Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра комп'ютерної інженерії та системного програмування

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Мультипроцесорна система загального призначення на основі топології «гіперкуб»
Назва теми

КвРКІ.170157.17.01.24 ПЗ
Шифр

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

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

Освітня програма *«Комп'ютерна інженерія»*
Назва

Виконав: студент IV курсу, група КІ–17–1 _____ О.В. Янчук
Підпис Ініціали, прізвище

Керівник _____ О.М. Березький
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ С.М. Лисенко
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної
Інженерії та системного
Програмування _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«____» червня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем

Кафедра Комп'ютерної інженерії та системного програмування

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорушенко

"11" 01 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Янчук Олександрі Віталіївні

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Мультипроцесорна система загального призначення на основі топології «гіперкуб»

Керівник проекту (роботи) Березький О.М., д.т.н., проф.

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

Затверджена наказом ректора університету від 05.02.2021 р. № 11

2. Строк подання студентом проекту (роботи) на кафедру 07.06.2021 р.

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

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

Дослідження предметної області та постановка задачі

Проектування мультипроцесорної системи загального призначення на основі топології «гіперкуб»

Програмно-апаратна реалізація та тестування мультипроцесорної системи загального призначення на основі топології «гіперкуб»

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

Загальна архітектура SMP-системи

Процесорний блок

Модуль когерентності кеш-пам'яті

6. Консультанти розділів дипломного проекту (роботи)

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

7. Дата видачі завдання « 11 » 01 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	При
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	11.01.2021	вик
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2021	вик
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2021	вик
4	Робота над розділом 2 – моделювання та проєктування мультипроцесорної системи	01.04.2021	вик
5	Робота над розділом 3 – програмно - апаратна реалізація мультипроцесорної системи	30.04.2021	вик
6	Оформлення пояснювальної записки згідно вимог	31.05.2021	вик
7	Попередній захист ВКР	02.06.2021	вик
8	Захист ВКР на засіданні ЕК	Червень 2021 року	

Студент _____ О.В. Янчук
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ О. М. Березький
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл.листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 170157.17.01.24 ПЗ	Пояснювальна записка	80		
			Графічні матеріали			
2		КвРКІ 170157.17.01.24 Е8	Загальна архітектура	1		
			SMP-системи			
3		КвРКІ 170157.17.01.24 Е8	Процесорний блок	1		
4		КвРКІ 170157.17.01.24 Е8	Модуль когерентності	1		
			кеш-пам'яті			

КвРКІ 170157.17.01.24 ВП

Зм	Арк	№ докум	Підпис	Дата
Розробив		Янчук		07.06.21
Перевір.		Березький		07.06.21
Н. контр.		Лисенко		07.06.21
Затв.		Говорущенко		07.06.21

Відомість проекту

Літера	Аркуш	Аркушів
У	1	1

ХНУ, КІ-17-1

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мультипроцесорна система загального призначення на основі топології гіперкуб».

Автор роботи: Янчук Олександра Віталіївна.

Керівник роботи: Березький Олег Миколайович.

Пояснювальна записка: 80 с., 13 рис., 7 табл., 4 дод., 50 джерел.

Графічна частина: 10 презентаційних слайдів.

МУЛЬТИПРОЦЕСОРНА СИСТЕМА ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ НА ОСНОВІ ТОПОЛОГІЇ «ГІПЕРКУБ»

Метою роботи є розробка мультипроцесорної системи загального призначення на основі топології гіперкуб.

У цій роботі була розроблена мультипроцесорна система загального призначення, на основі топології гіперкуб. Для розробки та моделювання роботи були використані такі компоненти як Quartus II, SOPC Builder та програмована плата Altera DE1-SoC. За основу системи було взято SMP-архітектуру, також у ході роботи було описано та запропоновано і частково розроблено рішення для усунення проблем «вузького місця», та когерентності кешу.

Підпис студента  Дата 04.06.2021р

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

SMP – симетрична мультипроцесорна система

ASMP – асиметрична мультипроцесорна система

ОС – операційна система

ПЗ – програмне забезпечення

ЕОМ – електронно-обчислювальна машина

RISC – reduced instruction set computer

CPU – central processing unit(центральний процесор)

ПЛІС – програмована логічна інтегральна схема

ЗСД – загальне сховище даних

МКК – модуль когерентності кеш-пам’яті

ISR – обробник переривань

АР – обчислювальні процесори

BSP – завантажувальний процесор

ВСТУП

Протягом усієї історії розвитку ЕОМ та обчислювальної техніки були спроби ввести загальну класифікацію, під яку попадали б усі комп'ютерні архітектури, але жодна з таких класифікацій так і не змогла охопити все різноманіття розроблювальних систем і не пройшла випробування часом. Проте ці спроби класифікації дали науці ряд широковикористовуваних термінів, які застосовуються і досі.

Будь яка ЕОМ, будь то суперкомп'ютерна система що використовується для надскладних обрахунків в сферах науки, чи звичний нам персональний комп'ютер, досягає найвищої продуктивності не лише завдяки використанню високошвидкісних елементів, а й можливості виконувати велике число операцій паралельно. І саме можливість паралельного виконання задач є основною, коли мова йде про пришвидшення роботи комп'ютера.

В сучасному світі актуальність цієї роботи дуже висока, адже сучасні технології, наука, розвиток суспільства ставить обширні та складні задачі, вирішення яких на однопроцесорних системах вимагає значних затрат часу. Сфера використання мультипроцесорних систем постійно розширюється і захоплює усе нові сфери науки, бізнесу, економіки та виробництва.

Саме тому однією з найважливіших тенденцій розвитку обчислювальної техніки полягає в активному запровадженні мультипроцесорних систем, та побудові цілих обчислювальних комплексів.

З появою багатоядерних процесорів, такі системи стали можливі навіть для використання в звичайних домашніх комп'ютерах, ноутбуках і уже навіть мобільних пристроях, хоча б з невеликою мультипроцесорною системою.

З'єднання великої кількості процесорів в одну єдину систему дозволяє досягти потужності, що в рази перевищує максимальну швидкодію класичних ЕОМ, по схемі фон Неймана.

Відповідно до того, як саме відбувається об'єднання процесорів у єдину систему і які ресурси використовуються, такі системи називаються паралельними комп'ютерами, мультикомп'ютерами або мультипроцесорними системами.

Такі ЕОМ потребують спеціальних операційних систем, в якості яких часто використовуються відомі ОС, але модифіковані, шляхом додавання до них спеціальних функцій зв’язку та синхронізації, а також більш скурпульозного підходу в програмуванні, адже саме програмне забезпечення дозволяє розкрити потужність таких систем на максимум.

Метою цієї роботи є створення саме мультипроцесорної системи загального призначення на основі топології «гіперкуб». У роботі буде проаналізовано різні архітектури, що можуть використовуватись в системах з загальною пам’яттю, проведено аналіз наявних архітектурних рішень. У другому та третьому розділах буде описано проектування та реалізацію такої системи з детальним обгрунтуванням вибору та застосування апаратно-програмної бази.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

У сучасному світі велику частину сфери покращення обчислювальної здатності комп'ютерів зайняла багатопроцесорність. Багатопроцесорність – це використання більше одного центрального процесора(CPU) у одній машині. Такий підхід дозволяє значно розширити можливості комп'ютера. Наприклад розподілити завдання між різними процесами, що призведе до зменшення часу, що затрачається на вирішення тих чи інших задач.

Вважається що вперше модель мультипроцесорної архітектури було описано в 1843 році італійським математиком та інженером Луїджі Федеріко Менабреа.[1] «Схожим чином, можна дати в роботу машині декілька задач одночасно, що значно пришвидшить час проведення обчислень» на ці слова його надихнуло вивчення Аналітичної машини розробленої Чарльзом Бєбіджем.

Проте терімін багатопроцесорність часто використовується у значенні виконання декілька процесів паралельно не залежно виконуються вони на одному CPU чи більше[2,3], тобто цей термін часто протиставляється терміну «багатозадачність» - можливість використовувати лише один процесор але працювати на різзних ядрах або перемикати процесор між завданнями в певних часових зрізах(фактично система розподілу часу).

У цій роботі мультипроцесорість розглядається з точки зору апаратного аспекту [4,5], тобто фізичної наявності більше одного центрального процесора і вони усі використовують спільну пам'ять.

Далі важливо визначитись з таким, уже програмним рішенням, як ранг та призначення кожного з процесорів, адже саме це визначає підтип цієї системи – тобто її симетричність або асиметричність.

У мультипроцесорній системі всі CPU можуть бути однаковими[6], а знчить рівнозначними, або певні процесори можуть бути зарезервованні під окремі цілі.

Наприклад один певний процесор буде реагувати не апаратне переривання, в той час як уся система буде виконувати задачі на інших процесорах і розподілиться між ними рівномірно[7]. Також процесори можуть ділитись між типами виконуваного коду, наприклад один процесор буде виконувати код ядра, в той час як інші обслуговуватимуть режим користувача.

Для проектування простішими є використання різнорангових процесорів, проте така архітектура є менш ефективною ніж рівнозначне використання усіх центральних процесорів[8].

Виділяють чотири типи архітектури[9, 11, 12]:

1) SMP – системи де усі процесори є рівнозначними і однаково обробляють команди;

2) ASMP – системи де процесори не є рівнозначними, а системні ресурси розподіляються різним чином;

3) NUMA – архітектура з неоднорідним доступом до пам’яті (використання адресного простору;

4) qq.v. – кластеризована багатопроцесорна обробка.

Згідно мого завдання допускається використання як симетричної так і асиметричної мультмипроцесорності. Далі про них детальніше.

Асиметрична багатопроцесорна система – це мультипроцесорна система, де всі зв’язані централі процесори обробляються неоднаково[10, 13]. Тобто у ASMP завдання від операційної системи виконує лише головний процесор. Така система може бути використана, наприклад для призначення конкретних задач CPU на основі пріоритету та важливості їх виконання[14]. Схематично робота такої системи представлена на рисунку 1.1

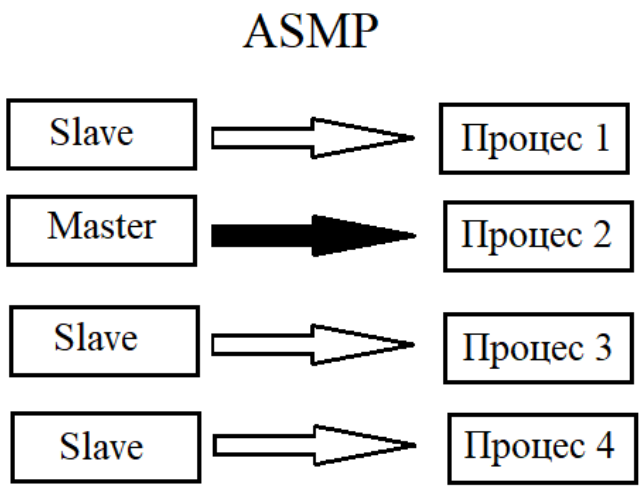


Рисунок 1.1 – принцип роботи ASMP

Асиметрична мультипроцесорна система включає багатопроцесорну програмно-апаратну архітектуру, де використовувані процесори підключені до єдиної спільної пам’яті, мають до неї повний доступ та здатні до самопланування.[6, 15] Схематично робота такої системи представлена на рисунку 1.2

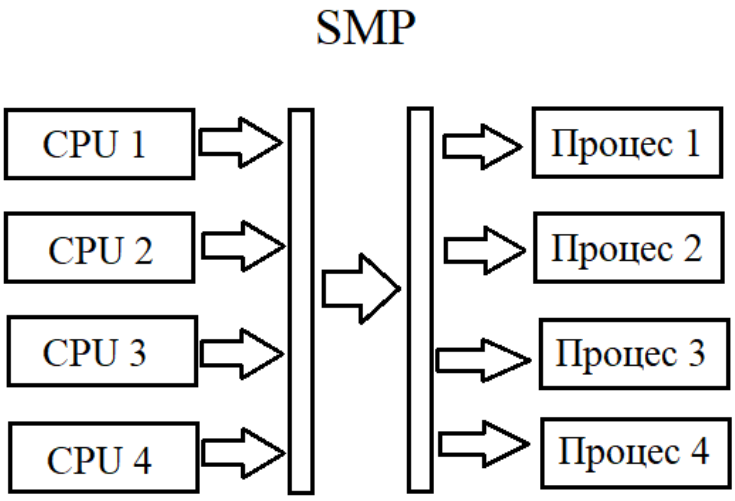


Рисунок 1.2 – Принцип роботи SMP

Таблиця 1.1 – Переваги та недоліки ASMP та SMP[16,17,18]

ASMP	SMP
Процесори не рівнозначні	Процесори рівнозначні
Завдання операційної системи виконує головний процесор.	Завдання операційної системи виконуються будь-яким процесором
Між процесорами зв’язок відсутній, оскільки ними керує головний процесор	Процесори спілкуються між собою за допомогою спільної пам'яті.
В асиметричних мультипроцесорних системах процеси працюють за принципом головний-дочірній (master-slave)	При симетричній мультипроцесорній обробці процес береться із готової черги.
Асиметричні мультипроцесорні системи дешевші.	Симетричні мультипроцесорні системи дорожчі.
Асиметричні мультипроцесорні системи легше спроектувати	Симетричні мультипроцесорні системи складні в проектуванні

При асиметричній мультипроцесорній обробці процесори не є рівнозначними.[19] При симетричній мультипроцесорній обробці процесори рівнозначні.[20,21]

Між процесорами зв’язок відсутній, оскільки ними керує головний процесор Процесори спілкуються між собою за допомогою спільної пам'яті[22].

В асиметричних мультипроцесорних системах процеси працюють за принципом головний-дочірній (master-slave)[23,24]. При симетричній мультипроцесорній обробці процес береться із готової черги.

Асиметричні мультипроцесорні системи дешевші. Симетричні мультипроцесорні системи дорожчі[25].

Асиметричні мультипроцесорні системи легше спроектувати. Симетричні мультипроцесорні системи складні в проектуванні[26]. Було обрано саме SMP, оскільки така архітектура є більш ефективною.

Симетричні мультипроцесорні системи включають в себе два або більше процесорів, що використовують одну, спільну для всіх, пам’ять; мають повний доступ до пристроїв вводу-виводу; працюють під управлінням однієї операційної системи [27,28].

Процесори можуть бути окремими мікросхемами або ядрами одного процесора(в такому випадку кожне ядро розглядається як окремий процесор)[29]. Цей вид є тісно пов’язаною багатопроцесорною системою з пулом однакових процесорів, які працюють незалежно один від одного[30]. Кожен процесор, працюючи над різними задачами, має можливітсть спільного використання загальних ресурсів, які, найяастіше, підключенні завдяки шині.

На рисунку 1.3 показана типова організація SMP машини. Загальна шина забезпечує зв’язок між процесорами та іншими елементами системи. Такий спосіб з’єднанан є одним з найпростіших і найвигідніших, проте він обмежує кількість процесорів, які можна використовувати [32,33].

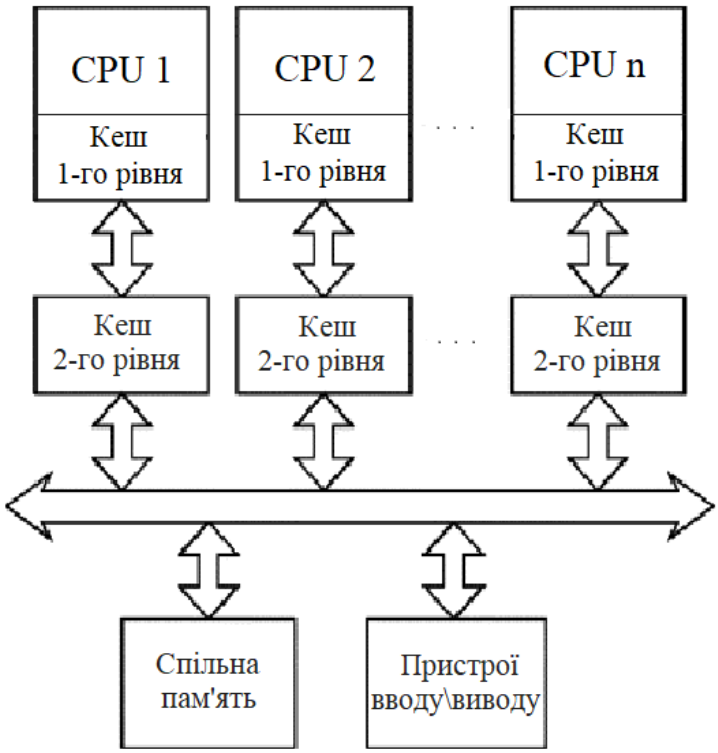


Рисунок 1.3 – загальна схема симетричної багатопроцесорної системи[31]

Так як усі запити до пам’яті та пристроїв вводу-виводу проходять через загальну шину, продуктивність системи обмежена максимальною пропускною здатністю використовуваної шини.

Одним із способів покращення роботи і усунення «вузького місця (Bottleneck)» [34, 35] у шині є активне використання кеш-пам’яті. Для цього потрібно використовувати, принаймі два рівні кеш-пам’яті, як показано на рисунку 1.1. Зазвичай кеш-пам’ять першого рівня знаходиться на тій ж мікросхемі, що й процесор, а кеш другого рівня є внутрішнім у самого процесора[37].

Кеш-пам’ять у SMP створює проблему когерентності кеш пам’яті[38], адже як можна гарантувати, що при зміні данних у певному кеші ця ж зміна відображається в інших та в основній пам’яті.

1.2 Аналіз наявного програмно-апаратного забезпечення предметної області

Комп’ютери з декількома процесорами існують уже десятки років, але саме зараз спостерігається швидкий ріст їх популярності. Зважаючи на їх високу вартість, та складність програмування під SMP системи, комп’ютерів, використовуючих таку технологію, не так багато. Найвідоміші та найцікавіші з нашої точки зору є наступні системи[39]:

1. Система Exemplar SPP2000, виробництва HP/Convex.
2. IBM Summit і IBM Sierra.
3. Процесори консолей Tegra від третього покоління і старше компаніїї NVIDIA.

Система Exemplar SPP2000. На сьогоднішній день в сімействі Exemplar анонсовано дві моделі суперкомп’ютерів

1. «S class" - 4-16 CPU, 256 Мбайт - 16 Гбайт пам’яті.
2. "X-class" - 4-64 CPU, 1 Гбайт- 64 Гбайт памяти.

При розробці цих двох комп’ютерів, за основу бралася можливість компромісного вирішення двох задач:

1) задовільнити зростаючі вимоги до обчислювальної потужності комп'ютерів;

2) зберегти усі переваги персонального комп'ютера, а саме: зробити можливим поступове нарощення потужності, проста модель програмування, сумісність з різними платформами, наявність графічного інтерфейсу.

У цих високопродуктивних архітектурах велику роль грає забезпечення масштабованості ресурсів системи, а також простота використання та мобільність такої системи [40].

В класичній моделі за основу береться архітектура SMP, яка забезпечує однорідність ресурсів, а уже масштабованість забезпечується за допомогою апаратної комутації компонентів системи, що побудовані по модульно[41]. В класичних SMP-системах, всі частини системи з'єднувались за допомогою системної шини, а суперкомп'ютерах, напромя – кабелем CrayLink. Обидва підходи мають значні недоліки.

Пропускна можливість системної шини встановлює верхню межу продуктивності усієї системи, а значить – межу масштабованості. Незважаючи на появу потужних шин, таких як SGI PowerPath2 (1.2 Гбайт/с), Sun Gigaplane (2.6 Гбайт/сек), межа все ще існує. В суперкомп'ютерах гарно себе показали прямі точкові з'єднання, проте вони дорогі і при збільшенні кількості компонентів їх число зростає комбінаторно.

Компанія Convex – це один з перших виробників що не пішли по жодному шляху остаточно, а використали дворівневу підтримку організації пам'яті NUMA, з підтримкою когерентності данних на на різних рівнях. Це відбулось ще 1994 році, в першому поколінні сімейства Exemplar - SPP1000.

Для забезпечення однорідного доступу до ресурсів було застосовано матричний комутатор – кроссбар (на першому рівні) та мережа обміну CTI (на другому)[42]. Таке рішення, наразі, стало загальним для системи класів суперкомп'ютерів та суперкомп'ютерних серверів: Sequent NUMA-Q, SGI Origin и Onyx2.

Платформи Exemplar S/X – це уже четверте та п'яте покоління масштабованих систем. Варто відмітити, що архітектура, яка була закладена ще в

перших поколіннях SPP1X00, виявилась досить вдалою та спроможною до адаптації для процесів, що потребують вищої потужності.

Поява кожного нового покоління SPP, фактично, базувалось на процесорі PA-RISC: PA-7100, PA-7200, PA-8000. Характерною особливістю систем Exemplar є те, що вони зберігають повну бінарну сумісність з попередніми поколіннями. Для Exemplar S/X це також не є винятком.

Детальніше щодо архітектури. У системі Exemplar SPP2000 було збережено основні риси архітектури SPP1X000. Як і раніше у ній можна виділити три рівня:

1) процесор (CPU);
2) гіпервузол;
3) кластер гіпервузлів.

Одна з основних задач – масштабованість у SPP2000 забезпечується саме на рівні гіпервузлів і підтримується двома апаратними засобами: матричним комутатором и мережею обміну.

Платформи Exemplar S/X зберігають просту модель програмування, характерну для SMP, але залишаються системами паралельних мультипроцесорів з високим рівнем масштабованості. В загальному, систему можна розглядати як комплекс з певної кількості MIMID-процесорів (число в межах від 4 до 64), що розділяє однорідну фізичну пам’ять та пристрої вводу\виводу.

Оскільки фізично пам’ять розподілена по різним частинам системи, така модель підтримується програмно і апаратно підсистемою управління загальною глобальною пам’яттю, яка в термінології Convex отримала назву GSM - фактично, це реалізація загального підходу до архітектури дворівневої пам’яті CC-NUMA[43]. GSM автоматично розподіляє данні і підтримує їх узгодженість. Такий підхід дозволяє забезпечити віртуальну однорідність пам’яті, а також звільняє програміста від потреби ручного управління процесами пересилки даних і їх оновлення в різних частинах системи.

IBM Summit і IBM Sierra. IBM Summit – суперкомп’ютер, розроблений компанією IBM для Окриджської Національної лабораторії. Він був запущений в використання в червні 2018 року, замінивши попередника Titan.

IBM Sierra – суперкомп’ютер, встановлений в Ліверморській національній лабораторії для користування Національним управлінням ядерної безпеки. Основним чином він використовується для прогнозування та управління ядерним персоналом. Цей комп’ютер створено для забезпечення надійності, безпеки і ефективності ядерної зброї у США.

Комп’ютер був розроблений в рамках спільного научного проекту трьох лабораторій:

1) національної лабораторії Ок-Ридж;
2) аргонської національної лабораторії;
3) ліверморської національної лабораторії.

Суперкомп'ютер IBM Sierra зайняв місце третє місце у рейтингу Топ-500 у редакціях за червень 2018 р. [44], потім піднявся до другої сходинки у версії за листопад 2018 р. [44]. У рейтингу Green 500 суперкомп'ютер займав шосту позицію станом на листопада 2018 р.

Суперкомп'ютер IBM Summit зайняв перше місце у рейтингу Топ-500 в окремих редакціях 2018 року [47]. У рейтингу Green 500 IBM Summit став на третю сходинку рейтингу в редакції листопада 2018 року.

Суперкомп'ютери IBM Summit та IBM Sierra, складаються з таких базових компонентів:

1) фрейми (frames);
2) стійки (racks);
3) вузли (nodes);
4) мережу (network);
5) файлову систему (file system);
6) архівне зберігання HPSS (archival HPSS Storage).

Вузли (ноди) існують декількох типів :

1) обчислювальні вузли (compute nodes);
2) вузли входу та запуску (login / launch nodes);
3) вузли введення-виведення (I / O nodes);
4) службові вузли (service / management nodes).

Обчислювальні вузли безпосерсдньо виконують саме обчислювальні задачі. Фізично це двухсокетні вузли IBM POWER9 (AC922), кожен з таких вузлів містить декілька графічних відеокарт NVIDIA Tesla V100. А саме таких відеокарти є 4 шт. в IBM Sierra [45] і 6шт. відповідно в IBM Summit [46].

Для забезпечення віддаленого доступу використовуються вузли входу і запуску. Ці вузли виконують задачу компіляції програм і додавання їх в чергу завдань. Ці вузли є загальними для всіх користувачів і тому не використовуються для запуску паралельних завдань. Так само як і обчислювальні вузли, вузли входу і запуску так само, мають два сокета IBM POWER9 і чотири графічних відеокарти NVIDIA Tesla V100.

Вузли вводу-виводу – це файлові сервери, вони складають паралельну файлову систему IBM Spectrum Scale parallel file systems. Користувачі цього комп'ютера не мають прямого доступу до цих вузлів. Вищеописані вузли є двухсокетнимі вузлами IBM POWER9, проте не мають графічних відеокарт.

Для системних процедур, зарезервовані службові вузли, вони аналогічні вузлам введення-виведення, не забезпечують прямого доступу для користувачів і мають два сокета IBM POWER9 також без графічних відеокарт.

Суперкомп'ютер IBM Sierra використовує мережу від Mellanox: Enhanced Data Rate (EDR) InfiniBand [44, 45]. Пропускна здатність якої становить 100 Гб / с.

Мережа такого типу використовується для внутрішніх комунікацій (зокрема і MPI) в рамках одного вузла, для трафіку введення-виведення між вузлами введення-виведення і обчислювальними вузлами, а також для доступу до інших кластерів та паралельним файлових серверів цього обчислювального комплексу.

Використовувана в комп'ютері мережа іншого типу - GigE, використовується для з'єднання мереж InfiniBand, HPSS, систем зберігання і зовнішніх мереж.

Основною файловою системою IBM Summit і IBM Sierra є паралельна файлова система IBM Spectrum Scale. В інших кластерах мультикомп'ютер використовує файлову систему Lustre. Окрім цього, для домашніх директорій користувача, інфраструктурних сервісів і тимчасових файлів використовується NFS.

Далі розглянемо кількісні характеристики обчислювальних систем IBM Sierra та IBM Summit.

Кількісні характеристики IBM Sierra:

1) загальна кількість вузлів – 4 474, з них:
   a) обчислювальні вузли – 4 320;
   b) вузли входу – 5:
2) загальна кількість ядр 1 572 480, з них:
   a) CPU ядр – 190 080
   b) загальна кількість GPU – 17 280;
3) об’єм RAM – 1 382 400 GB.

Кількісні характеристики IBM Summit:

1) загальна кількість вузлів – 4 608;
2) загальна кількість ядр – 2 397 824, з них:
   a) CPU ядр - 9 216;
3) загальна кількість GPU – 27 648.

Дуже важливим є і програмне середовище, в суперкомп’ютерних системах IBM Sierra та IBM Summit однакове, та має наступні характеристики:

1) ОС - Red Hat Enterprise Linux 7.4;
2) компілятори: IBM XLC, NVCC;
3) математичні бібліотеки: ESSL, CUBLAS 9.2;
4) MPI - IBM Spectrum MPI

Які ж у цих систем характеристики продуктивності і потужності? Згідно офіційної сторінки IBM Sierra, ця система має наступні характеристики[45]:

1) Linpack (Rmax) – 122 300.0 TFlop/s;
2) CPU: 4 666 TFlops;
3) GPU: 120 960 TFlops;
4) споживна потужність 7 438.28 kW;
5) енергоефективність 12.723 Gflops/W.

Характеристики потужності IBM Summit:

1) Linpack (Rmax) – 187 659.3 TFlop/s;
2) споживна потужність 9 783.00 kW;

3) енергоефективність 14.668 Gflops/W.

Процесори консолей Tegra від третього покоління і старше компаніїї NVIDIA.

Компанія NVIDIA також не лишається осторонь від SMP технології, але вона внесла свої зміни в її архітектуру.

Починаючи від консолі Tegra 3 компанія вносить в архітектуру процесора так звану «змінну SMP» (Variable SMP), скорочено – vSMP це особлива технологія використання процесорів в мобільних пристроях. Ця технологія має за собою включення п’ятого, додаткового, ядра в чотирьохядерному пристрої, що отримало назву Companion.

Проект Kal-El (Tegra 3)[48], який запатентувала NVIDIA, став першим SoC (System on Chip), що впровадив у свої процесори цю нову технологію – vSMP.

Ця технологія дозволяє максимізувати продуктивність чотирьох’ядерного процесора для використання програм, що потребують багато ресурсів, що в свою чергу, додатково, зменшує споживання енергії. Загалом це вирішує проблему збільшення автономної роботи пристрою, шляхом зменшення енергоспоживання процесора.

Помітною відмінністю SMP від vSMP є те, що ядро Companion є прозорим для операційної системи, а отже ОС та запущені програми, навіть не знають про це додаткове ядро, хоч і використовують його.

Додатков vSMP-архітектура вирішує або частково вирішує деякі проблеми архітектури SMP, детільніше про кожну з них:

1) когерентність кеш-пам’яті (не потрібно синхронізувати кеш-пам’ять на ядрах, оскільки ядро Companion не запускається одночасно з іншими чотирма ядрами);

2) ефективність роботи ОС: завдяки vSMP активні ядра процесора працюють на однакових частотах, що дозволяє оптимізувати планування роботи операційної системи;

3) оптимізація живлення: технологія vSMP здатна до енергозбереження, не лише завдяки описаному вище методу оптимізації продуктивності, а й завдяки

спроможності вимикати певні ядра для активного чи резервного використання, що дозволяє зменшити загальне використання енергії.

Отже, архітектура vSMP вирішує певні проблеми архітектури SMP, проте не можна сказати, що одна з архітектур здатна повністю замінити іншу. Адже в кожної з них є свої переваги та недоліки.

1.3 Постановка задачі

Метою цієї роботи – є проектування мультипроцесорної системи загального призначення на основі топології гіперкуб. Робота виконуватиметься у три етапи:

1) пошук інформації та аналіз наявних рішень;

2) проектування системи, підбір апаратних засобів, розробка детальної схеми організації апаратної частини; аналіз та вибір програмно-керуючого засобу;

3) налаштування програмно-керуючого засобу; аналіз спроектованої системи; способи покращення системи.

При проектуванні мультипроцесорної системи на основі топології гіперкуб, потрібно обов’язково врахувати, що елементи системи (процесори, плати, шина, пам’ять) повинні бути обґрунтовано підібрані, адже це дозволить отримати максимальну продуктивність системи. Також компоненти системи повинні мати гарне співвідношення ціна-потужність\продуктивність.

Також важливо правильно підібрати програмний засіб управління створеною системою, який забезпечить зручність її використання, розкриття можливостей апаратної частини та можливість налаштування створеної мультипроцесорної системи.

1.4 Висновки

У цьому розділі було порівняно два типи архітектури мультипроцесорних систем – симетричної та асиметричної. Шляхом порівняння та аналізу, для розроблювальної мультипроцесорної системи загального призначення, було обрано SMP архітектуру, оскільки при такому проектуванні усі процесори будуть

рівнозначні, матимуть змогу спілкуватись між собою і така система забезпечує кращу продуктивність, хоч і складніша в розробці.

Також проведено аналіз уже існуючих рішень, як і більш простих, схожих до того яке буде розроблено у цій роботі, так і максимально складних серверних систем на базі мультипроцесорів.

Визначено мету, цілі та кроки виконання роботи. Детально описано та сформульовано завдання.

2 ПРОЄКТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

2.1 Топологія гіперкуб

Гіперкуб (N-куб) – це мережа з 2^N вузлів, розташованих в вершинах n-мірного куба. Гіперкуб це узагальнення звичайного куба тривимірної форми.

Куб 2^0- це точка, для того щоб отримати куб 2^1 копіюється 0-куб і розміщується ця копія паралельно на відстані умовної одиниці та з'єднуємо ці копії за допомогою відрізка довжини 1. Щоб отримати гіперкуб розмірності 2^2, потрібно виконати ті ж кроки, так для куба будь-якої степені. Іншими словами n-куб будується з N парами паралельних (N-1) – площин, тобто має 2N гіперграні, кожна з яких сама по собі є (N-1)-кубом

Тобто, N-куб – це фігура, кожна вершина якої пов'язана за допомогою ребер з N іншими вершинами; в свою чергу число N визначає розмірність фігури.

Вузли гіперкуба позначаються двійковими числами, кількість бітів числа визначається степенем куба. На рисунку 2.1. зображено різні форми гіперкуба.

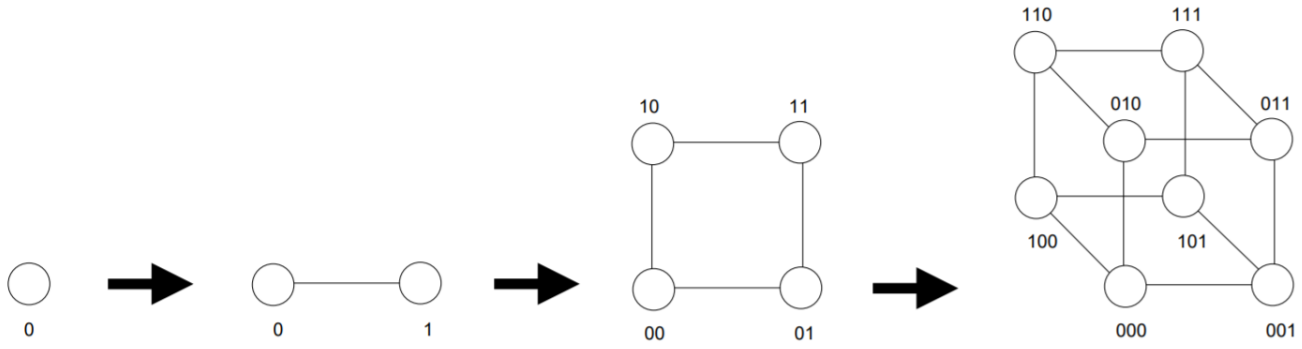


Рисунок 2.1 – Гіперкуб нульової, першої, другої та третьої степенів відповідно

Топологією для створення мультипроцесорної системи слугуватиме гіперкуб четвертого порядку, який будується способом, описаним вище на базі трьохвимірного n-куба і зображений на рисунку 2.2.

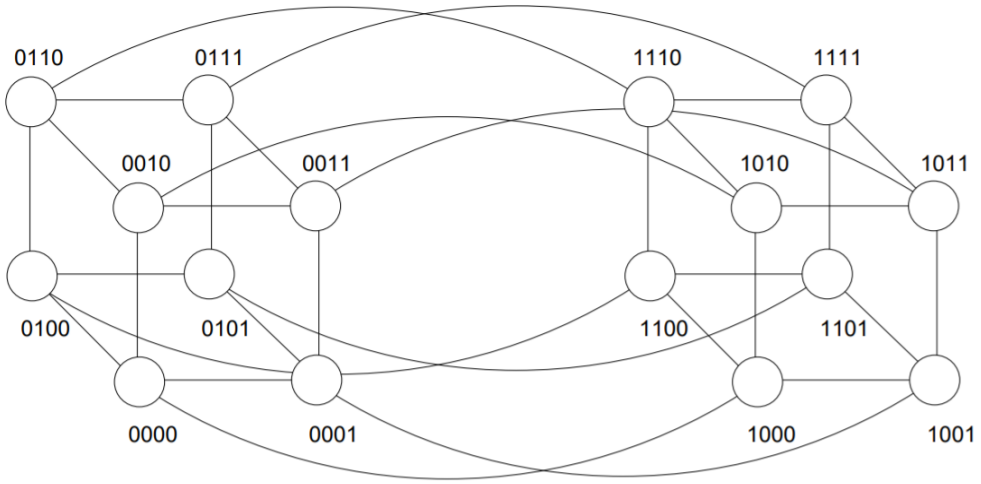


Рисунок 2.2 – 4-х вимірний гіперкуб

Найбільш значущими характеристиками для побудови мультипроцесорної системи, є часові затримки при передачі даних по каналах зв'язку для організації взаємодії окремих обчислювальних блоків (у нашому випадку – процесорів) в значній мірі впливають на ефективність паралельних обчислень.

Основні характеристики топологій:

1) діаметр – максимальна відстань між двома процесорами(у випадку складних топологій за значення береться мінімальна можлива відстань). Данна величина дозволяє охарактеризувати максимальний час передачі данних від одного процесора до іншого;

2) зв'язність (connectivity)– показник, що характеризує наявність різних маршрутів передачі даних між процесорами; за значення можна взяти, наприклад мінімальну кількість дуг, які потрібно видалити для поділу однієї мережі на дві;

3) ширина бінарного поділу (bisection width) – мінімальна кількість дуг, яку потрібно видалити для поділу однієї мережі, на дві непов'язаних мережі одного розміру;

4) вартість – показник, що визначається, наприклад, загальою кількістю ліній передачі данних в мультипроцесорній системі.

Оскільки у роботі використовується топологія гіберкуб, приведемо характеристики саме цієї топології:

1) діаметр – log p;

2) зв’язність (connectivity) – log p ;

3) ширина бінарного поділу (bisection width) – p / 2;

4) вартість – (p log p) / 2.

Алгоритми маршрутизації визначають шлях від процесора до процесора між якими повинно бути доставлено повідомлення. До найбільш поширених оптимальних алгоритмів відносяться методи покоординатної маршрутизації (dimension-ordered routing), в якій пошук шляху передачі визначається окремо для кожної розмірності топології.

У гіперкуба покоординатна схема маршрутизації – це циклічна передача даних процесору, що визначається першою відмінної бітовою позицією в номерах процесорів, на якому повідомлення знаходиться в данний момент часу і на який повідомлення повинно бути передано.

Також важливими характеристиками є передачі данних між двома процесорами мережі та передача даних від одного процесора всім іншим процесорам, що знаходяться в мережі.

Час передачі данних між двома процесорами у топології гіперкуб:

– передача повідомлень (формула 2.1):

$$t_н + mt_к \log p, \tag{2.1}$$

– передача пакетів (формула 2.2):

$$t_н + mt_k + t_c \log p\,. \tag{2.2}$$

У випадку передачі данних від одного процесора до всіх інших (one-to-all broadcast or single-node broadcast) все складніше.

При передачі повідомлень надсилання виконується в N етапів. На першому етапі процесор, що передає повідомлення передає данні свому сусідові(наприклад по першій розмірності). В результаті виконання цієї операції в мережі є два

процесора, що мають копію даних, для розсилки. В другому етапі уже два цих процесора розсилають повідомлення своїм сусідям другої розмірності і т.д.

В результаті таких операції надсилання час розраховується за формулою 2.3:

$$t_{пд} = (t_н + mt_к)\log p. \quad (2.3)$$

Якщо порівнювати з іншими топологіями, то гіперкуб має найкращі показники по затраченому часу на виконання операції розсилки. Більше того можна вважати що гіперкуб є найкращою топологією для данного виду розсилки повідомлень.

При передачі пакетів алгоритм і часові характеристики не відрізняються.

Передача даних усім процесорам до усіх процесорів мережі (all-to-all broadcast або multinode broadcast). На кожному етапі i, $1 \le I \le N$, виконання алгоритму задіяні всі процесори, які обмінюються своїми даними зі своїми сусідами і-тої розмірності і формують об'єднанні повідомлення.

$$t_{пд} = \sum_{i=1}^{\log p} \left(t_н + 2^{i-1} mt_к\right) = t_н \log p + mt_к(p-1) \quad (2.4)$$

2.2 Кеш-пам'ять у симетричних системах. Проблема когерентності та узгодженості пам'яті

Технологія SMP, завдяки своїй архітектурі, дозволяє виконувати велику кількість інструкцій і обробляти потоки даних паралельно. Кожен елемент такої системи, у нашому випадку – процесор, самостійно виконує інструкції зі свого власного потоку даних.

Мультипроцесорна система на архітектурі SMP може працювати і як типовий персональний комп'ютер, тобто фокусуватись на одному завданні, так і

виконувати кілька завдань одночасно, як багатопроцесорна машина. Системи SMP діляться ресурсом пам’яті через спільну шину, а отже елементи процесор-шина-пам’ять мають мати гарну сумісність, оскільки загальна швидкодія системи залежить від одного компонента з найменшою швидкодією.

Симетричність системи базується на трьох аспектах, кожен з яких є однаково важлививм:

1) усі процесори функціонально ідентичні, та ієрархічно рівні;

2) усі процесори знаходяться в одному адресному просторі;

3) усі просцесори мають однаковий доступ до підсистем вводу-виводу, а також до переривань.

Така симетричність допомагає запобігти утворенню критичних вузьких місць (bottlenecks), а також допомагає стандартизувати програмне забезпечення, завдяки чому розробники зможуть створювати системи з різною кількістю процесорів з однаковими виконуваними файлами.

Для того, щоб мати змогу вибрати основний процесор для ініціалізації під час запуску та дозволити програмному забезпеченню, в нашому випадку - операційній системі, призначати процеси і потоки для кожного з процесорів SMP-системи потребують засобів для одночасної ідентифікації процесорів.

Унікальна ідентифікація процесорів – це питання буде вирішено в ході роботи.

Як уже згадувалось, через кількість процесорів системна шина стає вузьким місцем у продуктивності системи. Для хоча б часткового вирішення цієї проблеми можна використовувати один або декілька рівнів кеш-пам’яті, таке рішення дозволє підвищити продуктивність процесора, навіть у однопроцесорних системах.

Така кеш-пам’ять складається з швидкодійної пам’яті, що є проміжними рівнем між процесором і основною пам’яттю. Вона використовується для того, щоб зменшити затримку шляхом повторення запитів до місця у пам’яті. Перевага кешу полягає в оптимальному використанні пам'яті та часу виконання коду за рахунок зберігання нещодавно використаних блоків пам'яті.

Коли процесор отримує доступ до кешованої пам’яті, він може отримати та передати данні, і жодна з таких дій не використовує шину пам’яті, такий підхід допомагає зменшити навантаження на шину, а отже покращити роботу системи в цілому. У порівнянні зі швидкістю роботи сучасних процесорів, основна пам’ять дуже повільна. Кеш допомагає підсистемі пам’яті передавати інструкціх та дані, до процесора з потрібною швидкістю, таким чином зменшуюючи простій.

На жаль, кешування теж має свої недоліки. Швидкість кеш-пам’яті – прямий результат збільшення кількості транзисторів, що використовуються для реалізації кожного біта накопичувача. Тобто архітектура кеш-пам’яті виключає можливість проектування великооб’ємної пам’яті для кожного з процесорів. Це призводить до того, що більшість систем, використовують ієрархічну підсистему організації пам’яті. Кеш – це швидка та малооб’ємна пам’ять. Сьогодні пристрої пам’яті стають більшими і повільнішими, така тенденція спостерігається від процесорною до основної пам’яті, включаючи магнітні жорсткі диски – найповільніша форма пам’яті в системах, проте останні використовуються все рідше і рідше.

Коли процесор записує дані у блок пам’яті кеш слідує одному із двох шляхів: прямий запис (write-through) або відкладений запис (write-back).

1) write-through - дані записуються безпосередньо на дисковий масив. Тобто як тільки дані отримані, вони відразу ж записуються на диски і після цього контролер подає сигнал керуючій ОС про завершення операції.

2) write-back - дані записуються спочатку в кеш, і тільки потім (або в міру заповнення кешу, або в моменти мінімального завантаження дискової системи) з кешу на диски. При цьому, сигнал про завершення операції запису передається керуючій ОС відразу ж після отримання даних кешом контролера.

Незважаючи на те, що write-through простіший у реалізації він породжує багато звернень до основної пам’яті, чого можна уникнути шляхом використання write-back, хоча він ефективно ефективно приховує записи з основної пам’яті та решти системи, поки рядок кешу не буде змінено.

Далі розглянемо два питання, що є дуже важливі для коректної роботи мультипроцеосрної системи – когерентність та узгодженість пам’яті та кеш-пам’яті.

Когерентність пам’яті. Когерентність кеш-пам’яті – це один з випадків когерентності пам’яті, фактично це властивість кешу, зберігати цілісність та забезпечувати актуальність даних, що знаходяться в локальних кешах роздільних ресурсів.

Система пам’яті вважається когерентною, якщо при зчитуванні з довільного місця у пам’яті повертається останнє записане за цією адресою значення. Однак це визначення охоплює два аспекти поведінки пам’яті у системі – когерентність та узгодженість. Створення мультипроцеорних системи загального призначення, що використовують спільну пам’ять вимагає ретельна врахування обох аспектів.

Когерентна система пам’яті має три властивості:

1) збереження порядку програмних запитів;
2) когерентний погляд на пам’ять;
3) серіалізація запису.

Збереження порядку програмних запитів означає, що якщо процесор читає місце в пам’яті після того, як відбувся запис в нього, повинно повертатись останнє записане значення.

Когерентна пам’ять означає, що якщо процес пише в місце пам’яті, за чим слідує зчитування іншим процесором, тоді записане значення повертається у випадку, якщо ці два процесори розділені і між ними не відбувається більше ніяких записів.

Серіалізація запису означає, що якщо відбувається записи до однієї адреси у пам’яті від двох різних процесорів, то усі процесори в системі бачать те, в якому порядку йде запис.

Використання кеш-пам’яті призводить до проблеми підтримки когерентності пам’яті в багатопроцесорних системах. Проблема полягає в тому, що представлення пам’яті різними процесорами може здійснюватись через їх кеші. Тобто копії спільних даних можуть знаходитись одночасно у декількох процесорних кешах, і коли один з процесорів змінює кешовані дані, то всі інші

процесори матимуть застарілі, неправильні значення(друга властивість когерентних систем пам’яті). Тобто усі кеші процесорів повинні бути проінформовані, про зміну даних, для належної роботи програми.

Для наглядності проблеми когерентності кешу в мультпроцесорних системах розгланемо таблицю 2.1.

Припустимо в системі є два процесори з власним кешом. Коли процесор А зчитує місце у пам’яті X, ці дані зберігаються в кеш процеора А. Те ж саме відбувається коли процесор Б зчитує місце у пам’яті X. Якщо згодом процесор Б записує значення у місце X, тоді кеш процесора А буде мати застаріле значення. Якщо процесор А знову зчитає місце X , він отримає застарілі дані з кешу.

Таблиця 2.1 – Проблема когерентності кешу в мультипроцеорних системах

Час	Подія	Кеш-пам’ять А	Кеш-пам’ять Б	Пам’ять
0				1
1	CPU А читає X	1		1
2	CPU Б читає X	1	1	1
3	CPU Б пише X	1	0	0

Існує два базових класи протоколів для забезпечення когерентності кеш-пам’яті:

1) snooping;
2) directory-based.

Протокол snooping включає в себе процесор, який містить кеш-пам’ять, загальну шину пам’яті для запису іншими процесорами. Якщо кеш процесора містить записані дані, протокол може змусити анулювати свій рядок

пам’яті(змушуючи звертатись до пам’яті ще раз), або оновити його вміст. До таких протоколів відносяться: Write Once, Synapse N+1, Berkeley, Illinois та Firefly.

В directory-based протоколах є директорія, що відстежує обмін даними та стан фізичних блоків пам’яті. Коли процеор пише в блок пам’яті, він зберігає ексклюзивний доступ для запису у цей блок. Повідомлення передаються, для того, щоб гарантувати відсутність застарілих блоків пам’яті у кешах процесорів. Приклади протоколів: Dir1NB та Dir0B схеми.

Узгодженість пам’яті. Узгодженість пам’яті – це система правил, яких дотримується певна комп’ютерна система, щодо упорядкування доступу до пам’яті (читання та запис). Модель узгодженості пам’яті забезпечує формальну специфікацію для програміста, про те як поводиться система пам’яті. Модель встановлює обмеження на значення, які може повертати система при читанні, під час роботи з спільною пам'яттю, і так поведінка обмежує можливості оптимізації апаратного та програмного забезпечення.

Визначення моделі узгодженості пам’яті є критично важливою для забезпечення коректної роботи мультипроцеорної системи з спільною пам’яттю і для використання цієї пам’яті програмами. Модель, яка застосовується для розроблювальної системи не може бути описана до моменту, поки система буде розроблена, але можливі до використання моделі описані нижче.

Модель послідовної узгодженості - це спрощена версія строгої моделі, в якій весь доступ до пам’яті серіалізується (доступ виконуються по одному або атомарно), і що операції процесорів виконуються в запрограмованому порядку. Ця модель проста і поводиться так як розробники очікують від комп’ютерів. На жаль, ця модель забороняє розширену оптимізацію мультипроцесорних систем, яка доступна в однопроцесорних системах. Як результат, існують більш гнучкі моделі, багато з яких використовуються реальними системами.

Для послідовної моделі додавання кешу даних це такий же виклик, як і оптимізація, а отже виникають два питання:

1) як виявити коли запис завершено (потрібно для збереження порядку виконнаня програми між записом та іншими операціями);

2) неатомарність анулювання запису в кеш-пам’яті інших процесорів в системі.

Перше питання вирішується впровадженням механізму відслідковування отримання повідомлень та оновлення кешу. Як тільки усі кеші підтвердили запис, записуючий процесор отримує повідомлення та може продовжувати роботу.

Проблему неатомарності можна вирішити шляхом примусової серіалізації, при записуванні в ту ж комірку пам’яті і одночасній забороні зчитувати цю інформацію, поки усі кеші не підтвердять отримання, аналювання або оновлення даних.

Окрім моделі послідовної узгодженості є і більш прості моделі. Такі моделі спрощують роботу програм, наприклад:

1) read after write (RAW) – зчитуйте лише після запису;

2) write after write (WAW) – пишіть після повного завершення запису;

3) any access after a read (RWAR) - доступ дозволяється лише після запиту на зчитування.

Як правило такі моделі часто є залежні одна від одної, а також часто змішують свої властивості. Наприклад можлива варіація таких двої правил як: читати чужі записи рано і читати свої записи рано, в такому випадку процесор не може прочитати запис іншого процесора і його власний запис, до повного підтвердження системою дозволу на читання\запис.

RAW визначає, коли записуючий процесор може прочитати нове значення після запису, відносно серіалізації та видимості даних для інших процесорів. Наприклад у архітектурі Intel x86, система спрощена так, що зчитування може відбутись до того як дані будуть серіалізовані та доступні для читання іншими процесорами.

WAW дозволяє процесорам конвеєрувати або забороняти записи в різні місця у пам’яті. Спрощення класичного підходу робить можливим переупорядкування будь-якої операції у пам’яті, якщо у кожного процесора доступ до різних адрес у пам’яті.

Незалежно від обраної моделі усі вони забезпечують захисні механізми, що дозволяють розробнику забезпечити виконання програмного коду у правильній

послідовності. Такі моделі часто тягнуть за собою явну серіалізацію, вказівки щодо синхронізації, послідовностей запитів, які разом зможуть забезпечити правильну послідовність роботи з даними.

2.3 Вибір програмно-апаратного забезпечення

У цьому розділі буде описано проведення підбору апаратних засобів для побудови мультипроцесорної системи загального призначення на основі топології гіперкуб. Проведено порівняльну характеристику процесорів, шини та інших компонентів системи. Система використовує класичну SMP- архітектуру зі спільною пам’яттю.

Цілі проектування:

1) реалізація з мінімальною затратою сил та ресурсів;
2) відсутність інвазивних змін в компонентах системи;
3) мінімальні втрати в обчислювальній потужності;
4) підтримка розширенаих функції, а саме – кешування.

За апаратну базу було вирішено взяти продукцію корпорації Altera. Altera виробляє програмовані логічні мікросхеми (ПЛІС) з 1983 року, а у 2015 році вона була придбана більш відомою компанією Intel.

Вона відома такими середньобюджетними серіями продуктів як Stratix та Arria, та бюджетною серією Cyclone, що базувалась на мікросхемі FPGA, розробила ПЛІС серії MAX, енергонезалежні FPGA.

Також ця компанія зпроектувала програмноге забезпечення для Intel Quartus Prime (Altera Quartus II) – середовище програмування та розробки під ПЛІС.

2.3.1 Порівняльна характеристика процесорів

Найпопулярнішими наразі процесорами Altera є ряд процесорів NIOS II, а також не такий популярний, але досить потужний вбудований процесор NIOS 3.0.

Щоб почати розгляд характеристик процесора варто нагадати наступні терміни:

1) Reduced Instruction Set Computing (RISK) – архітектура процесорів, у яких обчислення відбувається зі скороченим набором команд;

2) програмний мікропроцесор (soft microprocessor) – це таке ядро мікропроцесора, яке може бути повністю реалізовано метолом логічного синтезу. Зазвичай такий тип процесорів є одноядерним.

Спочатку по загальних характеристиках. Як і у Nios, так і у Nios II побудовані на архітектурі RISK з програмним мікропроцесором, зо має повноцінну 32-х розрядну архітектуру. Архітектура Nios представленна структурою системних команд (ISA). Загальна архітектура показана на рисунку 2.3, проте можливі певні модифікації, про які буде сказано далі[43].

Архітектура Nios визначається наступними функціональними модулями:

1) регістровий файл;
2) арифметико-логічний пристрій;
3) інтерфейс з логікою користувацьких інструкцій;
4) контролер виключень;
5) внутрішній або зовнішній контролер переривань;
6) шина даних;
7) диспечер пам’яті (MMU);
8) елемент захисту пам’яті (MPU);
9) кеш-пам’ять для інструкцій та даних;
10) здвоєний інтерфейс пам’яті для інструкцій та даних;
11) модуль налагодження JTAG.

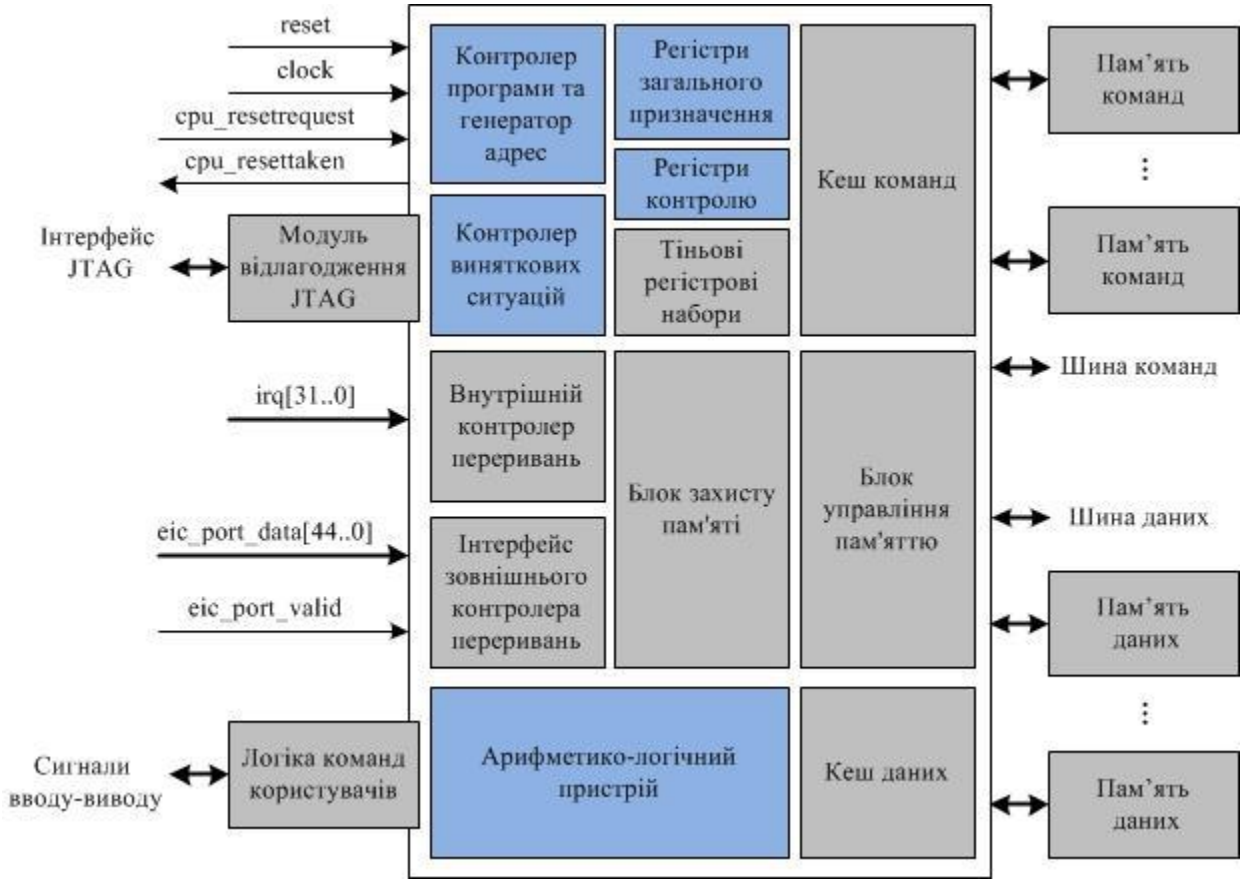


Рисунок 2.3 – Структурна схема процесорів Nios. Модулі, позначені синім кольором, є обов'язковими

У процесора є три типи команд – I, R та J, кожна з яких є 32-х розрядною. У командах 1-го класу використовуються константи, що записані в командному слові. Усі аргументи та результати команд 2-го класу визначаються через номери регістрів. Третій клас команд містить 26-бітне поле констант, яке використовується як адреси переходів.

Адресний простір для пам'яті та перефирійних пристроїв є загальним, оскільки Nios реалізує типову архітектуру load/store (завантажити/зберегти), у якій існує чіткий поділ інструкцій на команди доступу до оперативної пам'яті та операції арифметико-логічного пристрою.

Процесори Nios II доступні у трьох конфігураціях Nios II / f (fast), Nios II / s (standard) та Nios II / e (economy). Детальніше про конфігурацію кожного з них.

Nios II / f:

1) Nios II / f гарантує максимальну продуктивність, завдяки розміру ядра;

2) окремий кеш для команд та даних (від 512 байт до 64 КБ);

3) опціонально - MMU або MPU;

4) 2 ГБ зовнішнього адресного простору;

5) можлива наявність тісно пов’язаної пам’яті для інструкцій та даних;

6) шестиступінчатий конвеєр для досягнення максимального рівня DMIPS / МГц;

7) апаратні засоби множення та зсуву за один такт;

8) динамічний модуль передбачення переходів;

9) до 256 користувацьких інструкцій та необмежена кількість апаратних прискорювачів;

10) модуль налагодження JTAG;

11) можливість вдосконалення JTAG-модуля та апаратних точок розбиття.

Nios II / s:

1) ядро Nios II / s розроблено збалансовано між продуктивністю та вартістю;

2) спільний кеш для команд та даних;

3) 2 ГБ зовнішнього адресного простору;

4) опціонально тісно пов’язана пам’ять для інструкцій;

5) п’ятиступінчатий конвеєр;

6) статичний модуль передбачення переходів;

7) апаратні команди множення, ділення та зсуву;

8) до 256 користувацьких інструкцій

9) модуль налагодження JTAG

10) можливість вдосконалення JTAG-модуля та апаратних точок розбиття.

Nios II / e:

1) ядро Nios II / e розроблено для мінімально можливого використання FPGA. Така конфігурація ядра використовується для недорогих програм FPGA Cyclone II. Особливості Nios II / e включають

2) до 2 ГБ зовнішнього адресного простору

3) модуль налагодження JTAG;

4) повні системи складаються менш ніж з 700 логічних блоків;

5) необов’язкові вдосконалення налагодження;

6) до 256 користувацьких інструкцій.

7) Nios 3.0

8) ядро як і у Nios II/s, розроблено збалансовано між продуктивністю та вартістю;

9) 2 ГБ зовнішнього адресного простору;

10) до 512 внутрішніх реєстрів загального призначення;

11) п’ятиступінчатий конвеєр;

12) 16-бітні шина;

13) потужні режими адресації;

14) до 256 користувацьких інструкцій.

Для мультипроцесорної системи загального призначення, розроблюваної в цій роботі я обрала саме Nios 3.0, оскільки в ньому найкраще збалансовані показники вартості та якості, окрім цього він оптимізований для PLD Altera та має вбудовану кеш-пам’ять. Тому у наступному розділі детальніше про нього.

2.3.2 Nios 3.0. Вбудований кеш

Як вказано вище, Nios 3.0 має п’ятиступінчастий конвеєр зі зменшеним набором команд, побудований на RISC-архітектурі. На рисунку 2.4 зображена блок-схема ядра обраного процесора. Прозорість обраного процесора робить його зручним для проектування мультипроцесорної система, а також для створення програмного забезпечення під цю систему.

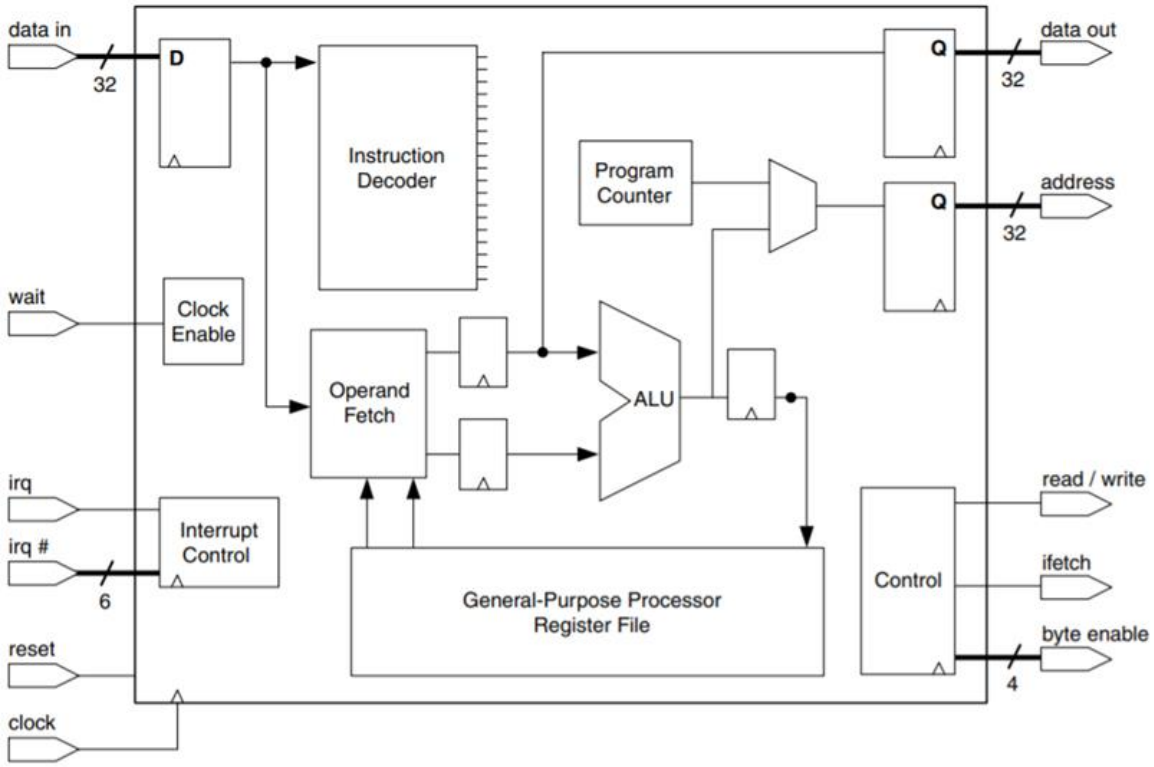


Рисунок 2.4. – Структурна схема ядра Nios 3.0

Обрана версія вбудованого процесора Nios включає функції, які використовують переваги вдосконалених функцій пам'яті високопродуктивного сімейства Stratix та недорогих сімейств Cyclone для поліпшення продуктивності системи, забезпечуючи при цьому нові інструменти для швидшого розвитку системи.

Настроювані користувачем кеші, версія 3.0 вбудованого процесора Nios включає настроювані користувачем інструкції першого рівня (L1) та кеші даних, які використовують велику двопортову пам’ять на недорогих пристроях Cyclone та Stratix для підвищення системних вимог .

Покращений контролер SDRAM, розробники вбудованих процесорів Nios можуть отримати доступ за один цикл до недорогих пристроїв SDRAM зі швидкістю вище 100 МГц. Поєднання настроюваої користувачем кеш-пам’яті L1 та контролера SDRAM дозволяє розробникам використовувати недорогу, пам’ять окремо від чіпа та досягати майже тієї ж продуктивності для великих вимог до пам’яті, що і при використанні високопродуктивної вбудованої SRAM.

Покращений комутатор Avalon, параметризована шина інтерфейсу Altera, що використовується вбудованим процесором Nios, тепер може підтримувати конвеєрні транзакції даних для усунення вузьких місць.

Новий налагоджувач у режимі реального часу на основі JTAG, ядро OCI (On-Chip Instrumentation) на основі JTAG від First Silicon Solutions (FS2), включене до нової версії вбудованого процесора Nios, надає розробникам програмного забезпечення потужний інструмент для налагодження коду в режимі реального часу . Доступ до OCI здійснюється через цільове з'єднання JTAG і підтримує функції емулятора, такі як складні апаратні тригери та трасування в реальному часі.

Надійне інтегроване середовище розробки, підтримка коду прискореної технології lab Developer Suite дозволяє розробникам програмного забезпечення швидко редагувати, компілювати, завантажувати та налагоджувати свій код на базі процесора Nios, забезпечуючи вбудовану підтримку розширених можливостей налагодження та відстеження процесора в реальному часі.

Включена бібліотека програмного забезпечення мережевого протоколу, підтримка програмного забезпечення для багатьох протоколів Ethernet, включаючи APR, IP, ICMP, TCP, UDP .

Архітектура набору інструкцій Nios (ISA) призначена для генерації на основі популярного С та мови програмування високого рівня на С ++. Підтримка додаткових бібліотек С/С++ та підпрограм дозволяє:

1) обробляти помилкові переривання - встановлений обробник переривань за замовчуванням. Незначно збільшує розмір коду і використання пам'яті;

2) виклик конструкторів С++ - використовується для ініціалізації статичних класів С ++;

3) window pointer manager - для обробки випадків переповнення реєстру вікон. Може зменшити розмір коду, якщо інженер знає, що глибина виклику програмної функції не перевищуватиме кількість вікон реєстру;

4) швидке множення - для програмних реалізацій множення;

5) small printf () - зменшує розмір коду (з 40 кБ для повної реалізації до 1 кБ), коли підтримка чисел з плаваючою комою не потрібна. Цілі числа, символи та рядки – підтримуються;

6) вбудований модуль налагодження апаратного забезпечення, який дозволяє розробникам використовувати апаратні точки зупинки та трасування за допомогою додаткового програмного забезпечення та / або обладнання.

Вбудована кеш-пам’ять Nios 3.0. Ядро Nios можна налаштувати за допомогою необов'язкових одноциклових інструкцій L1 та кешів даних. Розробник може вказати, що кожен кеш має бути розміром від 1 кБ до 16 кБ (розмір повинен бути кратний двом).

Кожен кеш безпосередньо відображається таким чином, що найменші біти адреси пам'яті використовуються як індекс кешу. Кеші з прямим відображенням простіші у реалізації та призводять до меншої апаратної схеми, але мають меншу швидкість, ніж повністю асоціативні або асоційовані кеші.

Кеш даних Nios використовує write-through політику запису (тобто повний запит на запит надходитьі у кеш, і у пам’ять,). Кеш інструкцій не підтримує запис, оскільки майстер інструкцій теж не підтримує. Крім того, кеш даних можна автоматично обійти при виконанні інструкції навантаження, використовуючи префікс-інструкції PFXIO. Це особливо корисно при доступі до периферійних пристроїв Nios, оскільки операції вводу-виводу не повинні кешуватися.

Система Nios вимагає ініціалізації та ввімкнення кешу даних, перш ніж їх можна буде використовувати. Ініціалізація досягається шляхом анулювання кожного рядка кешу. Це реалізується за допомогою контрольних регістрів ICACHE та DCACHE, що мають лише запис. Ці регістри анулюють рядок кешу, що відповідає адресі пам'яті, яка записана на них. Кожен кеш інструкцій та даних має біт увімкнення в регістрі керування STATUS, який повинен бути встановлений в 1, дозволяючи вмикати та вимикати кеш. Кеш-пам’ять потрібно вимкнути перед використанням реєстру недійсних рядків. Оскільки кеш-пам’ять Nios не має вбудованих засобів автоматичної когерентності кеш-пам’яті, ці регістри, є критично важливими для інформування кешу про те, що інший процесор записав дані в кешовану пам’ять.

2.3.3 Шина Avalon

Шина Avalon - це архітектура шини, яка була розроблена, для мережі взаємозв'язку для SOPC. З цією метою шина Avalon має простий інтерфейс, який визначає сигнали між головним та веденим портами. Окрім простоти, шина Avalon була розроблена для використання мінімальних логічних ресурсів в PLD та синхронної роботи, щоб уникнути складних проблем з аналізом часу.

Традиційна реалізація загальної шини використовує єдину тристатну шину, в якій пари master-slave арбітруються. Будь-які пристрої, підключені до шини, які не беруть участі в поточній транзакції, не повинні впливати на значеннями на цій шині, використовуючи драйвери трьох станів у режимі високого імпедансу.

Такий підхід є виграшним для розроблювальної SMP-системи, оскільки master і slave пристрої фізично відокремлені, розміщені на автономних друкованих платах або на різних платах. Конструкції використовують загальний набір шинних ліній для економії місця на платі та кількості доступних контактів вводу-виводу.

Проблеми таймінгу також спрощуються. Одна шина стає вузьким місцем пропускної здатності, оскільки одночасно на шині може відбуватися лише одна транзакція. У той час як більшість PLD забезпечують потрійні драйвери для комунікації поза мікросхемою, лише деякі PLD забезпечують внутрішні ресурси для підтримки обмеженої шини трьох станів. Як результат, більш поширеним є використання мультиплексорів для реалізації арбірованої шини, оскільки мультиплексори підтримуються усіма PLD.

Шина Avalon - це "комутаційна тканина", яка використовується Altera's SOPC Builder для з'єднання процесорів та інших пристроїв у вбудованій процесорній системі Nios, і насправді не є шиною в традиційному розумінні.

Зокрема, шина Avalon - це реалізація "спільної" шини від точки до точки з підтримкою одночасних декількох ведучих шин. Іншими словами, існує спеціальне з'єднання від кожного потенційного ведучого шини до кожного з підпорядкованих пристроїв, яким він може керувати. Хоча кожен процесор і

пристрій, підключаються до справжньої шини. Ця структура проілюстрована на малюнку 2.5 для N-процесорної системи.

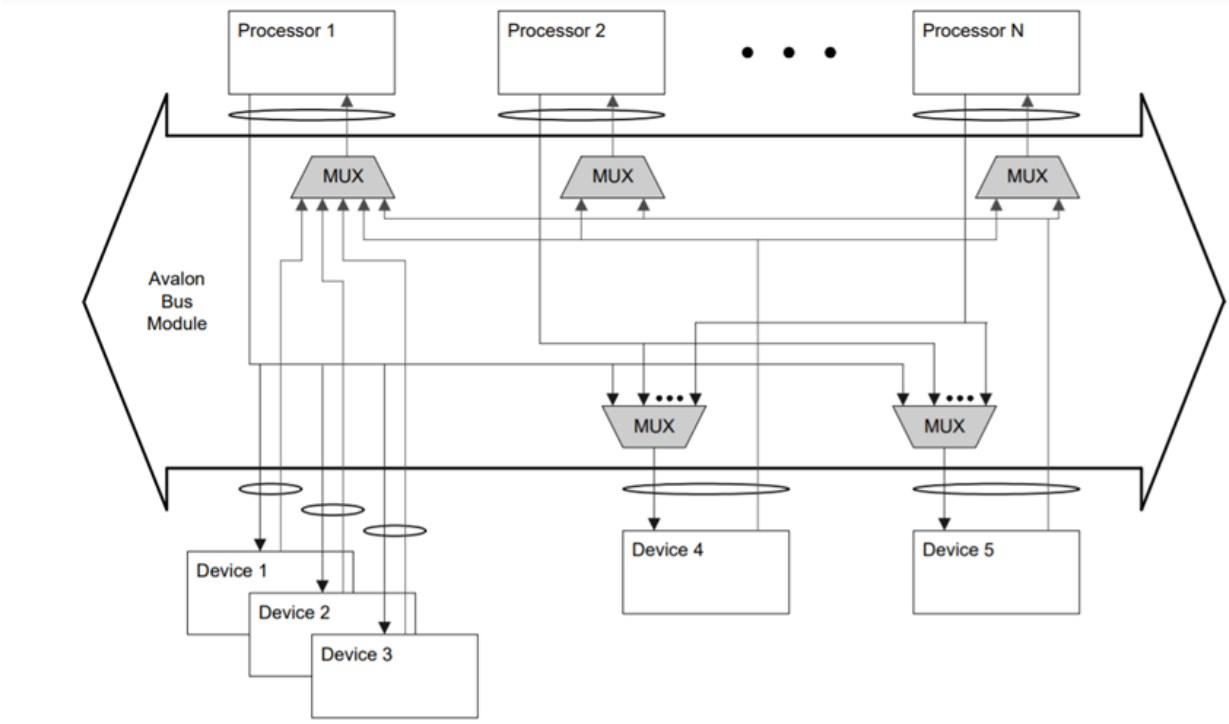


Рисунок 2.5 - Основна структура модуля шини Avalon

Архітектура з декількома master-вузли збільшує пропускну здатність системи, усуваючи вузьке місце однієї шини. Master-вузли системи борються за окремі slave-вузли, а не за саму шину. Цей прийом називається арбітражем на стороні master, це робить протокол досить гнучким для периферійних пристроїв і підвищує пропускну здатність. Арбітраж на підлеглій стороні означає, що будь-яка кількість транзакцій може відбуватися одночасно, доки не існує суперечок щодо одного і того ж slave-вузла. Якщо більше одного master-вузла вимагає одного і того ж slave, кожному master надається доступ по черзі, або за замовчуванням, або за допомогою схеми пріоритету, що налаштовується.

Цей арбітраж інкапсульований в модулі шини Avalon і прихований від конструктора системи (хоча правила арбітражу можна налаштувати за допомогою SOPC Builder). Після надання доступу до master пристрою, мультиплексори шини Avalon подають відповідні сигнали на master-вузол.

Шина Avalon також вказує на окремі адреси, дані та лінії управління. Це забезпечує простий інтерфейс до вбудованої логіки, уникаючи необхідності декодувати дані та адресувати цикли шини.

Крім того, шина Avalon підтримує динамічний розмір шини. Іншими словами, шини адреси та даних для кожного периферійного пристрою веденого пристрою є настільки великими, наскільки вони повинні бути.

Наприклад, ведений пристрій, що має лише чотири доступні регістри, матиме ширину адреси два. Динамічний розмір шини означає, що модуль шини Avalon також автоматично обробляє передачу даних між пристроями з данними різного розміру. Крім того, модуль шини Avalon автоматично обробляє стани очікування, генерація, прихованих передач та генерацію переривань.

Транзакції на шині Avalon можуть відбуватися у розмірі байтів, напівслів або слів (вісім, шістнадцять або тридцять два біти відповідно). Транзакція може розпочатися відразу після іншої транзакції, без втрати тактових циклів, незалежно від пари master-slave.

Протокол також визначає транзакції шини для периферійних пристроїв, що знають про затримки, потокових периферійних пристроїв та декількох master-вузлів шини. Кожен із цих вдосконалених режимів передачі дозволяє передавати кілька одиниць даних під час однієї транзакції (зменшуючи накладні витрати при переміщенні великих обсягів даних).

Nios використовує порти вводу-виводу для доступу до пам'яті та периферійних пристроїв на шині Avalon (процесор Nios, пов'язані з ним підлеглі периферійні пристрої та шина Avalon спільно називаються системним модулем). Nios використовує повний 4 Гб (32-розрядний) адресний простір, відображаючи адресу, яку модуль шини Avalon декодує у сигнал обраного slave-вузла та зміщення.

2.4 Висновки

У цьому розділі було розглянуто топологію «гіперкуб» при використанні її для мультипроцесорних систем загального призначення. Визначено її основні характеристики та покази ефективності роботи.

Також було висвітлено роботу симетричної системи зі спільною пам’ятю, піднято і пояснено проблему когерентності кеш-пам’яті та проілюстровано роботу кешу у розроблювальній системі. Представлено важливий аспект – забезпечення коректності виконання коду та актуалізація даних при розробці та виконанні паралельних та багатопотокових програм.

Проведено порівняльну характеристику декількох процесорів фірми Altera, описано основні особливості обраного процесора Nios 3.0 та інтерфейсу шини Avalon.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

3.1 Засоби для проектування мультипроцесорної системи

Для проектування, реалізації та тестування розроблюваної мультипроцесорної системи загального призначення на основі гуперкуб було обрано наступні компоненти:

1) Altera Quartus II – середовище розробки від компанії Altera, для програмування, проектування під ПЛІС. Це середовище дозволяє створювати проекти різної складності, аналізувати і синтезувати HDL-конструкції, тестувати, виконувати аналіз(у тому числі по таймінгах), імітує реакцію розроблюваної системи на різні подразники.

2) SOPC Builder – інструмент середовища Quartus II, що дозволяє створювати пристрої для тестування, змінювати їх конфігурацію, є можливість автоматичного генерування з’єднання

3) Програмована плата Altera DE1-SoC – надійна апаратна платформа, що містить великий набір інструментів розробки, та забезпечує гнучкість під будь-яке рішення.

3.1.1 Засоби візуалізації роботи – макетна плата DE1-SoC

DE1-SoC - комплект розробки, являє собою потужну апаратну платформу, побудовану на основі ALTERA System-on Chip (SoC) FPGA, яка є комбінацією новітнього вбудованого двоядерного процесора Cortex-A9 і провідною в галузі програмованої логіки. Тепер розробники можуть використовувати потужність конфігуруємої системи в поєднанні з високою продуктивністю і низьким споживанням процесора.

Системи на кристалі (SoC) від компанії ALTERA інтегрують в собі процесор на основі ARM, інтерфейси, периферію і пам'ять, органічно пов'язані з ПЛІС FPGA. Макетна лата DE1-SoC включає в себе високошвидкісну пам'ять

DDR3, аудіо- та відео-інтерфейси, Ethernet і багато іншого. Комплект містить всі компоненти, необхідні для його використання.

Рисунок 3.1 – Макетна плата DE1-SoC

Детальніше по характеристиках, що є важливими для нашої системи [50]:

1) встановлена мікросхема Cyclone V SoC 5CSEMA5F31C6;
2) двоядерний Cortex-A9 (HPS);
3) 85K елементів програмної логіки;
4) 4,450 Kbits пам’яті;
5) 2 контроллера пам’яті;
6) на платі встановлений USB BlasterII;
7) 64 MB (32Mx16) SDRAM on FPGA;
8) 1GB (2x256Mx16) DDR3 SDRAM on HPS;
9) MicroSD Card Socket on HPS;

10) два порта USB 2.0 Host (ULPI інтерфейс с USB порт типу A);
11) USB-UART (порт microUSB type B);
12) 10/100/1000 Ethernet;
13) PS/2 миш/клавіатура;
14) IR Emitter/Receiver;
15) 4 користувацьких ключа (FPGAx4);
16) 10 перемикачів (FPGAx10);
17) 11 світодіодів (FPGAx10; HPSx1);
18) 2 HPS клавіші скидання (HPS_RST_n and HPS_WARM_RST_n);
19) шестирозрядний семисигментний дисплей;
20) G-Sensor on HPS;
21) 12 V DC порт живлення.

3.2 Створення класичного вигляду SMP-системи

Як уже було сказано, у цій системі використовуватимуться процесори Nios 3.0 та шина Avalon. Оскільки ціль нашої задачі проектування системи та дослідження продуктивності та коректності роботи пам’ять та пристрої вводу-виводу були взяті типові, запропоновані системою SOPC Builder. Оскільки топологія гіперкуб – гнучка до кількості вузлів, у нашому випадку процесорів, для коректності проведених розрахунків цікавить 3 її варіації – гіперкуб 2, 3 та 4 степенів, тобто кількість процесорів у системі буде дорівнювати 4, 8 та 16 відповідно. На рисунку 3.2 зображена змодельована в SOPC Builder система.

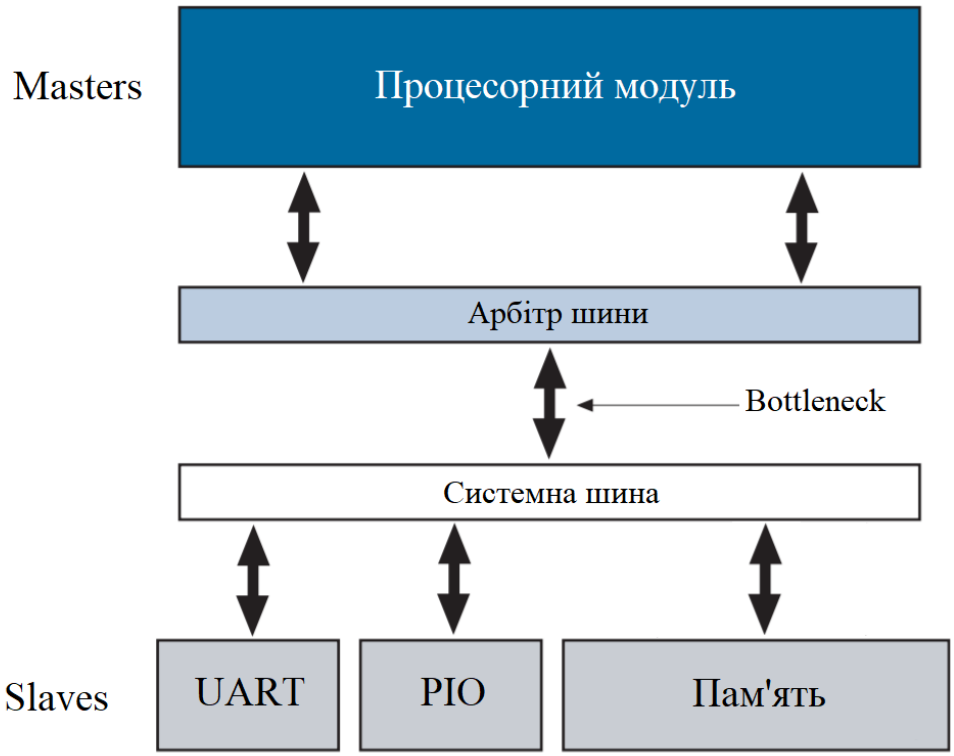


Рисунок 3.2 – Модель розроблюваної SMP-системи у середовищі SOPC Builder

Для зручності проектування та можливості масштабування системи буде введено окремий логічний блок – процесорний блок, де фізично знаходитись процесори, об’єднанні обраною топологією. Саме цей блок змінюватиметься для масштабованості. Варто відмітити, що у цьому блоці знаходитимуться і блоки кеш-пам’яті.

Реалізація симетричною мультипроцесорної системи на програмованій мікросхемі передбачає реалізацію безлічі програмних процесорів на одному програмованому логічному пристрої. Сучасні програмовані логічні пристрої забезпечують достатньо ресурсів, таких як LE та вбудована пам’ять, для реалізації складних систем 32-розрядних програмних процесорів з підтримкою кешу. Інструменти для розробки, забезпечують пряму підтримку впровадження декількох програмних процесорів на програмованому чіпі. Однак засоби розробки ще не забезпечують способу автоматичного впровадження функціонуючої системи SMP.

Системи на програмованому чіпі за архітектурою ідентичні своїм дискретним аналогам SMP. Сюди входить наявність однакових процесорів, кожен

з рівним доступом до пам'яті та підсистем вводу-виводу. Усі процесори повинні мати з'єднання з однаковим пріоритетом арбітражу для кожного периферійного вводу-виводу та пристрою пам'яті. Навіть при виконанні цих вимог, існує дві проблеми, які в даний час заважають повноцінно працювати таким системам:

1) неможливість однозначно ідентифікувати процесори в системі;

2) неможливість забезпечити узгодженість кеш-пам’яті.

Когерентність кеш-пам'яті є найскладнішим питанням при розробці симетричної мультипроцесорної системи на програмованому чіпі. Спеціальна розробка апаратного та програмного забезпечення необхідна для забезпечення узгодженості кеш-пам’яті

Спосіб однозначної ідентифікації процесорів потрібен для тимчасового вибору процесора, що завантажуватиме програмне забезпечення для виконання глобальної ініціалізації під час запуску та надання операційним системам можливості призначати процеси та потоки конкретним процесорам. Одним із аспектів глобальної ініціалізації є встановлення спільної векторної таблиці переривань. Локальна ініціалізація для кожного процесора включає включення переривань, встановлення маски пріоритету переривання, очищення та активування кеш-пам’яті тощо. У традиційних системах SMP материнська плата часто ідентифікує кожен процесор відповідно до фізичного сокета, в якому він знаходиться. Однак у ПЛІС фізичні сокети не існують.

Процесор Nios має регістр CPU ID - для керування ідентифікатором процесора, це регістр використовується лише для читання і повертає код, який є унікальним лише для певної версії Nios. Тому кожен Nios тієї самої версії повертає однаковий ідентифікатор. Існує кілька рішень цього питання:

1) зміна значення регістру управління ідентифікатором процесора;

2) створення контрольного реєстру для Nios;

3) реалізація невеликого ЗСД (загальне сховище даних) для кожного процесора, що містить унікальний ідентифікатор процесора (PID);

4) реалізація користувацької інструкції в кожному Nios для повернення унікального значення.

Перші два рішення не підпадають під мету бути неінвазивними для Nio. Четверте рішення є надто складним для цієї задачі. Тому було обрано ЗСД, оскільки воно, крім простої та неінвазивної, використовує переваги архітектури шини Avalon, щоб зробити кожне сховище доступним лише для відповідного процесора. Це рішення також дозволяє всім ЗСД призначати одну і ту ж адресу, таким чином зберігаючи адресний простір.

3.3 Проблема когерентності кешу. Створення модуля когерентності кешу

Класична система SMP-система на базі процесора Nios розроблена, але перш ніж говорити про завершення роботи необхідно підняти та вирішити проблеми, що стоять щодо її належного функціонування. Найбільшим викликом для впровадження високопродуктивної системи є забезпечення узгодженості кеш-пам’яті: типові процесори програмного забезпечення, доступні для побудови такої системи, не розроблені з урахуванням когерентності кеш-пам’яті.

Зокрема, архітектура шини, яка зазвичай використовується в ПЛІС (таких як шина Avalon), фактично робить неможливим пошук. Таким чином, для досягнення узгодженості кеш-пам'яті потрібні інші функції. У цьому розділі обговорюються проблеми, з якими стикається належне забезпечення узгодженості кеш-пам’яті, та викладається загальна архітектура системи, яка вирішує ці проблеми. Крім того, початковий прототип модуля когерентності кешу (МКК) розроблений як доказ концепції того, що когерентність кеш-пам'яті може підтримуватися в системі SMP.

Ця задача вимагає ретельного вивчення модуля Nios та шини Avalon, щоб зрозуміти, які функції спростять роботу, а які функції будуть перешкоджати когерентності кешу. Також вигідно створити когерентний модуль неінвазивним та процзорим, з незначним або відсутнім відхиленням від існуючих процесів системи. Цей прототип служить підтвердженням концепції того, що систему можна легко модифікувати для забезпечення узгодженості кеш-пам’яті.

3.3.1. Архітектура МКК

Перше архітектурне рішення проекту полягає в тому, чи слід застосовувати відслідковування (snooping), чи directory-based протокол. Можна використовувати протокол каталогів, але він не настільки ефективний, як протокол перегляду для маломасштабних систем, оскільки передача повідомлень, вимагає шини з високою пропускною здатністю, що тягне за собою високу вартість обладнання, або споживає додаткову пропускну здатність на вже перевантаженій системній шині. Будь-яка реалізація вимагає інвазивних змін до кожного процесора Nios, щоб його кеш міг надсилати, отримувати та розуміти повідомлення протоколу каталогу. Використання цього протоколу спричинить великі апаратні втручання та грошові трати.

В якості альтернативи може бути використаний протокол snooping. На архітектурному рівні є кілька місць, де можна змінити обладнання. Процесор Nios реалізує кеш інструкцій та даних у парі із записом. Традиційно когерентність кешу забезпечується шляхом створення апаратного модуля для кожного кешу, який контролює шину пам'яті процесора. На жаль, це неможливо через точковий характер шини Avalon. Таким чином, архітектуру відслідковування використовувати не можна.

Альтернативою є додавання периферійного пристрою до системного модуля для інформування процесорів про запис у пам’ять. Реалізація когерентності кеш-пам'яті через додатковий периферійний пристрій дозволяє розробникам систем просто створювати екземпляри модуля когерентності кешу за допомогою стандартного графічного інтерфейсу системи.

Це також легко реалізувати, оскільки шина Avalon - це специфікація інтерфейсу з чітко визначеними сигналами. Насправді це гібридний протокол перегляду, який переглядає шину, але використовує центральний «каталог» для забезпечення узгодженості. Введеному периферійному пристрою може бути наданий доступ до відповідних сигналів на різних інтерфейсах шини Avalon.

Ці інтерфейси можуть бути стандартними інтерфейсами для периферійних пристроїв, таких як вбудована оперативна пам'ять або контролер пам'яті, або

спеціальними інтерфейсами, такими як міст із трьома станами, який використовується для зв'язку з SRAM-пам'яттю та флеш-пам'яттю.

На рисунку 3.3 показано МКК щодо типової з шляхом N SMP-системи. Модуль когерентності повинен мати можливість виявляти записи (зазвичай за допомогою моніторингу сигналів, що дозволяють запис), а також зчитувати шину адрес. Це дозволить йому повідомляти процесори про адресу, на яку було записано, та про те що відповідний рядок кешу може бути анульованим.

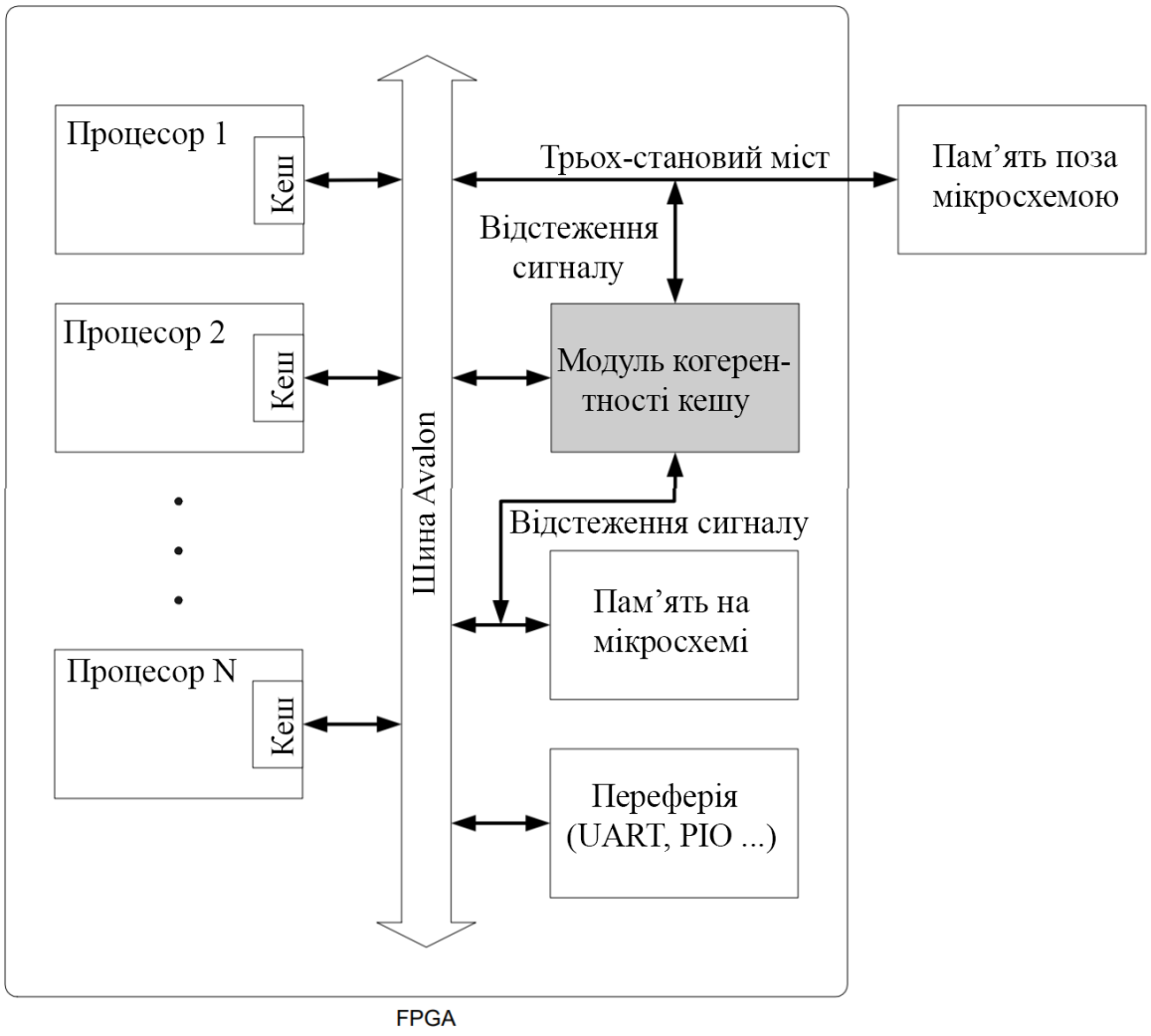


Рисунок 3.3 – Архітектура модуля когерентності кеш-пам’яті

Запису у кеш-пам'яті повинен бути анульований, оскільки Nios запрограмований анулювати певні рядки кешу, а не оновлювати їх. Анулювання виконується шляхом запису відповідної адреси до контрольних регістрів, реалізованих у кожному процесорі Nios. Політика анулювання була обрана з

метою мінімізації інвазивних змін у системі. Використовуваний протокол когерентності кешу зображений на рисунку 3.4.

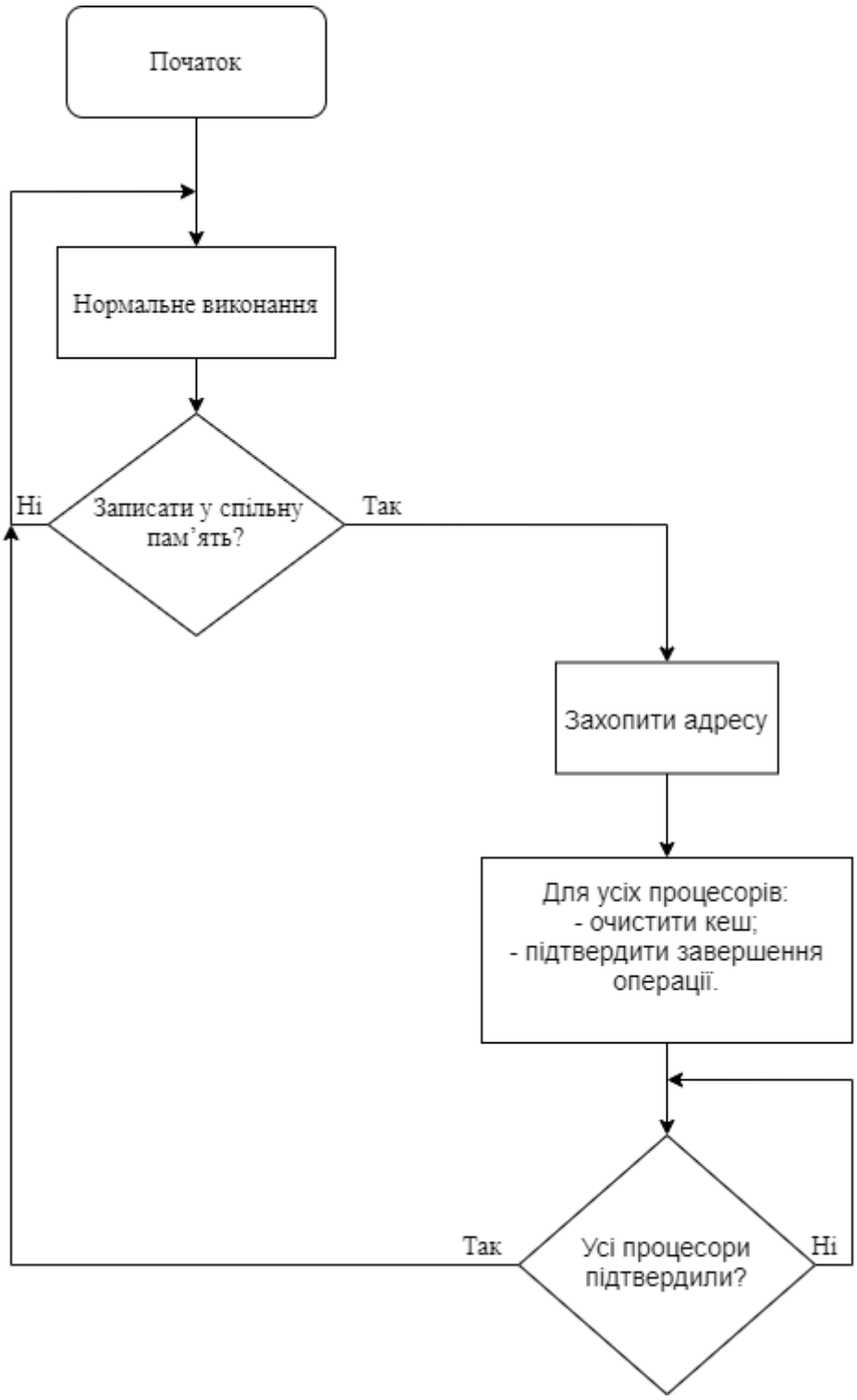


Рисунок 3.4 – Протокол когерентності кешу

Реалізація очищення кешу через регістри керування процесором вимагає, щоб програмне забезпечення відігравало роль у підтримці когерентності. Через

важливість підтримання узгодженості програмний компонент був написаний у формі високопріоритетного режиму обслуговування переривань. Це ідеально підходить для здатності введеного периферійного пристрою збільшити кількість переривань. Таким чином, забезпечення узгодженості кеш-пам’ятки є поєднанням апаратного та програмного рішень.

3.3.2. Реалізація модуля когерентності кешу у Quartus II

Модуль когерентності кеш-пам’яті відповідає за виявлення запису у пам’ять, та за повідомлення процесорів про таку подію. Код на мові VHDL для обладнання МКК наведено в додатку Б. На рисунку 3.5 наведено відповідну принципову схему.

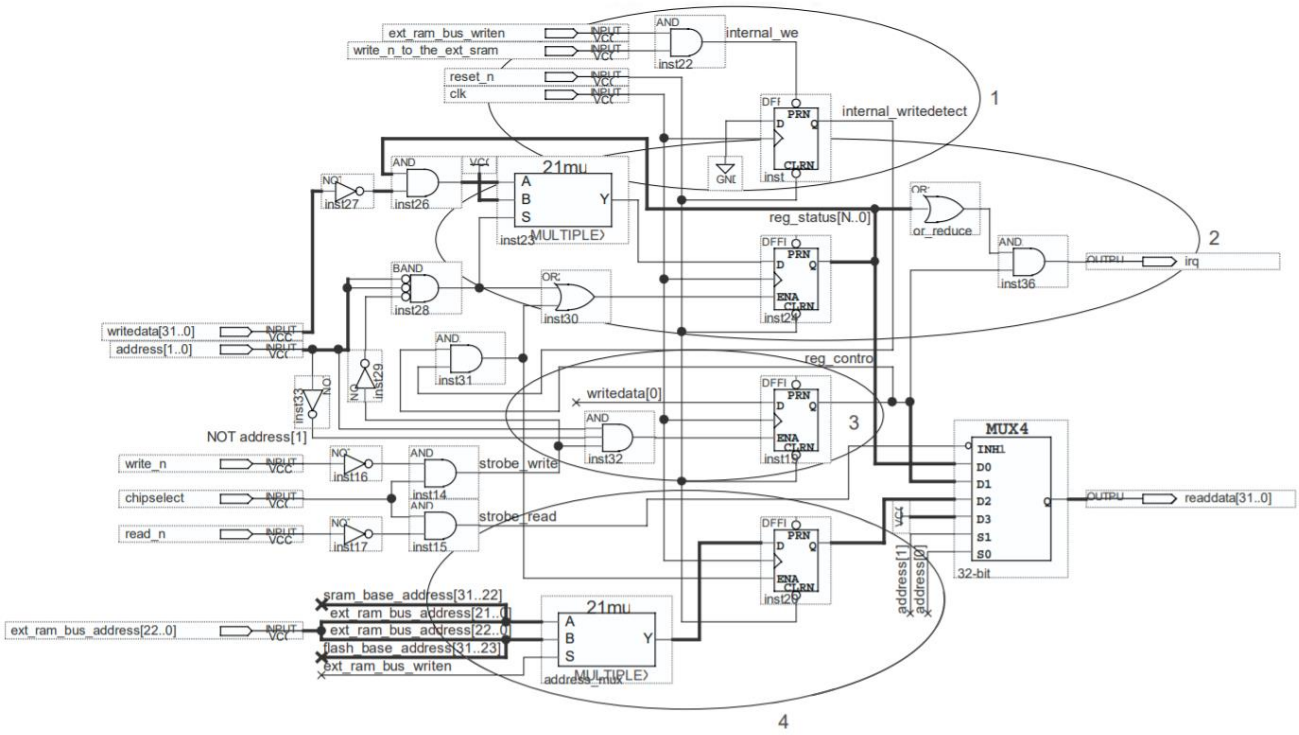


Рисунок 3.5 – Принципова схема модуля когерентності кеш-пам’яті

Процесор Nios повинен мати можливість вмикати та вимикати МКК. Це потрібно, оскільки існують ситуації, коли МКК не повинен викликати переривання (один з таких випадків - до завершення ініціалізації, де кеші

активовані і встановлена векторна таблиця переривань). Для вимкнення операції використовується однобітний регістр CONTROL (виділений овалом 3 на рисунку 3.4). Регістр CONTROL використовує системний CLOCK і сигнали скидання, і встановлює значення логічний «0» (вимкнено). Він спрацьовує при зсуві регістра 0x01, і кожен процесор в системі може записати в нього будь-яке значення.

Відстеження сигналів на шині дозволяє виявити транзакцію запису та захопити відповідну адресу пам'яті. Модуль когерентності виконує це завдання, зчитуючи рядки дозволу на запис і адресу на шині кожного пристрою пам'яті, і підтверджує сигнал виявлення запису (овал 1). Якщо МКК увімкнено, сигнал виявлення запису також встановлює регістр STATUS (овал 2) (один біт на процесор у системі). Існує два типи пристроїв для відстеження:

1) асинхронна пам’ять поза чіпом;
2) підлеглі інтерфейси шини Avalon.

Для асинхронних пам'яті поза мікросхемою, що використовує трьох-становий міст Avalon, МКК реалізує механізм виявлення запису, асинхронно встановлюючи триггер D, коли заявляється сигнал дозволу на запис. Цей тригер використовує системний CLOCK і сигнали скидання. Вхідні дані прив'язані до логічного "0", так що наступний такт піднімає фронт тактової частоти і скидає виявлення запису. Налаштування тригера є асинхронним, оскільки не всі пристрої пам’яті мають сигнал синхронного ввімкнення запису. Синхронний набір не зможе виявити запис.

І навпаки, для пам’яті (на чіпі чи поза ним), доступ до якої здійснюється через підпорядкований порт шини Avalon (тобто, вбудовану оперативну пам’ять та позачипову синхронну динамічну оперативну пам’ять - SDRAM), шина Avalon забезпечує чітко визначений синхронний інтерфейс для пошуку . У цьому випадку спосіб виявлення запису - це просто провід, який слідкує за синхронним сигналом активації запису Avalon, зберігаючи тригер.

Рядки адреси реєструються під час кожного тактового циклу, використовуючи функцію виявлення запису як ввімкнення clock-а. Такий підхід гарантує, що правильна адреса зареєстрована для більшості блоків, як синхронно, так і асинхронно, незалежно від технології. Це пояснюється тим, що сигнал

ввімкнення запису зазвичай запускає запис (асинхронно або зчитується при зміні такту), і, отже, адреса повинна бути вже на шині, щоб відповідати часу налаштувань.

Потрібно звернути увагу на час синхронізації пристроїв для різної пам’яті, щоб перевірити коректність роботи, оскільки для різних типів пам’яті (асинхронної флеш-пам’яті, асинхронної SRAM чи синхронної SDRAM) цей час різний.

Регістр ADDRESS (овал 4) записує лише адресу на висхідному положенні після виявлення сигналу дозволу запису. Однак адресні рядки забезпечують зміщення лише для певного пристрою пам'яті, який не відповідає адресі, що використовується процесором та кешем. Адреса перетворюється у придатну для використання форму, шляхом виконання логічного OR з базовою адресою пристрою пам’яті перед записом в ADDRESS. Інструмент SOPC Builder потребує узгодження всіх периферійних адресних просторів з діапазоном адрес веденого пристрою. Як результат, крайні біти, гарантовано встановляться в 0, що дозволить побудувати повну адресу за допомогою логічної операції АБО. Після того, як адреса збережена в реєстрі ADDRESS, процесори отримають дозвіл на її читання, щоб відповідний рядок кешу був анульований належним чином.

Однією з вимог систем, побудованих на процесорі Nios є те, що кожен процесор повинен підтвердити будь-яке переривання, яке він обслуговує. Процесор Nios підтверджує переривання в ISR, гарантуючи, що ISR обслуговується. МКК повинен відстежувати, які процесори мали можливість підтвердити переривання (а отже, очистили кеш-пам'ять), і відмінити сигнал irq лише тоді, коли всі процесори завершили операцію інвалідації даних. Реєстр STATUS служить саме для задоволення цієї вимоги.

N-бітний регістр STATUS - це унітарний код ідентифікатора процесора (тобто біт 0 відповідає процесору 0, біт 1 відповідає процесору 1 тощо). N - кількість процесорів, присутніх у системі. До регістру застосовується операція логічного АБО, що створює запит на сигнал переривання(irq), повідомляючи всі процесори про те, що відбувся запис. Сигнал irq також відображається регістром CONTROL. Процесор підтверджує переривання, записуючи унітарний код свого

PID, тим самим скидаючи відповідний біт у STATUS. Коли всі процесори підтверджують переривання, логічна операція АБО скасовує сигнал irq. При зчитуванні регістру STATUS отримаємо інформацію про те, які процесори ще не підтвердили переривання.

З такою схемою підтвердження переривань витрати на апаратне забезпечення мінімальні – потрібен лише один додатковий триггер на процесор. Це робить систему масштабованою до тридцяти двох процесорів, що становить ширину даних інтерфейсу шини Avalon МКК.

Модуль когерентності кеш-пам’яті включає можливість зчитування та запису до периферійних регістрів через інтерфейс шини Avalon. У таблиці 4.1 описані ці регістри та їх зміщення.

Таблиця 3.1 – Регістри модуля когерентності кеш-пам’яті

Регістр	Зміщення	Ширина (Біт)	Читання\запис
STATUS	00	N	Read-write
CONTROL	01	1	Read-write
ADDRESS	02	32	Read-only

Слід зазначити, що системи з більш ніж тридцятьма двома процесорами можуть використовувати модуль когерентності пам’яті з незначними змінами, потрібно лише додати обладнання. Замість цього, кожен процесор може записати свій PID, закодований методом транспортного кодування, до реєстру СТАТУС, який, у свою чергу, декодується для встановлення відповідного біта у внутрішньому N-бітовому регістрі. Внутрішній регістр може мати довільний розмір, оскільки він не обмежений шириною шини даних.

3.3.3. Процес обробки переривань

Далі показаний код збірки для обробки ISR для прототипу МКК. Константи `na_ccm`, `np_ccmaddress` та `na_ccmstatus` представляють базову адресу модуля когерентності кеш пам’яті та зміщення регістрів для ADDRESS та STATUS, відповідно.

```
1 nr_ccmisr: pfx %hi(0x20) ; MOVIA %l0,na_ccm
2 movi %l0,0x0
3 pfx %hi(0x80)
4 movhi %l0,0x10
5 pfxio %hi(0x0) ; address = na_ccm->np_ccmaddress;
6 ldp %l6,[%l0,np_ccmaddress]
7 rdctl %l5 ; nm_caches_disable();
8 movhi %l5,0x0
9 wrctl %l5
10 nop
11 pfx %hi(0xa0) ; nm_icache_invalidate_line(address);
12 wrctl %l6
13 pfx %hi(0xe0) ; nm_dcache_invalidate_line(address);
14 wrctl %l6
15 rdctl %l5 ; nm_caches_enable();
16 movhi %l5,0x3
17 wrctl %l5
18 nop
19 pfx %hi(0x0) : %l1 = _cpuid
20 movi %l1,0x0
21 pfx %hi(0xa0)
22 movhi %l1,0x00
23 movi %l5,0x1 ; %l5 = 1
24 ldp %l7,[%l1,0x0] ; %l7 = *_cpuid
25 ext16d %l7,%l1
26 lsl %l5,%l7 ; na_ccm->np_ccmstatus = 1 << (*_cpuid);
27 stp [%l0,np_ccmstatus],%l5
28 nr_ccmisr_loop: pfxio %hi(0x0)
29    ldp %l5,[%l0,np_ccmstatus]
30 skprz %l5
31 br nr_ccmisr_loop
32 nop
33 tret %o7
```

Розглянемо уривок коду:

1) рядки 1-6 – Переривання потоку даних починається з отримання, адреси клітинки пам’яті, яку потрібно очистити з реєстру ADDRESS модуля когерентності кеш-пам’яті;

2) рядки 7-10 – далі вимикаються обидва кеші;

3) рядки 11-14 – анулювання відповідного рядка кешу;

4) рядки 15–18 – увімкнення кешів, що боли вимкненні на початку;

5) рядки 19–27 – підтвердження переривання, та запис PID до реєстру STATUS модуля когерентності кеш-пам’яті;

6) рядки 28–32 – циклічне блокування (spinlock) до моменту повного очищення реєстру STATUS (відбувається, коли всі процесори в системі визнали переривання).

Після виконання переривання, робота повертається у звичайне русло.

ISR для прототипу модуля когерентності кеш-пам’яті має тридцять три інструкції. В них включено три інструкції доступу до підпорядкованого порту МКК Avalon і один до PID ROM. Доступ до PID ROM не викликає проблем доступу до шини, оскільки кожен процесор має ексклюзивний доступ до пам'яті через неподільну шину.

Тепер потрібно вирішити проблему самозмінюваного коду(Self-modifying code) – це програмний прийом, який дозволяє коду модифікуватись самостійно(змінювати або створювати частину свого коду самостійно, під час виконання).

Наслідком того, що код самостійно змінюється, є те, що тепер окремий кеш інструкцій може містити некогерентні інструкції. Як результат, значення, записане в пам’ять можуть бути як інструкцією, так і даними, тому обидва кеші повинні бути анульовані (оскільки неможливо визначити, що саме представляє записане значення – інструкцію чи дані). Заборона самопереробного коду зменшує кількість інструкцій в ISR на два (рядки 11 і 12) і дозволяє кешу інструкцій залишатися ввімкненим (що призводить до покращення продуктивності, якщо якийсь код ISR потрапляє в кеш інструкцій).

Як уже згадувалось, для розробника систем на базі Nios доступні вектори переривань 16 - 63, де вектор 16 є найпріоритетнішим для переривань, доступним для використання модулем когерентності кеш-пам’яті. Така логіка введена з наступних причин:

1) внутрішні модулі, в тому ж числі модуль налагодження апаратного забезпечення, повинні мати вищий пріоритет перед очищенням кеш-пам'яті, таким чином, вектори переривань 0–2 будуть мати перевагу над МКК;

2) загальноприйнято, що дебагер повинен мати вищий пріоритет, щоб фіксувати всю поведінку системи, включаючи будь-які переривання модуля когерентності кеш-пам’яті, отже, вектори переривань 3–5 будуть мати перевагу над МКК;

3) решта векторів переривань (6–15) наразі не використовуються.

Таким чином, важливо, щоб розробник системи забезпечив, що під час налаштування системи, МКК отримує номер переривання 16.

3.4 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення для підтримки когерентності кеш-пам'яті досить прості. Як правило, вони передбачають створення точки бар'єру ініціалізації після ініціалізації процесора для всіх AP (процесори, не позначені як BSP). Це змушує APs чекати, поки не завершиться глобальна ініціалізація. Це схоже на Intel SMP-метод утримання AP в скиданні, поки BSP не завершить ініціалізацію. У цьому випадку глобальна ініціалізація складається в основному з встановлення ISR та включення МКК.

Якщо використовується стандартна процедура ініціалізації запуску _ start для кожного процесора, тоді BSP також повинен мати бар’єр, щоб переконатися, що всі точки доступу досягли точки ініціалізації. Це пов’язано з тим, що процедура запуску для кожного процесора включає певну глобальну ініціалізацію, наприклад очищення векторної таблиці переривань.

Якщо BSP встановлює ISR (у загальній процедурі ініціалізації) до того, як усі AP закінчать процедуру запуску, AP може очистити таблицю векторів переривань, що призведе до того, що обробник фальшивих переривань буде викликаний винятком МКК, а не ISR. Така друга бар'єрна точка може бути реалізована холостим циклом (цикл який перевіряє настання умови, до моменту настання цієї умови, у системі використовується для очікування), довжина якого масштабується відповідно до кількості процесорів у системі.

Альтернативно, програміст може розробити власну програму запуску з усіма своїми правилами глобальної ініціалізації, переміщеними до загальної

підпрограми ініціалізації. Це віднімає необхідність в стандартному процесі завантаження. У тестах, що використовуються для перевірки системи, використовується стандартна процедура запуску. Однак для виробничих систем рекомендується забезпечити більш ефективну власну процедуру.

Нарешті, кожен Nios має регістри фреймів і стеків, що вказують на приватну пам’ять стека кожного процесора. Оскільки стеки є приватними, цими вказівниками слід керувати таким чином, щоб процесори випадково не перезаписували будь-які приватні дані в стеках інших процесорів. За це управління відповідає програма або операційна система.

3.5 Тестування

Тестування розроблювальної мультипроцесорної системи на основі топології гіперкуб є однією з найважливіших аспектів розробки вцілому. У цьому розділі будуть описані використовувані тести, сам процес тестування, його результати і проблеми виявлені(чи не виявлені) під час тестування.

3.5.1 Тест спільної пам’яті

Щоб перевірити, що система насправді використовує когерентність кеш-пам'яті, була написана проста програма, яка використовує спільну пам'ять, використовуючи набір інструментів розробки GCC, наданий Altera. Для цілей тесту один процесор був довільно визначений як BSP. Програма починається з ініціалізації за замовчуванням (процедура запуску), що включає включення переривань та ініціалізацію кешу. Усі процесори, у спільній пам’яті, крім BSP, змушені синхронізуватися і встановити змінну блокування.

Лише після цього може бути виконана будь-яка глобальна ініціалізація, наприклад ініціалізація векторної таблиці переривань та увімкнення модуля когерентності кешу. В кінці глобальної ініціалізації BSP встановлює блокуючу змінну в 1. Якщо встановлення відбулося, усі процесори можуть продовжувати виконання програми. Такі дії реалізують бар'єр ініціалізації, згаданий у

попередньому розділі. Оскільки APs в иконують холостий цикл, данні блокування зберігаються в кожному кеші даних.

Якщо когерентність кеш-пам’яті не застосовується, то процесори ніколи не пройдуть бар’єр. І навпаки, якщо система повністю функціональна, тоді процесори звільняються від блокування, коли BSP запише 1, а МКК зробить недійсною адресу спільної пам'яті. Цей тест називається тестом "спільної пам'яті". Список кодів для цього тесту можна знайти в додатку В.

Цей тест вважається пройденим, коли , усі процесори, при ввімкненому МКК, прозодять точку синхронізації, при цьому містять такий ID для кожного процесора, який зазначений в повідомленнях в консолі.

Коли МКК відключений, лише BSP надсилає своє повідомлення, оскільки у кеші є блокуюче значення, тоді як APs очікують в холостому циклі. Помилка означає, що запис BSP не поширюється на AP.

Очікується, що всі однопроцесорні системи повинні пройти цей тест, оскільки когерентність кешу не є проблемою в таких системах. Будь-які багатопроцесорні системи, які не оснащені модулем когерентності кеш-пам’яті, не повинні пройти цей тест, довівши, що когерентність кешу є проблемою. Отже, наша початкова система (без МКК), не повинна пройти тест, у той час як друга версія (з МКК) повинна пройти тест успішно.

3.5.2 Тест мультизапису

Другий тест, що названий тестом «мультизапису», виявляє критичний недолік оригінальної конструкції МКК та перевіряє, чи був усунутий недолік. Тест схожий на тест спільної пам'яті, але були додані блоки збірного коду, щоб забезпечити наявність декількох записів у конвеєрі процесора. Оскільки інші процесори завантажують значення зі спільної пам'яті в кеш, перед записом, система вважається повністю функціональною, лише якщо усі процесори басать всі нові значення. Код для цього тесту можна знайти в додатку Г.

Для проходження тесту на мультизапис, потрібно, щоб, усі процесори читали відповідні значення для адрес пам'яті, які були записані послідовними

інструкціями зберігання, при ввімкнутому МКК. Зчитування старого значення для будь-якого місця пам'яті будь-яким процесором вважається помилкою. Очікується, що всі однопроцесорні системи та системи, обладнані прогресивними модулями когерентності, пройдуть це випробування, показуючи, що критична вада багатозапису була усунена покращеною архітектурою. Очікується, що багатопроцесорні системи з розробленим МКК не пройдуть цей тест через недолік, наявний у розробленому модулі.

3.5.3 Результати тестування

Програма тестування спільної пам'яті виконувалася двічі в кожній системі, один раз із увімкненим модулем когерентності і один раз із вимкненим(щоб гарантувати, що очікувана різниця в результатах була обумовлена роботою МКК). Тестованими системами були однопроцесорна та 4-, 8-, 16-процесорні (гуперкуби 2, 3 та 4 степенів) системи SMPORC двох класів: без МКК, з прототипом МКК.

У таблиці 3.2 наведено результати тестування. Як можна бачити, результати збігаються з очікуваними, вказуючи на те, що розроблений модуль не здатний повністю забезпечити узгодженість кеш-пам'яті в системі.

Таблиця 3.2 – Результати тестування

Кількість процесорів	Тест спільної пам’яті		Тест мультизапису	
	Без МКК	З МКК	Без МКК	З МКК
1	Pass	Pass	Pass	Pass
4	Fail	Pass	Fail	Fail
8	Fail	Pass	Fail	Fail
16	Fail	Pass	Fail	Fail

Чому ж система з модулем когерентності кеш-пам’яті не пройшла тест мультизапису? Розроблений МКК, хоч і дозволяє виявити недійсні записи та рядки в кеш-пам’ять, але і містить свої недоліки.

Одним з недоліків є те, що всі процесори очищають вказаний рядок кешу незалежно від того, містить він справжню адресу чи ні (тобто Nios не перевіряє тег рядка кешу перед скиданням дійсного біта). Це обмеження процесора Nios та способу очищення кешу.

Іншим обмеженням є те і це саме той недолік, що спричиняє непроходження тесту, що оскільки для всіх процесорів існує одна спільна лінія переривань, усі процесори виконують ISR. Сюди входить процесор запису. Крім того, процесор запису очистить кеш, навіть якщо він містить поточне та правильне значення. Таким чином, зчитування з цього місця спричиняє промах кешу, збільшуючи час очікування.

Це можна виправити, визначивши, який процесор виконував запис, і не встановлюючи свій біт STATUS. Якщо припустити, що використовується однобітний унітарний код реєстру STATUS, це дозволить ISR перевірити відповідний біт STATUS і пропустити етап очищення кешу, для цього конкретного процесора біт не встановлено. Цю можливість можна застосувати, переглядаючи внутрішні, неопубліковані сигнали шини Avalon, однак це може бути поганою ідеєю, оскільки Altera не надає жодної конкретної реалізації для внутрішніх елементів модуля шини Avalon.

3.6 Висновки

У цьому розділі було розроблено мультипроцесорну систему загального призначення на основі топології гіперкуб. Розглянуто програмно апаратні засоби що дозволили розробити та змоделювати роботу такої системи.

Для масштабованості розробили окремий процесорний модуль, що дозволив гнучко змінювати кількість процесорів в системі і будувати різностепеневі гіперкуби.

Також була описана проблема когерентності кеш-пам’яті у таких системах, для вирішення цієї проблеми був створений модуль когерентності кеш-пам’яті. Відбулося тестування розробленої мультипроцесорної системи як з модулем так і без.

ВИСНОВКИ

Робота представляє собою розробку мультипроцесорної системи загального призначення на основі топології гіперкуб.

У ході роботи було описано важливість створення такої системи, розглянуто основні типи архітектур мультипроцесорних систем. Шляхом порівняння було обрано симетричну архітектуру, адже вона давала ширші можливості як для розробки так і для масштабування системи. Також розглянуто уже існуючі рішення, розглянуто основні проблеми готових систем, внаслідок чого сформульовано вимоги до розроблювальної системи.

Гіперкуб – це топологія що дає одні з найкращих характеристик швидкості та продуктивності, значною її перевагою є те, що вона підтримує велику кількість вузлів і легко масштабується, тому аналіз ведеться, фактично для трьох варіацій систем з 4-, 8- та 16- процесорами. При аналізі можливого програмно-апаратного забезпечення вибір впав на продукцію фірми Altera, як і в питаннях середовища розробки, так і в питаннях апаратного забезпечення та засобів візуалізації.

Спочатку було розроблено класичну модель SMP-системи, але вона є дуже проблемною у функціонуванні при використанні великої кількості процесорів, адже шина стає вузьким місцем. Для вирішення цієї проблеми було додано апаратний кеш кожному з процесорів, але це породило нову проблему – проблему когерентності кеш-пам’яті, тому було розроблено модуль когерентності кеш-пам’яті.

Для підтвердження правильності функціонування системи було проведено тестування двома видами тестів – тестом спільної пам’яті та тестом мультизапису. Це показало, що розроблений модуль має певні недоліки у своїй архітектурі і може бути покращений. Сама ж система працює досить коректно та може використовуватись для виконання різних задач.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. S. Jin, Z. Huang, R. Diao, D. Wu and Y. Chen. Comparative Implementation of High Performance Computing for Power System Dynamic Simulations. *IEEE Transactions on Smart Grid*, May 2017. vol. 8, no. 3, pp. 1387-1395, DOI: 10.1109/TSG.2016.2647220.

2. G. Valente et al. A Flexible Profiling Sub-System for Reconfigurable Logic Architectures. *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 2016. pp. 373-376, DOI: 10.1109/PDP.2016.86.

3. Nikov, K., Hosseinabady, M., Asenjo, R., Rodríguezz, A., Navarro, A., & Nunez-Yanez, J. *High-Performance Simultaneous Multiprocessing for Heterogeneous System-on-Chip.* 2020. DOI: arXiv preprint arXiv:2008.08883.

4. E. A. Rambo, O. P. Henschel and L. C. V. dos Santos. On ESL verification of memory consistency for system-on-chip multiprocessing. *Automation & Test in Europe Conference & Exhibition (DATE).* 2012. pp. 9-14, DOI: 10.1109/DATE.2012.6176424.

5. Peter Tröger, Andreas Polze. Proceedings of the 4th Many-core Applications Research Community Symposium. Potsdam : Universitätsverlag Potsdam 2012. p.57.

6. Girish Rao Bulusu. Asymmetric Multiprocessing Real Time Operating System on Multicore Platforms. ARIZONA STATE UNIVERSITY, December 2014. p. 157.

7. K. Yu, D. Han, C. Youn, S. Hwang and J. Lee. "*Power-aware task scheduling for big.LITTLE mobile processor*": International SoC Design Conference (ISOCC), 2013. pp. 208-212, DOI: 10.1109/ISOCC.2013.6864009.

8. K. Yu, D. Han, C. Youn, S. Hwang and J. Lee. "*Power-aware task scheduling for big.LITTLE mobile processor*": International SoC Design Conference (ISOCC), 2013. pp. 208-212, DOI: 10.1109/ISOCC.2013.6864009.

9. Kryzhanovsky, M.V., Malsagov, M.Y. Neuron network methods of task assignment in multiprocessing system. *Opt. Mem. Neural Networks*, 2010. pp. 213–219. DOI: https://doi.org/10.3103/S1060992X10030021

10. Dai Bui, Edward Lee, Isaac Liu, Hiren Patel, and Jan Reineke. "*Temporal isolation on multiprocessing architectures*". In Proceedings of the 48th Design

Automation Conference. Association for Computing Machinery, 2011. New York, NY, USA, pp. 274–279. DOI:https://doi.org/10.1145/2024724.2024787

11. T. Liang and V. Dinavahi. Real-Time Device-Level Simulation of MMC-Based MVDC Traction Power System on MPSoC. *IEEE Transactions on Transportation Electrification*, June 2018. vol. 4, no. 2, pp. 626-641. DOI: 10.1109/TTE.2018.2823059.

12. T. Liang and V. Dinavahi. Real-Time Device-Level Simulation of MMC-Based MVDC Traction Power System on MPSoC. *IEEE Transactions on Transportation Electrification*, June 2018. vol. 4, no. 2, pp. 626-641. DOI: 10.1109/TTE.2018.2823059.

13. Jaeyoung Yun, Jinsu Park and Woongki Baek. *HARS: A heterogeneity-aware runtime system for self-adaptive multithreaded applications*: 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2015. pp. 1-6. DOI: 10.1145/2744769.2744848.

14. S. Ahmad, V. Boppana, I. Ganusov, V. Kathail, V. Rajagopalan and R. Wittig. A 16-nm Multiprocessing System-on-Chip Field-Programmable Gate Array Platform. *IEEE Micro*, Mar.-Apr. 2016. vol. 36, no. 2, pp. 48-62. DOI: 10.1109/MM.2016.18.

15. A. Somdip Dey. SD-MARC: A New Multi-Processor Architecture. Department of Computer Science, St. Xavier's College. 2011. URL: http://worldcomp-proceedings.com/proc/p2012/CDE3285.pdf

16. J. Happe, H. Groenda, M. Hauck and R. H. Reussner. A Prediction Model for Software Performance in Symmetric Multiprocessing Environments. *Seventh International Conference on the Quantitative Evaluation of Systems*. 2010. pp. 59-68. doi: 10.1109/QEST.2010.15.

17. J. Devietti, B. Lucia, L. Ceze and M. Oskin. DMP: Deterministic Shared-Memory Multiprocessing. *IEEE Micro*, Jan.-Feb. 2010. vol. 30, no. 1, pp. 40-49. doi: 10.1109/MM.2010.14.

18. Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. Concurrency: the Works of Leslie Lamport. *Association for Computing Machinery*, 1019 New York, NY, USA, 179–196. DOI:https://doi.org/10.1145/3335772.3335934

19. Xu Zhou, Kai Lu, Xiaoping Wang, Xu Li. Exploiting parallelism in deterministic shared memory multiprocessing. *Journal of Parallel and Distributed Computing*. 2012. Volume 72, Issue 5,Pages 716-727. DOI: https://doi.org/10.1016/j.jpdc.2012.02.008.

20. Nunez-Yanez, J., Amiri, S., Hosseinabady, M. Simultaneous multiprocessing in a software-defined heterogeneous FPGA. *J Supercomput.* 2019. 75, 4078–4095 . DOI: https://doi.org/10.1007/s11227-018-2367-9

21. Kang, T.W., Hong, C.H. IFC-CityGML LOD mapping automation using multiprocessing-based screen-buffer scanning including mapping rule. *KSCE J Civ Eng*. 2018. 22, 373–383. https://doi.org/10.1007/s12205-017-0595-9

22. Tao Li, Shaokai Wang, Feng Luo, Fang-Xiang Wu, Jianxin Wang. MultiGuideScan: a multi-processing tool for designing CRISPR guide RNA libraries. *Bioinformatics*. February 2020. Volume 36, Issue 3, 1 Pages 920–921. DOI: https://doi.org/10.1093/bioinformatics/btz616

23. F. A. Samman, F. Philipp and M. Glesner. Reconfigurable interconnect infrastructure for multi-FPGA-based adaptive multiprocessing systems: *1st International Workshop on Computing in Heterogeneous, Autonomous 'N' Goal-Oriented Environments (CHANGE).* 2011. pp. 1-8. doi: 10.1109/CHANGE.2011.6172451.

24. Xu Zhou, Kai Lu, Xiaoping Wang, Xu Li. Exploiting parallelism in deterministic shared memory multiprocessing. *Journal of Parallel and Distributed Computing*. 2012. Volume 72, Issue 5, Pages 716-727. DOI: https://doi.org/10.1016/j.jpdc.2012.02.008.

25. Bienia C. Benchmarking modern multiprocessors. *Princeton University*, 2011.145 c.

26. Chi Zhang, Xin Yuan and A. Srinivasan. Processor affinity and MPI performance on SMP-CMP clusters. *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. 2010. pp. 1-8. DOI: 10.1109/IPDPSW.2010.5470774.

27. A. Mello, I. Maia, A. Greiner and F. Pecheux. Parallel simulation of systemC TLM 2.0 compliant MPSoC on SMP workstations. *Design, Automation & Test in*

Europe Conference & Exhibition (DATE 2010). 2010. pp. 606-609. DOI: 10.1109/DATE.2010.5457136.

28. W. Rekik, M. Ben Said, N. Ben Amor and M. Abid. Virtual prototyping of multiprocessor architectures using the open virtual platform. *International Conference on Computer Applications Technology (ICCAT)*. 2013. pp. 1-6. DOI: 10.1109/ICCAT.2013.6522061.

29. P. Abad, P. Prieto, L. G. Menezo, A. Colaso, V. Puente and J. Gregorio. TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers. *IEEE/ACM Sixth International Symposium on Networks-on-Chip*. 2012. pp. 99-106, DOI: 10.1109/NOCS.2012.19.

30. H. Yun, G. Yao, R. Pellizzoni, M. Caccamo and L. Sha. Memory Access Control in Multiprocessor for Real-Time Systems with Mixed Criticality. *24th Euromicro Conference on Real-Time Systems*. 2012. pp. 299-308. DOI: 10.1109/ECRTS.2012.32.

31. Orathai Sukwong and Hyong S. Kim. Is co-scheduling too expensive for SMP VMs? In Proceedings of the sixth conference on Computer systems. *Association for Computing Machinery*. 2011. New York, NY, USA, 257–272. DOI: https://doi.org/10.1145/1966445.1966469

32. K. Balston, M. Karimibiuki, A. J. Hu, A. Ivanov and S. J. E. Wilton. Post-silicon code coverage for multiprocessor system-on-chip designs. *IEEE Transactions on Computers*. Feb. 2013. vol. 62, no. 2, pp. 242-246. DOI: 10.1109/TC.2012.163.

33. B. Gerofi and Y. Ishikawa. RDMA Based Replication of Multiprocessor Virtual Machines over High-Performance Interconnects. *IEEE International Conference on Cluster Computing*. 2011. pp. 35-44. DOI: 10.1109/CLUSTER.2011.13.

34. J. Ras and A. M. K. Cheng. Response Time Analysis of the Abort-and-Restart Model under Symmetric Multiprocessing. *10th IEEE International Conference on Computer and Information Technology*. 2010. pp. 1954-1961. DOI: 10.1109/CIT.2010.332.

35. A. Bastoni, B. B. Brandenburg and J. H. Anderson. An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor EDF Schedulers. *31st IEEE Real-Time Systems Symposium*. 2010. pp. 14-24. DOI: 10.1109/RTSS.2010.23.

36. Paolo D'alberto, Marco Bodrato, and Alexandru Nicolau. Exploiting parallelism in matrix-computation kernels for symmetric multiprocessor systems: Matrix-multiplication and matrix-addition algorithm optimizations by software pipelining and threads allocation. *ACM Trans. Math*. November 2011. Softw. 38, 1, Article 2. DOI: https://doi.org/10.1145/2049662.2049664

37. A. Burns and A. J. Wellings. A Schedulability Compatible Multiprocessor Resource Sharing Protocol – MrsP. *25th Euromicro Conference on Real-Time Systems*. 2013. pp. 282-291. DOI: 10.1109/ECRTS.2013.37.

38. C. Su, D. Li, D. S. Nikolopoulos, K. W. Cameron, B. R. d. Supinski and E. A. León. Model-based, memory-centric performance and power optimization on NUMA multiprocessors. *IEEE International Symposium on Workload Characterization (IISWC)*. 2012. pp. 164-173. DOI: 10.1109/IISWC.2012.6402921.

39. Zoltan Majo and Thomas R. Gross. Memory system performance in a NUMA multicore multiprocessor. In Proceedings of the 4th Annual International Conference on Systems and Storage. *Association for Computing Machinery*. 2011. New York, NY, USA, Article 12, 1–10. DOI:https://doi.org/10.1145/1987816.1987832

40. V. Schwambach, S. Cleyet-Merle, A. Issard and S. Mancini. Fast parallel application and multiprocessor design space exploration from sequential code. *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2015. pp. 163-172. DOI: 10.1109/CODESISSS.2015.7331379.

41. Tanveer, M., Iqbal, M. A., & Azam, F. Using Symmetric Multiprocessor Architectures for High Performance Computing Environments. *International Journal of Computer Applications*. 2011. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.4182&rep=rep1&type=pdf

42. Anuradha, Dande. Simulation of Multiprocessor System Scheduling. *MS thesis*. 2014. URL: https://trepo.tuni.fi/handle/123456789/22144

43. Jiang Y., Tian K., Shen X. Combining Locality Analysis with Online Proactive Job Co-scheduling in Chip Multiprocessors. *Lecture Notes in Computer Science*: vol 5952. Springer, Berlin, Heidelberg, 2010. DOI: https://doi.org/10.1007/978-3-642-11515-8_16

44. SIERRA - IBM POWER SYSTEM AC922, IBM POWER9 22C 3.1GHZ, NVIDIA, VOLTA GV100, DUAL-RAIL MELLANOX EDR INFINIBAND, TOP 500, 2018, URL:https://www.top500.org/system/179398/

45. Using LC's Sierra Systems, *Lawrence Livermore National Laboratory*, 2018 URL: https://hpc.llnl.gov/hardware/platforms/sierra

46. Summit. *Oak Ridge National Laboratory*. 2015. URL: https://www.olcf.ornl.gov/summit/

47. SUMMIT - IBM POWER SYSTEM AC922, IBM POWER9 22C 3.07GHZ, NVIDIA VOLTA GV100, DUAL-RAIL MELLANOX EDR INFINIBAND, TOP500. 2018. URL: https://www.top500.org/system/179397/

48. NVIDIA Variable SMP – A Multi-Core CPU Architecture for Low Power and High Performance. 2011. URL: https://www.nvidia.com/content/pdf/tegra_white_papers/tegra-whitepaper-0911b.pdf

49. ALTERA Nios II Classic Processor Reference Guide. San Jose, 2016. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf

50. ALTERA DE1-SoC User Manual. March 2014. URL: https://courses.cs.washington.edu/courses/cse467/15wi/docs/DE1_SoC_User_Manual.pdf

Додаток А

ІНСТАЛЯЦІЯ ISR СИСТЕМИ VHDL

```
1 .include "excalibur.s"
2 .text
3 .global nr_installsystemisr
4 .global nr_ccmisr
5
6 ;---------------------------------------
7 ; void nr_installsystemisr(int trapNumber, nios_isrhandlerproc3 *trapProc);
8 ;
9 ; Description: Install a trap routine 84 System ISR Installer 85
10 ; Input: %o0 = trap number
11 ; %o1 = trap handler routine
12 ; Output: none
13 ; Side Effects: %g0 & %g1 altered
14 ; CWP Depth: 0
15 ;
16
17 .ifdef __nios32__
18 .equ nmul,2
19 .else
20 .equ nmul,1
21 .endif
22
23 .equ _ISRManagerDebugging_,0
24
25 nr_installsystemisr:
26    MOV %g0,%o0 ; %g0 = Trap Number
27    LSLI %g0,nmul ; %g0 = offset into vector table
28
29    MOVIA %g1,nasys_vector_table ; %g1 -> vector table
30    ADD %g1,%g0 ; %g1 -> entry for this trap in table
```

```
31
32    ST [%g1],%o1 ; install the handler routine last
33
34    JMP %o7 ; return 35 NOP ; delay slot
```

Додаток Б

VHDL-КОД МОДУЛЯ КОГЕРЕНТНОСТІ КЕШ-ПАМ'ЯТІ

```
1 library altera_vhdl_support;
2 use altera_vhdl_support.altera_vhdl_support_lib.all;
3
4 library ieee;
5 use ieee.std_logic_1164.all;
6
7 ENTITY ccm IS
8 GENERIC(NUM_NIOS : integer:= 2);
9 PORT( -- Avalon slave port
10 signal address : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
11 signal chipselect : IN STD_LOGIC;
12 signal clk : IN STD_LOGIC;
13 signal read_n : IN STD_LOGIC;
14 signal reset_n : IN STD_LOGIC;
15 signal write_n : IN STD_LOGIC;
16 signal writedata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
17 signal irq : OUT STD_LOGIC;
18 signal readdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
19
20 -- external tri-state bus signals
21 signal ext_ram_bus_address : IN STD_LOGIC_VECTOR(22 DOWNTO 0);
22 signal ext_ram_bus_writen : IN STD_LOGIC;
23 signal write_n_to_the_ext_sram : IN STD_LOGIC
24 );
25 END ccm;
26
27
28 ARCHITECTURE europa OF ccm IS
29 SIGNAL internal_we : STD_LOGIC;
30 SIGNAL internal_writedetect : STD_LOGIC;
31 SIGNAL strobe_read : STD_LOGIC;
32 SIGNAL strobe_write : STD_LOGIC;
33 -- interrupt driver, address 0x00
34 SIGNAL reg_status : STD_LOGIC_VECTOR(NUM_NIOS-1 DOWNTO 0);
35 -- interrupt enable/disable, address 0x01
36 SIGNAL reg_control : STD_LOGIC;
37 -- byte address store, address 0x02
38 SIGNAL reg_address : STD_LOGIC_VECTOR(31 DOWNTO 0);
39 SIGNAL read_mux_out : STD_LOGIC_VECTOR(31 DOWNTO 0);
40 SIGNAL address_mux : STD_LOGIC_VECTOR(31 DOWNTO 0);
41 BEGIN
42
43 -- write detect signal for crossing into a synchronous domain
44 process (clk, reset_n, internal_we) begin
45 if reset_n = '0' then
46 internal_writedetect <= '0';
47 elsif internal_we = '0' then
48 internal_writedetect <= '1';
49 elsif clk'event and clk = '1' then
50 internal_writedetect <= '0';
51 end if;
52 end process;
53
54 strobe_read <= chipselect AND NOT read_n;
55 strobe_write <= chipselect AND NOT write_n;
```

```
56 -- STATUS REGISTER: interrupt status bit
57 process (clk, reset_n) begin
58 if reset_n = '0' then
59 reg_status <= (OTHERS => '0');
60 elsif clk'event and clk = '1' then
61 if std_logic'((strobe_write AND
62 to_std_logic(address = "00"))) = '1' then
63 reg_status <= reg_status AND
64 NOT writedata(NUM_NIOS-1 DOWNTO 0);
65 elsif std_logic'(internal_writedetect) = '1' AND
66 reg_control = '1' then 88
67 reg_status <= (OTHERS => '1');
68 end if;
69 end if;
70 end process;
71
72 -- CONTROL REGISTER: bit 0 is the interrupt enable bit
73 process (clk, reset_n) begin
74 if reset_n = '0' then
75 reg_control <= '0';
76 elsif clk'event and clk = '1' then
77 if std_logic'(strobe_write AND
78 to_std_logic(address = "01")) = '1' then
79 reg_control <= writedata(0);
80 end if;
81 end if;
82 end process;
83
84 -- ADDRESS REGISTER
85 process (reset_n, clk) begin
86 if reset_n = '0' then
87 reg_address <= x"00000000";
88 elsif clk'event AND clk = '1' then
89 if internal_writedetect = '1' AND reg_control = '1' then
90 reg_address <= address_mux;
91 end if;
92 end if;
93 end process;
94
95 -- Combinational register reads (read_wait_states = "0")
96 read_mux_out <= A_EXT(reg_status, 32) WHEN address = "00" else
97 A_REP(reg_control, 32) WHEN address = "01" else
98 reg_address WHEN address = "10" else
99 x"FFFFFFFF";
100 readdata <= read_mux_out when strobe_read = '1' else
101 x"00000000";
102
103 -- Combinational address mux
104 address_mux <= A_WE_StdLogicVector(
105 (std_logic'(write_n_to_the_ext_sram) = '1'),
106 "00000000" & (("0" & ext_ram_bus_address)
107 OR "10000000000000000000000"),
108 "000000000" & ((ext_ram_bus_address
109 OR "00000000000000000000000")) );
110
111 internal_we <= ext_ram_bus_writen AND write_n_to_the_ext_sram;
112 irq <= or_reduce(reg_status) AND reg_control;
113
114 END europa;
```

Додаток В

VHDL-КОД ТЕСТУ СПІЛЬНОЇ ПАМ'ЯТІ

```
1 #include "nios.h"
2 #include
3
4 #define BOOT_CPU 0
5 #ifdef na_ccm_s0
6 #define na_ccm na_ccm_s0
7 #define na_ccm_irq na_ccm_s0_irq
8 #endif
9
10 void global_initialize(int cpuid);
11
12 const char *_cpuid = (char *)na_cpuid_cpu0;
13 int *synch = (int *)0x008FF014;
14
15 int main(void) {
16 int context = *_cpuid;
17
18 /* pre-load caches */
19 (*synch) = 0;
20
21 printf("%d\n", context);
22 global_initialize(context);
23
24 /* AP synchronization point */
25 while((*synch) == 0) {
26 nr_delay(1000); printf("-%d", context);
27 }
28
29 nr_delay(100);
30 printf("+%d\n", context);
31
32 while(1) {;}
33
34 return 0;
35 }
36
37 /* global_initialize: setup that must be performed by only the BSP */
38 void global_initialize(int cpuid) {
39 if(cpuid == BOOT_CPU) {
40 #ifdef na_ccm
41 /* FIXME: This nr_delay is to "synchronize" the system...
42 * allow all APs in the system to get to the while loop
43 * before proceeding. The delay value scales with the
44 * number of processors in the system. */
45 nr_delay(5000);
46
47 /* install CCM ISR */
48 nr_installsystemisr(na_ccm_irq, nr_ccmisr);
49
50 /* enable CCM */
51 na_ccm->np_ccmcontrol = 1;
52 #endif
53
54 /* synchronize system */
55 (*synch) = 1;
```

```
56 }
57 }
```

Додаток Г

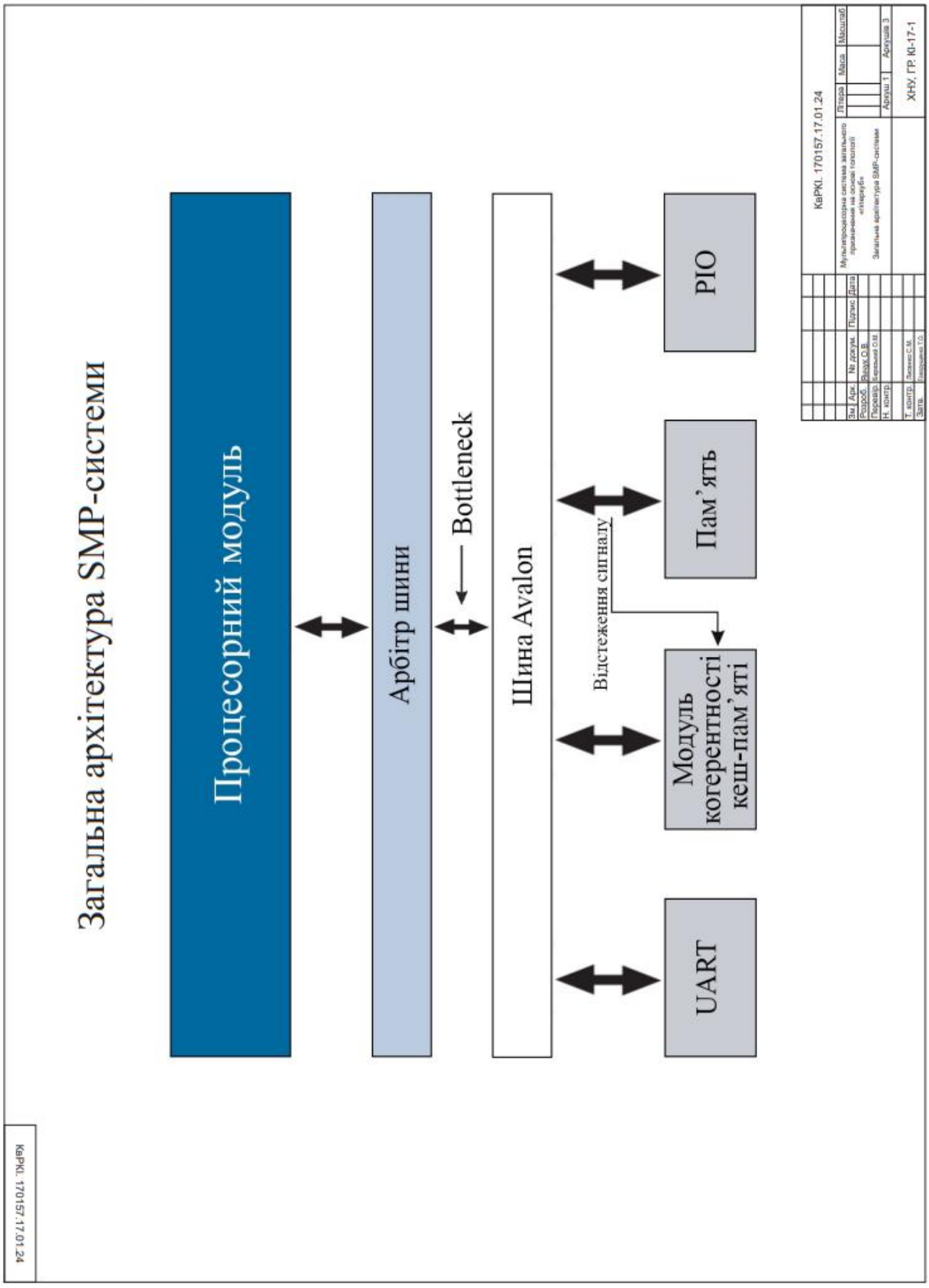
VHDL-КОД ТЕСТУ МУЛЬТИЗАПИСУ

```
1 #include "nios.h"
2 #include
3
4 #undef WORST_CASE
5 #define BOOT_CPU 0
6 #ifdef na_ccm_s0
7 #define na_ccm na_ccm_s0
8 #define na_ccm_irq na_ccm_s0_irq
9 #endif
10
11 void global_initialize(int cpuid);
12
13 const char *_cpuid = (char *)na_cpuid_cpu0;
14 int *synch = (int *)0x008FF014;
15
16
17 int main(void) {
18 int context = *_cpuid;
19 int i;
20
21 /* pre-load caches */
22 (*synch) = 0;
23 *(synch + 1) = 0;
24
25 printf("%d\n", context);
26 global_initialize(context);
27 global_initialize(context);
28
29 /* AP synchronization point */
30 while((*synch) == 0) {
31 nr_delay(1000);
32 if(*(synch + 1) == 0) printf("-%d", context);
33 else printf("*%d", context);
34 }
35
36 printf("+%d\n", context);
37
38 while(1) {;}
39
40 return 0;
41 }
42
43
44 /* global_initialize: setup that must be performed by only the BSP */
45 void global_initialize(int cpuid) {
46 static int i = 0;
47 if(cpuid == BOOT_CPU) {
48 #ifdef na_ccm
49 /* FIXME: This nr_delay is to "synchronize" the system...
50 * allow all APs in the system to get to the while loop
51 * before proceeding. The delay value scales with the
52 * number of processors in the system. */
53 nr_delay(5000);
54
55 /* install CCM ISR */
```

```
56 nr_installsystemisr(na_ccm_irq, nr_ccmisr);
57
58 /* enable CCM */
59 na_ccm->np_ccmcontrol = i;
60 #endif
61
62 /* synchronize system */
63 (*synch) = i;
64
65 #ifndef WORST_CASE
66 /* a second consecutive write to memory */
67 asm( "stp [%0,0x1],%1 \n", : : "r" (synch), "r" (i) );
68 #else
69 /* worst-case: maximum in-flight writes to memory */
70 asm( "stp [%0,0x1],%1 \n stp [%0,0x2],%1 \n \
71 stp [%0,0x3],%1 \n stp [%0,0x4],%1 \n \
72 stp [%0,0x5],%1 \n stp [%0,0x6],%1 \n \
73 stp [%0,0x7],%1 \n stp [%0,0x8],%1 \n \
74 stp [%0,0x9],%1 \n stp [%0,0xA],%1 \n \
75 stp [%0,0xB],%1 \n stp [%0,0xC],%1 \n \
76 stp [%0,0xD],%1 \n stp [%0,0xE],%1 \n" \
77 : \
78 : "r" (synch), "r" (i) );
79 #endif
80
81 i = 1;
82 }
83
84 }
```

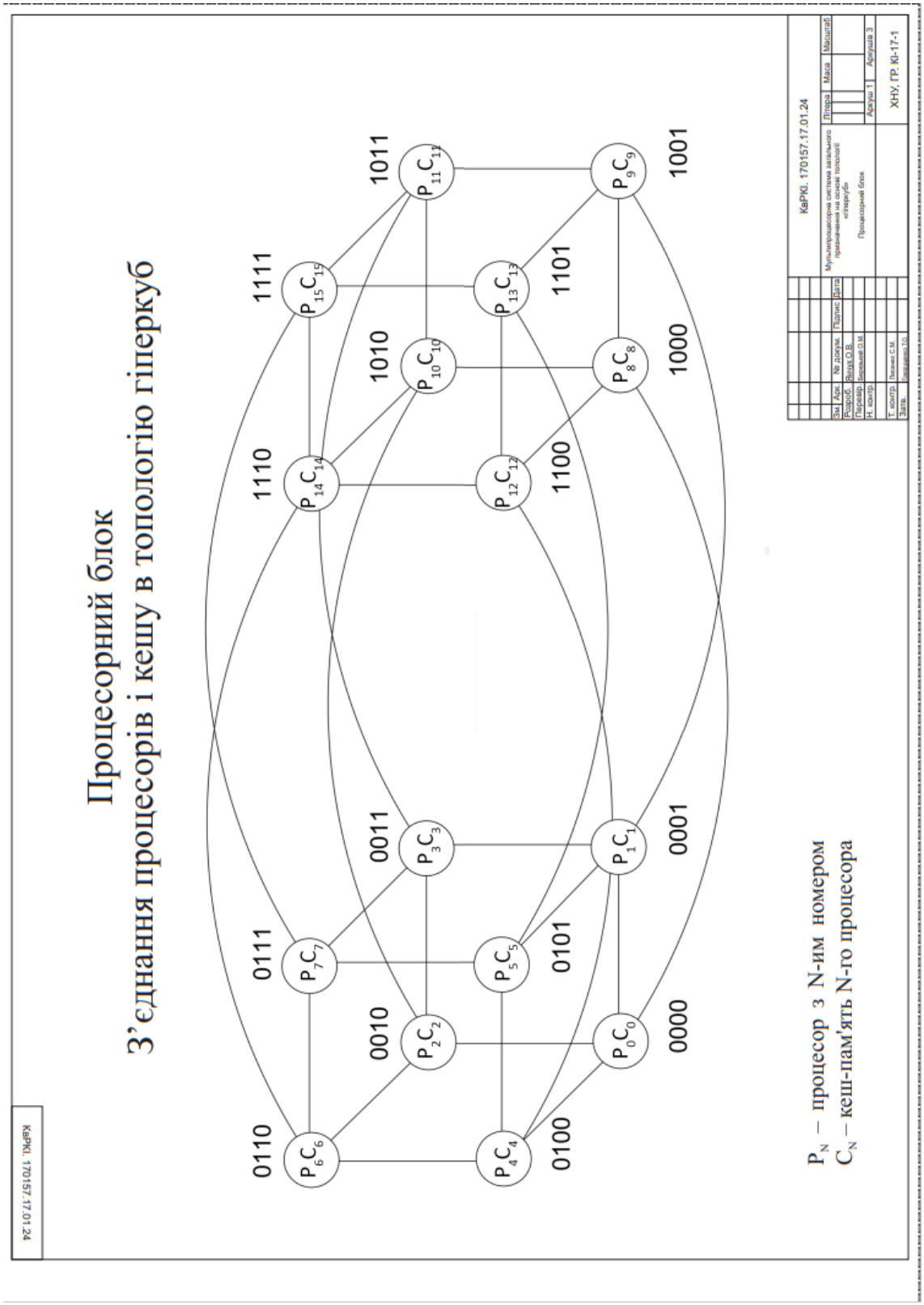
Додаток Д

Копія креслення «Загальна архітектура SMP-системи»



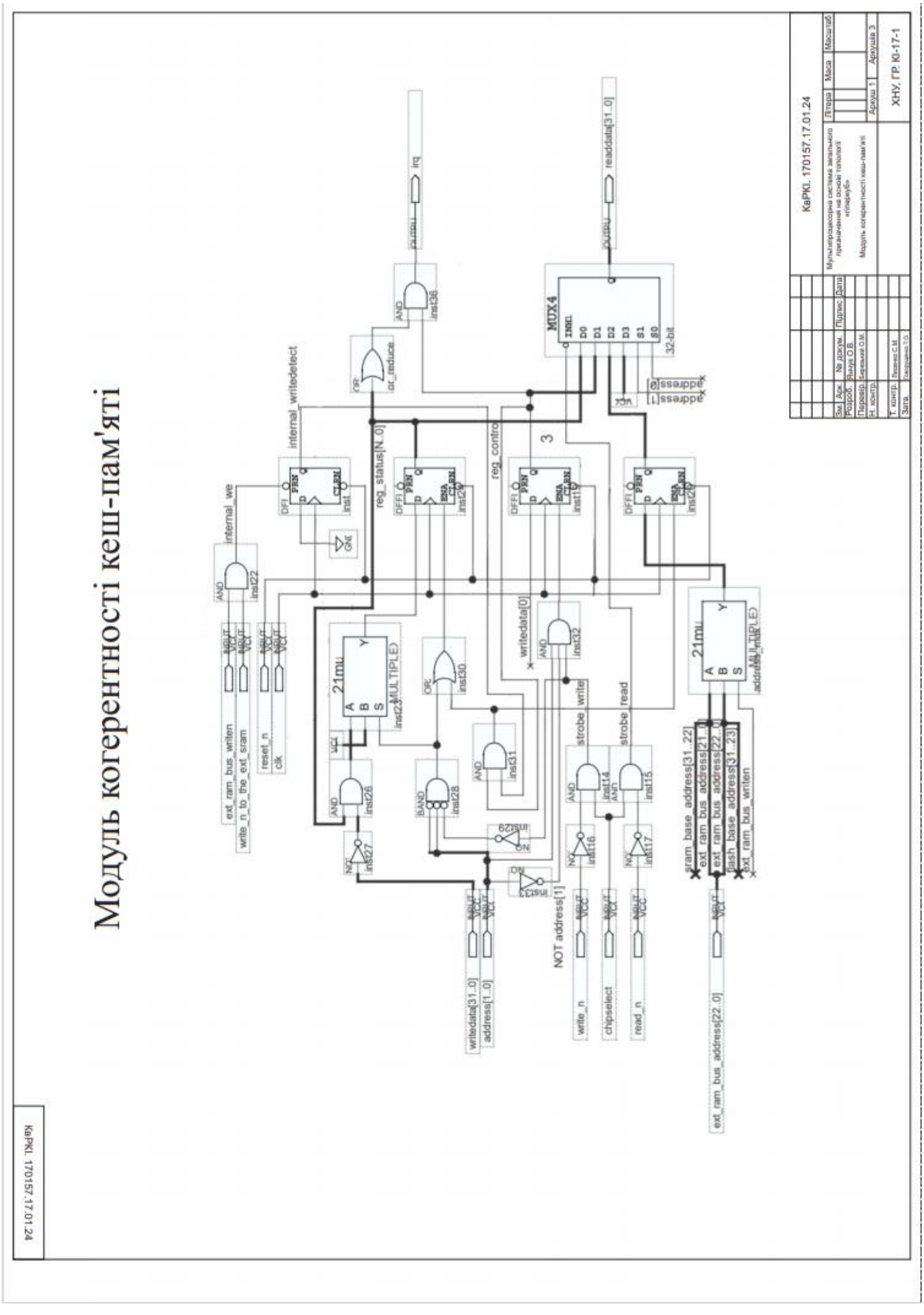
Додаток Е

Копія креслення «Процесорний блок»



Додаток Ж

Копія креслення « когерентності кеш-пам’яті»



UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1008133519

Дата перевірки:
02.06.2021 08:16:24 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.06.2021 08:23:15 EEST

ID користувача:
100005591

Назва документа: Янчук_Мультипроцесорна система загального призначення на основі топології «гіперкуб»

Кількість сторінок: 63 Кількість слів: 13477 Кількість символів: 103164 Розмір файлу: 1.66 MB ID файлу: 100821469

6.35%
Схожість

Найбільша схожість: 5.06% з Інтернет-джерелом (https://studopedia.net/1_48103_harakteristika-obiektiv-obslugovuvann

5.97% Джерела з Інтернету | 35 ..... Сторінка 65

0.6% Джерела з Бібліотеки | 60 ..... Сторінка 65

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 0.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 13%**

ID: 91875 Название: Мультипроцесорна система загального призначення на основі топології «гіперкуб» Добавлено в БД: 2021-06-02 Авторы: Янчук О.В. Руководители: Березький О.М. Консультанты: Оппоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	83333	703	368 (0%)	7 (1%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник Янчук Олександра Віталіївна
Тема Мультипроцесорна система загального призначення на основі топології «гіперкуб»
Спеціальність 123 Комп'ютерна інженерія

Обсяг кваліфікаційної роботи:

кількість листів креслень 3 ; кількість сторінок записки 80

1. Короткий зміст КП та прийнятих рішень В рамках кваліфікаційної роботи було розроблено мультипроцесорну систему на основі топології «гіперкуб». За базу взято SMP-архітектуру, для покращення роботи системи було додано процесорний кеш та розроблено модуль когерентності кеш-пам'яті для належного її функціонування.

2. Висновок про відповідність КП дипломному завданню Кваліфікаційна робота у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому, теоретичному, розділі кваліфікаційної роботи якісно та в повній мірі розглянуті методи вирішення поставленої задачі, був проаналізований кожен аспект, який стосується теми кваліфікаційної роботи. У наступному розділі було проведено порівняльну характеристику модулів, що можуть використовуватись в системі. Рішення про використовувані модулі обґрунтовано. Також були розглянуті переваги та недоліки топології «гіперкуб» У основній проектній частині роботи була реалізована сучасними методами та рішеннями мультипроцесорна система загального призначення на основі топології «гіперкуб». За проведеним у попередніх розділах аналізом, систему було покращено шляхом додавання кеш-пам'яті та створення модуля когерентності кеш-пам'яті. Було проведено тестування системи у двох варіантах – класичного вигляду, та покращеного. В загальному усі розділи відповідають завданню та містять сучасні методи вирішення поставлених завдань.

4. Позитивні сторони проекту Кваліфікаційна робота відповідає сучасним вимогам до проектування мультипроцесорних систем. Для проектування системи були вукористані сучасні програмно-апаратні рішення. Окремо можна виділити підняття питання про «вузьке місце» таких систем, та спроектовано і протестовано рішення для усунення такого недоліку.

5. Негативні сторони проекту Надмірна деталізація в плані питання вибору процесора. Добре було б детальніше розглянути питання масштабованості та організації системи. Вказаний недолік не зменшує позитивне враження від роботи.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконане відповідно до суті кваліфікаційної роботи. У першому листі відображено загальну схему мультипроцесорної системи загального призначення на основі, обраної, SMP-архітектури. У другому листі детально показано об'єднання процесорів та їх кешу один між одним та у топологію «гіперкуб». Третій лист представляє собою схему когерентності кеш-пам'яті. В загальному графічне оформлення виконане на належному рівні. Пояснювальна записка відповідає задекларованим нормам для її оформлення.

7. Відгук про роботу в цілому В загальному кваліфікаційна робота заслуговує схвальних відгуків. Весь матеріал роботи структурований, чіткий та послідовний. Усі розділи кваліфікаційної роботи йдуть у вірній послідовності, що дозволяє чітко розуміти викладений матеріал в рамках даної роботи. Графічний матеріал дозволяє наочно побачити принцип побудови, та методи покращення мультипроцесорної системи на основі топології «гіперкуб».

8. Інші зауваження

9. Оцінка кваліфікаційної роботи На основі результатів розгляду позитивних та негативних сторін представленого дипломного проекту, можна зробити висновок, що він заслуговує оцінку «відмінно»/5,0/A.

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи) доцент кафедри автоматизації, комп'ютерно-інтегрованих технологій та телекомунікацій, к.т.н. Федула Микола Васильович

« _____ » _____ 2021 р. _____ (підпис)

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

Янчук Олександра Віталіївна
ПІБ здобувача вищої освіти

ФПКТС, 4 курсу, групи КІ-17-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 29.09.2020 (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.

04. 06. 2021
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА СИСТЕМНОГО ПРОГРАМУВАННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

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

Назва: Мультипроцесорна система загального призначення на основі топології «гіперкуб»

Автор: Янчук Олександра Віталіївна

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Березький О.М., д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

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

Підтвердження:

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

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з більш ніж 10 джерелами на один фрагмент речення;
4) серед запозичень знаходяться загальновідомі терміни, скорочення та визначення.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 6.33 і адресується до 35 першоджерела, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О.М. Березький

Гарант ОП _____ С.М. Лисенко

Завідувач кафедри КІСП _____ Т. О. Говорущенко