

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій

(повне найменування факультету)

Комп'ютерні системи та мережі

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ПОСЛІДОВНИЙ ОБЧИСЛЮВАЧ НА ПЛІС З

(назва теми)

АПАРАТУРНОЮ ОПТИМІЗАЦІЄЮ

Виконав(ла): студент(ка) 2 курсу, групи КНТ-613м

Спеціальності 123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Спеціалізовані комп'ютерні системи

ЧАРОВАТИЙ П. Г

(ПРИЗВИЩЕ та ініціали)

Керівник ЗЕЛЕНЬОВА І. Я

(ПРИЗВИЩЕ та ініціали)

Рецензент СТЕПАНЕНКО О. О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра Комп'ютерні системи та мережі
Ступінь вищої освіти магістерський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Спеціалізовані комп'ютерні системи
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри _____

«_____» _____ 20_24 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЧАРОВАТОГО Павла Геннадійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Послідовний обчислювач на ПЛІС із апаратною оптимізацією

керівник проєкту (роботи) к. т. н., доцент, ЗЕЛЕНЬОВА Ірина Яковлівна
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 18 » жовтня 2024 року №149

2. Строк подання студентом проєкту (роботи) 20 січня 2025 року

3. Вихідні дані до проєкту (роботи) література за темами синтезу операційних пристроїв, програмні пакети для проєктування на мікросхемах Altera, Xilinx.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз структур операційних пристроїв процесора;
2) Розробка операційного автомата з узагальненими мікроопераціями;
3) Дослідження схем цифрового М-автомата на різних мікросхемах.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ЗЕЛЕНЬОВА І. Я., доцент		
нормоконтроль	ПОЛЬСЬКА О.В., ст. викл.		

7. Дата видачі завдання 01.10.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз архітектур і структур процесорів	10.10.2024 р.	
2	Проектування операційного автомата	17.10.2024 р.	
3	Розробка керуючого автомата	25.10.2024 р.	
4	Розробка схеми формування мікрооперацій	05.11.2024 р.	
5	Реалізація цифрової системи у середовищі VHDL	10.11.2024 р.	
6	Тестування і дослідження системи	18.11.2024 р.	
7	Оформлення звіту та графічного матеріалу	27.11.2024 р.	

Студент(ка)

(підпис)

Павло ЧАРОВАТИЙ

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Ірина ЗЕЛЕНЬОВА

(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 62 с., 20 рис., 4 табл., 22 джерел.

ОПЕРАЦІЙНИЙ АВТОМАТ, АВТОМАТ МУРА, МІКРОСХЕМА,
QUARTUS, АРХІТЕКТУРА, VHDL, FPGA

Об'єкт дослідження – методи синтезу операційних автоматів.

Предмет дослідження – архітектура операційного автомата типу М (mutual) з узагальненими мікроопераціями, методи його проектування, верифікації та тестування на сучасних мікросхемах.

Мета роботи – розробка схеми спеціалізованого обчислювача із мінімальними апаратними витратами.

У першому розділі здійснено огляд сучасних архітектур процесорів, проаналізовано основні компоненти операційних блоків процесорів, такі як регістри, лічильники і шини даних і окрему увагу приділено огляду мікросхем від компанії Altera, їх особливостям та областям застосування в сучасних цифрових системах.

У другому розділі виконано розробку операційного автомата типу М, цей тип автомата вирізняється узагальненою комбінаційною логікою. Описано етапи формування пам'яті автомата, розподіл регістрів між шинами, синтез таблиці операторів та логічних умов для забезпечення коректної роботи автомата.

Третій розділ присвячений реалізації операційного автомата типу М та тестуванню коректності його роботи у середовищі розробки Active-HDL. Також проводиться дослідження роботи цифрового пристрою на ПЛІС з використанням середовища Quartus. Було проведено верифікацію логічної схеми та перевірку сумісності схеми з різними мікросхемами.

Розроблено операційний автомат типу М з узагальненою комбінаційною частиною, що дозволяє оптимізувати апаратні витрати.

ABSTRACT

Explanatory note to the master's work: 62 p., 20 figures, 4 tables, 22 sources.

OPERATION UNIT, MOORE MACHINE, MICROCHIP, QUARTUS, ARCHITECTURE, VHDL, FPGA

The object of research – methods for synthesizing operational units.

The subject of research – the architecture of an M-type (mutual) operation unit with generalized micro-operations, methods for its design, verification, and testing on modern microchips.

Aim of the work – development of a specialized computing scheme with minimal hardware costs.

In the first chapter, a review of modern processor architectures is conducted, analyzing the main components of processor operational units, such as registers, counters, and data buses, with particular attention given to microchips from Altera, their features, and areas of application in modern digital systems.

The second chapter focuses on the development of an M-type operation unit, distinguished by generalized combinational logic. The stages of memory formation for the machine, the distribution of registers across buses, the synthesis of operator tables and logical conditions for ensuring the correct operation of the machine are described.

The third chapter is dedicated to the implementation of the M-type operation unit and testing its correct functionality in the Active-HDL development environment. Additionally, the operation of the digital device on FPGA is explored using the Quartus environment. The logical circuit verification and compatibility testing of the design with various microchips were performed.

An M-type operation unit with a generalized combinational logic part was developed, allowing for the optimization of hardware costs.

ЗМІСТ

Скорочення та умовні позначки	8
Вступ.....	9
1 Аналіз структур операційних пристроїв процесора	10
1.1 Огляд сучасних архітектур процесорів.....	10
1.2 Основні структури операційного блока процесора	12
1.3 Аналіз базису ПЛІС	13
1.4 Огляд сучасних мікросхем Altera	17
1.5 Огляд середовища розробки Quartus II	21
1.6 Постановка задачі розробки	23
2 Розробка операційного автомата з узагальненими мікроопераціями.....	24
2.1 Структура операційного автомата типу М	24
2.2 Формування пам'яті автомата та розподіл регістрів між шинами	25
2.3 Формування таблиці операторів автомата.....	29
2.4 Формування логічних умов	31
2.5 Синтез функціональної схеми автомата та формування перетвореної мікропрограми	31
2.6 Визначення основних характеристик автомата типу М.....	35
2.7 Синтез керуючого автомата по граф-схемі алгоритму	37
2.8 Висновок за розділом 2.....	45
3 Дослідження схем цифрового м-автомата на різних мікросхемах	46
3.1 Реалізація схем М-автомата на VHDL	46
3.2 Тестування і перевірка логічних схем.....	48
3.3 Дослідження спецобчислювача на мікросхемах Altera\Intel	52
3.4 Порівняння апаратних витрат та практичні рекомендації	55
3.5 Висновки за результатами дослідження	57
Висновки	59
Перелік джерел посилання	60

Додаток А.....	63
----------------	----

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

АЛП - арифметико логічний пристрій;

ГСА – граф-схема алгоритму;

КА – керуючий автомат;

МО – мікрооперація;

ОА – операційний автомат;

ПЛІС – програмована логічна інтегральна схема;

ІІІ – штучний інтелект;

ARM - advanced RISC Machine;

CISC – complex Instruction Set Computing;

CLK – clock;

CPU – central Processing Unit;

FPGA – field-programmable gate array;

FPU – floating Point unit;

FSM – finite-state machine;

GPU – graphics processing unit;

HDL – hardware description language;

RISC – reduced instruction set computing;

RISC-V - risk-five;

SoC - system-on-a-chip;

VHDL – (Very high-speed integrated circuits) Hardware Description Language

VLIW – very long instruction word.

ВСТУП

Процесор є серцем будь-якої обчислювальної системи, від найпростіших мікроконтролерів до найпотужніших суперкомп'ютерів. З розвитком цифрових технологій і зростанням вимог до продуктивності, енергоефективності та паралельної обробки даних, розуміння структури процесора стало актуальним як ніколи. Процесори визначають здатність пристрою швидко виконувати складні обчислення, працювати з великими обсягами даних і підтримувати сучасні інноваційні технології, такі як штучний інтелект, обробка відео та графіки, мережеві комунікації.

Вивчення структури процесора дозволяє краще зрозуміти, як працюють сучасні обчислювальні системи, а також оптимізувати їх використання. Оптимізація архітектури процесорів забезпечує покращення роботи операційних систем, додатків і складних обчислювальних завдань.

Крім того, з появою багатоядерних процесорів і гетерогенних обчислювальних середовищ (наприклад, комбінацій CPU та GPU), структура процесора набула ще більшої складності. Розуміння того, як організовані ядра процесора, системи кешування даних, управління пам'яттю і внутрішня комунікація між компонентами, дозволяє ефективно вирішувати задачі оптимізації продуктивності та зниження енергоспоживання.

Окрім того, вивчення структури процесорів є ключовим для розробки нових типів пристроїв, таких як системи на кристалі (SoC), що поєднують в собі процесор, графічний блок, контролери вводу/виводу та інші компоненти на одному кристалі. Такий підхід дозволяє створювати більш компактні та ефективні системи, що є критично важливим для мобільних та портативних пристроїв.

Таким чином, актуальність дослідження структури процесора полягає не тільки в оптимізації сучасних систем, але й у розвитку майбутніх технологій, які покращать продуктивність, енергоефективність та здатність обробляти величезні обсяги інформації.

1 АНАЛІЗ СТРУКТУР ОПЕРАЦІЙНИХ ПРИСТРОЇВ ПРОЦЕСОРА

1.1 Огляд сучасних архітектур процесорів

Архітектура CISC (Complex Instruction Set Computing), яка використовується в більшості настільних комп'ютерів і серверів [1]. Основною особливістю цієї архітектури є наявність складних інструкцій, які можуть виконувати декілька операцій за один цикл. Це дозволяє зменшити кількість інструкцій, необхідних для виконання програм, але робить сам процесор більш складним [2].

Основні характеристики:

- інструкції в архітектурі CISC можуть включати кілька операцій, що спрощує програмування, але вимагає більшого обсягу ресурсів для їх декодування;
- архітектура CISC має велику історію і забезпечує зворотну сумісність, що дозволяє запускати старі програми на нових системах. Це важлива перевага для підприємств, які покладаються на старе програмне забезпечення;
- сучасні процесори на архітектурі CISC підтримують віртуалізацію, багатоядерність, SIMD (Single Instruction, Multiple Data) та інші сучасні технології, що підвищують продуктивність і енергоефективність.

Архітектура RISC (Reduced Instruction Set Computing), яка широко використовується в мобільних пристроях, вбудованих системах та сучасних обчислювальних пристроях [3]. Вона фокусується на простих командах, які виконуються за один цикл, що робить її швидкою і енергоефективною [4].

Основні характеристики:

- RISC архітектура спроектована для досягнення високої продуктивності при низькому енергоспоживанні, що робить її ідеальною для мобільних пристроїв, таких як смартфони та планшети;
- ця архітектура підтримує фіксовану довжину інструкцій і обмежену кількість адресних режимів, що спрощує розробку компіляторів і знижує складність процесора;

- RISC надає ліцензії на свою архітектуру іншим виробникам, що дозволяє створювати різноманітні варіанти процесорів, адаптовані до специфічних потреб.

RISC-V є відкритою архітектурою процесорів, яка стала популярною завдяки своїй гнучкості та можливостям модульності. На відміну від комерційних архітектур, таких як CISC чи RISC, RISC-V дозволяє розробникам адаптувати інструкційний набір для своїх специфічних потреб. Це означає, що користувачі можуть додавати або видаляти інструкції, а також налаштовувати їх, щоб оптимізувати продуктивність для різних застосувань [5].

Основні характеристики:

- RISC-V підтримує базовий набір інструкцій, а також кілька додаткових розширень, таких як інструкції для обробки векторних даних, математичні операції, а також інструкції для оптимізації енергоефективності;
- як відкрита архітектура, RISC-V не має ліцензійних зборів, що робить її привабливою для розробників, стартапів та академічних установ;
- вона знайшла застосування в різних сферах, включаючи вбудовані системи, суперкомп'ютери, мобільні пристрої та штучний інтелект.

Архітектура VLIW дозволяє одночасне виконання декількох інструкцій, пакуючи їх в одну довгу інструкцію [6]. Це досягається шляхом використання довгих інструкційних слів, що включають кілька операцій. Оскільки планування виконання операцій покладається на компілятор, VLIW може бути дуже ефективною, якщо програмне забезпечення розроблено так, щоб максимально використовувати можливості архітектури [7].

Основні характеристики:

- VLIW розроблена для досягнення високого рівня паралелізму в обробці даних, що дозволяє використовувати всі ресурси процесора без потреби в складному управлінні виконанням;
- через необхідність точного планування інструкцій на етапі компіляції, VLIW вимагає ефективних компіляторів, які можуть оптимізувати код для досягнення максимальної продуктивності;

- VLIW часто використовуються в специфічних сферах, таких як цифрова обробка сигналів (DSP), де можуть забезпечити значну перевагу в продуктивності.

1.2 Основні структури операційного блока процесора

Арифметико-логічний пристрій (АЛП) виконує широкий спектр арифметичних (додавання, віднімання, множення, ділення) та логічних (І, АБО, НЕ, XOR) операцій над двійковими числами. Його внутрішня структура складається з логічних елементів (І, АБО, НЕ), суматорів, мультиплексорів та інших спеціалізованих блоків.

Типи АЛП:

- цілочисельний АЛП, який працює з цілими числами;
- АЛП з плаваючою точкою (FPU), який виконує операції над числами з плаваючою точкою (дійсними числами);
- векторний АЛП, що спеціалізований для обробки великих масивів даних.

Регістри – це швидкодіюча пам'ять, яка зберігає дані, що безпосередньо беруть участь в обчисленнях. Вони складаються з тригерів, які можуть зберігати один біт інформації.

Типи регістрів:

- загального призначення, які можуть використовуватися для зберігання різних типів даних;
- спеціального призначення, що мають фіксоване призначення (наприклад, лічильник команд, регістр стану).

Мультиплексори вибирають один із декількох інформаційних входів і передають його на вихід, в залежності від значення адресних входів.

Використовуються для вибору операндів, які будуть оброблятися АЛП, або для вибору адреси пам'яті.

Дешифратори перетворюють двійковий код команди в набір керуючих сигналів для інших блоків процесора. Використовуються для визначення типу операції, що виконується, та для вибору необхідних операндів.

Блок керування координує роботу всіх блоків процесора, генерує керуючі сигнали. Він складається з лічильника команд, дешифратора команд та логічних схем для формування керуючих сигналів.

Шина даних з'єднує всі компоненти процесора, забезпечуючи передачу даних між ними. Її структура це сукупність ліній для передачі бітів.

1.3 Аналіз базису ПЛІС

Програмовані логічні інтегральні схеми (ПЛІС) або FPGA - це тип інтегральної схеми, що дозволяє гнучко змінювати апаратну конфігурацію після виготовлення. Це дозволяє використовувати FPGA у різноманітних сферах. Внутрішня структура ПЛІС наведена на рисунку 1.1.

Основні компоненти ПЛІС:

- логічні блоки (CLB) – фундаментальна одиниця ПЛІС, що складається з тригерів і таблиць пошуку (LUT), для реалізації, як комбінаційної так і послідовної логіки;
- тригери – елементи послідовної логіки, які дозволяють зберігати дані та маніпулювати ними;
- LUT (таблиці пошуку) – використовуються для реалізації комбінаційної логіки;
- блоки введення/виведення (I/O) – забезпечують інтерфейс між зовнішнім світом і самою ПЛІС;

- інтерконекти - маршрути для з'єднання різних логічних блоків. Дозволяють мінімізувати час затримки між компонентами;
- блоки цифрової обробки сигналів (DSP) – оптимізовані для виконання операцій, таких як множення і фільтрація, що використовуються в високопродуктивних додатках [8].

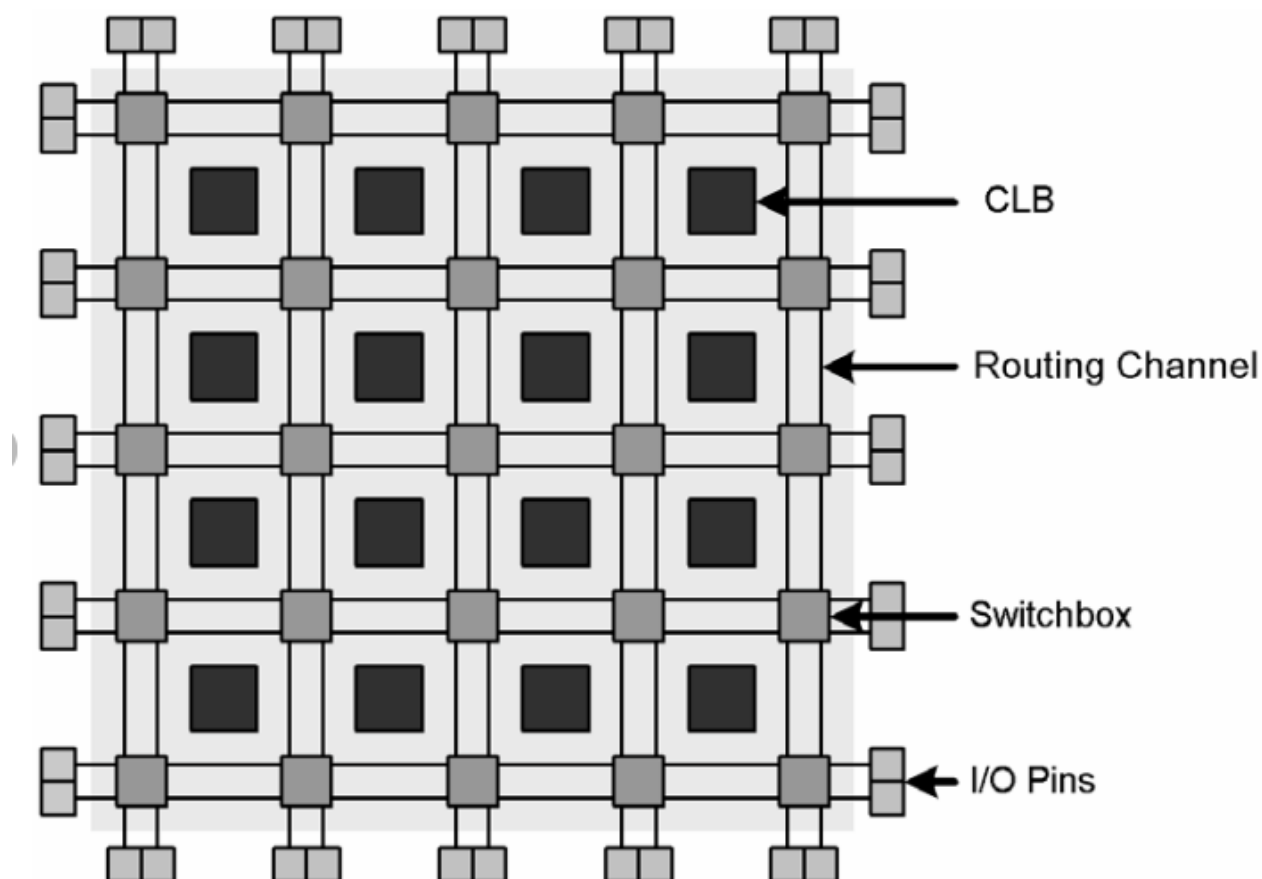


Рисунок 1.1 – Внутрішня структура ПЛІС [9]

Виробники ПЛІС мають можливість інтегрувати власні цифрові схеми та налаштовувати компоненти відповідно до вимог користувача. Це дозволяє створювати рішення, адаптуючись до конкретних специфікацій застосувань.

Відомі постачальники ПЛІС на ринку:

- Xilinx - один з двох лідерів ринку, який пропонує різноманітні серії ПЛІС;
- Altera – це другий лідер ринку, який зосередив увагу на високопродуктивності своїх ПЛІС;

- Lattice Semiconductor – компанія, що фокусується на низькому енергоспоживанні і компактності при розробці ПЛІС.

1.3.1 Різниця між FPGA, ASIC та CPLD

FPGA, CPLD та ASIC є ключовими компонентами сучасних обчислювальних систем, і їх порівнюють між собою в багатьох галузях. Тому, як для новачків так і для досвідчених інженерів і фахівців необхідно знати відмінності між цими типами схем і намагатися вибрати найефективніший варіант для конкретної задачі [8].

На рисунку 1.2, який наведений нижче подано порівняння між FPGA, ASIC та CPLD за їх основними характеристиками, щоб допомогти зрозуміти, який типаж схеми підходить для ваших потреб [8].

Параметр	Filed-Programmable Gate Array	Application-Specific Integrated Circuit	Complex Programmable Logic Device
Найкраще застосування	Прототипування, адаптивні системи	Спеціалізовані, високопродуктивні рішення	Простіша логіка і системи з низькими вимогами
Кастомізація	Високі можливості налаштування для широкого діапазону застосувань	Висока ступінь налаштування для конкретного застосування	Обмежена кастомізація у порівнянні з FPGA
Переконфігурація	Можливість реконфігурації після виготовлення	Не можна змінювати конфігурацію після виготовлення	Може пропонувати обмежену часткову реконфігурацію
Продуктивність	Змінна, залежить від конфігурації	Оптимізована для конкретного завдання	Може бути швидшою за FPGA
Ціна	Низька ціна порівняно з ASIC	Найдорожча через спеціалізацію	Дешевша за FPGA та ASIC
Час розробки	Давший через необхідність налаштування	вимагає найдовшого часу розробки	швидший час розробки порівняно з FPGA
Енергоспоживання	високе енергоспоживання через накладні витрати на конфігурацію	Найнижче серед усіх типів схем оскільки дизайн оптимізований під конкретні завдання	зазвичай споживає менше енергії, ніж FPGA

Рисунок 1.2 – Різниця між FPGA, ASIC та CPLD

Вибір однієї з цих інтегральних схем залежить від специфіки розроблюваного проекту. FPGA забезпечують велику гнучкість і можуть бути перконфігуровані для багатьох програм, а ASIC використовуються в конкретному застосуванні і виконують строго обмежені функції. CPLD є менш складними і підходять для простих завдань, де важливий час розробки і невисока вартість.

1.3.2 Обґрунтування вибору елементної бази

Вибір між FPGA, CPLD, та ASIC залежить від вимог до пристрою, від його вартості, продуктивності та гнучкості в налаштуванні. В моєму проекті використання FPGA обґрунтовано гнучкістю цих мікросхем, високою продуктивністю і підтримкою складних логічних схем. Але в той же час важливо розглянути переваги й обмеження кожного типу елементної бази [10].

FPGA мають конфігураційну пам'ять на базі енергонезалежних технологій, як SRAM, що означає, що конфігурацію потрібно завантажувати кожен раз після ввімкнення пристрою [10]. Вони ідеально підходять для розробки і тестування нових проектів, оскільки дозволяють швидко вносити зміни в роботу пристрою. Проте ПЛІС споживають більше енергії, ніж CPLD, через їх більші розміри та більшу кількість логічних ресурсів [10]. Цей тип мікросхем рекомендовано використовувати для середніх і великих проектів, що вимагають високої продуктивності.

CPLD в той же час мають конфігураційну пам'ять на основі енергонезалежних технологій, таких, як FLASH або EEPROM, що дозволяє зберігати конфігурацію навіть при вимкненому живленні [10]. Ці мікросхеми споживають менше енергії, особливо в режимі очікування, і підходять для невеликих пристроїв, які вимагають енергонезалежності. CPLD зазвичай мають фіксовану кількість логічних блоків і обмежені ресурси, що робить їх придатними для невеликих проектів, які не потребують частих змін в конфігурації.

ASIC створюються для виконання конкретних завдань і забезпечують неймовірну продуктивність за рахунок оптимізації архітектури [11]. Ці типи мікросхем забезпечують найкращу мініатюризацію і енергоефективність для серійного виробництва. Але розробка ASIC надзвичайно дорога – її вартість може

коливатися від ста до п'ятисот тисяч доларів, а процес створення може тривати місяці чи навіть роки [12]. Тому саме ASIC мікросхеми доцільно використовувати для масових проектів із повністю відлагодженим дизайном. Виробництвом ASIC займаються такі компанії, як Intel, Broadcom, Texas Instruments, Infineon Technologies [13]. Основні сфери застосування включають телекомунікації, криптомайнінг, автомобільні системи, побутову електроніку [11].

Таким чином у розробці гнучких рішень ідеальним варіантом є FPGA, для простих і енергонезалежних і бюджетних проектів – CPLD [10]. Для серійного виробництва оптимальним є ASIC, за умови того, що проект підлагоджений і готовий до виробництва.

1.4 Огляд сучасних мікросхем Altera

Мікросхеми компанії Altera (Intel) представлені в різних цінових сегментах, що дозволяє обирати оптимальні і ефективні рішення для конкретних поставлених завдань. На рисунку 1.3 зображені розподіл мікросхем за ціновим сегментом.

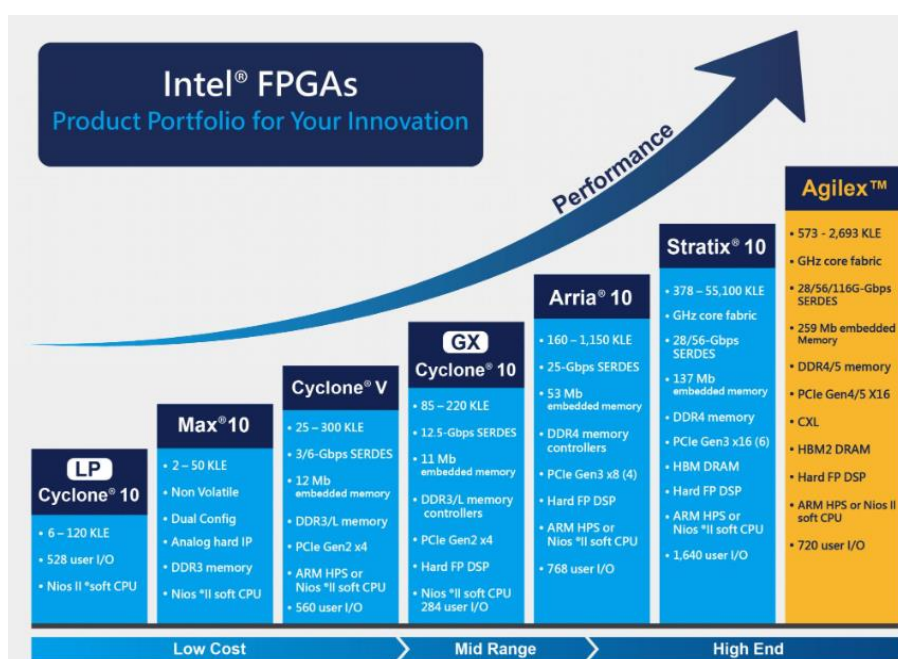


Рисунок 1.3 – Розподілення мікросхем за ціновим сегментом [14]

1.4.1 Мікросхеми сімейства Arria

Мікросхема Arria V від компанії Altera розроблена для середнього класу задач, де важливий баланс між продуктивністю і енергоспоживанням. Використання 28-нм технології виготовлення забезпечує високу щільність логічних елементів, покращене управління енергоспоживанням і високу продуктивність. Вбудовані трансивери підтримують швидкість передачі 10 Гбіт/с, що важливо для сьогоденних високошвидкісних інтерфейсів, як PCI Express. Деякі моделі містять двоядерний ARM Cortex-A9 для інтеграції процесорних обчислень та логічної обробки на одній мікросхемі. Підтримка спеціалізованих блоків DSP для швидкої обробки складних математичних операцій з фіксованою крапкою.

Сімейство FPGA Arria 10 встановлює нову оптимальну продуктивність і енергоефективність у середньому класі. Виготовлені за 20-нм технологією, вони досягають вищої щільності компонентів та нижчого енергоспоживання, ніж високоякісні FPGA попереднього покоління. Підвищення швидкодії та зниження енергоспоживання на 40% у порівнянні з попередніми поколіннями. Високошвидкісні трансивери підтримують швидкість до 28,3 Гбіт/с. Також це сімейство підтримує понад 1500 GFLOP операцій з плаваючою комою завдяки апаратно реалізованим блокам цифрової обробки сигналів. Ця інформація надає чітке розуміння основних переваг нових мікросхем Altera, що включає їхню енергоефективність та високу обчислювальну можливість.

Сімейство FPGA Arria 10:

- забезпечує повну сумісність із попередніми поколіннями Altera Soc FPGA, що дозволяє знизити час на перехід між поколіннями;
- широка екосистема ARM для розробки програмного забезпечення включаючи бібліотеки, SDK, інструменти для програмування;
- пропускна здатність пам'яті до 2666 Мбіт/с;
- підтримка компілятора OpenCL дозволяє створювати високопродуктивні програми для ПЛІС;

- у 4 рази більша пропускна здатність трансивера порівняно з попереднім поколінням, що забезпечує значно більшу швидкість обміну даними, що важливо для мережевих і комунікаційних застосувань.

1.4.2 Мікросхеми сімейства Cyclone

Модельний ряд Altera Cyclone являє собою сімейство пристроїв, оптимально поєднуючи функціональність і доступність за ціною. Вони застосовуються в різних конструкціях і додатках, де важливі параметри економічного споживання енергії в поєднанні з малою собівартістю проекту. Саме ці ПЛІС підходять для застосувань у системах аудіо та відеообробки, вбудованих системах і промислових контролерах.

Сімейство Cyclone FPGA включає кілька поколінь і кожне з них має свої особливості.

Cyclone III виготовлені за 65-нм технологічним процесом, ці ПЛІС призначені для портативних пристроїв, що працюють з великими обсягами даних і потребують низького енергоспоживання. Вони забезпечують гнучкість та оптимізовані характеристики для різних застосувань, починаючи від бездротових комунікацій і до відеообробки.

Cyclone IV представлено двома варіантами:

- Cyclone IV GX, що містить вбудований прийомопередавачі зі швидкістю до 3,125 Гбіт/с. що дозволяє значно збільшити пропускну здатність без збільшення витрат;

- Cyclone IV E під сімейство з низьковольтним ядром для зниження енергоспоживання, що на 25% менше порівняно з попередніми поколіннями. Ці ПЛІС ідеально підходять для застосувань у бездротових та провідних комунікаціях.

Забезпечуючи низьку собівартість, не втрачаючи при цьому продуктивності, та ще й маючи високошвидкісні прийомопередачтики, сімейство CycloneIV є ідеальним для застосування в виробках невеликого розміру, таких як бездротова передача, широкомовна радіопередача, індустриальні і користувацькі рішення [15].

Сімейство Cyclone V вирізняється, як одне з найекномічніших рішень серед ПЛІС, виготовлених за сучасною 28-нм технологією. Завдяки поєднанню високої продуктивності, низького енергоспоживання та доступної вартості, ці мікросхеми

стали ідеальним вибором для промислових додатків, телекомунікаційних пристроїв, військових систем [16].

Для захисту інтелектуальної власності мікросхеми Cyclone V підтримують шифрування за стандартом AES-256 з енергонезалежними та енергозалежними ключами [16].

Завдяки інтеграції двоядерного процесора ARM Cortex-A9 з логікою ПЛІС на одному кристалі, Cyclone V здатний забезпечити високу енергоефективність та продуктивність для різноманітних вбудованих систем.

1.4.3 Мікросхеми сімейства MAX

Мікросхеми MAX 10 від компанії Intel поєднують у собі високу інтеграцію, низьке енергоспоживання і доступну вартість. Вони ідеально підходять для застосувань, де важливі компактність, економічність та швидка конфігурація [17].

Основні характеристики MAX 10:

- Від 2000 до 50000 логічних елементів, що дозволяє реалізувати різноманітні цифрові функції;
- до двох конфігураційних образів, що забезпечує швидку і надійну конфігурацію пристрою;
- вбудовані аналого-цифрові перетворювачі з роздільною здатністю 12 біт, що дозволяє мікросхемі здійснювати обробку аналогових сигналів;
- дана мікросхема має вбудовані блоки статичної пам'яті та блоки цифрової обробки сигналів, що підвищує ефективність обробки даних;
- можливість реалізації синтезованого процесора Nios II для виконання програмованих завдань;
- до 500 користувацьких виводів, що забезпечує гнучкість у підключенні до інших компонентів системи;
- також для забезпечення стабільного живлення різних частин системи, дана мікросхема має вбудований перетворювач напруги.

Переваги MAX 10:

- оптимізація для застосувань з обмеженими ресурсам енергії, що важливо для портативних та автономних пристроїв;

- дуже швидке завантаження конфігурації з вбудованої флеш-пам'яті, що забезпечує швидкий старт системи;
- висока інтеграція різних функцій в одному чипі зменшує кількість компонентів.

Мікросхема MAX 10 від фірми Altera зображена на рисунку 1.4.

Завдяки описаним вище характеристикам мікросхема MAX 10 є відмінним вибором для різних галузей, включаючи вбудовані системи і застосування, які потребують ефективного управління енергоспоживанням.

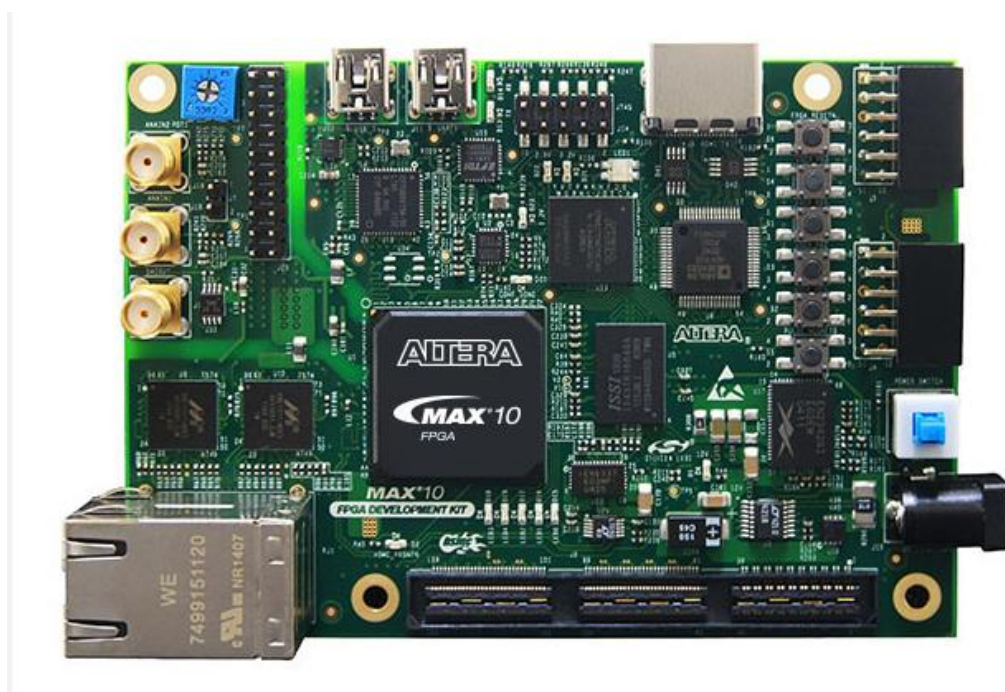


Рисунок 1.4 – Приклад мікросхеми MAX 10 [18]

1.5 Огляд середовища розробки Quartus II

Quartus II – це потужне середовище автоматизованого проектування розроблене компанією Altera, яка забезпечує комплексний набір інструментів для проектування та програмування пристроїв ПЛІС. Це програмне забезпечення

об'єднує в собі функції компіляції, моделювання та синтезу цифрових схем, що дозволяє розробникам ефективно створювати складні цифрові системи.

Основні можливості та особливості Quartus II:

- Quartus II підтримує популярні мови апаратного опису апаратури, як Verilog, VHDL, AHDL, що дає змогу розробникам вибрати найбільш зручний для них підхід до опису апаратурної логіки;
- програмне забезпечення містить велику бібліотеку попередньо розроблених цифрових блоків (IP-блоків), які можна використовувати для прискорення процесу проектування та зменшення часу розробки;
- Quartus II також надає інструменти для аналізу та синтезу HDL-коду, що дозволяє перевіряти правильність опису та оптимізувати його для цільовго пристрою;
- програмне забезпечення включає функції моделювання та верифікації проектів. Що дає змогу користувачу тестувати дизайн перед його реалізацією на фізичному пристрої;
- також це програмне забезпечення підтримує різні серії FPGA від Altera, включаючи Cyclone, Stratix, Arria та MAX, що забезпечує опціональність у виборі відповідного пристрою для конкретного проекту;
- Quartus II надає засоби для підключення сигналів проекту до виводів FPGA, які вільні від основних функцій проекту і призначені для контролю функціонування пристроїв в системі. До таких засобів відносяться налагодження виводів і інтерфейс для підключення зовнішнього логічного аналізатора [19];
- Quartus II підтримує блокову методологію проектування, яка дозволяє організувати дизайн у вигляді блоків, що спрощує управління складними проектами та підвищує ефективність розробки [20];
- Quartus II доступний для операційних систем Windows і Linux, що забезпечує гнучкість у виборі платформ для розробки [20].

Підсумовуючи, можна сказати, що середовище Quartus II – це потужний програмний пакет, призначений для проектування схем ПЛІС і який надає розробникам великий набір інструментів для спрощення процесу розробки [20].

1.6 Постановка задачі розробки

На основі проведеного огляду сформулюємо задачу розробки: розробити проєкт операційного автомата та керуючого автомата цифрової системи, а також виконати тестування логічної схеми на різних мікросхемах фірм-виробників.

На етапі розробки проєкту операційного автомата та керуючого автомата розглядається їх побудова з урахуванням таких аспектів:

Структура керуючого автомата — визначається основна архітектура керуючого автомата, який генерує мікрокоманди для подальшої обробки в системі. Описується, як керуючий автомат організує послідовність виконання операцій і керує іншими компонентами, які виконують конкретні операції.

Схема формування мікрооперацій — описується, як мікрокоманди, що генеруються керуючим автоматом, перетворюються на мікрооперації. Це включає розподіл мікрокоманд на сукупності операторів, необхідних для виконання певної операції, і їх подальшу передачу до операційного автомата.

Структура операційного автомата — визначається архітектура операційного автомата, який приймає оператори і виконує відповідні мікрооперації. Описується, як операційний автомат організує обробку мікрооперацій на основі отриманих команд.

Синтез функціональної схеми — на основі заданих вимог синтезується функціональна схема, що реалізує генерацію мікрокоманд, процеси формування мікрооперацій і виконання мікрооперацій. Для цього використовуються інструменти проектування апаратних систем на VHDL.

На етапі тестування схеми операційного та керуючого автоматів на різних мікросхемах виконується реалізація функціональної схеми на VHDL і проводиться верифікація логіки на основі симуляції. Основне тестування проводиться на сучасних мікросхемах фірм Altera (Intel), зокрема на платформах FPGA

2 РОЗРОБКА ОПЕРАЦІЙНОГО АВТОМАТА З УЗАГАЛЬНЕНИМИ МІКРООПЕРАЦІЯМИ

2.1 Структура операційного автомата типу М

Операційний втомат типу М (mutual micro instructions, тобто із узагальненими мікрокомандами) - це одна з моделей цифрових автоматів, яка використовується для моделювання та реалізації алгоритмів.

Основне призначення М-автомата – послідовне виконання мікрокоманд алгоритмів, при цьому узагальнення комбінаційної частини призводить до скорочення апаратних витрат в схемі автомата, тобто певної економії. Послідовне виконання мікрокоманд має також і негативний наслідок – сповільнення виконання алгоритму. М-автомат використовується для:

- моделювання поведінки цифрових систем, таких як процесори, контролери тощо;
- реалізації алгоритмів, виконує різноманітні алгоритми, від простих арифметичних операцій до складних обчислень.

Автомат типу М має модульну структуру, що складається з таких основних компонентів:

- пам'ять - скупчення регістрів для зберігання даних;
- операційні елементи комбінаційної частини, які виконують арифметичні та логічні операції над даними;
- мультиплексори, що вибирають дані з різних регістрів для подачі їх на вхід комбінаційних схем і мультиплексор для вибору даних з різних операційних елементів комбінаційної частини і запису цих даних в один з регістрів пам'яті;
- шини, які сполучають між собою різні блоки автомата для передачі даних.

Особливості

розрядності

слів.

У цьому автоматі всі дані передаються і зберігаються у вигляді 8-бітних слів, де найстарший біт використовується як знаковий. Значення старшого біта визначає,

чи є число додатним (0) або від'ємним (1). Решта 7 бітів використовуються для представлення величини числа. Це дозволяє працювати з числами в діапазоні від -128 до 127. Така розрядність застосовується в компараторах і арифметичних операціях для коректної роботи з умовними переходами та математичними обчисленнями.

Автомати зі структурою М типу використовуються у тих випадках коли обчислювальному пристрою не потрібна висока швидкодія, отже ми можемо зекономити на апаратних витратах. М-автомати виконують операції лише послідовно, тобто можуть виконувати в кожному такті роботи тільки одну мікрооперацію, за рахунок чого буде достатньо лише однієї комбінаційної схеми для кожного виду операцій.

2.2 Формування пам'яті автомата та розподіл регістрів між шинами

Закон функціонування ОА задається функціональною граф-схемою алгоритму, яку часто називають функціональною мікропрограмою.

Вхідними даними для проектування є система функцій, яка показана за формулою (2.1):

$$S = \begin{cases} 9 * (A \oplus B) + \left(4A + \frac{B}{8}\right) \oplus \overline{(5B + A)}, & \text{якщо } A < \text{const} \\ \frac{A+B}{8} + \overline{(4A \text{ and } 2B)} \text{ or } (A \oplus B) & \text{в іншому випадку} \end{cases}, \quad (2.1)$$

де А, В – операнди, які в подальшому будуть ініціалізовані для виконання обчислень.

Функціональна мікропрограма М-автомата показана на рисунку 2.1.

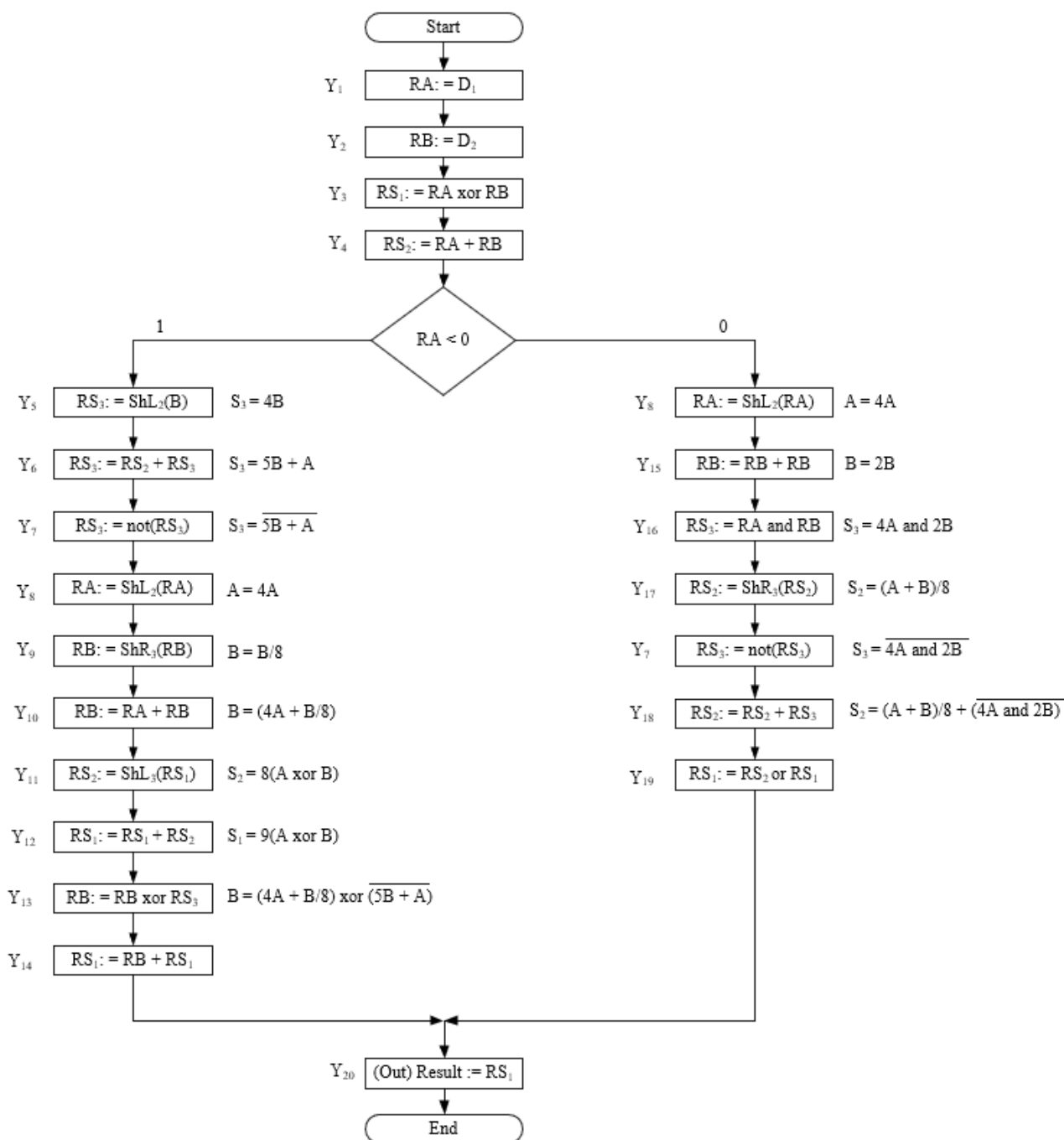


Рисунок 2.1 – Функціональна мікропрограма

В функціональній мікропрограмі існують слова трьох типів:

- вхідні слова (тип І). Ці слова беруть участь в МО ініціалізації і пов'язані з формуванням початкових значень операндів;
- вихідні слова (тип О). Ці слова відповідають результатам виконуваних обчислень;

- внутрішні слова (тип L). Значення слів цього типу змінюються в процесі виконання обчислень.

Кожне вхідне слово визначає один вхідний полюс або порт схеми, а кожне вихідне слово відповідає вихідному полюсу схеми.

Наша функціональна мікропрограма містить слова наступних типів: A (тип IL), B (тип IL), S_2 (тип L), S_3 (тип L) і S_1 (тип OL). Тобто автомат типу M має два вхідні полюси $\langle d_1 \rangle$, $\langle d_2 \rangle$. Початкові значення операндів завантажуються через ці полюси відповідно по керуючим сигналам. І також має один вихідний полюс, оскільки є тільки одне слово типу O. Усі слова, що використовуються в нашому операційному автоматі мають розрядність 8 біт.

Кожному слову типу L відповідає регістр пам'яті автомата. Число тригерів в кожному регістрі співпадає з розрядністю (числом біт) однойменного слова, тобто усі регістри автомата мають по 8 тригерів.

Для реалізації алгоритму, представленого функціональною мікропрограмою, потрібно п'ять регістрів пам'яті. Вони позначені, як RA, RB, RS_1 , RS_2 , RS_3 .

Опис регістрів:

- RA – регістр, що зберігає значення слова типу IL (вхідно-внутрішнє слово) для операнда A. Він використовується для отримання початкового значення A через вхідний порт d_1 ;

- RB – регістр, що зберігає значення слова типу IL для операнда B. Він відповідає за отримання початкового значення B через вхідний порт d_2 ;

- RS_1 – регістр, що відповідає за зберігання значення слова типу OL (вихідно-внутрішнє слово). Це результуюче слово, яке записується на вихідний полюс схеми, тобто на вихідний порт автомата;

- RS_2 – регістр для зберігання значення внутрішнього слова типу L, що змінюється під час виконання обчислень і використовується в процесі обробки даних;

- RS_3 – регістр для зберігання значення ще одного внутрішнього слова типу L, яке також бере участь у виконанні алгоритму.

Роль регістрів:

- RA і RB служать для ініціалізації операндів A і B через вхідні порти автомата, забезпечуючи початкові дані для подальших обчислень;
- RS₁ використовується для зберігання і передачі вихідного результату через вихідний порт автомата;
- RS₂ і RS₃ відповідають за зберігання проміжних результатів, які змінюються під час виконання функціональної мікропрограми.

Шини A₁ і A₂ використовуються для передачі інформації з регістрів пам'яті до операційних елементів. Ці шини у свою чергу є виходами мультиплексорів M₁ і M₂.

У автоматі типу M кожна з шин A₁ і A₂ має 8 розрядів, що збігається з розрядністю переданих слів.

Для оптимізації схем мультиплексорів необхідно розділити множину регістрів на два класи, в яких немає загальних елементів. Якщо таке розбиття неможливо через особливості реалізованого алгоритму обчислень, то необхідно знайти покриття множини регістрів, що складається з підмножин з мінімальним числом загальних елементів. Таким чином, перетин $A_1 \cap A_2$ має бути мінімальним. Тут A_i означає множину регістрів, інформація з яких передається через мультиплексор M_i.

В моєму автоматі типу M можна знайти наступний розподіл регістрів на два класи: $A_1 = \{A, B, S_2\}$, $A_2 = \{B, S_1, S_3\}$.

Регістр B повинен бути розташований в обох класах цього покриття, так як МО зсуву вліво зводиться до підсумовування і виконується згідно формули:

$$L_1(B) = B + B, \quad (2.2)$$

де B - це операнд, який представляє значення, збережене у регістрі RB;

L₁(B) – операція зсуву вліво на 1 над операндом B.

2.3 Формування таблиці операторів автомата

Таблиця операторів — це основний елемент, який визначає структуру операційного автомата і формує мікрокоманди, що виконує цей автомат. Вона відображає зв'язки між регістрами пам'яті, мультиплексорами і комбінаційною частиною. Кожен рядок таблиці відповідає одній мікрокоманді у початковій мікропрограмі і визначає, які апаратні елементи будуть задіяні під час виконання цієї МК.

Для знаходження операторів необхідно побудувати таблицю із стовпцями, що визначають зв'язки між ключовими елементами структури автомата, зокрема:

- y_n — номер або ідентифікатор мікрокоманди;
- A_1 — зв'язок між регістрами пам'яті та мультиплексором A_1 ;
- A_2 — зв'язок між регістрами пам'яті та мультиплексором A_2 ;
- Z — комбінаційна частина схеми, яка використовується для виконання мікрооперацій;
- RS — зв'язок між комбінаційною частиною (шина Z) та регістрами пам'яті.

У випадку автомата типу M , який виконує одну мікрокоманду за такт, таблиця операторів включає лише одну таблицю для всіх операцій. У ній присутні зазначені вище стовпці (y_n , A_1 , A_2 , Z , RS). Оператори, пов'язані з кожним з цих стовпців, необхідно вписати у відповідні колонки в таблиці. Таким чином, формується таблиця операторів, приклад якої наведено у таблиці 2.1.

Таблиця 2.1 – Таблиця операторів М-ОА

y_n	A_1	A_2	Z	RS
y_1	-	-	$\langle d_1 \rangle$ ϕ_1	RA c_1
y_2	-	-	$\langle d_2 \rangle$ ϕ_2	RB c_2
y_3	RA a_1	RB b_1	$A_1 \text{ xor } A_2$ ϕ_3	RS ₁ c_3
y_4	RA a_1	RB b_1	$A_1 + A_2$ ϕ_4	RS ₂ c_4
y_5	RB a_2	-	$L_2(A_1)$ ϕ_5	RS ₃ c_5
y_6	RS ₂ a_3	RS ₃ b_2	$A_1 + A_2$ ϕ_4	RS ₃ c_5
y_7	-	RS ₃ b_2	$\bar{A}_2$ ϕ_6	RS ₃ c_5
y_8	RA a_1	-	$L_2(A_1)$ ϕ_5	RA c_1
y_9	RB a_2	-	$R_3(A_1)$ ϕ_7	RB c_2
y_{10}	RA a_1	RB b_1	$A_1 + A_2$ ϕ_4	RB c_2
y_{11}	-	RS ₁ b_3	$L_3(A_2)$ ϕ_8	RS ₂ c_4
y_{12}	RS ₂ a_3	RS ₁ b_3	$A_1 + A_2$ ϕ_4	RS ₁ c_3
y_{13}	RB a_2	RS ₃ b_2	$A_1 \text{ xor } A_2$ ϕ_3	RB c_2
y_{14}	RB a_2	RS ₁ b_3	$A_1 + A_2$ ϕ_4	RS ₁ c_3
y_{15}	RB a_2	RB b_1	$A_1 + A_2$ ϕ_4	RB c_2
y_{16}	RA a_1	RB b_1	$A_1 \text{ and } A_2$ ϕ_9	RS ₃ c_5
y_{17}	RS ₂ a_3	-	$R_3(A_1)$ ϕ_7	RS ₂ c_4
y_{18}	RS ₂ a_3	RS ₃ b_2	$A_1 + A_2$ ϕ_4	RS ₂ c_4
y_{19}	RS ₂ a_3	RS ₁ b_3	$A_1 \text{ or } A_2$ ϕ_{10}	RS ₁ c_3
y_{20}	-	-	ϕ_{11}	-

Відмінність таблиць між автоматами типу М і типу М з паралельною комбінаційною частиною: автомат типу М виконує одну мікрокоманду за такт, тому має одну таблицю операторів, яка відображає всі мікрокоманди послідовно.

Автомат типу М з паралельною комбінаційною частиною може виконувати одночасно дві мікрокоманди. Для цього використовуються дві таблиці операторів: одна для операцій з одним операндом, а інша для операцій з двома операндами. У результаті структура таблиць цих автоматів відрізняється кількістю стовпців та їхнім призначенням.

2.4 Формування логічних умов

З аналізу структури ОА типу М виходить, що значення логічних умов обчислюються комбінаційною схемою компаратором. У моєму випадку логічна умова не пов'язана безпосередньо з шиною Z, а перевіряється на рівні окремих розрядів операндів. Це означає, що формули для обчислення логічної умови мають бути представлені у вигляді перевірки окремих бітів.

Наприклад, логічну умову $A < 0$ необхідно представити у вигляді $A[7] == 1$, де $A[7]$ — старший розряд операнда A, який визначає знак числа.

2.5 Синтез функціональної схеми автомата та формування перетвореної мікропрограми

Функціональна схема автомата типу М реалізується на основі таблиці операторів, яка визначає зв'язки між операційними елементами, мультиплексорами та регістрами пам'яті. Кожен рядок таблиці операторів відповідає певній мікрокоманді і визначає, які апаратні компоненти будуть використовуватися в

кожному такті роботи автомата. Основні компоненти функціональної схеми включають операційні елементи, мультиплексори та регістри пам'яті. Функціональна схема автомата М наведена на рисунку 2.2.

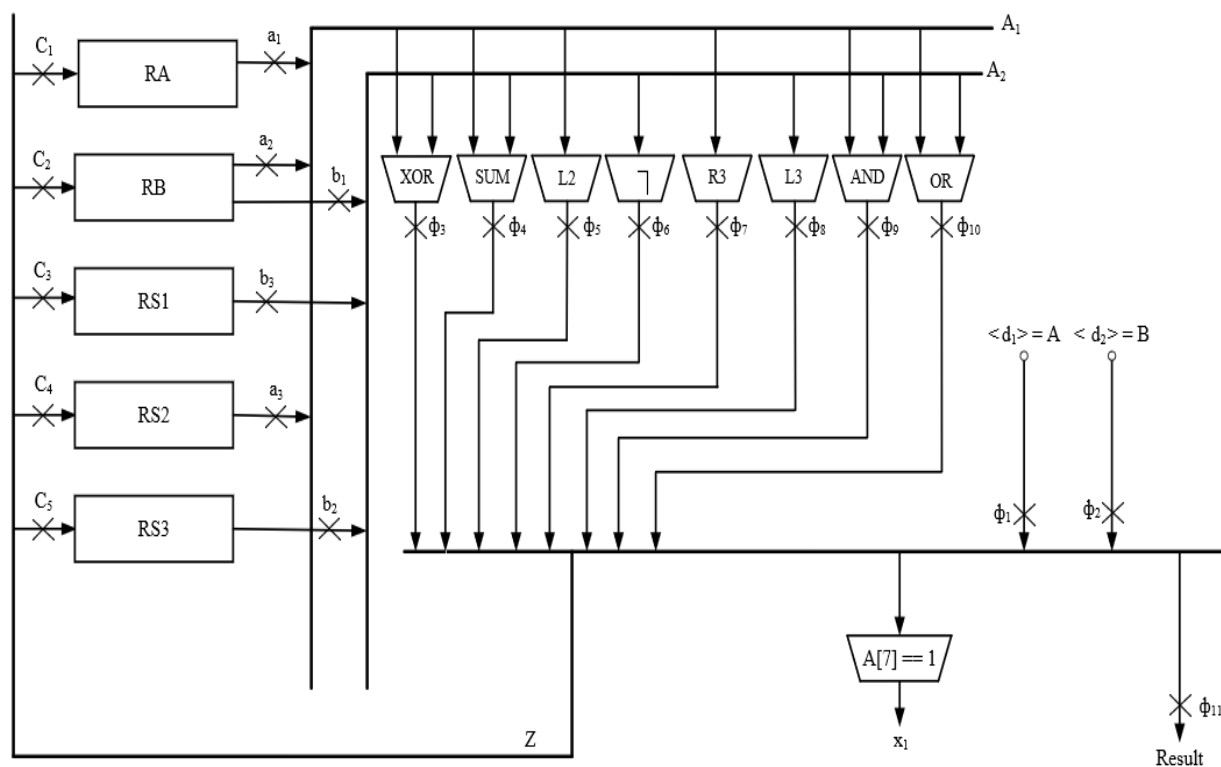


Рисунок 2.2 – Функціональна схема М-ОА

Стовпець RS визначає регістри пам'яті і задає операції передачі даних з шини Z до відповідних регістрів. Це відповідає збереженню результатів обчислень у пам'яті автомата.

Стовпці A_1 , A_2 визначають вхідні зв'язки мультиплексорів. Ці стовпці вказують, які регістри пам'яті підключаються до мультиплексорів A_1 і A_2 для передачі інформації до комбінаційної частини автомата. Мультиплексори обирають, які з регістрів пам'яті будуть використовуватися для подальших операцій, виконуючи вибір між різними джерелами даних.

Стовпець Z відповідає за операційні елементи комбінаційної частини, які виконують обчислення над переданими даними. Ці елементи можуть виконувати

арифметичні, логічні або інші операції, залежно від завдань. Оператори в стовпці Z визначають передачу результатів цих операцій на шину Z для подальшого використання або збереження.

Операційні елементи — це компоненти комбінаційної частини схеми, які виконують основні обчислення, такі як додавання, множення, логічні операції тощо. Вони отримують дані від мультиплексорів і виконують необхідні мікрооперації.

Мультиплексори (A_1, A_2) — це ключові елементи, що здійснюють вибір між кількома вхідними сигналами, визначаючи, які дані будуть використовуватися в обчисленнях. Мультиплексори приймають вхідні дані з регістрів пам'яті і передають їх до операційних елементів для виконання операцій.

Шина Z — це передавальна лінія, через яку результати операцій надсилаються назад до регістрів або до інших частин автомата. Стовпець RS таблиці операторів визначає, в які регістри будуть записані результати після виконання операцій.

Процес перетворення мікропрограми полягає у тому, щоб замінити кожен мікрооперацію початкової мікропрограми на відповідні оператори, що містяться в таблиці операторів операційного автомата типу M . Основна ідея полягає в тому, що M -ОА може виконувати тільки одну мікрокоманду в кожному такті. Тому кожна вершина початкової мікропрограми, яка може містити кілька мікрокоманд, повинна бути розділена на послідовність операторних вершин, кожна з яких містить тільки одну мікрокоманду. Перетворена мікропрограма операційного автомата типу M представлена на рисунку 2.3.

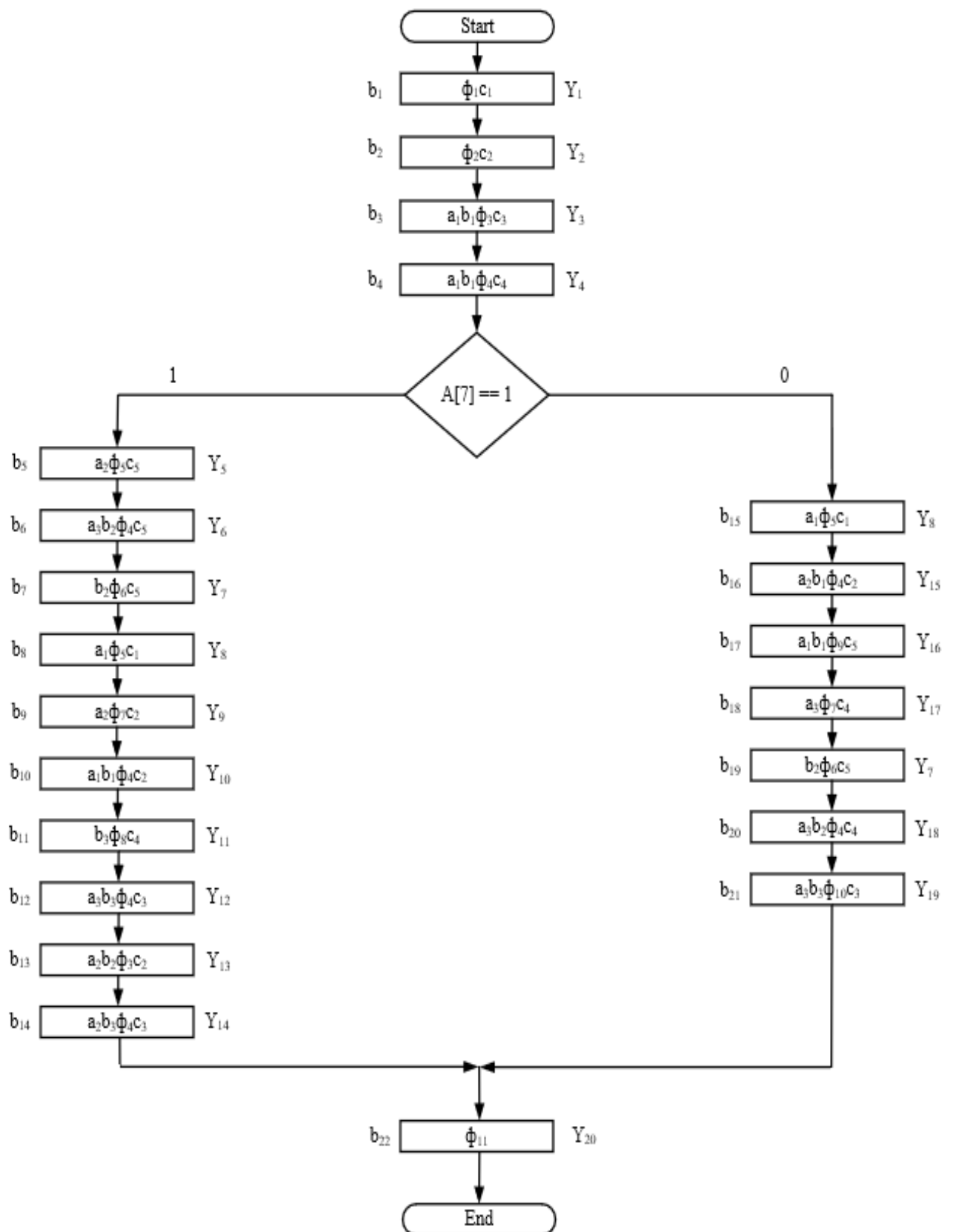


Рисунок 2.3 – Перетворена мікропрограма М-ОА

При перетворенні необхідно дотримуватись послідовності мікрокоманд. Якщо результат однієї мікрокоманди використовується, як операнд іншої, порядок їх виконання має це враховувати. У деяких випадках виникають складні ситуації, коли мікрокоманди взаємозалежні за даними, і жоден з очевидних порядків їх виконання не дозволяє уникнути помилок в обчисленні результату. Для вирішення таких конфліктів доцільно додати додаткову мікрокоманду, що призведе до введення додаткового такту або навіть регістра.

Додавання додаткових тактів збільшує загальний час виконання мікропрограми, але це необхідно для правильного обчислення і перевірки логічних умов.

2.6 Визначення основних характеристик автомата типу М

Позначимо символом $M(\Gamma)$ той факт, що деякий М-ОА синтезується по функціональній мікропрограмі Γ .

Визначення характеристик операційного автомата типу М, а саме апаратних витрат, час такту і оцінку швидкодії.

Апаратні витрати можна визначити як оцінку в деяких умовних одиницях. Прийmemo, що вартість суматора дорівнює 1, а інші операційні елементи коштують в 10 разів менше, тобто 0,1. Схема ОА типу М включає 1 суматор, 1 інвертор, 1 суматор по модулю 2, 1 логічний елемент OR, 1 логічний елемент AND, 3 зсувачі і 3 мультиплексори. Таким чином, $Q_M(\Gamma) = 2$.

Щоб визначити час такту, необхідно знати час затримки для кожного операційного елементу. Нехай час затримки для вентиля І-НЕ становить τ деяких одиниць. Час затримки суматора $t_s = 20\tau$, а для інших елементів цей час однаковий і рівний 2τ . Нехай тригери регістрів пам'яті переключаються за час $t_r = 6\tau$. У разі М-ОА час такту можемо визначити за формулою:

$$T_M(\Gamma) = \tau_{cc} + \tau_r, \quad (2.3)$$

де τ_{cc} - час затримки в комбінаційній частині ОА;

τ_r – час переключення тригерів регістрів пам'яті.

Знайти час затримки в комбінаційній частині ОА можна за формулою:

$$\tau_{cc} = \max(\tau_s, \tau_{sh}, \tau_i, \tau_{or}, \tau_{and}, \tau_{xor}) + 2\tau_M = \tau_s + 2\tau_M, \quad (2.4)$$

де τ_s - час затримки суматора;

τ_{sh} – час затримки зсувача;

τ_i – час затримки інвертора;

τ_{or} – час затримки логічної операції or;

τ_{and} – час затримки логічної операції and;

τ_{xor} – час затримки логічної операції xor;

τ_M – час затримки суматора;

Для даного прикладу можна знайти, що $T_M(\Gamma) = 24\tau + 6\tau = 30\tau$.

Для визначення швидкодії $LT_M(\Gamma)$ необхідно знайти середнє число циклів $n_M(\Gamma)$, потрібне для виконання обчислень по алгоритму Γ для М-ОА. Потім ці параметри ($n_M(\Gamma)$ и $T_M(\Gamma)$) перемножуються, даючи значення $LT_M(\Gamma)$. Параметр $n_M(\Gamma)$ можливо знайти використавши наступну процедуру:

- знайти усі шляхи $\alpha_1, \dots, \alpha_J$ між вершинами Start і End;
- знайти число операторних вершин n_j для кожного шляху α_j ;
- знайти параметр $n_M(\Gamma)$ як суму усіх n_j ;
- знайти $LT_M(\Gamma)$ як результат ділення $n_M(\Gamma)$ на J .

Отже, швидкодію ОА можна знайти по формулі:

$$LT_M(\Gamma) = \sum_{j=1}^J n_j (T_M(\Gamma) + t_{CU})/J, \quad (2.5)$$

де J – кількість шляхів між вершинами Start і End;

n_j – число операторних вершин для кожного шляху;

$T_M(\Gamma)$ – час такту;

t_{CU} – час циклу пристрою управління.

При цьому ми припускаємо, що вірогідність для усіх шляхів α_j однакова.

У мікропрограмі Γ існують два шляхи: $\alpha_1 = \langle b_1, \dots, b_{14}, b_{22} \rangle$ - при виконанні рівності умови $x_1 = 1$; $\alpha_2 = \langle b_1, \dots, b_4, b_{15}, \dots, b_{22} \rangle$ - при значенні логічної умови $x_1 = 0$. Отже, маємо $J = 2$, при цьому $n_j = 15$ і 12 , таким чином можна отримати, що $LT_M(\Gamma) = 405\tau + 13,5t_{CU}$.

Отже, для М-ОА знайдені наступні значення його основних характеристик: апаратні витрати $Q_M(\Gamma) = 2$, час такту $T_M(\Gamma) = 30\tau$ і латентність $LT_M(\Gamma) = 405\tau + 13,5t_{CU}$.

2.7 Синтез керуючого автомата по граф-схемі алгоритму

В моїй роботі блок керування цифрової системи реалізований на основі FSM Мура.

2.7.1 Формування відзначеної ГСА

Вихідні сигнали автомата Мура залежать тільки від станів, тому кожна операторна вершина ГСА повинна бути відзначена окремим станом. Відзначена ГСА автомата Мура формується за наступними правилами:

- початкова і кінцева вершини відзначаються станом A_0 ;
- кожна операторна вершина відзначається унікальною позначкою, починаючи з A_1 .

Застосування цієї методики до ГСА приводить до формування відзначеної ГСА, яка представлена на рисунку 2.4. Автомат Мура, що реалізує ГСА має $M = 22$ стани.

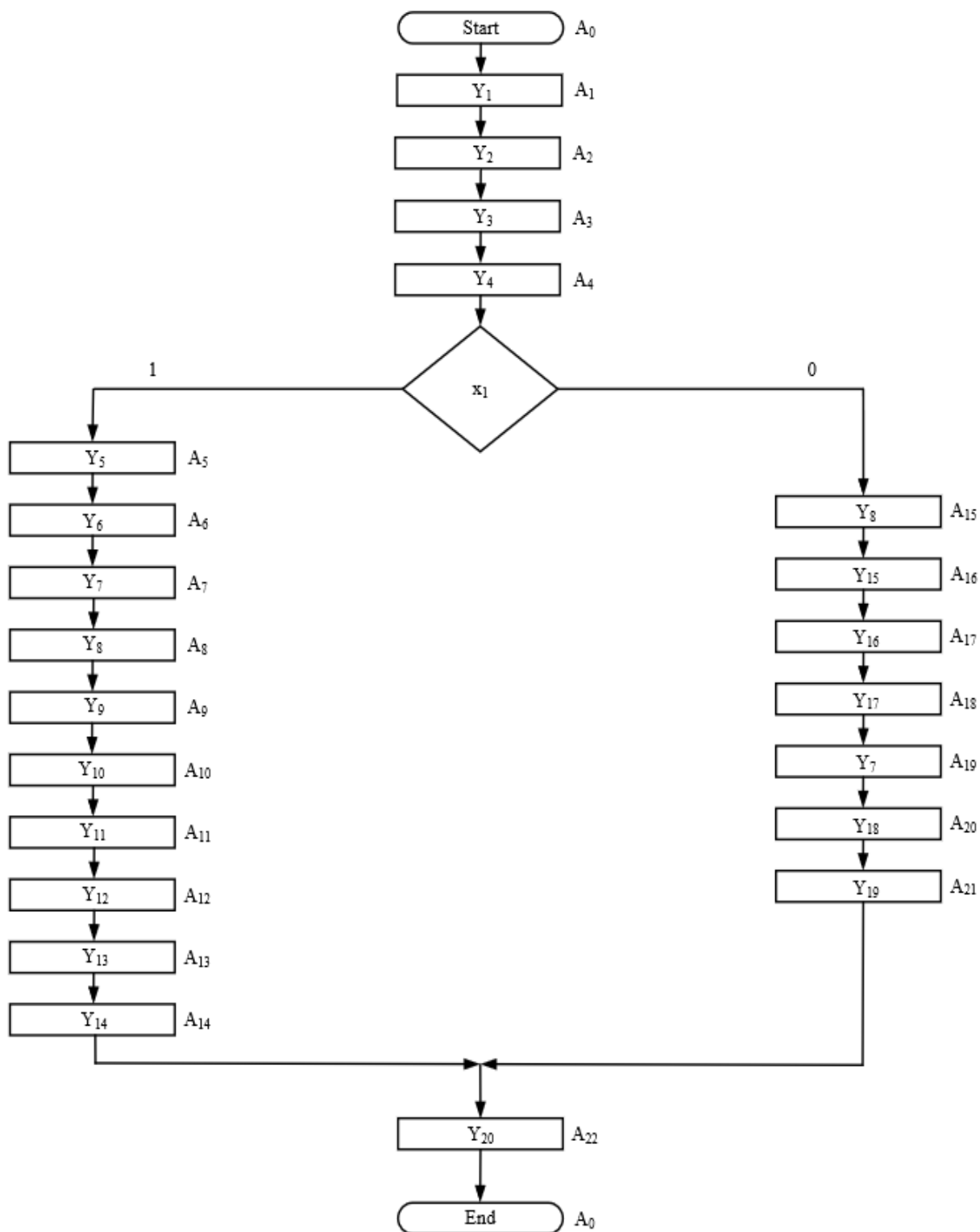


Рисунок 2.4 – Відзначена граф-схема алгоритму

2.7.2 Формування прямої структурної таблиці

FSM Мура подано у вигляді структурної таблиці зі стовпцями: a_m це початковий стан автомата, $a_m \in A$, де $A = \{A_1, \dots, A_m\}$ множина внутрішніх станів. $K(a_m)$ це код стану a_m . Кількість бітів коду стану a_m можна знайти за формулою:

$$R = \lceil \log_2 M \rceil, \quad (2.6)$$

де M – кількість станів автомата Мура.

Для мого автомата Мура кількість бітів коду стану $R = 5$. Кожен стан a_m кодується за допомогою множини внутрішніх змінних $T_r \in T = \{T_1, T_2, \dots, T_R\}$, де $R \geq \lceil \log_2 M \rceil$.

a_s це стан переходу. $K(a_s)$ код стану $a_s \in A$. X_h – вхідний сигнал, який визначає перехід з a_m до a_s і дорівнює кон'юнкції деяких елементів (або їх доповнень) множини логічних умов $X = \{x_1, \dots, x_L\}$. Φ_h це набір функцій вхідної пам'яті з множини $\Phi = \{D_1, \dots, D_R\}$, які дорівнюють 1, щоб переключити пам'ять автомата з $K(a_m)$ на $K(a_s)$. $h = 1, \dots, H$ це номер рядка. В стовпці a_m записаний набір мікрокоманд $Y(a_m) \subseteq Y$, які генерується в стані $a_m \in A$, де $Y = \{Y_1, \dots, Y_N\}$. Таким чином формується пряма структурна таблиця, приклад якої наведено у таблиці 2.2.

Таблиця 2.2 – Пряма структурна таблиця автомата Мура

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Φ_h	h
$A_0(-)$	00000	A_1	00001	1	D_1	1
$A_1(Y_1)$	00001	A_2	00010	1	D_2	2
$A_2(Y_2)$	00010	A_3	00011	1	D_1D_2	3
$A_3(Y_3)$	00011	A_4	00100	1	D_3	4
$A_4(Y_4)$	00100	A_5	00101	x_1	D_1D_3	5
		A_{15}	01111	$\bar{x}_1$	$D_1D_2D_3D_4$	6
$A_5(Y_5)$	00101	A_6	00110	1	D_2D_3	7
$A_6(Y_6)$	00110	A_7	00111	1	$D_1D_2D_3$	8
$A_7(Y_7)$	00111	A_8	01000	1	D_4	9
$A_8(Y_8)$	01000	A_9	01001	1	D_1D_4	10
$A_9(Y_9)$	01001	A_{10}	01010	1	D_2D_4	11
$A_{10}(Y_{10})$	01010	A_{11}	01011	1	$D_1D_2D_4$	12
$A_{11}(Y_{11})$	01011	A_{12}	01100	1	D_3D_4	13
$A_{12}(Y_{12})$	01100	A_{13}	01101	1	$D_1D_3D_4$	14
$A_{13}(Y_{13})$	01101	A_{14}	01110	1	$D_2D_3D_4$	15
$A_{14}(Y_{14})$	01110	A_{22}	10110	1	$D_2D_3D_5$	16
$A_{22}(Y_{20})$	10110	A_0	00000	1	—	17
$A_{15}(Y_8)$	01111	A_{16}	10000	1	D_5	18
$A_{16}(Y_{15})$	10000	A_{17}	10001	1	D_1D_5	19
$A_{17}(Y_{16})$	10001	A_{18}	10010	1	D_2D_5	20
$A_{18}(Y_{17})$	10010	A_{19}	10011	1	$D_1D_2D_5$	21
$A_{19}(Y_7)$	10011	A_{20}	10100	1	D_3D_5	22
$A_{20}(Y_{18})$	10100	A_{21}	10101	1	$D_1D_3D_5$	23
$A_{21}(Y_{19})$	10101	A_{22}	10110	1	$D_2D_3D_5$	24
$A_{22}(Y_{20})$	10110	A_0	00000	1	—	25

2.7.3 Синтез структурної схеми автомата Мура

Пряма структурна таблиця визначає структурну схему FSM Мура, яку зображено на рисунку 2.5.

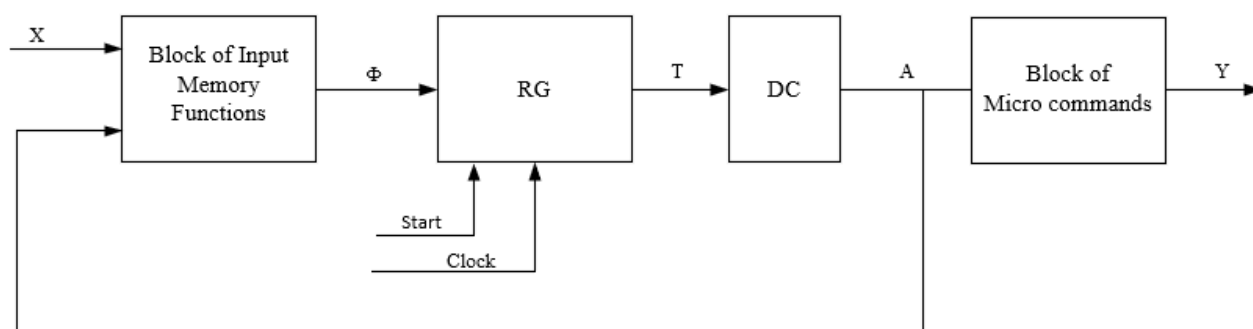


Рисунок 2.5 - Структурна схема автомата Мура

Регістр пам'яті RG використовується для збереження і відображення станів автомата $a_m \in A = \{A_1, A_2, \dots, A_m\}$. Дешифратор, що з'єднаний із RG, забезпечує відновлення стану A, який формує вихідні сигнали (мікрокоманди) і стани переходу.

Алгоритм роботи автомата:

а) після подання сигналу Start у регістр RG записується код початкового стану $K(A_0)$, який відповідає стану $A_0 \in A$;

б) код стану $K(a_m)$, що знаходиться у RG, передається на дешифратор. Дешифратор перетворює код $K(a_m)$ на конкретний стан A, після чого виконує дві основні функції:

1) формування функції збудження: комбінація стану A і вхідних сигналів x на виході блока функцій вхідної пам'яті генерує функції $\Phi = \Phi(T, x)$. Ці функції визначають код стану переходу $K(a_s)$, який буде записаний у RG;

2) формування мікрокоманд: на основі стану A блок мікрокоманд генерує набір мікрокоманд, обумовлений виразом $Y = Y(T)$;

в) за сигналом Clock регістр RG оновлюється — поточний код стану $K(a_m)$ замінюється на код стану переходу $K(a_s)$.

Цей процес повторюється циклічно, доки автомат не перейде в кінцевий стан.

2.7.4. Формування системи функцій автомата

У випадку структурної схеми, блок функцій вхідної пам'яті (BIMF) генерує систему функцій згідно формули:

$$\Phi = \Phi(T, x), \quad (2.7)$$

де T - це набір змінних стану, які використовуються для кодування станів $a_m \in A$;
 x – вхідні сигнали.

А блок мікрокоманд (ВМО) генерує систему функцій згідно формули:

$$Y = Y(T), \quad (2.8)$$

де T - це набір змінних стану, які використовуються для кодування станів $a_m \in A$.

A_m^h – кон'юнкція внутрішніх змінних, відповідна коду $K(a_m)$ вихідного стану a_m з h -го рядка прямої структурної таблиці. Кон'юнкція A_m^h визначається за формулою:

$$A_m^h = T_1^{l_{m1}} \dots T_R^{l_{mR}}, \quad (2.9)$$

де $l_{mr} \in \{0,1\}$ – значення r -го розряду коду $K(a_m)$, $T_r^0 = \bar{T}_r$, $T_r^1 = T_r$, ($r = \overline{1, R}$).

Як видно із систем (2.4) і (2.5), функції збудження пам'яті автомата визначені на множині термів $A = \{A_0, \dots, A_m\}$, які відповідають станам автомата Мура в комбінації з вхідними сигналами $X = \{x_1, \dots, x_L\}$, що визначають перехід із поточного стану до наступного, тоді як мікрокоманди визначені лише на множині термів $A = \{A_0, \dots, A_m\}$.

Система функцій (2.4) формується у виді формули:

$$\Phi_r = \bigvee_{h=1}^H C_{rh} * A_m * x_L, \quad (2.10)$$

де C_{rh} — булева змінна, рівна одиниці, якщо і тільки якщо в h -му рядку прямої

структурної таблиці записана функція Φ_r , яка активується у стані A_m під дією вхідного сигналу x_L .

Для мого автомата Мура система (2.7) має вид:

$$D_1 = A_0 \vee A_2 \vee (A_4 \wedge x_1) \vee (A_4 \wedge (\bar{x}_1)) \vee A_6 \vee A_8 \vee A_{10} \vee A_{12} \vee A_{16} \vee A_{18} \vee A_{20}.$$

Система функцій (2.5) формується у виді формули;

$$Y_n = \bigvee_{m=1}^M C_{nm} * A_m, \quad (2.11)$$

де C_{nm} – булева змінна, рівна одиниці, якщо і тільки якщо мікрокоманда Y_n формується в стані $a_m \in A$.

Для мого автомата Мура система (2.8) має вид:

$$Y_6 = A_6;$$

$$Y_7 = A_7 \vee A_{19}.$$

2.7.5 Синтез логічної схеми автомата

Логічну схему керуючого автомата Мура зображено на рисунку 2.6.

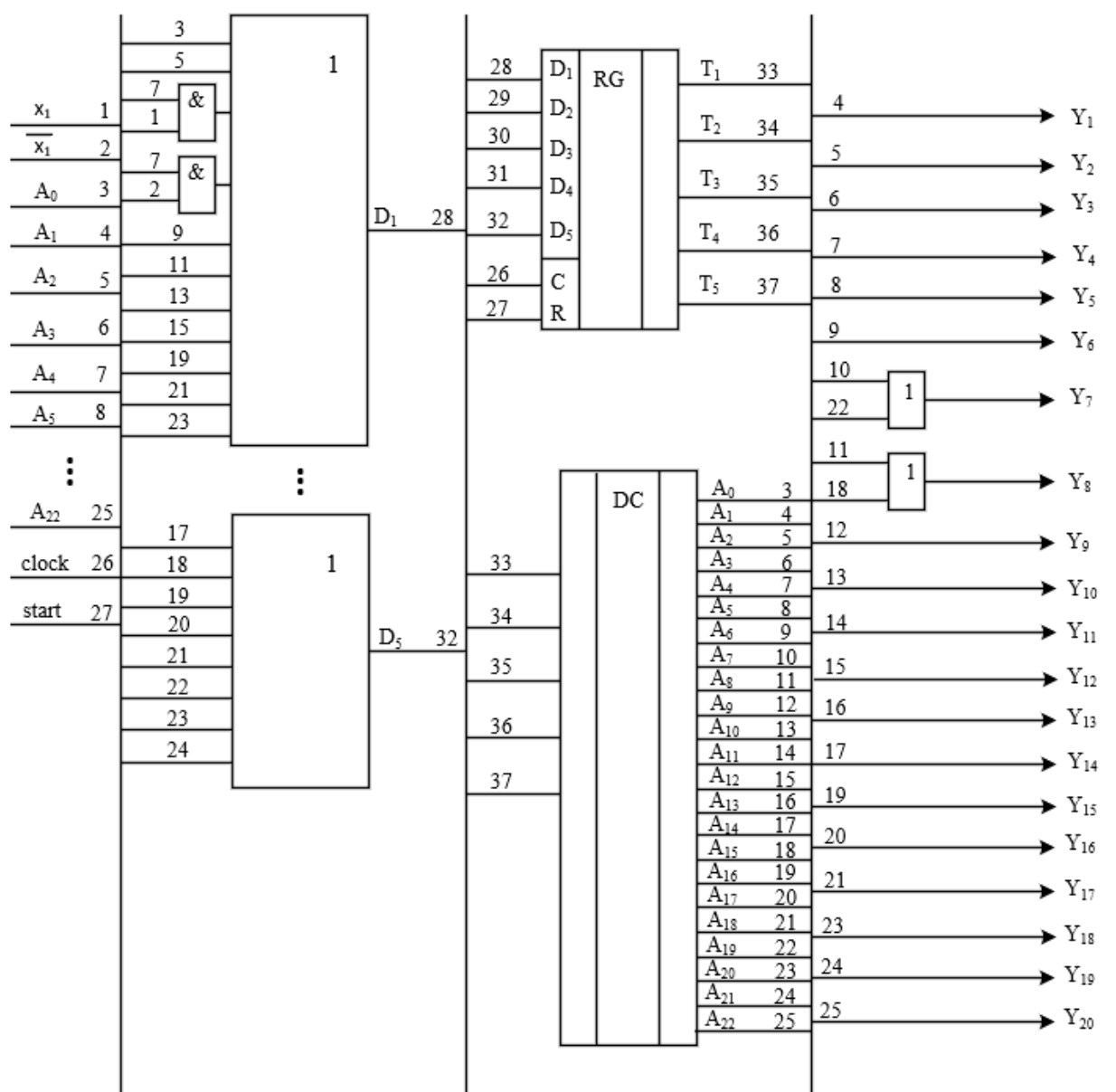


Рисунок 2.6 - Логічна схема керуючого автомата Мура

В цій логічній схемі пам'ять автомата реалізована на входовому регістрі RG з інформаційними входами D-типу, дешифратор станів реалізований на стандартному дешифраторі.

Кожна мікрооперація формується згідно з таблицею операторів (2.1).

Множина мікрооперацій для даного операційного автомата складає $MO = \{a_1, a_2, a_3, b_1, b_2, b_3, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7, \phi_8, \phi_9, \phi_{10}, \phi_{11}, c_1, c_2, c_3, c_4, c_5\}$.

Наприклад мікрооперація a_1 формується в наступних мікрокомандах Y_i :

$$a_1 = Y_3 \vee Y_4 \vee Y_8 \vee Y_{10} \vee Y_{16}.$$

Аналогічно складемо формули для всіх мікрооперацій із даного алгоритма. Ось приклади деяких з них:

$$b_1 = Y_3 \vee Y_4 \vee Y_{10} \vee Y_{15} \vee Y_{16};$$

$$\phi_4 = Y_4 \vee Y_6 \vee Y_{10} \vee Y_{12} \vee Y_{14} \vee Y_{15} \vee Y_{18};$$

$$\phi_5 = Y_5 \vee Y_8;$$

$$c_2 = Y_2 \vee Y_9 \vee Y_{10} \vee Y_{13} \vee Y_{15};$$

$$c_3 = Y_3 \vee Y_{12} \vee Y_{14} \vee Y_{19}.$$

2.8 Висновок за розділом 2

У другому розділі було розглянуто розробку операційного автомата з узагальненими мікроопераціями. Детально проаналізовано структуру операційного автомата, проведено формування пам'яті та виконано розподіл регістрів між шинами. Розробка таблиці операторів та формування логічних умов забезпечили чіткий опис поведінки автомата в різних станах.

Особливу увагу приділено синтезу функціональної схеми автомата, а також формуванню перетвореної мікропрограми, що є основою для подальшого керування мікроопераціями. У цьому ж розділі виконано синтез керуючого автомата на основі граф-схеми алгоритму. Зокрема, було сформовано відзначену граф-схему алгоритму, пряму структурну таблицю, побудовано структурну схему автомата Мура та системи функцій. Завдяки цьому вдалося оптимізувати функціонування керуючого автомата, який забезпечує ефективне управління взаємодією всіх компонентів системи.

Насамкінець було здійснено синтез схеми формування мікрооперацій, яка дозволяє реалізувати необхідні дії у системі.

3 ДОСЛІДЖЕННЯ СХЕМ ЦИФРОВОГО М-АВТОМАТА НА РІЗНИХ МІКРОСХЕМАХ

3.1 Реалізація схем М-автомата на VHDL

Середовище розробки дозволяє інженерам створювати складні цифрові схеми двома основними методами: графічне проектування блок-схем та програмування на мові VHDL. В даному дослідженні реалізовано операційний автомат типу М за допомогою блок-схеми та керуючий автомат як VHDL-код.

Побудова блок-схеми:

- у графічному редакторі створюються логічні блоки, які можуть бути взяті з бібліотеки стандартних компонентів або розроблені вручну;
- кожен блок виконує свою специфічну функцію: наприклад, суматори додають числа, регістри зберігають стан, мультиплексори керують потоком даних;
- з'єднання між блоками реалізуються за допомогою провідників, що відповідають за передачу сигналів та даних між компонентами;
- блоки можна налаштовувати: для регістрів визначається розрядність, для суматорів – розрядність операцій.

Створення проєкту:

Для створення нового проєкту в середовищі розробки необхідно обрати тип проєкту (блок-діаграма), вказати його назву (операційний автомат типу М) і визначити основні параметри, такі як кількість керуючих сигналів, сигнал скидання і синхронізації, розрядність вхідних і вихідних шин даних.

Використання блоків:

У проєкті були використані різні блоки, кожен з яких виконує окрему функцію. Ось список блоків використаних у проєкті:

- арифметичні блоки (суматори, двох бітні суматори). Блоки `Adder.bde`, `2bit_adder.bde` відповідають за виконання арифметичних операцій, таких як додавання та додавання двох біт з переносом;

- тригери (D-тригери): використовуються для зберігання біта даних. Блок D-trigger.bde відповідає за збереження бітів в операційному автоматі;
- регістри: використовуються для зберігання даних. Синхронізовані із сигналом тактового генератора. Блок Register.bde виконує роль регістра для ініціалізації і зберігання проміжних результатів. Також складається із D-тригерів кількість яких дорівнює розрядності однойменного слова. Значення цих блоків можуть бути скинуті в залежності від значення сигналу скидання;
- логічні блоки. Блоки AND.bde, XOR.bde, OR.bde, NOT.bde, ShL2.bde, ShL3.bde, ShR3.bde реалізують базові логічні операції;
- мультиплексори (MUX): використовуються для вибору одного з кількох вхідних сигналів, в залежності від значення керуючого сигналу. Блоки mul.bde, mul2.bde, mul3.bde виступають як мультиплексори, керуючи передачею даних між різними частинами проєкту;
- блок "enable" використовується як внутрішній блок для мультиплексорів, який дозволяє передавати дані на вихід, коли активується відповідний керуючий сигнал. Блок Enable.bde забезпечує контроль за потоком даних в певний момент часу.

Генерація VHDL-коду:

- після побудови блок-схеми середовище розробки автоматично генерує VHDL-код;
- генерація коду відбувається шляхом аналізу структури блоків, їхніх функцій та зв'язків між ними.

Для керуючого автомата використано мовний підхід, де логіка автомата описана на VHDL. Це дозволяє створити більш точну та гнучку реалізацію станів та переходів.

Лістинг VHDL-коду для керуючого автомата реалізує автомат Мура, де вихідні сигнали залежать лише від поточного стану автомата. Нижче наведені основні компоненти коду.

Опис сутності, який містить усі вхідні та вихідні сигнали визначаються у розділі port, наприклад, CLK для синхронізації, Reset для скидання, а Y1 ... Y20 є вихідними сигналами станів.

Архітектура автомата:

- використовуються сигнали Q для збереження поточного стану та D для переходу до наступного стану;
- процес синхронізації керується сигналами CLK та Reset. Якщо Reset активний, автомат повертається у початковий стан;
- логіка переходів між станами реалізована через сигнали D, які залежать від умов на вхідних сигналах x_i та поточного стану;
- виходи Y1 ... Y20 задаються відповідно до поточного стану.

3.2 Тестування і перевірка логічних схем

Для перевірки коректності реалізації проєктованих логічних схем було проведено їх тестування за допомогою часових діаграм у середовищі моделювання Active-HDL. Верифікація охоплювала три основні компоненти: схему формування мікрооперацій, операційний автомат і керуючий автомат.

3.2.1 Тестування операційного автомата

Тестування операційного автомата виконувалося з метою перевірки правильності виконання заданих мікрооперацій та відповідності вихідних сигналів заданому алгоритму. Часову діаграму тестування операційного автомата зображено на рисунку 3.1. Вона демонструє реакцію логічної схеми на зміну вхідних даних.

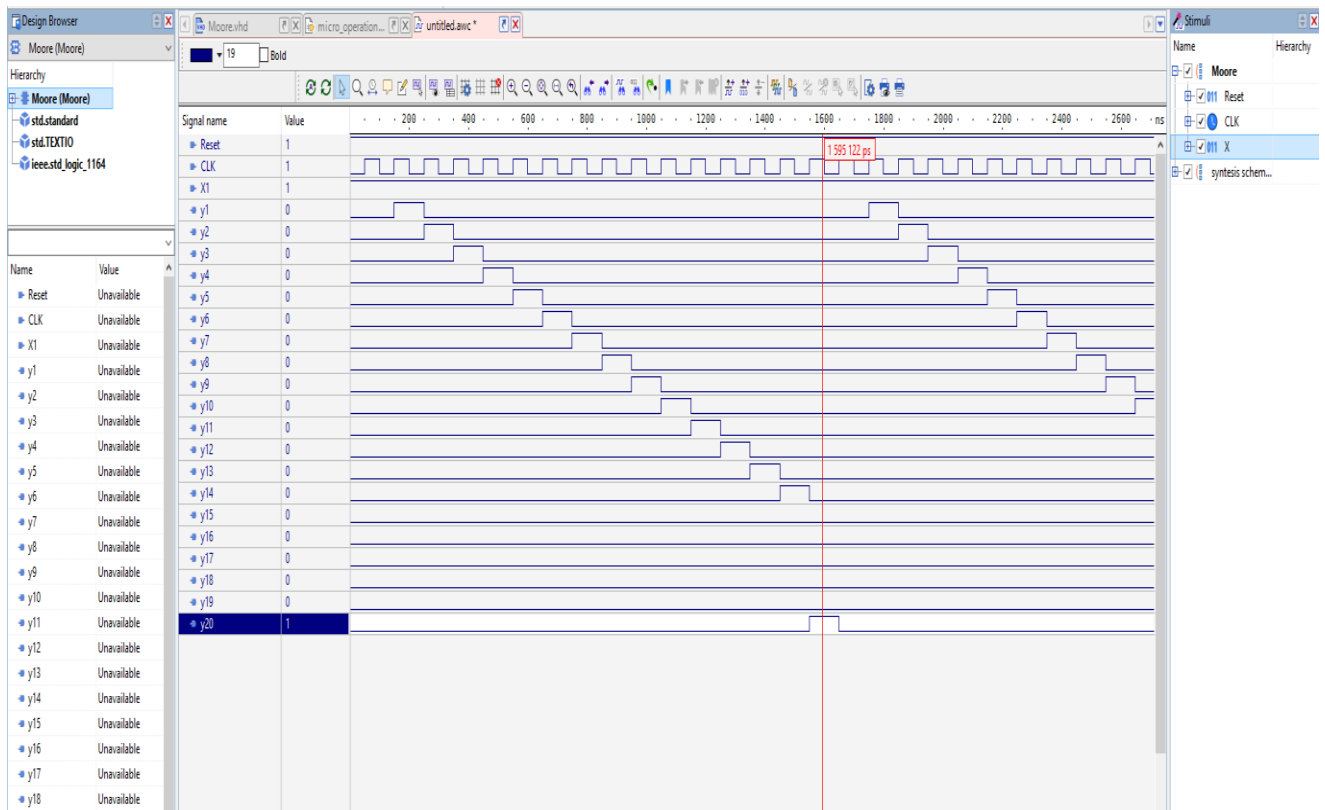


Рисунок 3.2 – Часова діаграма тестування керуючого автомата

Особливу увагу було приділено коректності формування вихідних сигналів (мікрокоманд), які впливають на роботу інших компонентів системи.

3.2.3 Тестування схеми формування мікрооперацій

Для перевірки коректності функціонування схеми формування мікрооперацій було змодельовано логіку її роботи у середовищі Active-HDL. Часову діаграму тестування схеми формування мікрооперацій зображено на рисунку 3.3.

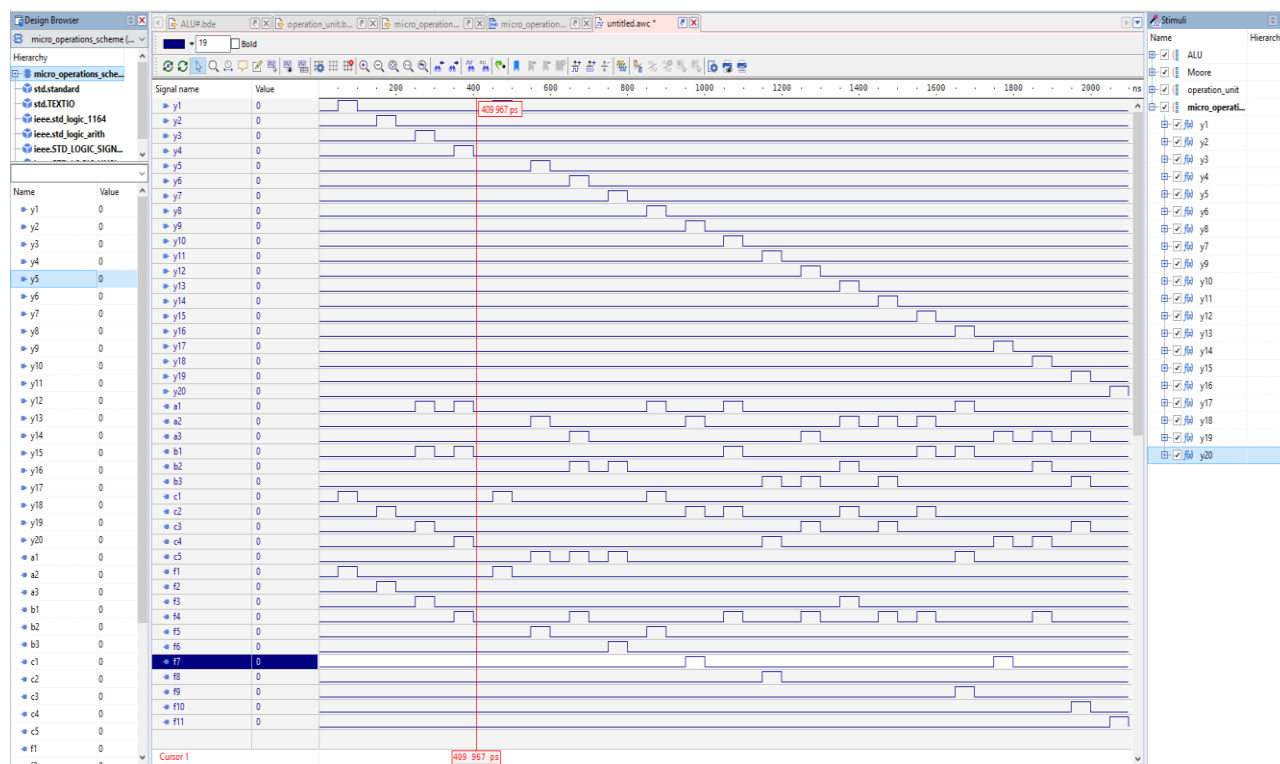


Рисунок 3.3 – Часова діаграма тестування схеми формування мікрооперацій

Таблиця відповідності мікрокоманд і мікрооперацій, приклад якої наведено у таблиці 3.1, містить інформацію про перетворення мікрокоманд у сукупність операторів, які використовуються для виконання мікрооперацій згідно з алгоритмом, на підставі даних взятих із таблиці операторів.

Таблиця 3.1 – Відповідності мікрокоманд і мікрооперацій

Мікрокоманда	Y_1	Y_2	Y_3	Y_4	Y_5
Мікрооперація	$\phi_1 c_1$	$\phi_2 c_2$	$a_1 b_1 \phi_3 c_3$	$a_1 b_1 \phi_4 c_4$	$a_2 \phi_5 c_5$
Мікрокоманда	Y_6	Y_7	Y_8	Y_9	Y_{10}
Мікрооперація	$a_3 b_2 \phi_4 c_5$	$b_2 \phi_6 c_5$	$a_1 \phi_5 c_1$	$a_2 \phi_7 c_2$	$a_1 b_1 \phi_4 c_2$
Мікрокоманда	Y_{11}	Y_{12}	Y_{13}	Y_{14}	Y_{15}
Мікрооперація	$b_3 \phi_8 c_4$	$a_3 b_3 \phi_4 c_3$	$a_2 b_2 \phi_3 c_2$	$a_2 b_3 \phi_4 c_3$	$a_2 b_1 \phi_4 c_2$
Мікрокоманда	Y_{16}	Y_{17}	Y_{18}	Y_{19}	Y_{20}
Мікрооперація	$a_1 b_1 \phi_9 c_5$	$a_3 \phi_7 c_4$	$a_3 b_2 \phi_4 c_4$	$a_3 b_3 \phi_{10} c_3$	ϕ_{11}

Відповідність сукупності операторів мікрокоманд із таблиці, операторам, що спостерігаються на часовій діаграмі, підтверджує коректність виконання алгоритму і правильність його реалізації.

3.3 Дослідження спецобчислювача на мікросхемах Altera/Intel

Я виконав верифікацію проекту методом моделювання, використовуючи схематичний редактор Quartus II. У процесі роботи було отримано звіти з імплементації проекту на різних мікросхемах фірми Altera, що дозволило оцінити відповідність дизайну вимогам синтезу та оптимізувати його для обраного пристрою. На рисунках 3.4, 3.5, 3.6 зображені звіти з імплементації у середовищі Quartus.

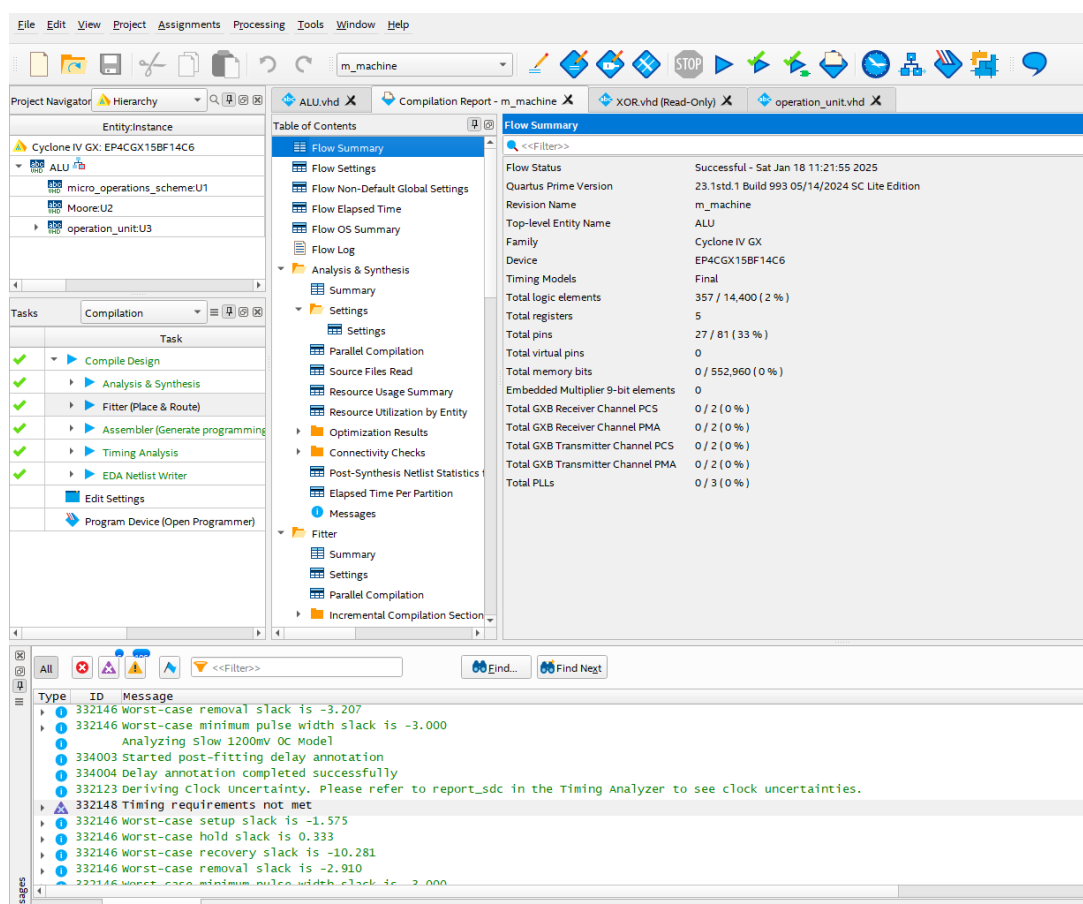


Рисунок 3.4 – Результати імплементації на мікросхемі Cyclone IV

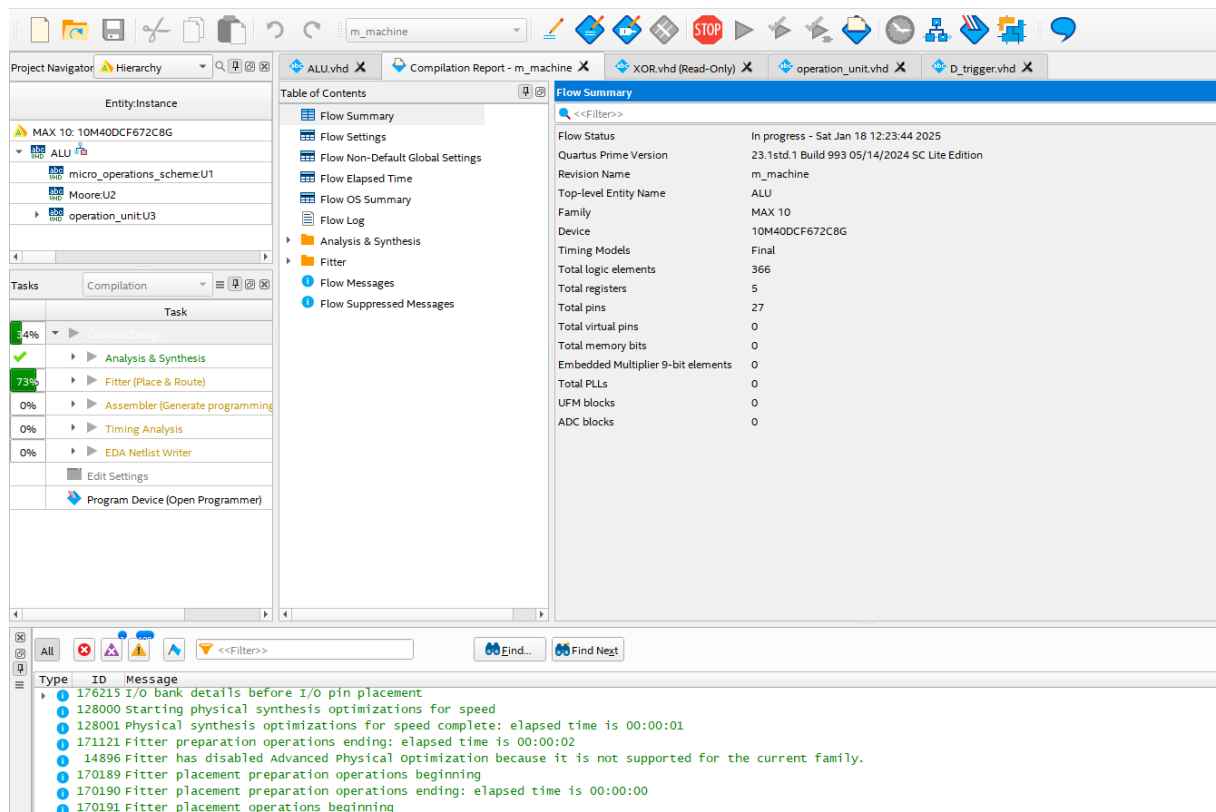


Рисунок 3.5 – Результати імплементації на мікросхемі MAX 10

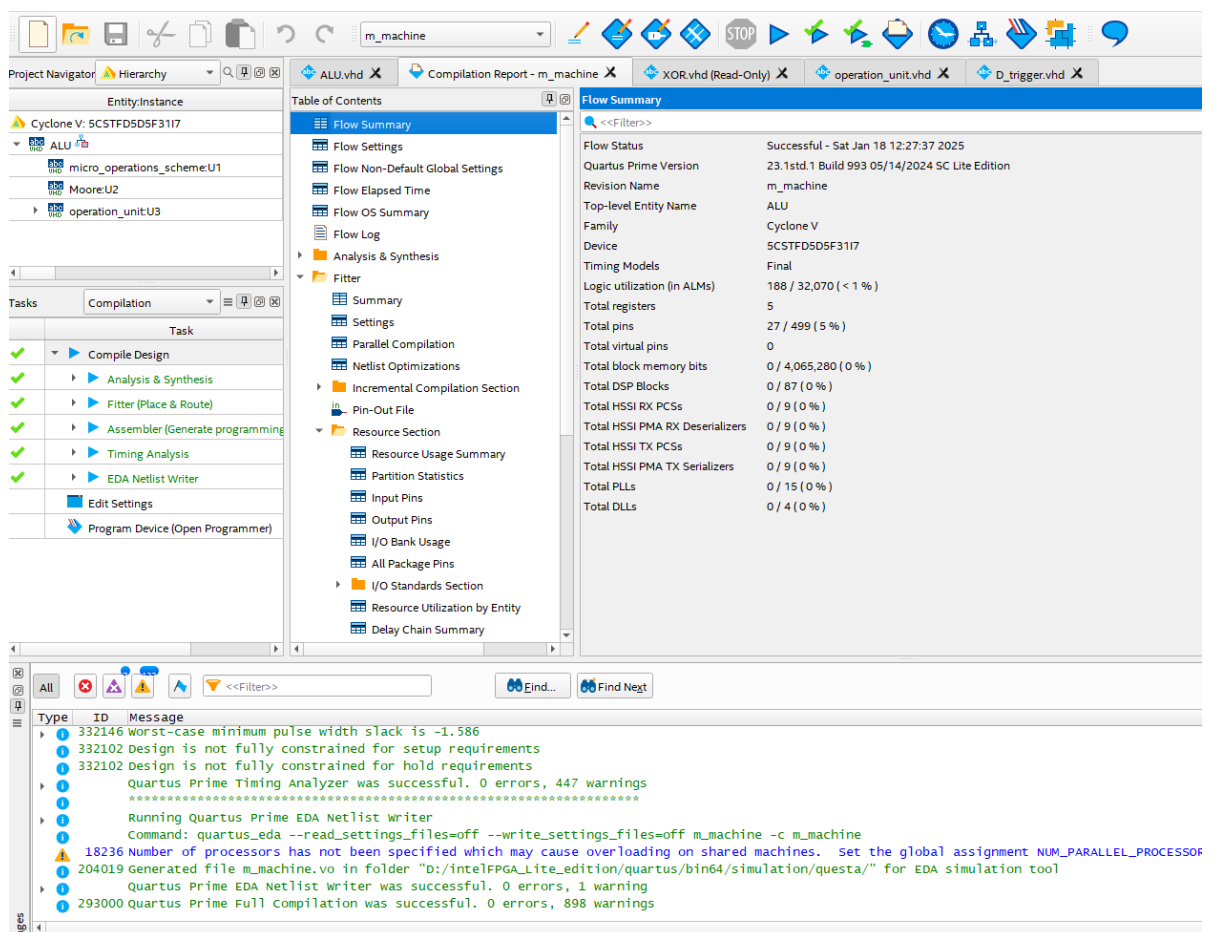


Рисунок 3.6 – Результати імплементації на мікросхемі Cyclone V

На рисунках 3.7, 3.8, 3.9 наведені топології мікросхем у середовищі Quartus.

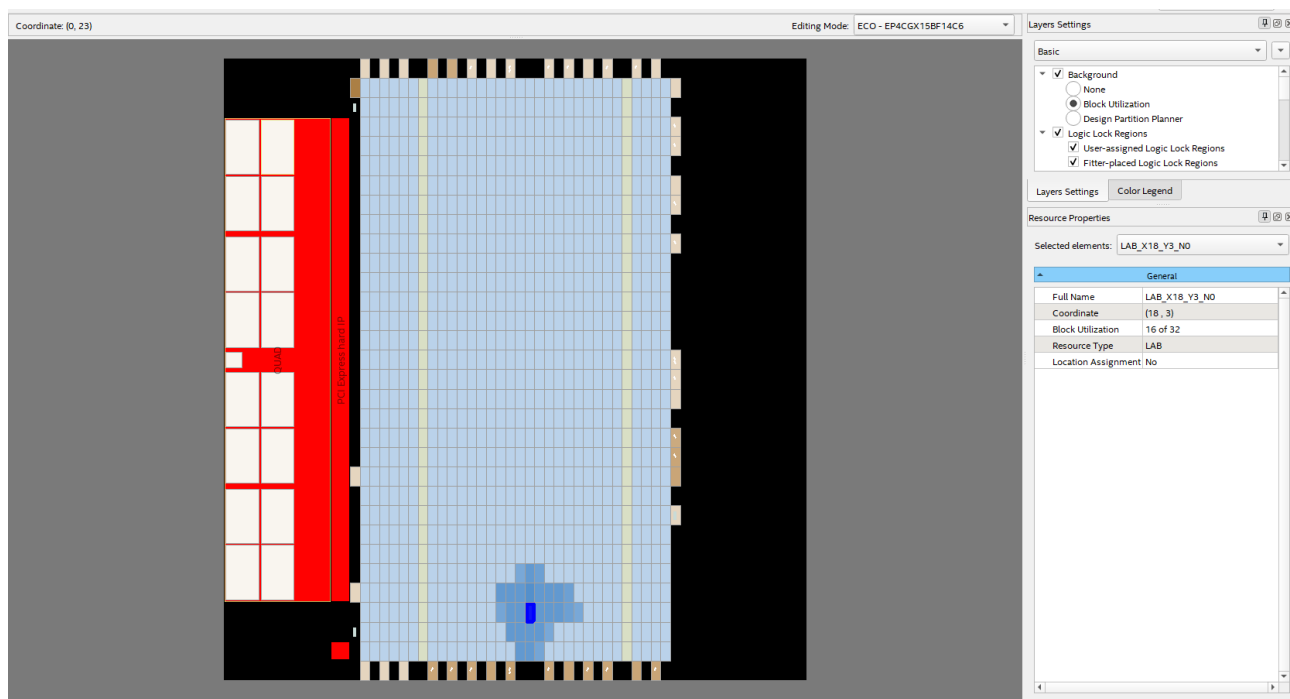


Рисунок 3.7 – Топологія мікросхеми Cyclone IV

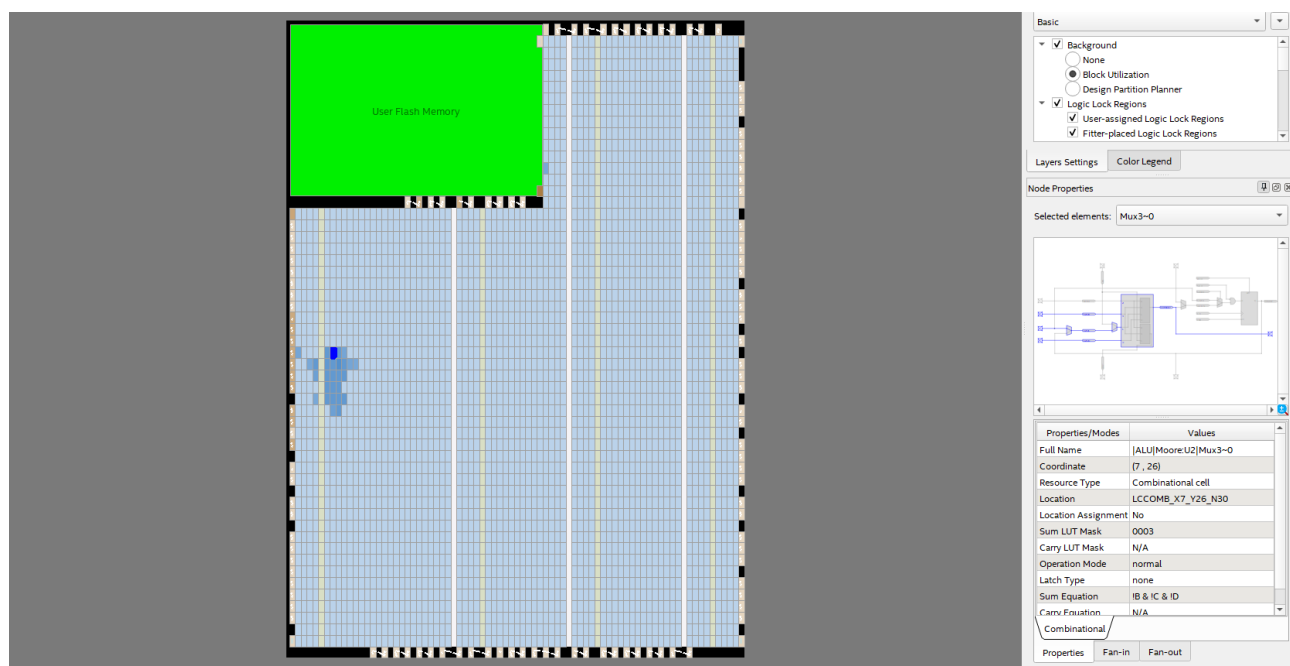


Рисунок 3.8 – Топологія мікросхеми MAX 10

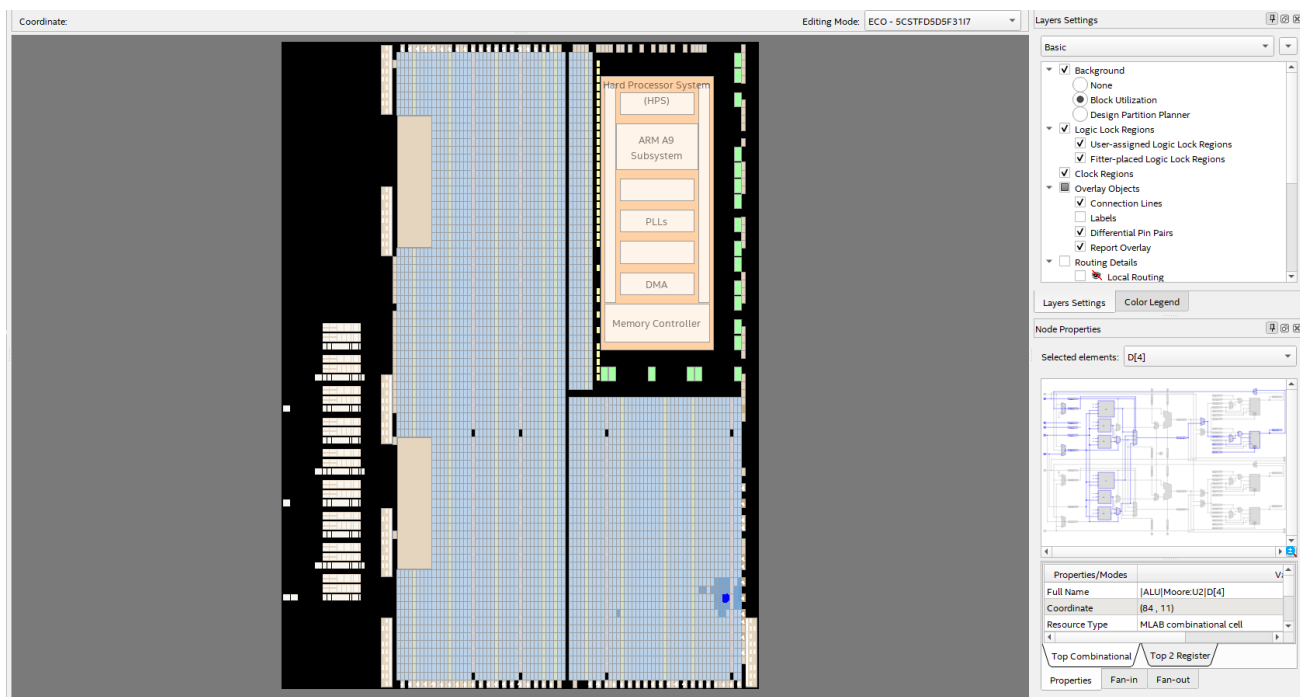


Рисунок 3.7 – Топологія мікросхеми Cyclone V

3.4 Порівняння апаратних витрат та практичні рекомендації

У таблиці 3.1 наведені апаратні затрати на різних мікросхемах для одного проекту та для більш наочного порівняння на графіку на рисунку 3.8 наведені дані з цієї таблиці.

Таблиця 3.1 – Апаратурні затрати на різних мікросхемах

Family	Logic elements	Total registers	Total pins	Total memory bits
Cyclone IV GX EP4CGX15BF14C6	357	5	27	0
MAX 10 10M40DCF672C8G	366	5	27	0
Cyclone V GX 5CSTFD5D5F31I7	188	5	27	0

На графіку нижче відображається зміна кількості логічних елементів залежно від мікросхеми.

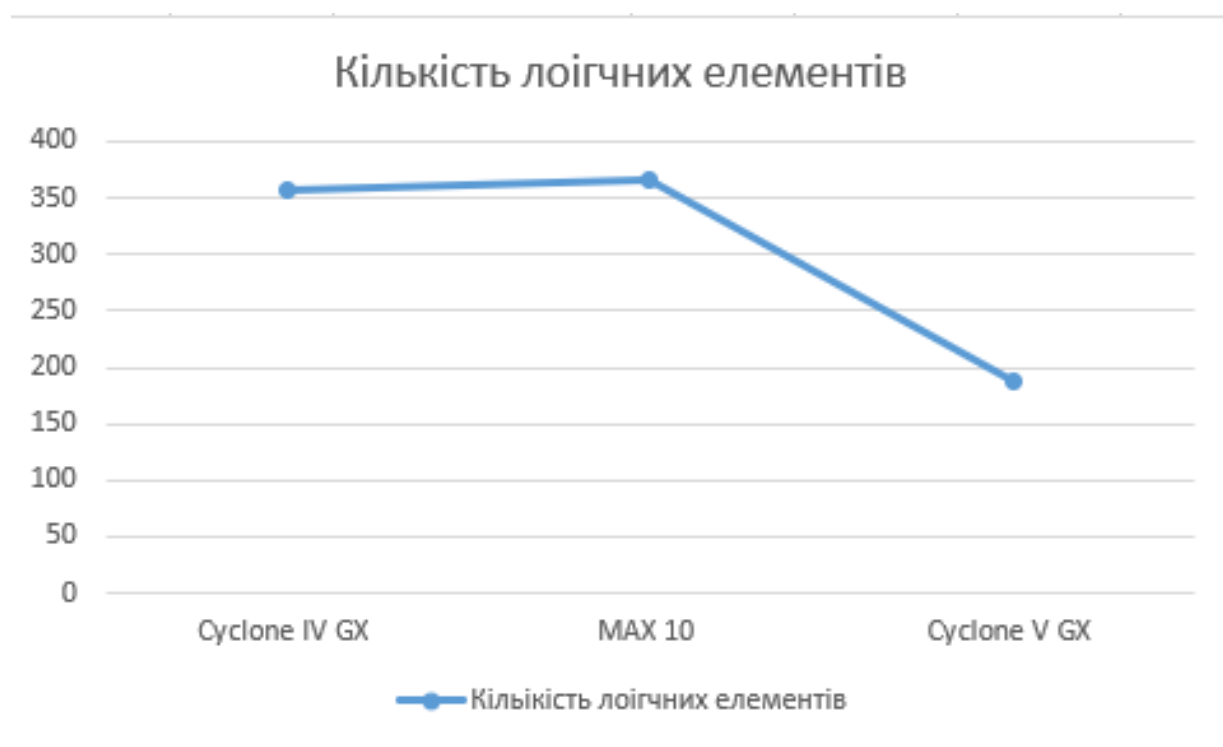


Рисунок 3.8 – Затрати логічних елементів на різних мікросхемах

Незважаючи на те, що кількість регістрів у всіх трьох мікросхемах залишається сталою, кількість логічних елементів змінюється, що може вказувати на різницю в архітектурі цих мікросхем. Мікросхеми Cyclone IV GX і MAX 10 мають схожі показники щодо кількості логічних елементів, що може свідчити про спільні технічні характеристики в цих моделях.

Таким чином можна зробити висновок, що на різних мікросхемах навіть з однаковою кількістю регістрів, ресурси використовуються по-різному. Мікросхема Cyclone V GX має найменше число логічних елементів, що може бути результатом більшої щільності упаковки логічних елементів на кристалі або інших специфічних особливостей її архітектури.

3.5 Висновки за результатами дослідження

Процес проєктування:

- операційний автомат типу М був реалізований за допомогою блок-схем у графічному редакторі середовища розробки;
- для створення керуючого автомата використано мову VHDL, що забезпечило більш точне та гнучке описання станів і переходів;
- генерація VHDL-коду виконувалася автоматично на основі блок-схем, що спрощувало процес інтеграції та тестування компонентів.

Логіка роботи автомата:

- керуючий автомат формує вихідні сигнали (мікрокоманди) залежно від поточного стану і стани переходу в залежності від поточного стану і результату логічних умов ;
- схема формування мікрооперацій ділить мікрокоманди на сукупність операторів, які передаються операційному автомату для виконання обчислень;
- операційний автомат приймає сукупності операторів і виконує мікрооперації за заданим алгоритмом.

Симуляція роботи:

- симуляція проводилася з використанням часових діаграм у середовищі Active-HDL, які підтвердили коректність функціонування автомата на кожному етапі;
- моделювання цифрової схеми виконувалося у середовищі Quartus II, що дало змогу отримати звіти про використання логічних елементів, інших ресурсів, а також топології мікросхем;
- виявлені під час тестування проблеми були усунуті до реалізації на апаратному рівні.

Оптимізація ресурсів:

- мінімізація кількості тригерів і регістрів шляхом зменшення кількості станів автомата;

- зменшення числа операторних вершин, якщо у верхній і нижній частинах вираження можна виділити еквівалентні операції, то їх доцільно виконати перед перевіркою логічної умови (перед розгалуженням). Це зменшить кількість тактів для отримання результату.

Подальший розвиток операційного автомата типу М:

- операційний автомат може бути інтегрований в складніші системи або використаний як частина багатокомпонентної обчислювальної платформи. Це дозволить масштабувати автомат для складніших завдань і покращити його взаємодію з іншими пристроями;

- для подальшого розвитку автомата можна покращити засоби його тестування, впровадивши більш гнучкі і ефективні інструменти налагодження, які дозволять відслідковувати помилки в реальному часі.

ВИСНОВКИ

У межах цього дослідження було реалізовано послідовний обчислювач на ПЛІС із апаратною оптимізацією, що базується на принципах операційного автомата типу М. Основною метою роботи було створення ефективної архітектури, яка забезпечує мінімізацію використання апаратних ресурсів при виконанні обчислень із заданою точністю.

Розробка пристрою виконувалася у середовищі проектування, що підтримує графічне проектування та програмування на VHDL. Графічний підхід використовувався для моделювання арифметичних, логічних і керуючих блоків, тоді як VHDL-код забезпечив реалізацію гнучкого керуючого автомата. Така комбінація методів дозволила не лише досягти апаратної оптимізації, але й підвищити гнучкість та масштабованість пристрою.

Апаратна оптимізація була досягнута завдяки раціональному розподілу функцій між окремими компонентами та використанню мінімальної кількості логічних елементів. Узагальнена комбінаційна частина операційного автомата дозволила зменшити витрати на реалізацію складної логіки. Послідовний режим виконання мікрооперацій забезпечив узгодженість роботи пристрою при мінімальних апаратних затратах, що особливо важливо для систем, які реалізуються на ПЛІС із обмеженими ресурсами.

Функціональність пристрою була перевірена шляхом моделювання та тестування у середовищі Active-HDL із використанням часових діаграм. Часові діаграми дозволили проаналізувати роботу кожного блоку на різних етапах виконання обчислень, виявити можливі помилки та підтвердити узгодженість функціонування системи загалом.

Окрім цього, тестування продуктивності виконувалося на FPGA платформах Altera (Intel). Результати показали стабільну роботу обчислювача, підтвердили його сумісність із різними архітектурами та довели доцільність використання такого підходу у задачах автоматизації та цифрової обробки даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Огляд архітектури Intel [Електронний ресурс] – Режим доступу до ресурсу: <https://www.sciencedirect.com/topics/computer-science/intel-architecture>.
2. Розуміння архітектури x86 [Електронний ресурс] – Режим доступу до ресурсу: <https://www.allaboutcircuits.com/technical-articles/an-introduction-to-x86-processor-architecture/>.
3. Огляд архітектури ARM [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/arm-processor-and-its-features/>.
4. Введення в архітектуру ARM [Електронний ресурс] – Режим доступу до ресурсу: <https://ebooks.inflibnet.ac.in/csp13/chapter/introduction-to-arm-processor/>.
5. RISC-V: відкритий стандарт [Електронний ресурс] – Режим доступу до ресурсу: <https://www.wevolver.com/article/risc-v-instruction-set>.
6. Пояснення архітектури VLIW [Електронний ресурс] – Режим доступу до ресурсу: <https://courses.cs.washington.edu/courses/csep548/06au/lectures/vLIW.pdf>.
7. Розуміння архітектури VLIW [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/very-long-instruction-word-vliw-architecture/>.
8. What is an FPGA? Definition, Types, Programming, and More [Електронний ресурс] – Режим доступу до ресурсу: <https://www.datacenterknowledge.com/hardware/what-fpga-definition-types-programming-and-more#close-modal>.
9. Зображення внутрішньої структури ПЛІС [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/figure/linux-style-FPGA-architecture-contains-an-array-of-CLBs-switchboxes-and-vertical-and_fig1_243558701.
10. CPLD vs FPGA what is the difference? [Електронний ресурс] – Режим доступу до ресурсу: <https://fpgainsights.com/fpga/cpld-vs-fpga-what-is-the-difference/>.
11. What is ASIC used for? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.maven-silicon.com/blog/what-is-asic-used-for/>.

12. What is an Application-specific Integrated Circuit (ASIC)? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.allaboutcircuits.com/technical-articles/what-is-an-asic-application-specific-ic/>.

13. Programmable ASIC Companies [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mordorintelligence.com/industry-reports/programmable-application-specific-integrated-circuit-asic-market/companies>.

14. Зображення розподілення сімейств ПЛІС за цінами [Електронний ресурс] – Режим доступу до ресурсу: <https://www-gfec-production.onemacnica.com/en/products/semiconductor/intel-fpga>.

15. Огляд сімейства ПЛІС Cyclone IV [Електронний ресурс] – Режим доступу до ресурсу: <https://biakom.com/news/130-cyclone-iv--nedorogie-fpga-s-vysokoskorostnymi-priemoperedatchikami-i-nizkim-energopotrebleniem>.

16. Огляд сімейства ПЛІС Cyclone V [Електронний ресурс] – Режим доступу до ресурсу: <https://biakom.com/news/236-kompaniya-altera-nachinaet-postavki-plis-cyclone-v>.

17. Огляд сімейства ПЛІС Altera MAX10 [Електронний ресурс] – Режим доступу до ресурсу: <https://marsohod.org/11-blog/279-altera-max10>.

18. Зображення мікросхеми сімейства MAX10 [Електронний ресурс] – Режим доступу до ресурсу: <https://www.rcscomponents.kiev.ua/product/dk-dev-10m50-a.html>.

19. Лахно В.А., Гусєв Б.С., Смолій В.В., Місюра М.Д., Касаткін Д.Ю. Технології проектування комп'ютерних систем. – Національний Університет Біоресурсів і Природокористування України, 2019. – 250 с.

20. Quartus II 7.2 [Електронний ресурс] – Режим доступу до ресурсу: <https://quartus-ii.updatestar.com/>.

21. Adamski M., Barkalov A. Architectural and sequential synthesis of digital devices. – Zielona Gora: University of Zielona Gora Press, 2006. – 199 pp.

22. Baranov S. Logic and system design of digital systems. – Tallinn: TUT Press, 2008. – 267 pp.

Додаток А

Таблица для обчислення результату

Тестування роботи спецобчислювача при вхідних даних:

$A = D_1 = 01011000 = 88_{10};$

$B = D_2 = 11001010 = 202_{10};$

$X_1 = 0.$

Показано тестування по одній гілці алгоритма.

Таблица А.1 – Тестове обчислення значення результату алгоритма по одній гілці

	Вершини змістовного графа	Вершини закодованого графа																						Вміст регістрів					
№	Мікрооперація (зміст)	a ₁	a ₂	a ₃	b ₁	b ₂	b ₃	φ ₁	φ ₂	φ ₃	φ ₄	φ ₅	φ ₆	φ ₇	φ ₈	φ ₉	φ ₁₀	φ ₁₁	c ₁	c ₂	c ₃	c ₄	c ₅	RA	RB	RS ₁	RS ₂	RS ₃	RES
1	RA := A	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	0101 1000	00	00	00	00	00
2	RB := B	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-	-	-	0101 1000	1100 1010	00	00	00	00
3	RS ₁ := RA xor RB	1	-	-	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-	-	0101 1000	1100 1010	1001 0010	00	00	00
4	RS ₂ := RA + RB	1	-	-	1	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-	0101 1000	1100 1010	1001 0010	0010 0010	00	00
5	RA := shl ₂ (RA)	1	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	-	-	-	-	0110 0000	1100 1010	1001 0010	0010 0010	00	00
6	RB := RB + RB	-	1	-	1	-	-	-	-	-	1	-	-	-	-	-	-	-	-	1	-	-	-	0110 0000	1001 0100	1001 0010	0010 0010	00	00
7	R ₃ := RA and RB	1	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	0110 0000	1001 0100	1001 0010	0010 0010	0000 0000	00
8	RS ₂ := shr ₃ (RS ₂)	-	-	1	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	1	-	0110 0000	1001 0100	1001 0010	0000 0100	0000 0000	00
9	RS ₃ := not(RS ₃)	-	-	-	-	1	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	1	0110 0000	1001 0100	1001 0010	0000 0100	1111 1111	00
10	RS ₂ := RS ₂ + RS ₃	-	-	1	-	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	1	-	0110 0000	1001 0100	1001 0010	0000 0011	1111 1111	00
11	RS ₁ := RS ₂ or RS ₁	-	-	1	-	-	1	-	-	-	-	-	-	-	-	-	1	-	-	-	1	-	-	0110 0000	1001 0100	1001 0011	0000 0011	1111 1111	00
12	(Out) Result := RS ₁	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	0110 0000	1001 0100	1001 0011	0000 0011	1111 1111	1001 0011