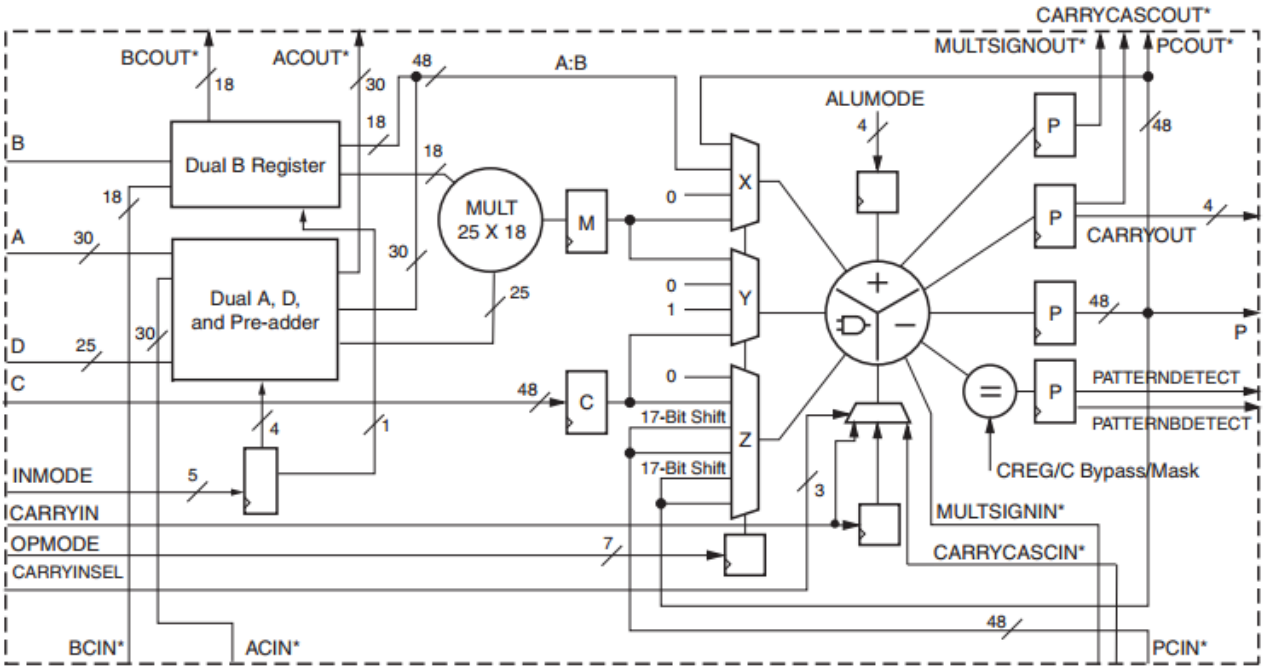


Рисунок 5.5 – Стандартний блок DSP

У багатопроцесорної системі проводяться арифметичні операції над комплексними числами, а ПЛІС із сімейства сімейства Artix™ 7. Тому була вибрана із універсальних блоків цифрової обробки сигналів була вибрана DSP48E1, яка виконує функції множення, додавання і віднімання.

Графічне позначення DSP48E1 зображено на рисунку 5.6.

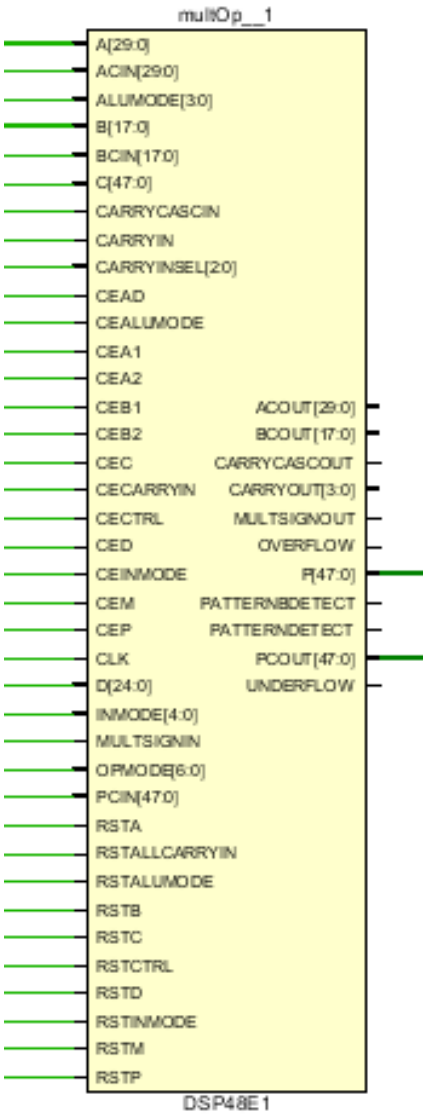


Рисунок 5.6 – Графічне позначення DSP48E1

5.4 VHDL-опис операційної частини багатопроцесорної системи

Після побудови функціональної схеми багатопроцесорної системи та процесора, в САПР Vivado Design Suite 2018.2 був написано VHDL–код, який складається з локального керуючого блока і чотирьох арифметико-логічних блоків.

VHDL -код наведено у додатку А.

Локальний керуючий блок відповідає за те, щоб повідомляти арифметико-логічним блокам, в якій послідовності треба виконувати дії. Локальний керуючий блок має чотири виходи, які визначають активний стан конвеєра в даний момент, а також входи скидання і синхронізації. По кожному такту послідовно змінюються стани, і якщо сигнал скидання буде активним, то локальний керуючий блок повернеться у початковий стан. У блоці формуються 4 сигнали enable, які запускають певні процеси:

- mult_enable – запускає процес множення;
- sum_enable – запускає процес додавання;
- sub_enable – запускає процес віднімання;
- div_enable – запускає процес ділення.

Фрагмент VHDL-коду локального керуючого блоку приведений нижче (Лістинг 5.1).

Лістинг 5.1 – Фрагмент VHDL-коду локального керуючого блоку

```
 process(Current_State)
begin
    case Current_State is
      when intl =>
         mult_enable <= '0';
         sum_enable <= '0';
```

```
            sub_enable <= '0';
            div_enable <= '0';
        when multiplic =>
            mult_enable <= '1';
            sum_enable <= '0';
            sub_enable <= '0';
            div_enable <= '0';
        when sume =>
            mult_enable <= '0';
            sum_enable <= '1';
            sub_enable <= '0';
            div_enable <= '0';
        when subtract =>
            mult_enable <= '0';
            sum_enable <= '0';
            sub_enable <= '1';
            div_enable <= '0';
        when divis =>
            mult_enable <= '0';
            sum_enable <= '0';
            sub_enable <= '0';
            div_enable <= '1';
        when others =>
            mult_enable <= '0';
            sum_enable <= '0';
            sub_enable <= '0';
            div_enable <= '0';
    end case;
end process;
```

Для позначення АЛП у коді та на схемах застосовується скорочення ALU – arithmetic logic unit.

Арифметично-логічний пристрій працює з цілими числами і виконує такі арифметичні операції як додавання, віднімання, множення та ділення;

Оскільки АЛП має в своєму складі елементи пам’яті, то перед використанням потрібно виконати скидання його стану за сигналом reset (rst).

АЛП спрацьовує при спадаючому фронту сигналу clk за умови позитивного сигналу en (enable). Код операції задається вхідним сигналом code, операнди – сигналами x1, x2, y1, y2. Вихідний вектор res надає результат виконання операції на вихід.

Для реалізації арифметико-логичного пристрою множення, додавання і віднімання використовувався VHDL-опис поведінковим стилем, а для реалізації процесу ділення - структурним стилем.

Фрагмент VHDL-коду компоненту ділення двох чисел (cur_divident на cur_divisor, результат - cur_qt) приведений нижче (Лістинг 5.2). Оскільки, в формулі 5.4 повинно виконуватися два ділення, тому в АЛП використовуються два компоненти ділення – u1 і u2 (див. додаток А).

Лістинг 5.2 – Фрагмент VHDL-коду ділення комплексних чисел.

```
divisionn: process (clk,rst,ce)
variable cur_divident,R: std_logic_vector(31 downto 0) ;
variable cur_divisor: std_logic_vector(31 downto 0) ;
variable cur_qt: std_logic_vector(31 downto 0) ;
begin
   if rst='1' then
      cur_divident:=x"00000000" ;
      cur_divisor:=x"00000000" ;
      R:=x"00000000";
      cur_qt := x"00000000";
```

```
    elsif rising_edge(Clk) then
      if ce='1' then
        cur_divident := dd ;
        cur_divisor := dr ;
        cur_qt := x"00000000";
        R:=x"00000000";
            for i in 31 downto 0  loop
            R := R(30 downto 0) & cur_divident(i);
          if R>=cur_divisor then
            R := R-cur_divisor ;
            cur_qt(i):='1' ;
          end if ;
          end loop ;
      end if ;
    end if ;
  qt<=cur_qt;
  end process divisionn;
```

5.5 Дослідження створеної багатопроцесорної системи

Після написання коду на VHDL був згенерований Test Bench і введені початкові данні в порти(числа приведені в шістнадцятирічному системі числення):

1) X1 – (8, A, B);
2) X2 – (3, 5, 7);
3) Y1 – (A, C, E);
4) Y2 – (2, 4, 5).

Після чого він готовий до симулювання. На рисунках 5.7 -5.13 приведені тестування з послідовним роботи чотирьох процесорів.

Операції виконуються над двома комплексними числами: $Z_1 = X_1 + i * Y_1$; $Z_2 = X_2 + i * Y_2$, де $i = \sqrt{-1}$ - уявна одиниця.

На рисунках 5.7, 5.8 наведено результати моделювання АЛП при виконанні операції додавання за формулою (5.1):

$$Z_1 + Z_2 = (X_1 + X_2, Y_1 + Y_2) \quad (5.1)$$

де по сигналу sum_enable починається процес додавання і подається результат на порти zr2 (дійсна частина) і zi2 (уявна частина). На рисунку 5.7 додаються комплексні числа 8+i*10, 3+i*2, відповідно результат zr2=11, zi2=12. На рисунку 5.8 додаються комплексні числа 10+i*12, 5+i*4, відповідно результат zr2=15, zi2=16.

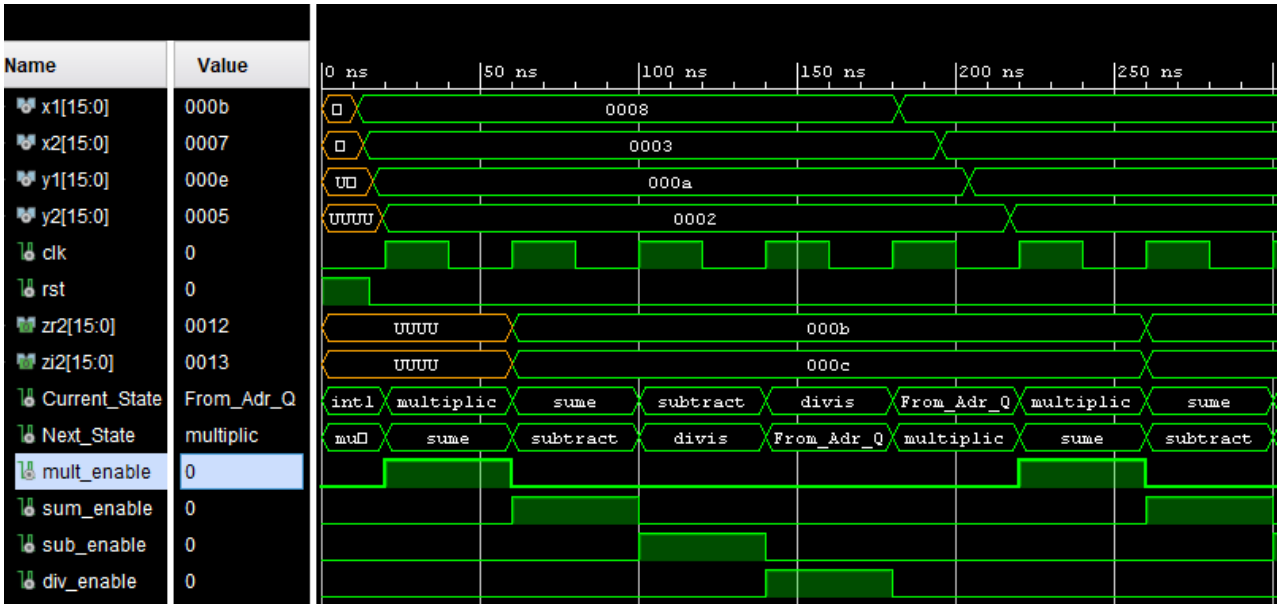


Рисунок 5.7 – Тестування АЛП при виконанні операції додавання, частина 1

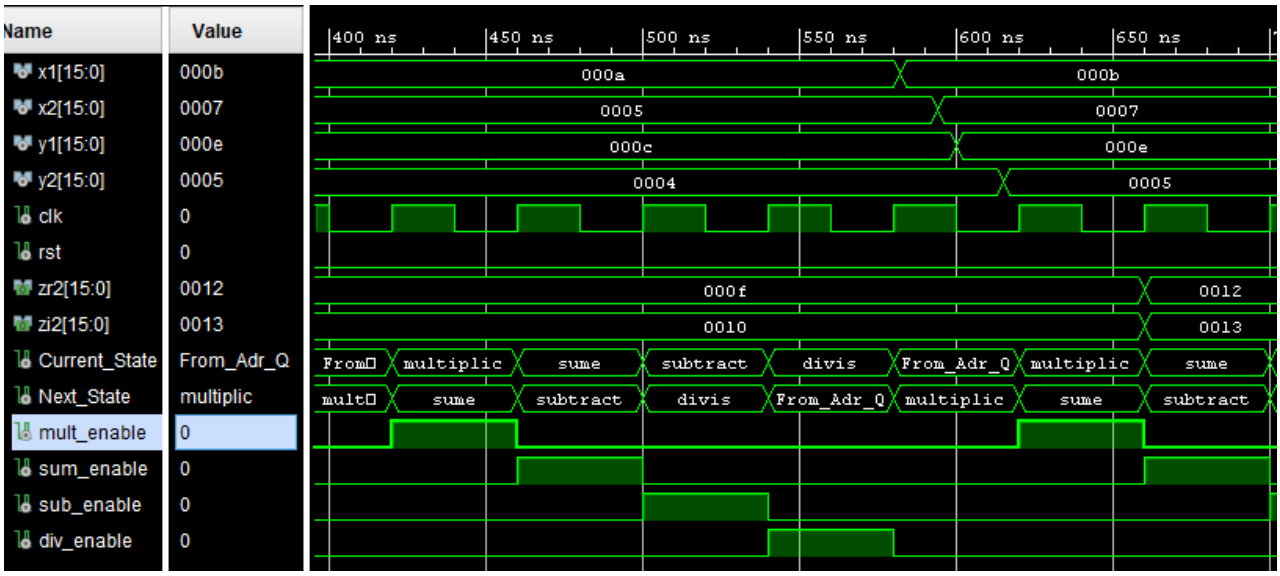


Рисунок 5.8 – Тестування АЛП при виконанні операції додавання, частина 2

На рисунках 5.9, 5.10 проводиться тестування АЛП при виконанні операції віднімання за формулою (5.2):

$$Z_1 - Z_2 = (X_1 - X_2, Y_1 - Y_2) \tag{5.2}$$

де по сигналу sub_enable починається процес віднімання і подається результат на порти zr3 (дійсна частина) і zi3 (уявна частина).. На рисунку 5.9 віднімаються комплексні числа 8+i*10, 3+i*2, відповідно результат zr3=5, zi3=8. На рисунку 5.10 віднімаються комплексні числа 11+i*14, 7+i*5, відповідно результат zr3=4, zi3=9.

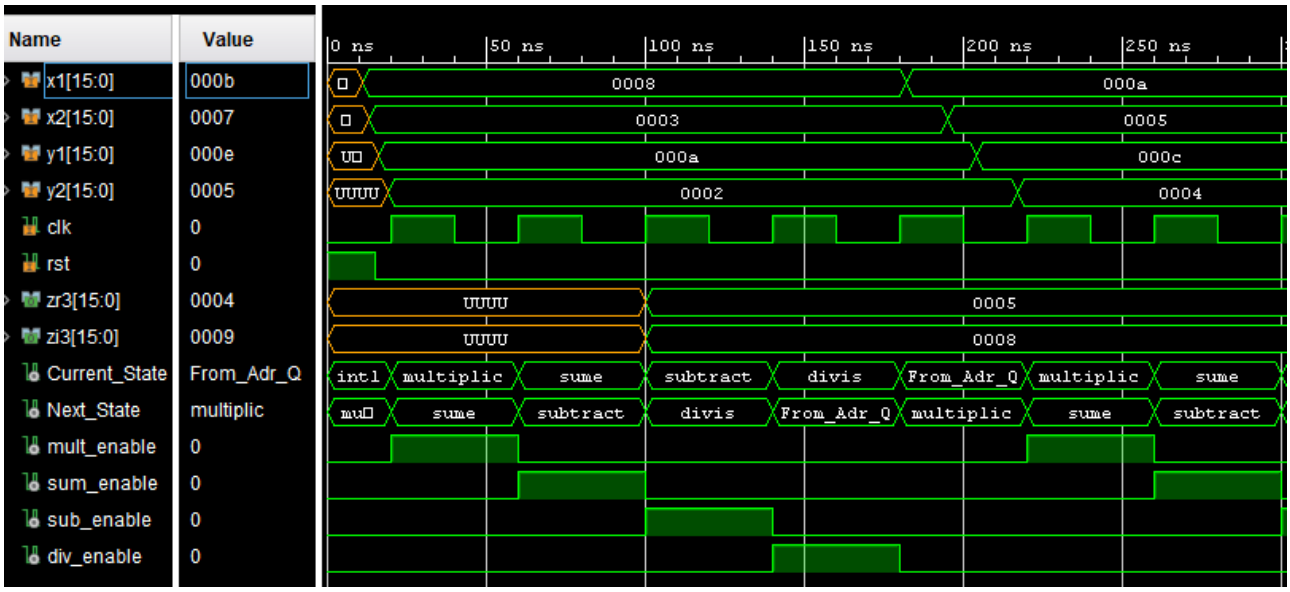


Рисунок 5.9 – Тестування АЛП при виконанні операції віднімання, частина 1

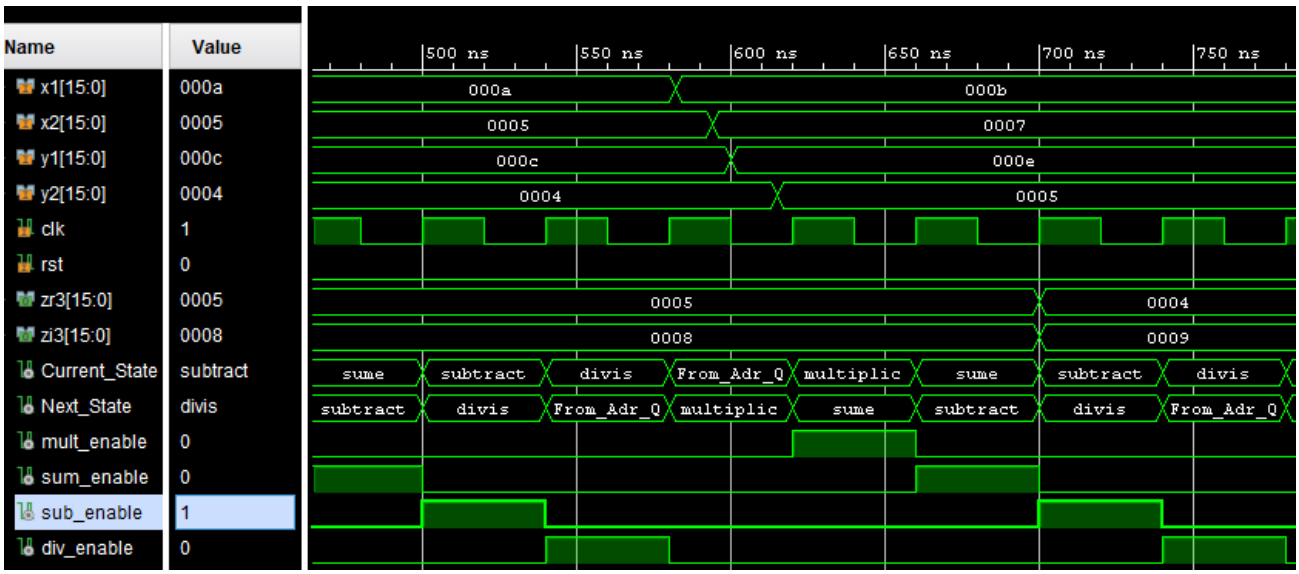


Рисунок 5.10 – Тестування АЛП при виконанні операції віднімання, частина 2

На рисунках 5.11, 5.12 проводиться тестування АЛП при виконанні операції множення за формулою (5.3):

$$Z_1 * Z_2 = (X_1 * X_2 - Y_1 * Y_2, X_1 * Y_2 + X_2 * Y_1) \quad (5.3)$$

де по сигналу mult_enable починається процес множення і подається результат на сигнали a1, b1, a2, b2. Після цього результат віднімання виводиться на порт zr1 (дійсна частина), а результат додавання на порт zi1 (уявна частина). На рисунку 5.11 умножаються комплексні числа 12+i*10, 7+i*2, відповідно результат zr1=4, zi1=46. На рисунку 5.12 умножаються комплексні числа 11+i*14, 7+i*5, відповідно результат zr1=2, zi1=100.

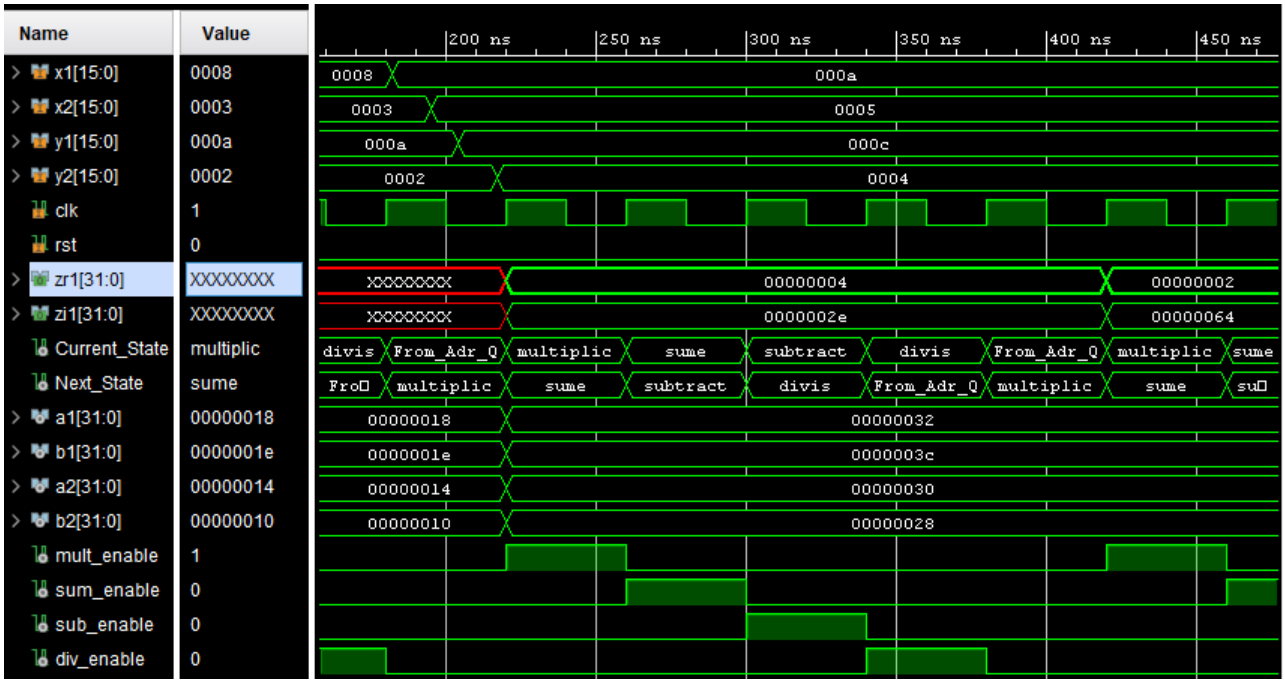


Рисунок 5.11 – Тестування АЛП при виконанні операції множення, частина 1

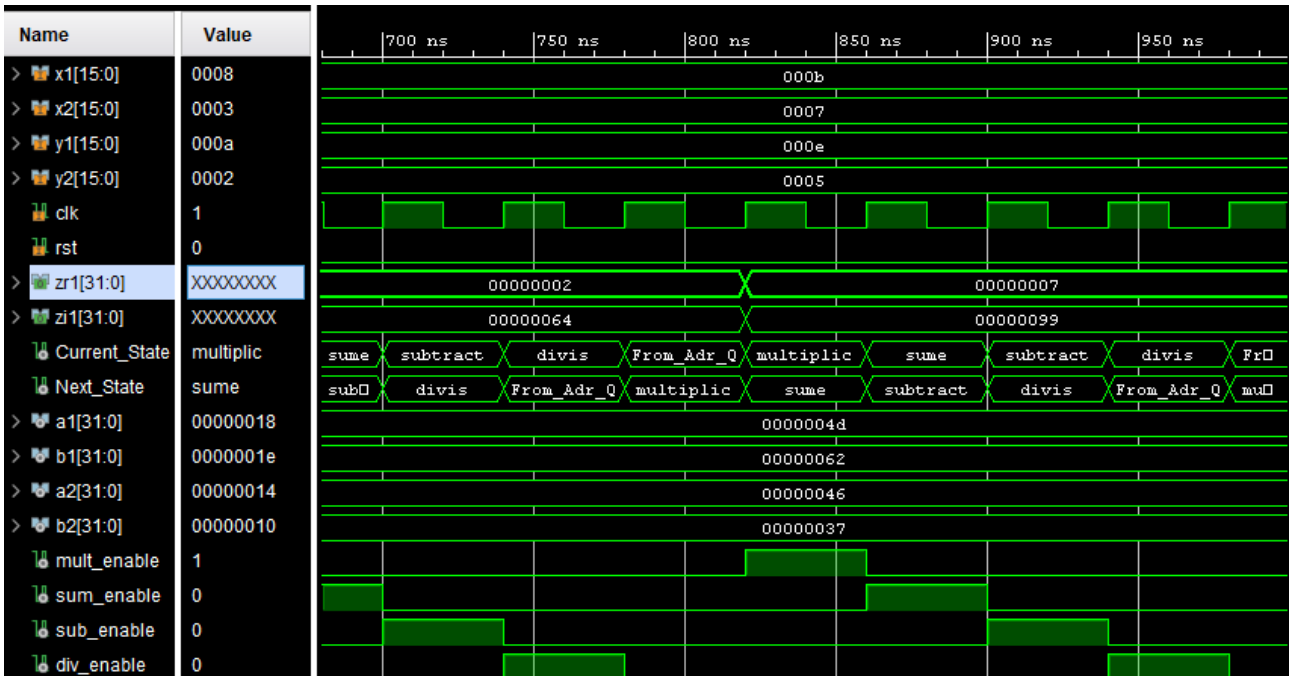


Рисунок 5.12 – Тестування АЛП при виконанні операції множення, частина 2

На рисунках 5.13, 5.14 проводиться тестування АЛП при виконанні операції ділення за формулою(5.4):

$$\frac{Z_1}{Z_2} = \left(\frac{X_1X_2+Y_1Y_2}{X_2^2+Y_2^2}, \frac{X_2Y_1-X_1Y_2}{X_2^2+Y_2^2}\right) \qquad (5.4)$$

де по сигналу div_enable починається процес розрахунку двух чисельників (ddr, ddi) і одного знаменника (drri). Після чого на порт zr (дійсна часть) виводиться результат ділення ddr на drri, а на порт zi (уявна часть) результат ділення ddi на drri. На рисунку 5.13 умножаються комплексні числа 12+i*10, 7+i*2, відповідно результат zr=3, zi=1. На рисунку 5.14 умножаються комплексні числа 11+i*14, 7+i*5, відповідно результат zr=2, zi=0.

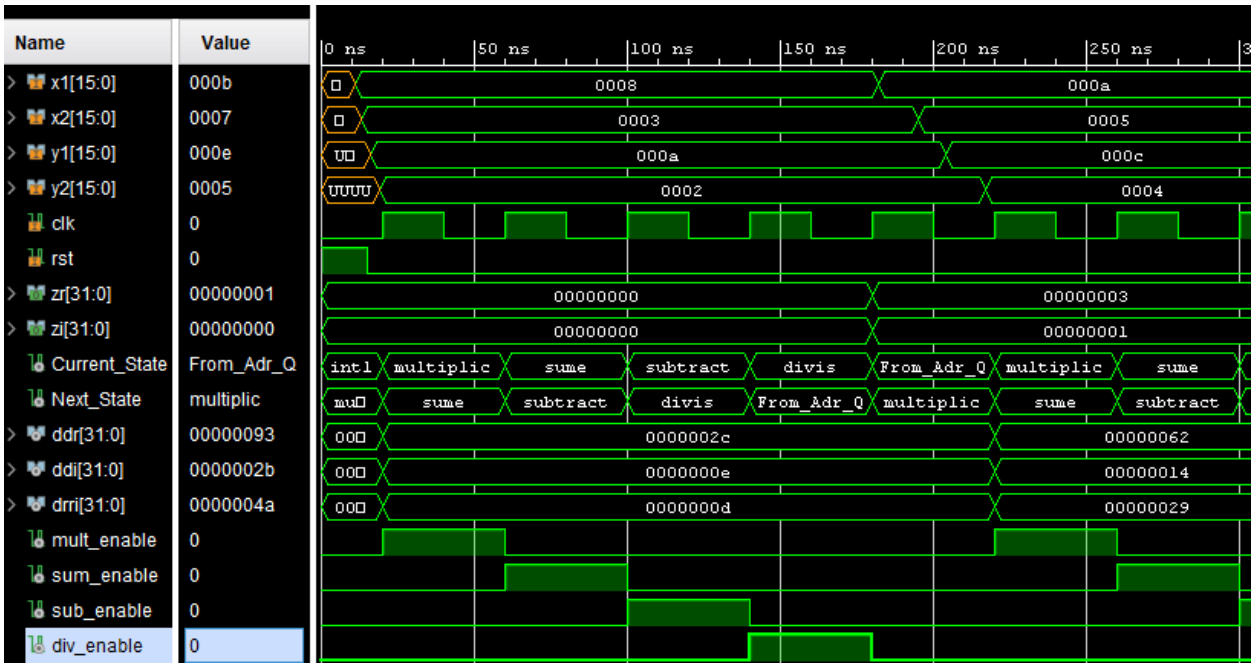


Рисунок 5.13 – Тестування АЛП при виконанні операції ділення, частина 1

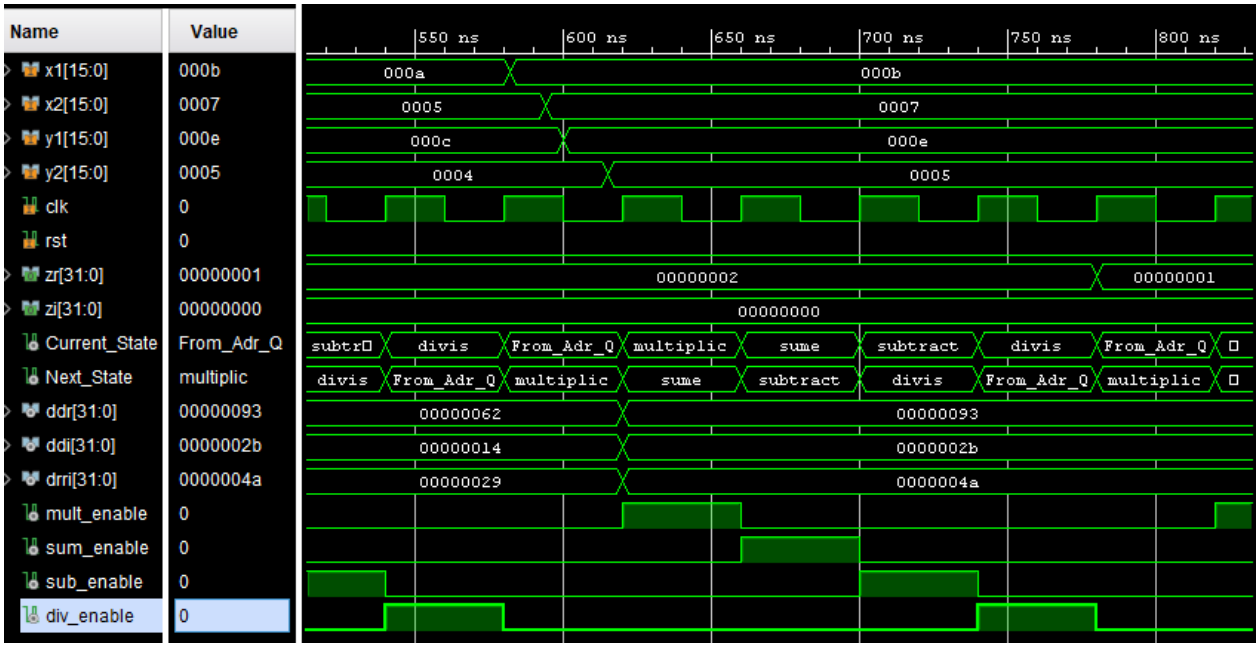


Рисунок 5.13 – Тестування АЛП при виконанні операції ділення, частина 1

На рисунках 5.14, 5.15 приведені результати тестування пристрою з паралельною роботою чотирьох процесорів, які одночасно виконують операції додавання, віднімання, множення і ділення комплексних чисел з видачою результатів відповідно на порти zr і zi, zr1 і zi1, zr2 і zi2, zr3 і zi3.

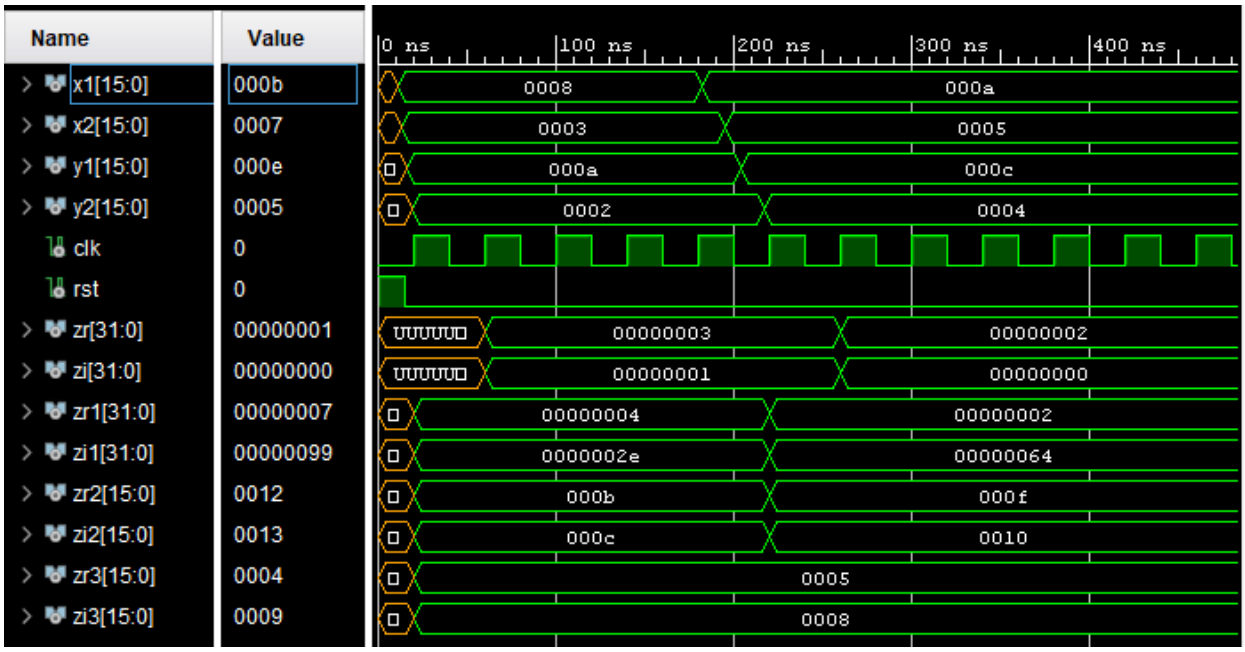


Рисунок 5.14 – Тестування АЛП при їх паралельній роботі, частина1

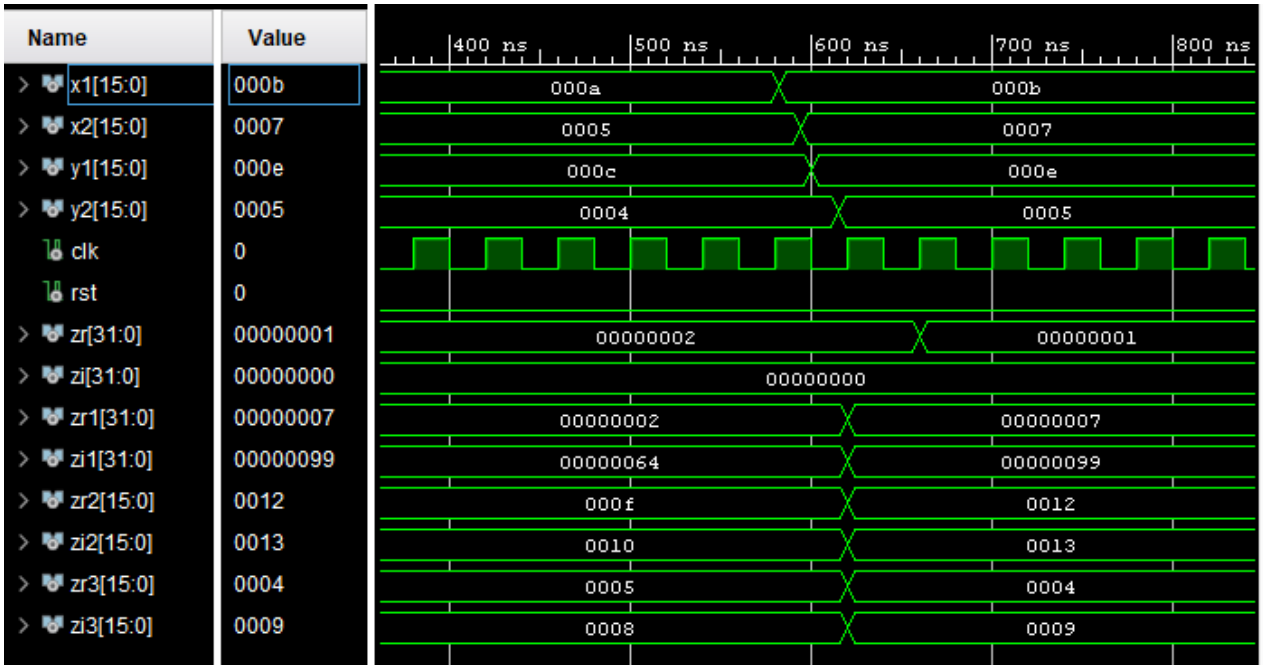


Рисунок 5.15 – Тестування АЛП при їх паралельній роботі, частина2

ВИСНОВКИ

У першому розділі був зроблений огляд суперкомп’ютерів, типів багатопроцесорних систем, класифікації М. Флінна і проблем когерентності та способів їх вирішення.

У другому розділі були розглянуті різні особливості ПЛІС, їх переваги над ASIC. Були проаналізовані програмні ядра на ПЛІС для багатопроцесорних систем та зроблено аналіз багатопроцесорної архітектури на ПЛІС.

У третьому розділі наведені сімейства ПЛІС від фірм AMD (XILINX) та Intel (ALTERA). Також були розглянуті найбільш популярні САПР і засоби високорівневого проектування на ПЛІС. Після чого була обрана ПЛІС сімейства Artix™ 7 та САПР Vivado Design Suite 2018.2 від фірми AMD.

У четвертому розділі була побудована структурна та функціональна схеми багатопроцесорної системи та одного процесора.

У п’ятому розділі були наведені етапи проектування на ПЛІС та створення проекту у Vivado 2018.2. Також були розглянуті блоки цифрової обробки сигналів, вчасності DSP48E1. Після чого на мові VHDL була реалізована операційна частина багатопроцесорної системи для виконанні арифметичних операцій над комплексними числами. Було проведено дослідження (шляхом моделювання) цієї операційної частини багатопроцесорної системи при послідовному і паралельному виконанні математичних операцій додавання, віднімання, множення і ділення комплексних чисел.

Результати дипломної роботи апробовані на конференціях [10, 11].

ПЕРЕЛІК ПОСИЛАННЬ

1. Multiprocessor Systems [Електронний ресурс] - Режим доступу: https://www.tutorialspoint.com/Multiprocessor-Systems (дата звернення: 03.06.2023).
2. Організація когерентності системи кеш-пам'яті в багатопроцесорних системах із загальною оперативною пам'яттю [Електронний ресурс] - Режим доступу: http://um.co.ua/3/3-3/3-38722.html (дата звернення: 09.07.2023).
3. Xilinx [Електронний ресурс] - Режим доступу: https://www.xilinx.com/products/silicon-devices/fpga.html (дата звернення: 09.07.2023).
4. Intel® FPGAs and SoC FPGAs [Електронний ресурс] - Режим доступу: https://www.xilinx.com/products/silicon-devices/fpga.html (дата звернення: 09.07.2023).
5. Design Tools to Develop Solutions on all AMD Adaptive Computing Platforms [Електронний ресурс] - Режим доступу: https://www.xilinx.com/products/design-tools.html (дата звернення: 09.07.2023).
6. Programming with MATLAB [Електронний ресурс] - Режим доступу: https://www.mathworks.com/products/matlab/programming-with-matlab.html (дата звернення: 09.07.2023).
7. Open Computing Language OpenCL [Електронний ресурс] - Режим доступу: https://developer.nvidia.com/opencl (дата звернення: 09.07.2023).
8. 7 Series DSP48E1 Slice User Guide (UG479) [Електронний ресурс] - Режим доступу: https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1 (дата звернення: 09.07.2023).

9. Опанасенко В. Н. Лисовий А. Н. Сахарин, В.Г. Реалізація арифметичних операцій із комплексними числами на ПЛІС. Технология и конструирование в электронной аппаратуре. 2008. № 5. С. 24-30.
10. Ванін М. В. Використання ПЛІС в багатопроцесорних системах / Всеукраїнська науково-технічна конференція студентів і молодих учених «Наука і сталий розвиток транспорту». – 2023.
11. Ванін М. В. Шаповалов В.О Використання ПЛІС в багатопроцесорних системах Міжнародна науково-практична конференція «сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті». – 2023.

Додаток А

VHDL-код операційної частини багатопроцесорної систем

```
library IEEE, work;
use work.ALL;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rt is
generic (N: integer:=16);
   Port (x1 : in std_logic_vector (N-1 downto 0);
      x2 : in std_logic_vector (N-1 downto 0);
      y1 : in std_logic_vector (N-1 downto 0);
      y2 : in std_logic_vector (N-1 downto 0);
      clk : in std_logic;
      rst:  IN std_logic;
      zr : out std_logic_vector (2*N-1 downto 0);
      zi : out std_logic_vector (2*N-1 downto 0);
      zr1 : out std_logic_vector (2*N-1 downto 0);
      zi1 : out std_logic_vector (2*N-1 downto 0);
      zr2 : out std_logic_vector (N-1 downto 0);
      zi2 : out std_logic_vector (N-1 downto 0);
      zr3 : out std_logic_vector (N-1 downto 0);
      zi3 : out std_logic_vector (N-1 downto 0));
end rt;

architecture Behavioral of rt is
Type State is(intl, multiplic, sume, subtract,divis,From_Adr_Q);
Signal Current_State, Next_State: State;
signal x1q, x2q, y1q, y2q :std_logic_vector(N-1 downto 0);
signal a1, b1, a2, b2 :std_logic_vector(2*N-1 downto 0);
signal ddr, ddi, drri :std_logic_vector(2*N-1 downto 0);
signal mult_enable, sum_enable, sub_enable, div_enable: STD_LOGIC;
component division
```

```
   PORT(dd : IN std_logic_vector(31 downto 0);
      dr : IN std_logic_vector(31 downto 0);
      ce : IN std_logic;
      clk : IN std_logic;
      rst:  IN std_logic;
      qt : OUT std_logic_vector(31 downto 0));
   --    rmd : OUT std_logic_vector(31 downto 0));
   END COMPONENT;

begin
St_Reg: Process(Clk,Rst)
        Begin
                If Rst = '1' Then
                        Current_State <= intl;
                Elsif Rising_Edge(Clk) Then
                        Current_State <= Next_State;
                End If;
        End Process;

Log_Next: Process(Current_State)
        begin
                Case Current_State is
                   when intl => Next_State <=multiplic;
                        when multiplic => Next_State <=sume;
                        when sume => Next_State <= subtract;
                        when subtract => Next_State <= divis;
                        when divis => Next_State <= multiplic;

                End Case;
End Process Log_Next;

  process(Current_State)
begin
     case Current_State is
```

```
        when intl =>
            mult_enable <= '0';
            sum_enable <= '0';
            sub_enable <= '0';
            div_enable <= '0';
        when multiplic =>
            mult_enable <= '1';
            sum_enable <= '0';
            sub_enable <= '0';
            div_enable <= '0';
        when sume =>
            mult_enable <= '0';
            sum_enable <= '1';
            sub_enable <= '0';
            div_enable <= '0';
        when subtract =>
            mult_enable <= '0';
            sum_enable <= '0';
            sub_enable <= '1';
            div_enable <= '0';
        when divis =>
            mult_enable <= '0';
            sum_enable <= '0';
            sub_enable <= '0';
            div_enable <= '1';
        when others =>
            mult_enable <= '0';
            sum_enable <= '0';
            sub_enable <= '0';
            div_enable <= '0';
    end case;
end process;

u1: division port map (dd=>ddr, dr=>drri, ce=>div_enable, clk=>clk, rst=>rst, qt=>zr);
```

```
u2: division port map (dd=>ddi, dr=>drri, ce=>div_enable, clk=>clk, rst=>rst, qt=>zi);
   ddr<=x1q*x2q+y1q*y2q;
   ddi<=x2q*y1q-x1q*y2q;
   drri<=x2q*x2q+y2q*y2q;
process (clk,rst)
       begin
          if rst = '1' then
             x1q<=(others=>'0'); x2q<=(others=>'0'); y1q<=(others=>'0'); y2q<=(others=>'0');
          elsif (clk'event and clk='1') then
             x1q<=x1; x2q<=x2; y1q<=y1; y2q<=y2;
          end if;
       end process;

multiplicationn: process (mult_enable)--
begin
   If mult_enable = '1' Then
      a1<=x1q*x2q;
      a2<=y1q*y2q;
      b1<=x2q*y1q;
      b2<=x1q*y2q;
      zr1<=a1-a2;
      zi1<=b1+b2;
   End If;
end process multiplicationn;

suman: process (sum_enable)
begin
   If sum_enable = '1' Then
      zr2<=x1q+x2q;
      zi2<=y1q+y2q;
   End If;
end process suman;

subtractionn: process (sub_enable)
```

```
begin
   If sub_enable = '1' Then
      zr3<=x1q-x2q;
      zi3<=y1q-y2q;
   End If;
end process subtractionn;

end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity division is
   Port(dd : IN std_logic_vector(31 downto 0);
      dr : IN std_logic_vector(31 downto 0);
      ce : IN std_logic;
      clk : IN std_logic;
      rst:  IN std_logic;
      qt : OUT std_logic_vector(31 downto 0));
end division;

architecture Behavioral of division is

begin
divisionn: process (clk,rst,ce)
variable cur_divident,R: std_logic_vector(31 downto 0) ;
variable cur_divisor: std_logic_vector(31 downto 0) ;
variable cur_qt: std_logic_vector(31 downto 0) ;

begin
   if rst='1' then
      cur_divident:=x"00000000" ;
```

```
        cur_divisor:=x"00000000" ;
        R:=x"00000000";
        cur_qt := x"00000000";
    elsif rising_edge(Clk) then
        if ce='1' then
            cur_divident := dd ;
            cur_divisor := dr ;
            cur_qt := x"00000000";
            R:=x"00000000";
                 for i in 31 downto 0  loop
                R := R(30 downto 0) & cur_divident(i);
              if R>=cur_divisor then
                R := R-cur_divisor ;
                cur_qt(i):='1' ;
              end if ;
              end loop ;
        end if ;
    end if ;
qt<=cur_qt;
end process divisionn;

end Behavioral;
```

Додаток Б

Test Bench

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity rt_tb is
end;

architecture bench of rt_tb is

  component rt
  generic (N: integer:=16);
     Port (x1 : in std_logic_vector (N-1 downto 0);
         x2 : in std_logic_vector (N-1 downto 0);
         y1 : in std_logic_vector (N-1 downto 0);
         y2 : in std_logic_vector (N-1 downto 0);
         clk : in std_logic;
         rst:  IN std_logic;
         zr : out std_logic_vector (2*N-1 downto 0);
         zi : out std_logic_vector (2*N-1 downto 0);
         zr1 : out std_logic_vector (2*N-1 downto 0);
         zi1 : out std_logic_vector (2*N-1 downto 0);
         zr2 : out std_logic_vector (N-1 downto 0);
         zi2 : out std_logic_vector (N-1 downto 0);
         zr3 : out std_logic_vector (N-1 downto 0);
         zi3 : out std_logic_vector (N-1 downto 0));
  end component;

  signal x1: std_logic_vector (15 downto 0);
  signal x2: std_logic_vector (15 downto 0);
  signal y1: std_logic_vector (15 downto 0);
  signal y2: std_logic_vector (15 downto 0);
```

```
  signal clk: std_logic := '0';
  signal rst: std_logic := '0';
  signal zr: std_logic_vector (31 downto 0);
  signal zi: std_logic_vector (31 downto 0);
  signal zr1: std_logic_vector (31 downto 0);
  signal zi1: std_logic_vector (31 downto 0);
  signal zr2: std_logic_vector (15 downto 0);
  signal zi2: std_logic_vector (15 downto 0);
  signal zr3: std_logic_vector (15 downto 0);
  signal zi3: std_logic_vector (15 downto 0);
  signal rdy: std_logic;
  signal err: std_logic;

  constant clock_period: time := 10 ns;
  signal stop_the_clock: boolean;

begin

  -- Insert values for generic parameters !!
  uut: rt generic map ( N   => 16  )
          port map ( x1  => x1,
                     x2  => x2,
                     y1  => y1,
                     y2  => y2,
                     clk => clk,
                     rst => rst,
                     zr  => zr,
                     zi  => zi,
                     zr1 => zr1,
                     zi1 => zi1,
                     zr2 => zr2,
                     zi2 => zi2,
                     zr3 => zr3,
                     zi3 => zi3 );
```

```
        Rst <= '1', '0' after 15 ns;


    x1 <= x"0008" after 11 ns, x"000A" after 182 ns, x"000B" after 582 ns;
    x2 <= x"0003" after 13 ns, x"0005" after 195 ns, x"0007" after 594 ns;
    y1 <= x"000A" after 16 ns, x"000C" after 204 ns, x"000E" after 600 ns;
    y2 <= x"0002" after 19 ns, x"0004" after 217 ns, x"0005" after 615 ns;

        clk<= not clk  after 20 ns;

END;
```

Додаток В

ВИКОРИСТАННЯ ПЛІС В БАГАТОПРОЦЕСОРНИХ СИСТЕМАХ

Ванін М.В., керівник доц. Шаповалов В.О.
Український державний університет науки і технологій

Багатопроцесорні системи на мікросхемах використовуються в комп'ютерних системах для досягнення більш високої продуктивності, а кількість процесорних ядер постійно збільшується.

Програмовані логічні інтегральні схеми (ПЛІС) дозволяють реалізувати реконфігуровані багатопроцесорні системи. При цьому нові серії ПЛІС мають ряд особливостей, які впливають на вибір архітектури і конфігурованих модулів. Застосування можливостей реконфігурування в багатопроцесорних системах значно розширює їх сфери використання і дозволяє оптимізувати для вирішення конкретних задач. Протягом останніх років розвиток технології ПЛІС просунувся з точки зору використовуваних ресурсів, швидкість обробки, споживання енергії та вартості. Це включає швидкий розвиток циклів, висока гнучкість і багаторазове використання, помірні витрати, легка модернізація (за рахунок використання абстрактного опису обладнання мови (VHDL)) і розширення функцій (за умови, що ПЛІС не вичерпано). Крім того, поточні ПЛІС дозволяють інтегрувати процесори з програмним ядром. Тобто ПЛІС може використовувати типові можливості процесора. На додаток до нових розробок, рішення на основі ПЛІС можна вигідно використовувати в існуючих, «природно вирощених» системах. «Ключовою особливістю» таких природних систем є те, що з часом оригінальну апаратну платформу було доповнено різними платами розширення. Як наслідок, такі системи важко обслуговувати та розширювати, а також чутливі до поломок компонентів. Ще одна проблема виникає через розподіл функціональних можливостей, що взаємодіють, оскільки потрібна передача (великих) обсягів даних. Це ще більше ускладнює систему, що у свою чергу, знижує гнучкість і надійність. Крім того, найстаріші компоненти обмежують подальші модифікації таких гетерогенних систем.

Багатопроцесорна система побудована на ПЛІС та реалізована з використанням паралельного потоку даних дозволяє зменшити апаратні витрати та підвищити продуктивність обробки даних.

Додаток Г

Використання ПЛІС в багатопроцесорних системах

Ванін М. В., Шаповалов В. О., Український державний університет науки і технологій

Протягом багатьох років розробники апаратного забезпечення покладалися на збільшення тактової частоти системи як на спосіб підвищення продуктивності. Однак цей підхід більше не є життєздатним, оскільки такі проблеми, як розсіювання тепла і тепловідводи, стали занадто складними для подолання. У пошуках більш простих способів підвищення продуктивності багатопроцесорні системи стають все більш поширеним рішенням. Багатопроцесорна система - це система з декількома процесорами, які можуть виконувати кілька процесів одночасно. З розвитком технологій з'явилася можливість інтегрувати цілі багатопроцесорні системи на одному кристалі. Такі системи називаються MPSoC (багатопроцесорні системи на кристалі).

На сьогоднішній день MPSoC є дуже привабливим рішенням в області вбудованих систем, дозволяючи вбудованим системам виконувати завдання в режимі реального часу і в той же час долати значні обмеження по енергоспоживанню і займаному простору.

У цьому контексті ПЛІС (програмовані логічні інтегральні схеми) стали новою та перспективною платформою для реалізації багатопроцесорних систем. ПЛІС дозволяє швидко створювати прототипи та досліджувати нові архітектури без проблем, пов'язаних з ASIC (інтегральна схема для конкретного застосування). Однак проектування на мові HDL (Hardware Description Language) займає багато часу, і альтернативою проектуванню на HDL є використання програмних процесорів у ПЛІС для побудови багатопроцесорних систем. Програмні процесори - це конфігуровані процесори, розроблені відповідно до дизайну ПЛІС. Десятки процесорів і більше можуть бути інтегровані в сучасну ПЛІС, що значно збільшує потужність паралельних обчислень.

Підтримка потужних інструментів, використання абстрактної мови опису апаратного забезпечення та застосування IP-ядер може призвести до значної економії часу та коштів при проектуванні.

Для математичних обчислень можуть використовуватися спеціалізовані багатопроцесорні системи на ПЛІС. Такі системи повинні підтримувати одночасне виконання інструкцій над декількома даними. З чого можна зробити висновок, що найбільш підходящими архітектурами для такого процесора є SIMD (single instruction — multiple data), або MIMD (multiple instructions — multiple data).

На сьогодні провідними компаніями у виробництві ПЛІС та САПР (системи автоматизованого проектування) є AMD (Xilinx) і Intel (Altera). Зараз основними інструментами розробки пристроїв на ПЛІС є САПР Vivado (Xilinx) і Quartus(Altera). Проектування виконується за допомогою низькорівневих мов проектування обладнання, таких як VHDL та Verilog. Існують також високорівневі інструменти проектування, які можуть значно скоротити час проектування. Наприклад, до таких інструментів відносяться система Matlab/Simulink з пакетом System Generator for DSP, або фреймворк OpenCL(Open Computing Language).

Ще одним моментом, який слід враховувати при використанні ПЛІС, є те, що опис апаратного забезпечення, тобто VHDL-код, залишається у розробника і, крім того, конфігураційні файли не можуть бути прочитаними третьою стороною. Це означає, що вся інформація про систему може залишатися конфіденційною.

Всі перераховані вище переваги є вагомими аргументами на користь використання багатопроцесорних систем на ПЛІС в промислових додатках.