Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему Спецобчислювач на FPGA із паралельною структурою операційної частини

Виконав: студент 2 курсу, групи КНТ-613М
Спеціальності 123 «Комп'ютерна інженерія»
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Спеціалізовані комп'ютерні системи

КУХТІН К. І.
(прізвище та ініціали)

Керівник ЗЕЛЕНЬОВА І. Я.
(прізвище та ініціали)

Рецензент СТЕПАНЕНКО О. О.
(прізвище та ініціали)

2024

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти магістрський

Спеціальність 123 Комп'ютерна інженерія

(код і найменування)

Освітня програма (спеціалізація) Спеціалізовані комп'ютерні системи

(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри КУДЕРМЕТОВ Р.К.

_____

"____" _____ 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

КУХТІНА Костянтина Ігоровича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Спецобчислювач на FPGA із паралельною структурою операційної частини

керівник проєкту (роботи) доцент, ЗЕЛЕНЬОВА Ірина Яківна

(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від "18" жовтня 2024 року № 149

2. Строк подання студентом проєкту (роботи) 20 грудня 2024 року

3. Вихідні дані до проєкту (роботи) Aldec Active-HDL 13, теоретичні матеріали з синтезу операційних пристроїв, Quartus II

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

1) Аналіз технічного завдання, дослідження архітектур процесорних ядер сучасних процесорів компаній AMD та Intel;

2) Розробка алгоритму та структури цифрового пристрою;

3) Моделювання та тестування розробленого цифрового пристрою.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	ЗЕЛЕНЬОВА І. Я., доцент		
нормоконтроль	ПОЛЬСЬКА О. В., ст. викл.		

7. Дата видачі завдання 24.10.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз технічного завдання та сучасних архітектур	23.09.2024 р.	
	ядер процесорів компаній AMD та Intel		
2	Розробка алгоритму пристрою	08.10.2024 р.	
3	Розробка структурних елементів пристрою	10.11.2024 р.	
4	Моделювання структурних елементів пристрою	19.11.2024 р.	
5	Тестування структурних елементів пристрою	21.11.2024 р.	
6	Оформлення графічного матеріалу	16.12.2024 р.	
7	Оформлення пояснювальної записки	17.12.2024 р.	

Студент (ка) _____ **Костянтин КУХТІН**
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ **Ірина ЗЕЛЕНЬОВА**
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 70 с.; 45 рис.; 9 табл.; 19 джерел; 1 додаток

АРХІТЕКТУРА ЯДЕР ЦП, ОПЕРАЦІЙНИЙ АВТОМАТ, FPGA, СПЕЦОБЧИСЛЮВАЧ, ПАРАЛЕЛЬНІ ОПЕРАЦІЇ, ОБ'ЄДНАННЯ ОПЕРАЦІЙ.

Об'єкт дослідження – спецобчислювач на FPGA із паралельною структурою операційної частини.

Предмет дослідження – операційні частини обчислювачів різних архітектур.

Мета роботи – підвищення швидкодії операційного пристрою спецобчислювача із операційною частиною паралельного типу.

Матеріали, методи та технічні засоби: програмне забезпечення Aldec Active-HDL, персональний комп'ютер з процесором Intel Core i5 під управлінням операційної системи Microsoft Windows 10.

Результати. За допомогою ПЗ Aldec Active-HDL створено спецобчислювач із операційною частиною паралельного типу та досліджено на різних мікросхемах Altera\Intel.

Висновки. Розроблено спецобчислювач із операційною частиною паралельного типу, оптимізовано його роботу та перевірено коректність виконання ним заданих йому операцій.

Галузь використання – схемотехніка.

ABSTRACT

Explanatory note to the master's work: 70 p., 45 figures, 9 tables, 19 sources.

CPU CORE ARCHITECTURE, OPERATIONAL UNIT, FPGA, SPECIAL FPGA-BASED COMPUTER, PARALLEL OPERATIONS, AGGREGATED OPERATIONS

Object research – special FPGA-based computer with a parallel structure of the operating part.

The subject of the research is the operating parts of various processing units with different architectures.

The purpose of the work is to improve the performance of the operatinon unit of a special FPGA-based computer with a parallel structure of the operating part.

Materials, Methods, and Techniques: the program product Aldec Active-HDL, a personal computer with an Intel Core i5 CPU, and the Windows 10 operating system.

Results – using the program product Aldec Active-HDL, a special FPGA-based computer with a parallel structure of the operating part was created and tested on different circuits from Altera/Intel.

The field of use is electronic circuit design.

ЗМІСТ

ВСТУП

На сьогодні є лише два основні виробники центральних процесорів, це компанії Intel та AMD які конкурують між собою періодично, перехоплюючи першість один в одного. Кожна компанія намагається розробити свою максимально ефективну, у своєму баченні ефективності, процесорну систему. І архітектура процесора та його окремих ядер відіграє тут як мінімум настільки ж важливу роль як і фізична кількість транзисторів які вдається розмістити на кристалі. Це можна схарактеризувати як ріст в ширину – збільшення елементної бази, водночас розробка архітектури це ріст у глибину.

Як результат, процесорні системи мають велику кількість різних архітектур. Вони відрізняються залежно від виробника самого кристалу процесора та задач, які цей самий виробник покладає на процесорну систему. Також, звісно, архітектури самих процесорів можуть відрізнятися навіть в межах одного покоління. Як приклад, архітектура процесора з вбудованим графічним ядром буде сильно відрізнятись від процесора який буде використовувати такі самі процесорні ядра, але не матиме графічного ядра. І так само сильно відрізнятись від згаданих вище типів центральних процесорів буде й архітектура «серверних» процесорів. Від архітектури самого процесора може залежати не тільки його швидкодія, а й енергоефективність, надійність, ефективність в цілому взаємодії між собою внутрішніх компонентів. Тому задача проєктування операційного та керуючого блоків процесора є актуальною.

Найпростішим і наочним прикладом відмінностей архітектур і їх впливу на процесорну систему можна вважати архітектури спецобчислювачів. Це зумовлено тим що їх архітектури являють собою дві крайнощі та варіант по середині що дозволить досить ясно побачити відмінності архітектур, їх переваги та недоліки в порівнянні з аналогами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз різних архітектур сучасних процесорів

1.1.1 Архітектури центральних процесорів компанії AMD

На сьогодні компанія AMD використовує у своїх процесорах ядра лінійки архітектур Zen які прийшли на заміну архітектури Excavator. Виробництво процесорів із ядрами цієї архітектури почалось у кінці 2017 року. Зараз у цій лінійці вже є чотири покоління і нещодавно відбувся реліз вже п’ятого покоління [1].

Кожне наступне покоління не змінює структуру принципово, а скоріше покращує попередню. Так, наприклад порівняно з Zen, у Zen 2 було розширено шину для I-кешу з чотирьох до восьми, але вдвоє зменшено сам обсяг пам’яті. Також додали один додатковий модуль генерації адрес, у Zen їх було два, а починаючи з Zen 2 і до Zen 4 їх було три, а вже з Zen 5 їх буде чотири. Це скоріше за все пов’язано із збільшенням швидкості роботи арифметико-логічних модулів. Але окрім пришвидшення, самих «арефмів» також стало більше на два, якщо порівнювати Zen та Zen 5 [2].

На рис. 1.1 маємо можливість роздивитись схематичне зображення структури одного ядра центрального процесора з архітектурою Zen. Можемо побачити що ядро розділено на блоки в які об’єднано модулі згідно з їх задачами. Чітко видно і зрозуміло призначення блоків для обробки цілочисельних значень та значень із плаваючою точкою. «Над» цими двома блоками бачимо керуючу частину яка інтерпретує інструкції та передає їх у вигляді мікрооперацій функціональним блокам. Останні два блоки призначені для роботи із даними які необхідно обробити або являють собою вже результат обробки.

Звісно це красиве розділення на блоки лише умовне, зроблене для кращого розуміння “загальної картини” як звичайних людей, так і самих розробників. Приклад того як виглядає ядро процесора фактично наведено на рис. 1.2.

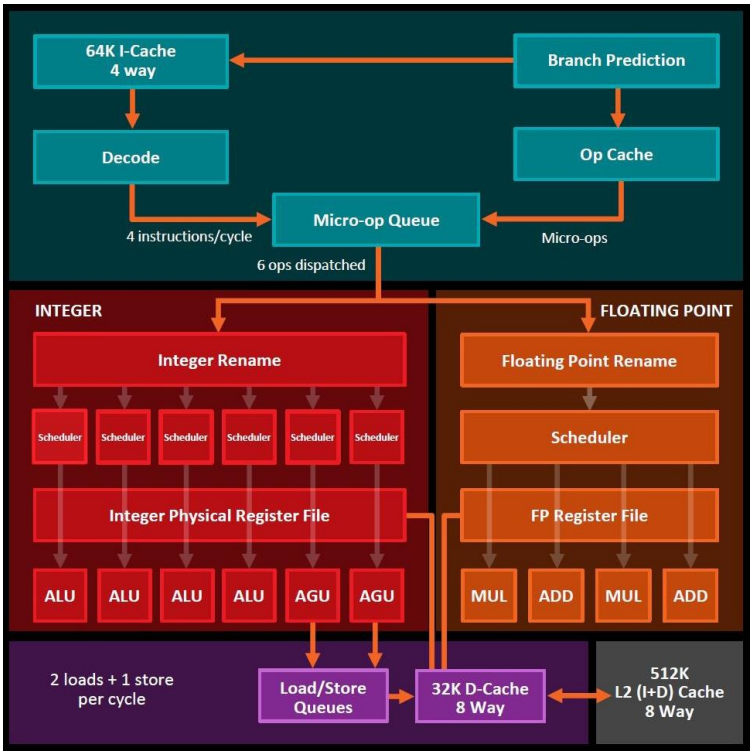

Рисунок 1.1 – Схематичне зображення процесорного ядра архітектури Zen [2]

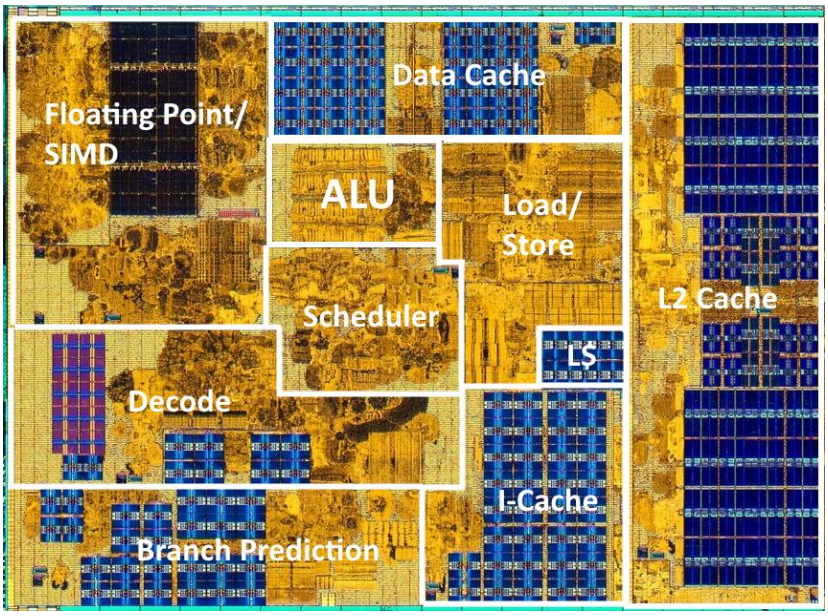

Рисунок 1.2 – Фактичний вигляд процесорного ядра архітектури Zen [3]

Для порівняння можемо також розглянути архітектуру ядра Zen 5, її схематичне зображення відображено на рис. 1.2. І власне можемо побачити ті два додаткові арифметико-логічні пристрої та генератори адрес, збільшення кеш-пам’яті і її канальності, за виключенням І-кешу об’єм якого зменшили ще у Zen 2. Також можна помітити зміни у модулях планувальників. У Zen 5 на

цілочисельні операції тепер два планувальники замість шести, в той час, як для операцій з плаваючою точкою навпаки, цільний планувальник було розділено на три незалежні [1].

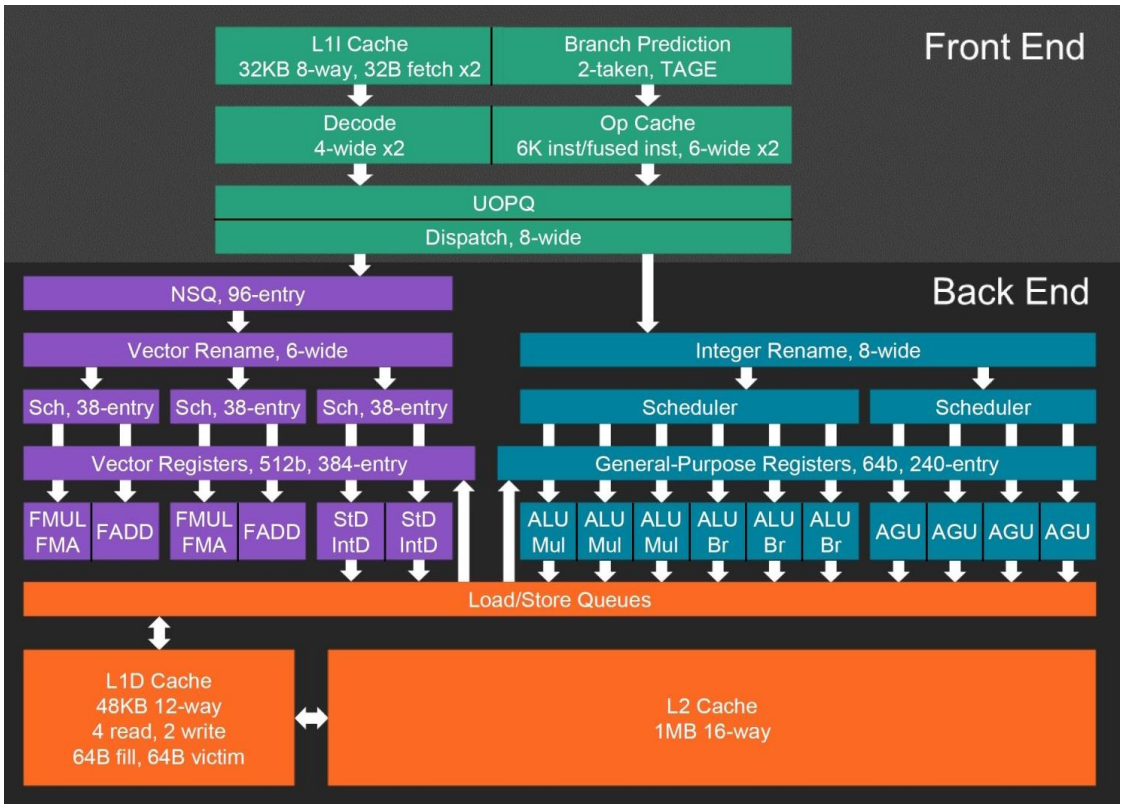

Рисунок 1.3 – Схематичне зображення процесорного ядра архітектури Zen 5 [1]

Хоч зараз і складно вже знайти процесори з ядрами архітектури Zen або Zen 2, особливо не вживані. Однак враховуючи тенденції сьогодення на зниження навантаження центрального процесора, за рахунок перенесення частини обчислень на графічні ядра, моделі із Zen 3 ядрами ще довго можуть лишатись актуальними оскільки для сьогоднішніх задач їх цілком достатньо.

1.1.2 Архітектури центральних процесорів компанії Intel

Схожим образом як і в AMD, серед архітектур процесорних ядер компанії Intel можна виділити сімейство архітектур Cove. Першим її представником є Palm Cove який презентували на заміну Goldmont Plus у 2018 році. Загалом зараз у цьому сімействі налічується вісім архітектур, починаючи з вже згаданого Palm Cove, закінчуючи останнім доступним на цей момент Raptor Cove.

Якщо порівнювати архітектури Palm Cove на рис. 1.4 з архітектурою Zen на рис. 1.1 можна помітити як схожість, особливо підсистем пам’яті, так і значні відмінності як от область безпосередньо операційної частини. У архітектурах від компанії Intel як можемо бачити нема явного розділення на блоки для цілочисельних розрахунків та обчислень значень з плаваючою точкою. Також для генераторів адрес є свій окремий модуль планувальника [4].

В цілому розвиток всередині сімейства цих ядер процесорів, як і в AMD, здебільшого зводиться до збільшення кількості пам’яті та/або каналів, збільшення кількості модулів та їх покращення.

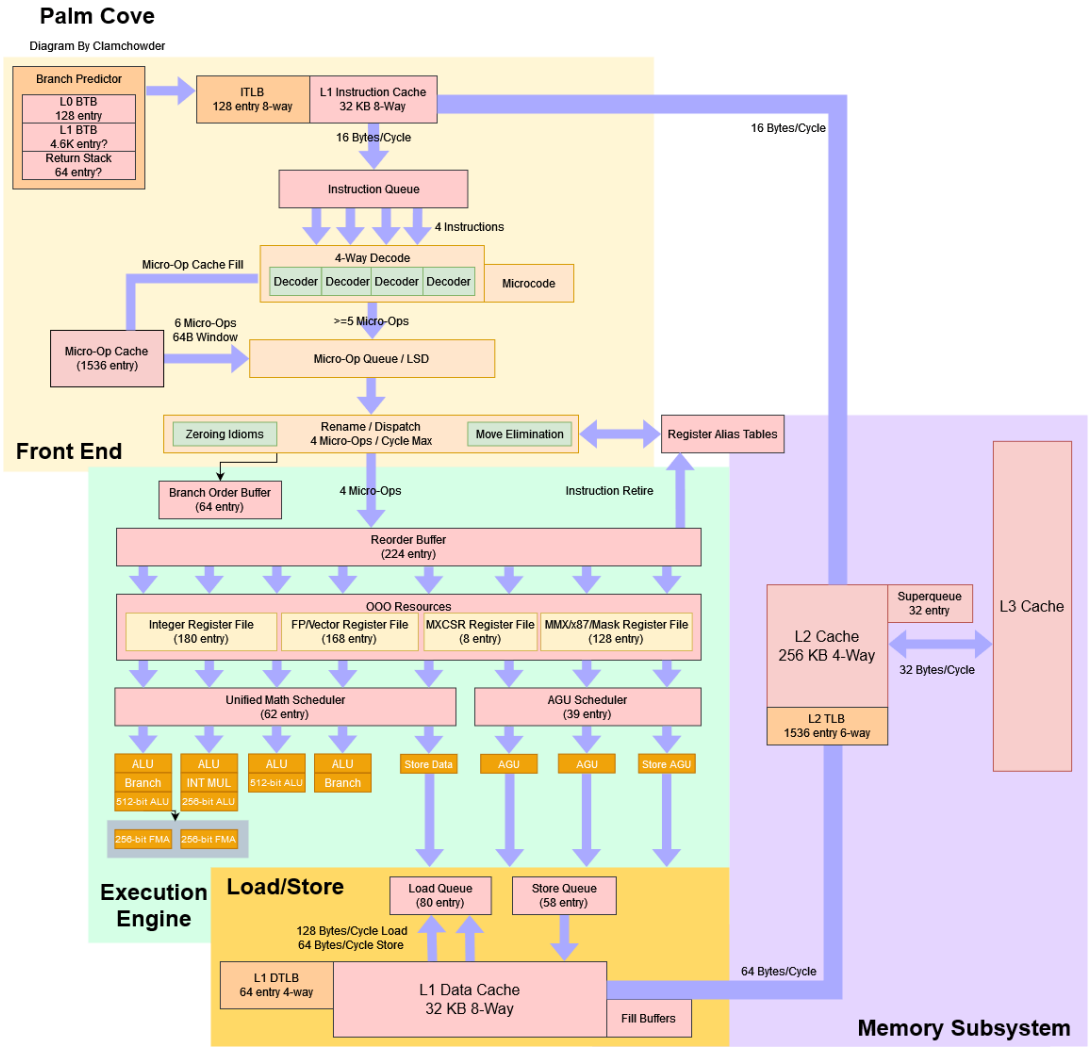

Рисунок 1.4 – Схематичне зображення процесорного ядра архітектури Zen [4]

1.2 Архітектури операційних частин спецобчислювачів

У основні ідеї архітектури операційної частин I типу полягає у досягненні обчислюючим пристроєм максимальної швидкодії при виконанні операцій, не зважаючи на кількість необхідних ресурсів для цього. Відповідно така архітектура буде мати як свої переваги, так і недоліки. Головною перевагою очевидно буде швидкодія пристрою. Вона досягається внаслідок паралельного виконання максимально можливої кількості операцій за такт. Тож, чим більше операцій з алгоритму виконається паралельно, тим менше тактів на виконання операції потрібно буде і відповідно розрахунок завершиться швидше.

Що ж до недоліків, головним недоліком можна назвати ресурсомісткість. Якщо більш конкретно, то це кількість необхідних розрахункових блоків для побудови системи, та об’єм споживаної схемою енергії. Також серед проблем можна відмітити значне тепловиділення через велику швидкодію та енергоспоживання.

Своєю чергою архітектура операційної частин M типу виглядає як інший бік монети. Ідея цієї архітектури полягає у максимально економному використанні ресурсів що відповідно впливає на швидкодію. Зумовлено це тим що подібний пристрій може виконувати лише одну операцію за такт. Тож можемо зазначити це як недолік такої архітектури порівняно з I типом, однак M тип не позбавлений сенсу, оскільки буде найкращим рішенням коли швидкодія не є важливим фактором.

Вище згадані архітектури являють собою дві крайнощі, і відповідно до цього існує ще третій тип, який являє собою компроміс між архітектурами I та M типів. Пристрої архітектури I-M виконують не більше декількох паралельних операцій що дозволяє пришвидшити розрахунки і при цьому не надто збільшувати ресурсомісткість.

1.3 Теоретичне підґрунтя для синтезу цифрових пристроїв

1.3.1 Операційний автомат

Операційний автомат це один з двох основних модулів які мінімально необхідні для синтезу будь-якого цифрового пристрою. До його задач може входити збереження слів інформації, обчислення значень логічних умов, і обов’язково входить виконання певного набору мікрооперацій які в ньому реалізовані. Мікрооперації які виконуються операційним автоматом, задаються множиною керуючих сигналів з кожним з яких ототожнюється визначена мікрооперація. Мікрооперації - це фактично команда для операційного автомата, яка однозначно визначає яку операцію він має виконати у цьому такті. Кожна мікрооперація повинна відповідати конкретній функції, яку автомат може виконувати, інакше можуть виникнути проблеми з некоректними інтерпретуванням операції.

1.3.2 Керуючий автомат

Керуючий автомат призначений для того, щоб генерувати запропоновану мікропрограмою послідовність керуючих сигналів, відповідно до значень логічних умов. Відповідно, керуючий автомат задає порядок дій, які буде виконувати операційний автомат, згідно з закладеним алгоритмом виконання операцій та вхідними значеннями логічних умов на кожному етапі виконання алгоритму. Цей алгоритм визначається розробником залежно від поствленої задачі і частіше за все подається у вигляді граф-схеми. Ця граф-схема складається з вузлів, які позначають собою окремі операції, та ліній, які вказують на те за яких умов і куди відбуваються переходи між станами. Кожен з вузлів може позначати як одну операцію, так і їх групу, або взагалі не містити ніяких операцій.

1.3.3 Компаратор

Цифрові компаратори відомі ще як схеми порівняння, призначені для порівняння між собою щонайменше двох вхідних значень. Вихідним сигналом з

компаратора може бути виключно або логічна “1”, коли умова порівняння була виконана, або логічний “0” у випадках невиконання умови.

1.4 Постановка задачі проєктування

Необхідно спроєктувати спецобчислювач із паралельною операційною частиною. Для цього необхідно сформувати щонайменше два рівняння одне з яких буде обиратись згідно з визначеною умовою у ході роботи пристрою. Ці рівняння повинні бути достатньо великими для того, щоб була можливість проводити паралельні обчислення.

Засновуючись на описах архітектур операційних частин у розділі 1.2 у якості операційної частини цифрового пристрою нам підходить архітектура I типу та I-M типу. Оскільки у I-типу виконується максимальна кількість паралельних операцій оберемо цю архітектуру для операційної частини.

Будь-який керуючий автомат може генерувати будь-яку кількість керуючих команд, тож для поточної задачі нема різниці від того який тип буде обрано. В такому разі оберемо керуючий автомат Мура, як достатньо простий у реалізації та надійний. Для виконання умови перемикання між формулами достатньо буде комбінаційної схеми компаратора.

2 ОСНОВНІ ЕТАПИ ПРОЄКТУВАННЯ ЦИФРОВОГО ПРИСТРОЮ

Спецобчислювач повинен складатись з операційного автомата, спроєктованого за архітектурою I-типу (Individual microoperations), керуючого автомату Мура, та компараторів. При цьому спецобчислювач буде зчитувати два

восьмибітних числа, оброблювати їх згідно з визначеними рівняннями та умовами після чого виведе результат обчислень на шину вихідних даних.

Порядок роботи спецобчислювача має бути наступним:

- записати вхідні дані у внутрішні регістри;

- провести порівняння вхідних значень із константами визначеними умовами;

- залежно від результату порівняння обрати формулу для виконання операційною частиною;

- згідно з алгоритмом виконання формули провести усі необхідні операції над записаними даними;

- вивести результат на шину вихідних даних;

- очистити вміст внутрішніх регістрів від тимчасових даних які там зберігаються.

Для зберігання даних всередині операційного автомата необхідно визначити які елементи пам’яті будемо використовувати.

Виходячи з того, що згідно з завданням пристрій має оброблювати восьмибітні числа цілком достатньо буде використати восьмибітні регістри засновані на D-тригерах. Оскільки, згідно з завданням, після завершення всіх операцій треба очистити вміст регістрів при їх проєктуванні необхідно врахувати можливість скидання поточних значень цих регістрів. Відповідно такий сигнал має бути і у кожного D-тригера.

2.1 Розробка операційного автомата

Згідно з завданням операційна частина обчислювача повинна мати архітектуру І-типу і відповідно виконувати частину операцій паралельно. Оскільки задача не надто велика нема потреби розміщувати елементи пам’яті в окремий модуль, і відповідно в операційний автомат має бути вбудовано

щонайменше два восьмибітних регістри. Для визначення функціональних блоків, які безпосередньо будуть проводити обчислення, необхідно розібрати рівняння на атомарні операції та скласти з них алгоритм дій за яким буде працювати операційний автомат.

2.1.1 Формування рівнянь та розробка алгоритму їх обчислення

Сформуємо рівняння за якими спецобчислювач буде виконувати розрахунки. Рівняння мають бути достатньо великими, щоб була можливість розпаралелити частину обчислення.

Використаємо для проєктування наступні два рівняння:

$$S = \begin{cases} 9(A \oplus B) + \left(4A + \frac{B}{8}\right) \oplus \overline{(5B + A)} \\ \frac{A+B}{8} + \overline{(4A \; and \; 2B)} \; or \; (A \oplus B) \end{cases}, \quad (2.1)$$

Умовою для визначення рівняння по якому буде виконуватись обчислення поставимо А більше 20-ти та В менше 26-ти, при виконанні умов буде виконуватись перше рівняння. Такі умови трохи знизять шанс того що результат обчислень вийде за межі розрядної сітки.

Отже, маючи тепер рівняння можемо утворити на їх основі алгоритм за яким вони будуть обчислюватись. Для цього розберемо кожну формулу на атомарні операції, тобто в нашому випадку такі які можна виконати однією мікрооперацією за один такт і запишемо рівняння у такому вигляді.

$$S = \begin{cases} (A \oplus B) + 8(A \oplus B) + \left(4A + \frac{B}{8}\right) \oplus \overline{(4B + A + B)} \\ \frac{A+B}{8} + \overline{(4A \; and \; 2B)} \; or \; (A \oplus B) \end{cases}, \quad (2.2)$$

Враховуючи те, що при множенні або діленні на число кратне ступеням двійки ми можемо виконати звичайний зсув на певну кількість біт в ліво або право, залежно від операції, ми зможемо спростити проєктування модулів для

виконання цих операцій. Для того, щоб скористатись цим у першому рівнянні деякі операції множення було розділено на множення та складання змінних, як наприклад 5B == B + 4B. Такий запис дозволить спочатку скористатись зсувами, а потім додаванням об’єднати результат.

Визначившись з операціями, які будуть виконуватись, можемо тепер оцінити приблизну кількість необхідних регістрів які будуть зберігати як вхідні значення змінних, так і регістри для проміжних результатів. Враховуючи що вхідних значень два ми матимемо два “Input” регістри. Назвемо їх відповідно до змінних які в них будуть зберігатись, тобто будуть регістри A та B. Однак, ці регістри не обов’язково мають зберігати лиш вхідні дані, за потреби можемо використати їх для зберігання проміжних даних і відповідно вони вже будуть “Input - Local” регістрами. Враховуючи кількість операцій які необхідно виконати для початку додамо два регістри “Local” для проміжних даних. Подібні регістри будуть мати назву S з номером, відповідно буде S_1 та S_2.

Тепер маючи атомарні операції у рівняннях та знаючи кількість регістрів можемо утворити з урахуванням цього алгоритм у вигляді граф-схеми. Цей алгоритм буде відображенням послідовності операцій для обчислення рівнянь і його ж у подальшому можна буде використати як основу для алгоритму роботи спецобчислювача загалом. На початку алгоритму очевидно необхідно розмістити операцію запису вхідних даних у регістри A та B для того, щоб мати можливість використовувати їх в обчисленнях.

Граф-схема алгоритму відображена на рис. 2.1

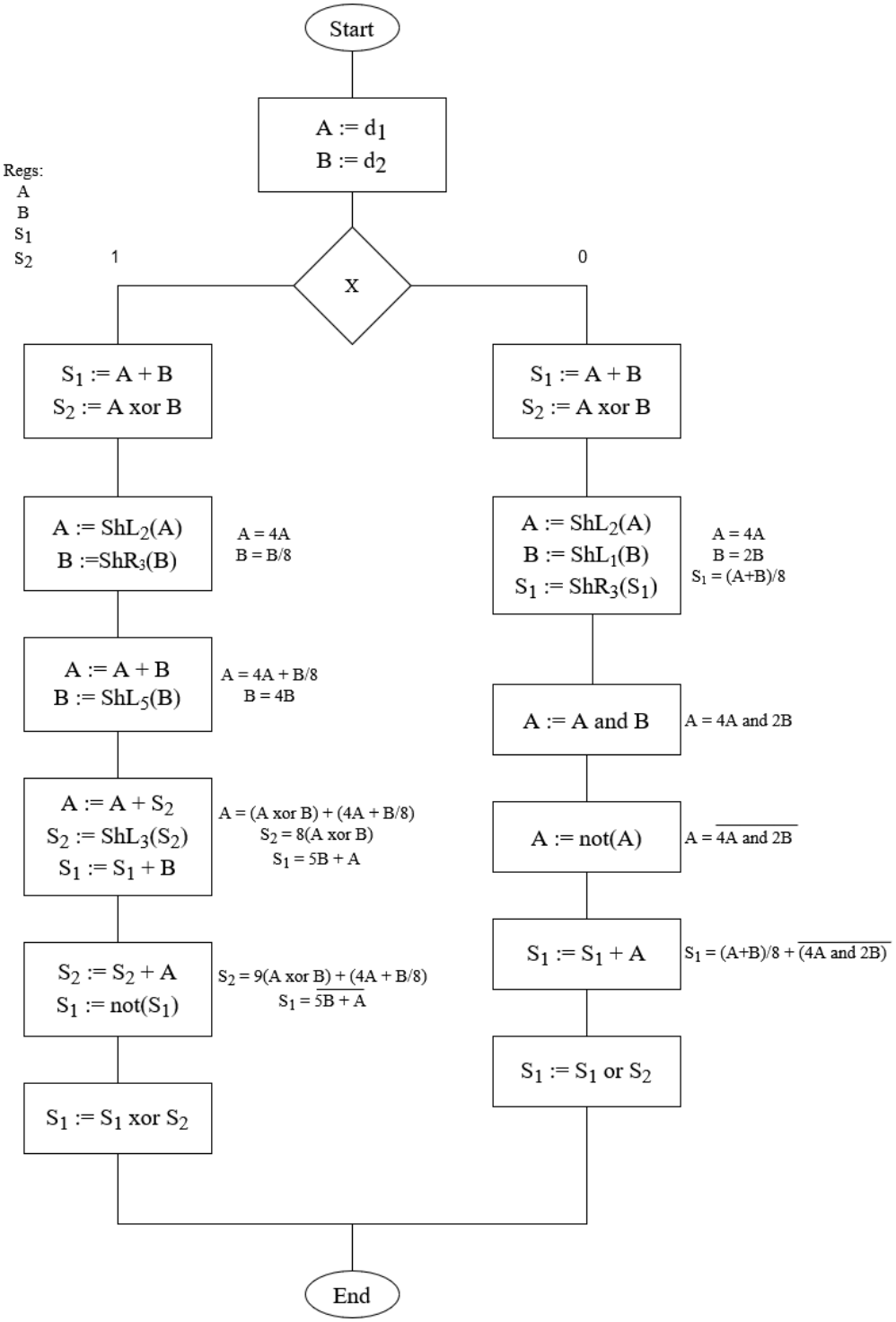

Рисунок 2.1 – Граф-схема алгоритму обчислення рівнянь

На рис. 2.1 можемо побачити два цікаві моменти. Перший, на початку обчислень обох рівнянь стоять дві ідентичні операторні вершини, відповідно ці вершини можемо винести перед оператором логічної умови. Друге, через нестачу регістрів, для забезпечення коректності, довелось спочатку змінну B поділити на вісім, згідно з рівнянням, і після використання цього значення

помножити його на 32, тобто зсунути число на п’ять біт, щоб отримати 4B яке необхідне для подальших розрахунків. Відповідно було прийнято рішення додати ще один додатковий регістр у другому варіанті алгоритму.

На рис. 2.2 відображено оптимізований варіант алгоритму в якому вже було враховано два попередні зауваження. Після декількох спроб подальшої оптимізації алгоритму покращень досягти не вдалось, тож можемо вважати другий варіант фінальним.

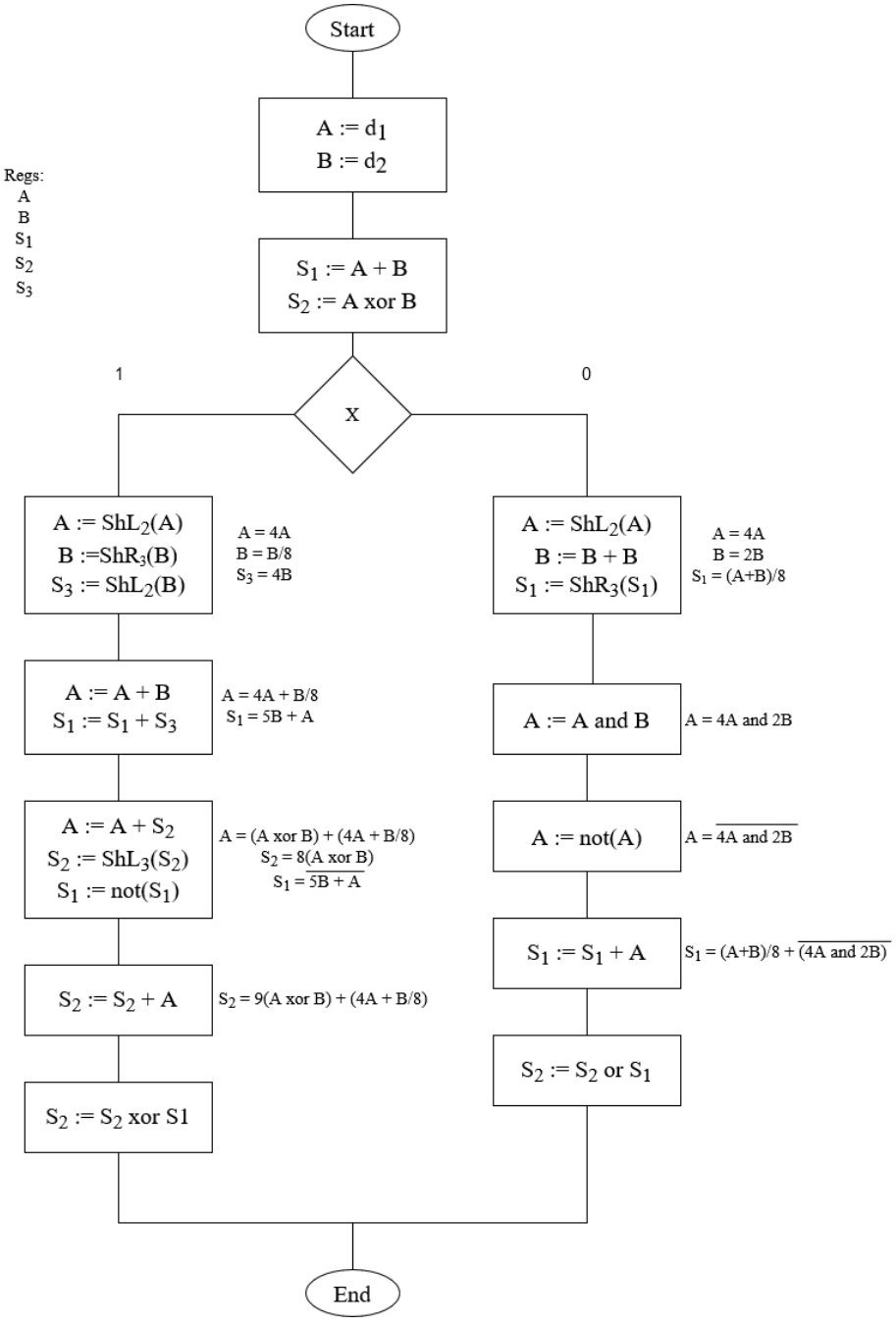

Рисунок 2.2 – Граф-схема оптимізованого алгоритму обчислення рівнянь

Отже, доповнимо алгоритм з рис. 2.2 згідно із задачами поставленими безпосередньо до спецобчислювача. Граф-схема цього алгоритму відображена на рис. 2.3

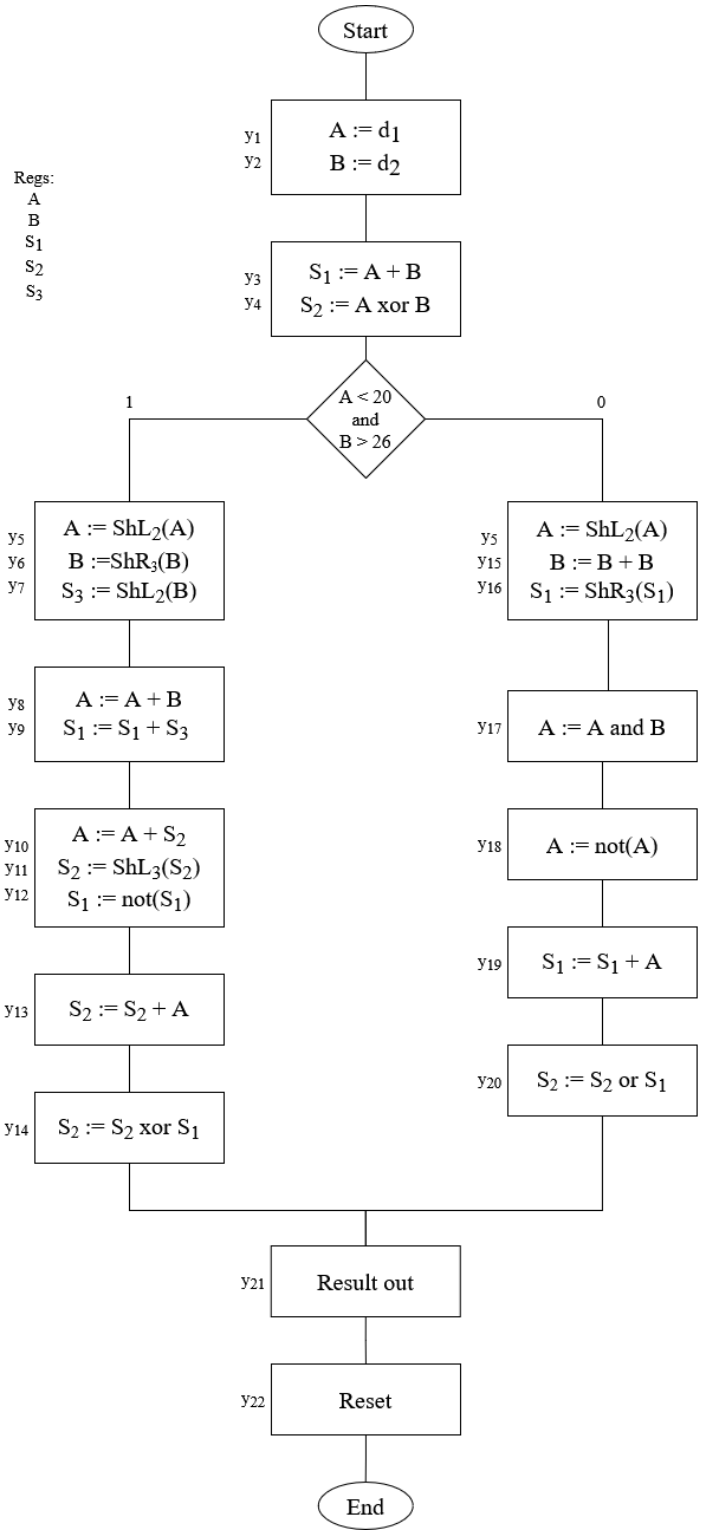

Рисунок 2.3 – Граф-схема алгоритму роботи спецобчислювача з позначенням мікрокоманд

2.1.2 Формування таблиці операторів та оптимізація операцій

Тепер маючи алгоритм роботи спецобчислювача для зручності подальшої оптимізації та в цілому розуміння кількості й складності операторів запишемо їх усі у табл. 2.1 згрупувавши їх по регістрах у які відбувається запис результату обчислень по горизонталі, та типом оператора по вертикалі.

Таблиця 2.1 – Таблиці операторів

Регістр	Add	Xor	And	Not	Or	ShL_2	ShR_3	ShL_3
A	A+B		A and B	Not(A)		ShL_2(A)		
	A+S_2							
B	B + B						ShR_3(B)	
S_1	A + B			Not(S_1)			ShR_3(S_1)	
	S_1 + S_3							
	S_1 + A							
S_2	S_2 + A	A xor B			S_2 or S_1			ShL_3(S_2)
		S_2 xor S_1						
S_3						ShL_2(B)		

Тепер маючи таблицю операторів нам буде простіше провести оптимізацію. Вона полягає в узагальненні операторів які мають однаковий тип і запис результату йде в один і той самий регістр. Це дозволить нам зменшити кількість необхідних функціональних блоків.

Як приклад, операції Add із записом у регістр S_1. Таких операторів три й відповідно до того як проєктуються операційні автомати типу I ми б мали використати три суматори. Але якщо ці операції узагальнити нам потрібно буде вже тільки один суматор. Однак, потрібна буде додаткова схема для контролю того з якого регістра потрібно зчитувати інформацію, це можна реалізувати за допомогою мультиплексора.

Розглянемо усі оператори, які можна узагальнити. Кожен оператор розпишемо з урахуванням його мікрокоманди. Це зумовлено тим, що в такому випадку керуючий сигнал для мікрокоманди можна використати як сигнал

логічної умови для мультиплексора, що дозволить йому пропускати на вхід функціонального блоку дані з потрібних регістрів.

Почнемо узагальнення з операторів результат яких записується у регістр A. Таких операторів буде два: “A + B” з керуючим сигналом y_8, і “A + S_2”з керуючим сигналом y_{10}. Позначення керуючих сигналів важливе, оскільки за рахунок них у подальшому буде визначатись те, які дані надійдуть у функціональний блок. Таким чином узагальнення для цих двох операторів суми буде мати вигляд $y_{8,10}(A) = O_1 + O_2$, де O_i це операнд поточної операції. У двох операторів які наразі розглядаємо відрізняються тільки другий операнд. Відповідно ми можемо прив’язати значення операнду O_2 до керуючих сигналів y_8 та y_{10}. Коли сигнал y_8 буде активним має передаватись значення лінії B, коли активний y_{10} відповідно має передаватись значення з лінії S_2. Таким чином формула для прорахунку значення другого операнду матиме наступний вигляд:

$$O_2 = (y_8 \text{ and } B) \text{ or } (y_{10} \text{ and } S_2), \tag{2.3}$$

Наступним проведемо узагальнення для операторів які записують результат у регістр S_1. Таких операторів буде три: “B + A” при сигналі y_3, “$S_1 + S_3$” при сигналі y_9, “S_1 + A” при сигналі y_{19}. Узагальнення для цих операторів суми буде мати вигляд $y_{3,9,19}(S_1) = O_1 + O_2$. У кожного оператора між собою відрізняється як мінімум один операнд, отже необхідно описати формули для кожного узагальненого операнда. Формула для узагальнених операндів матиме наступний вигляд:

$$\begin{aligned} O_1 &= (y_3 \text{ and } B) \text{ or } ((y_9 \text{ or } y_{19}) \text{ and } S_1), \\ O_2 &= ((y_3 \text{ or } y_{19}) \text{ and } A) \text{ or } (y_9 \text{ and } S_3), \end{aligned} \tag{2.4}$$

Залишилось зробити опис для операторів які записують результат у регістр S_2. Таких операторів буде два: “A xor B” при сигналі y_4, “S_2 xor S_1” при сигналі y_{14}. Узагальнення для цих операторів суми буде мати вигляд $y_{4,14}(S_2) = O_2 \text{ xor } O_1$.

У обох операторів всі операнди відрізняються, тож необхідно описати формули для кожного узагальненого операнда. Формула для узагальнених операндів матиме наступний вигляд:

$$O_1 = (y_4 \text{ and } A) \text{ or } (y_{14} \text{ and } S_2),$$
$$O_2 = (y_4 \text{ and } B) \text{ or } (y_{14} \text{ and } S_1), \quad (2.5)$$

Тепер маючи інформацію про кількість регістрів та усі оператори які необхідно реалізувати на схемі операційного автомата можемо утворити його схематичне зображення.

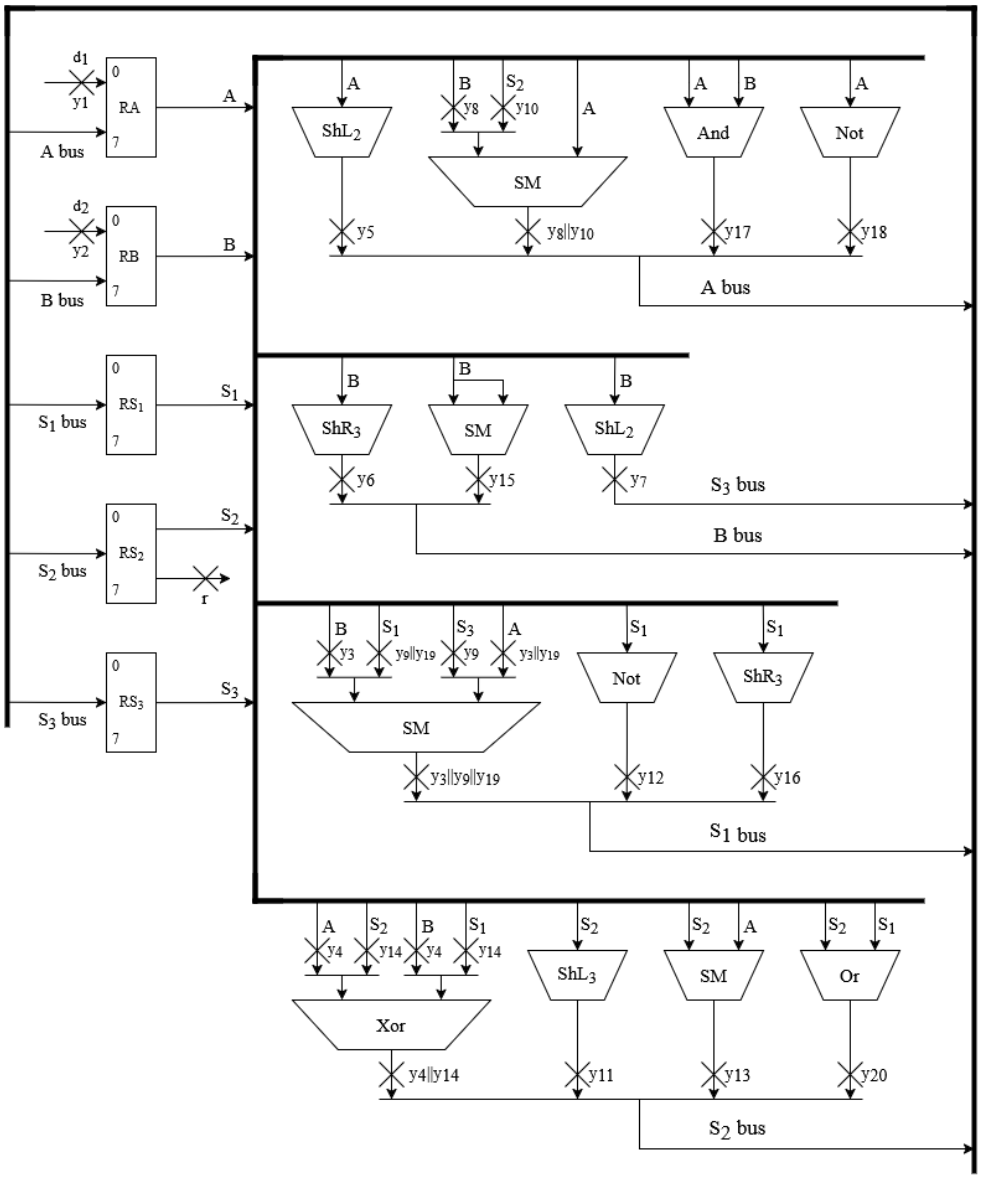

Рисунок 2.4 – Схематичне зображення будови операційного автомата I типу

2.2 Розробка керуючого автомата

Задачу керуючого автомату виконуватиме автомат Мура. Його функціоналу буде цілком достатньо для того, щоб керувати операційною частиною розробленою у попередньому розділі.

2.2.1 Формування та розмічення граф-схеми керуючого автомату

Для початку розробки керуючого автомата буде необхідно сформувати функціональну граф-схему його алгоритму. Ця схема дозволить візуально представити послідовність операторів алгоритму та зрозуміти логіку його роботи.

Як вже згадувалось раніше для побудови граф-схеми керуючого автомата можемо як основою скористатись граф-схемою алгоритму спецобчислювача відображену на рис. 2.3. В результаті ми можемо одразу перейти до етапу розмітки станів у яких може знаходитись керуючий автомат для того, щоб у подальшому формувати таблицю переходів.

Для будь-якого керуючого автомата початок і кінець алгоритму завжди позначаються станом з індексом 0. Враховуючи те що в нас автомат Мура то кожну операторну вершину позначаємо станом S_i.

Додатково для спрощення подальшого формування таблиці переходів і простоти читання граф-схеми замінимо мікрокоманди на відповідні до них керуючі сигнали y_i, і також позначимо умову переходу як x.

Таким чином, після виконання розмітки граф-схеми керуючого автомата отримаємо схему відображену на рис. 2.5.

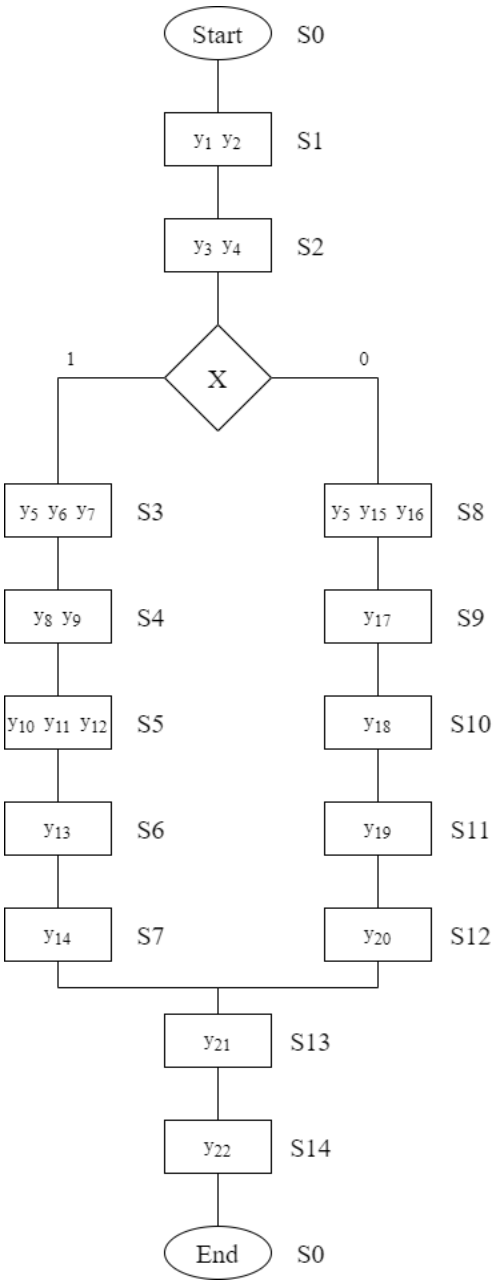

Рисунок 2.5 – Граф-схема алгоритму з розміченими станами автомату Мілі

2.2.3 Побудова графа переходів керуючого автомата

Відповідно до розміченої до цього граф-схеми керуючого автомата, яка відображена на рис. 2.5, побудуємо граф переходів цього керуючого автомату.

До кожної вершини графу ставимо у відповідність стан автомата S_i. Для відображення переходів між станами проставимо стрілки між станами. Обов’язково необхідно враховувати напрямок стрілки, бо саме це вказує на те з якого стану до якого відбувається перехід. Також на лініях цих стрілок за наявності вказується умова переходу та функція яка виконується при переході.

Результуючий вигляд графу переходів відображено на рис. 2.6.

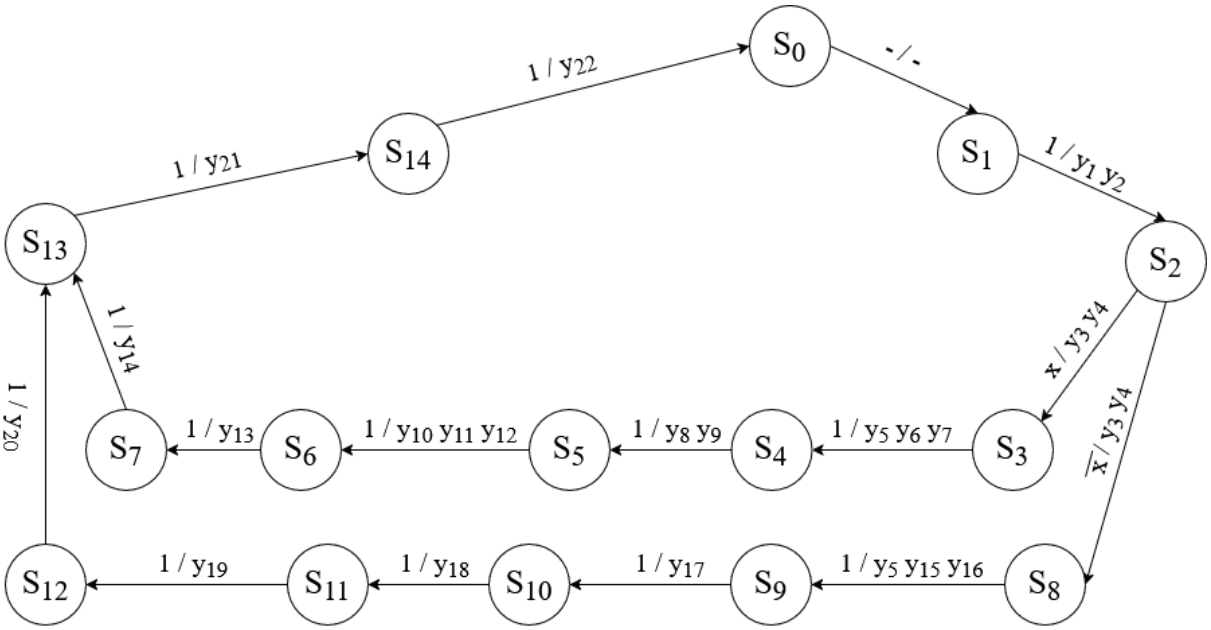

Рисунок 2.6 – Граф переходів керуючого автомата

2.2.4 Визначення кількості елементів пам’яті та кодування станів керуючого автомату

Для подальшого проєктування необхідно провести кодування станів керуючого автомату. Це означає що кожному стану автомата необхідно поставити у відповідність кодову комбінацію. Однак спершу треба визначити тип коду та його розрядність.

Враховуючи кількість станів автомата двійкового коду для їх кодування буде цілком достатньо. Розрядність числа такого коду можемо визначити за допомогою наступної формули:

$$n = \langle log_2 S \rangle, \tag{2.6}$$

де n – кількість розрядів числа;

S - число станів автомата;

⟨ ⟩ - округлення в більшу сторону.

Відповідно, оскільки автомат має 15 різних станів, підставимо це значення у формулу та отримаємо наступний результат:

$$n = \langle log_2 15 \rangle = 4, \tag{2.7}$$

Враховуючи результат розрахунку, для кодування станів нам буде достатньо числі з чотирма розрядами. В такому разі чотирьох елементів пам’яті буде цілком достатньо.

Отже, знаючи розрядність і вид коду, можемо записати закодовані комбінації усіх станів автомата у табл. 2.2.

Таблиця 2.2 – Кодові комбінації станів автомата

State	Code			
	Q_3	Q_2	Q_1	Q_0
S0	0	0	0	0
S1	0	0	0	1
S2	0	0	1	0
S3	0	0	1	1
S4	0	1	0	0
S5	0	1	0	1
S6	0	1	1	0
S7	0	1	1	1
S8	1	0	0	0
S9	1	0	0	1
S10	1	0	1	0
S11	1	0	1	1
S12	1	1	0	0
S13	1	1	0	1

2.2.5 Побудова таблиці переходів та відповідних їм мікрокоманд

Тепер, маючи граф-схему керуючого автомата та коди його станів ми можемо сформувати таблицю переходів. Вона буде відображати перетворення кодів при переході між станами, при якій умові відбувається перехід та які при цьому мають формуватися мікрокоманди. Результати результат відображено у табл. 2.3.

Таблиця 2.3 – Таблиця переходів автомата

Q_t	Код поточного стану S_i				Q_{t+1}	Код наступного стану S_j				Умова переходу	Code D_i				Мікро-команди $y(S_i)$
	Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0		D_3	D_2	D_1	D_0	
S_0	0	0	0	0	S_1	0	0	0	1	1	0	0	0	1	-
S_1	0	0	0	1	S_2	0	0	1	0	1	0	0	1	0	y_1 y_2
S_2	0	0	1	0	S_3	0	0	1	1	x	0	0	1	1	y_3 y_4
S_2	0	0	1	0	S_8	1	0	0	0	$\bar{x}$	1	0	0	0	
S_3	0	0	1	1	S_4	0	1	0	0	1	0	1	0	0	y_5 y_6 y_7
S_4	0	1	0	0	S_5	0	1	0	1	1	0	1	0	1	y_8 y_9
S_5	0	1	0	1	S_6	0	1	1	0	1	0	1	1	0	y_{10} y_{11} y_{12}
S_6	0	1	1	0	S_7	0	1	1	1	1	0	1	1	1	y_{13}
S_8	1	0	0	0	S_9	1	0	0	1	1	1	0	0	1	y_5 y_{15} y_{16}
S_9	1	0	0	1	S_{10}	1	0	1	0	1	1	0	1	0	y_{17}
S_{10}	1	0	1	0	S_{11}	1	0	1	1	1	1	0	1	1	y_{18}
S_{11}	1	0	1	1	S_{12}	1	1	0	0	1	1	1	0	0	y_{19}
S_7	0	1	1	1	S_{13}	1	1	0	1	1	1	1	0	1	y_{14}
S_{12}	1	1	0	0	S_{13}	1	1	0	1	1	1	1	0	1	y_{20}
S_{13}	1	1	0	1	S_{14}	1	1	1	0	1	1	1	1	0	y_{21}
S_{14}	1	1	1	0	S_0	0	0	0	0	1	0	0	0	0	y_{22}

2.2.6 Формування та мінімізація функцій збудження елементів пам’яті й функцій виходів

Маючи таблицю переходів можемо тепер сформувати системи рівнянь для збудження тригерів та формування мікрокоманд. Рівняння збудження тригерів дозволять нам власне спроєктувати автомат який буде циклічно змінювати свої стани. Своєю черго система рівнянь формування мікрокоманд, як випливає з назви, дозволить керуючому автомату формувати мікрокоманди для управління операційним автоматом.

Сформована, але ще не мінімізована система рівнянь збудження тригерів відображені у формулі 2.8.

$$\begin{cases} D_0 = \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} \vee \overline{Q_3}\,\overline{Q_2}\,Q_1\,\overline{Q_0}\,x \vee \overline{Q_3}\,Q_2\,\overline{Q_1}\,\overline{Q_0} \vee \overline{Q_3}\,Q_2\,Q_1\,\overline{Q_0} \vee \\ \quad \vee Q_3\,\overline{Q_2}\,Q_1\,\overline{Q_0} \vee \overline{Q_3}\,Q_2\,Q_1\,Q_0 \vee Q_3\,Q_2\,\overline{Q_1}\,\overline{Q_0} \vee Q_3\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} \\ D_1 = \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,Q_0 \vee \overline{Q_3}\,\overline{Q_2}\,Q_1\,\overline{Q_0}\,x \vee \overline{Q_3}\,Q_2\,\overline{Q_1}\,Q_0 \vee \overline{Q_3}\,Q_2\,Q_1\,\overline{Q_0} \vee \\ \quad \vee Q_3\,\overline{Q_2}\,\overline{Q_1}\,Q_0 \vee Q_3\,\overline{Q_2}\,Q_1\,\overline{Q_0} \vee Q_3\,Q_2\,\overline{Q_1}\,Q_0 \\ D_2 = \overline{Q_3}\,\overline{Q_2}\,Q_1\,Q_0 \vee \overline{Q_3}\,Q_2\,\overline{Q_1}\,\overline{Q_0} \vee \overline{Q_3}\,Q_2\,\overline{Q_1}\,Q_0 \vee \overline{Q_3}\,Q_2\,Q_1\,\overline{Q_0} \vee \\ \quad \vee Q_3\,\overline{Q_2}\,Q_1\,Q_0 \vee \overline{Q_3}\,Q_2\,Q_1\,Q_0 \vee Q_3\,Q_2\,\overline{Q_1}\,\overline{Q_0} \vee Q_3\,Q_2\,\overline{Q_1}\,Q_0 \\ D_3 = \overline{Q_3}\,\overline{Q_2}\,Q_1\,\overline{Q_0}\,\bar{x} \vee Q_3\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} \vee Q_3\,\overline{Q_2}\,\overline{Q_1}\,Q_0 \vee Q_3\,\overline{Q_2}\,Q_1\,\overline{Q_0} \\ \quad \vee Q_3\,\overline{Q_2}\,Q_1\,Q_0 \vee \overline{Q_3}\,Q_2\,Q_1\,Q_0 \vee Q_3\,Q_2\,\overline{Q_1}\,\overline{Q_0} \vee Q_3\,Q_2\,\overline{Q_1}\,Q_0 \end{cases}, \quad (2.8)$$

Проведемо мінімізацію цих рівнянь за допомогою карт Карно. Результат мінімізації записано у формулі 2.9. Карти Карно відображено на рис. 2.7-2.10. На картах Карно є одиниці які відмічені червоним кольором який позначає що у цих термах окрім комбінації Q_i-х також присутній х, відповідно ці терми не можна було використовувати для скорочень за правилами карт Карно.

$$\begin{cases} D_0 = \overline{Q_1}\,\overline{Q_0} \vee \overline{Q_3}\,Q_2\,Q_1 \vee Q_3\,\overline{Q_2}\,\overline{Q_0} \vee \overline{Q_3}\,Q_2\,\overline{Q_0} \vee \overline{Q_3}\,\overline{Q_2}\,Q_1\,\overline{Q_0}\,x \\ D_1 = \overline{Q_1}\,Q_0 \vee \overline{Q_3}\,Q_2\,Q_1\,\overline{Q_0} \vee Q_3\,\overline{Q_2}\,Q_1\,\overline{Q_0} \vee \overline{Q_3}\,\overline{Q_2}\,Q_1\,\overline{Q_0}\,x \\ D_2 = \overline{Q_3}\,Q_2 \vee Q_2\,\overline{Q_1} \vee \overline{Q_3}\,Q_1\,Q_0 \vee \overline{Q_2}\,Q_1\,Q_0 \\ D_3 = Q_3\,\overline{Q_2} \vee Q_3\,\overline{Q_1} \vee \overline{Q_3}\,Q_2\,Q_1\,Q_0 \vee \overline{Q_3}\,\overline{Q_2}\,Q_1\,\overline{Q_0}\,\bar{x} \end{cases}, \quad (2.9)$$

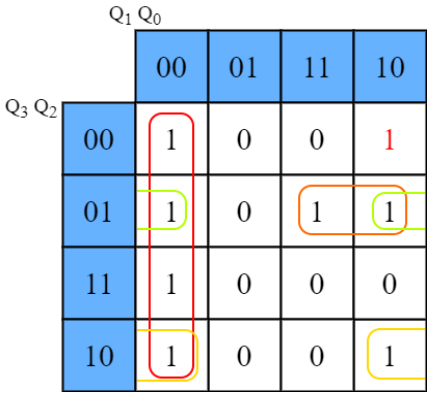

Рисунок 2.7 – Карта Карно для рівняння D_0

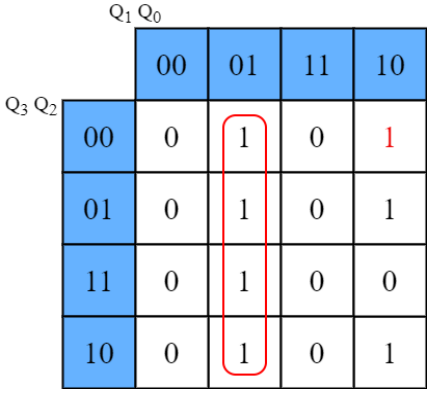

Рисунок 2.8 – Карта Карно для рівняння D_1

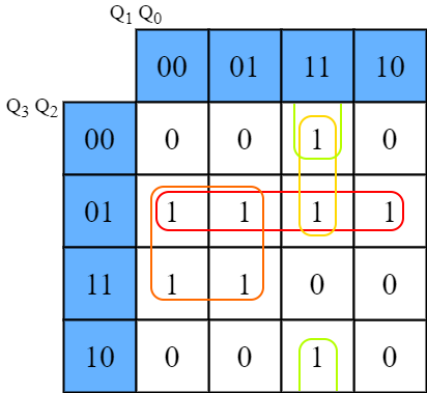

Рисунок 2.9 – Карта Карно для рівняння D_2

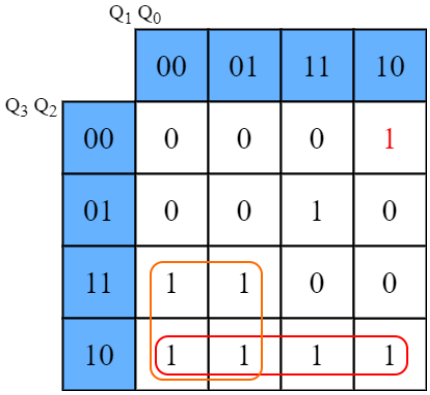

Рисунок 2.10 – Карта Карно для рівняння D_3

Тепер так само сформуємо систему рівнянь формування мікрокоманд, яка буде відображена у формулі 2.10.

$$\begin{cases} y_1 = \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,Q_0 \\ y_2 = \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,Q_0 \\ y_3 = \overline{Q_3}\,\overline{Q_2}\,Q_1\,\overline{Q_0} \\ y_4 = \overline{Q_3}\,\overline{Q_2}\,Q_1\,\overline{Q_0} \\ y_5 = \overline{Q_3}\,\overline{Q_2}\,Q_1\,Q_0 \text{ v } Q_3\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} \\ y_6 = \overline{Q_3}\,\overline{Q_2}\,Q_1\,Q_0 \\ y_7 = \overline{Q_3}\,\overline{Q_2}\,Q_1\,Q_0 \\ y_8 = \overline{Q_3}\,Q_2\,\overline{Q_1}\,\overline{Q_0} \\ y_9 = \overline{Q_3}\,Q_2\,\overline{Q_1}\,\overline{Q_0} \\ y_{10} = \overline{Q_3}\,Q_2\,\overline{Q_1}\,Q_0 \\ y_{11} = \overline{Q_3}\,Q_2\,\overline{Q_1}\,Q_0 \\ y_{12} = \overline{Q_3}\,Q_2\,\overline{Q_1}\,Q_0 \\ y_{13} = \overline{Q_3}\,Q_2\,Q_1\,\overline{Q_0} \\ y_{14} = \overline{Q_3}\,Q_2\,Q_1\,Q_0 \\ y_{15} = Q_3\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} \\ y_{16} = Q_3\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} \\ y_{17} = Q_3\,\overline{Q_2}\,\overline{Q_1}\,Q_0 \\ y_{18} = Q_3\,\overline{Q_2}\,Q_1\,\overline{Q_0} \\ y_{19} = Q_3\,\overline{Q_2}\,Q_1\,Q_0 \\ y_{20} = Q_3\,Q_2\,\overline{Q_1}\,\overline{Q_0} \\ y_{21} = Q_3\,Q_2\,\overline{Q_1}\,Q_0 \\ y_{22} = Q_3\,Q_2\,Q_1\,\overline{Q_0} \end{cases}, \quad (2.10)$$

Як можемо побачити здебільшого усі рівняння складаються з одного терму, відповідно і мінімізовувати в цих рівняннях нічого.

2.2.7 Визначення термів

Наступним кроком із сформованих систем рівнянь визначимо список термів керуючого автомата. Це дозволить виключити вірогідність повторення одних і тих самих термів у різних рівняннях. А це своєю чергою дозволить більш

економно і раціонально побудувати схему керуючого автомату. Список унікальних термів занесемо у табл. 2.4.

Таблиця 2.4 – Список термів керуючого автомата

Терм	Кодування терму	Терм	Кодування терму	Терм	Кодування терму
T_0	$\overline{Q_1}\ \overline{Q_0}$	T_{10}	$\overline{Q_3}\ Q_1\ Q_0$	T_{20}	$\overline{Q_3}\ Q_2\ \overline{Q_1}\ \overline{Q_0}$
T_1	$\overline{Q_3}\ Q_2\ Q_1$	T_{11}	$\overline{Q_2}\ Q_1\ Q_0$	T_{21}	$\overline{Q_3}\ Q_2\ \overline{Q_1}\ Q_0$
T_2	$Q_3\ \overline{Q_2}\ \overline{Q_0}$	T_{12}	$Q_3\ \overline{Q_2}$	T_{22}	$Q_3\ \overline{Q_2}\ \overline{Q_1}\ Q_0$
T_3	$\overline{Q_3}\ Q_2\ \overline{Q_0}$	T_{13}	$Q_3\ \overline{Q_1}$	T_{23}	$Q_3\ \overline{Q_2}\ Q_1\ Q_0$
T_4	$\overline{Q_3}\ \overline{Q_2}\ Q_1\ \overline{Q_0}$ x	T_{14}	$\overline{Q_3}\ Q_2\ Q_1\ Q_0$	T_{24}	$Q_3\ Q_2\ \overline{Q_1}\ \overline{Q_0}$
T_5	$\overline{Q_1}\ Q_0$	T_{15}	$\overline{Q_3}\ \overline{Q_2}\ Q_1\ \overline{Q_0}\ \bar{x}$	T_{25}	$Q_3\ Q_2\ \overline{Q_1}\ Q_0$
T_6	$\overline{Q_3}\ Q_2\ Q_1\ \overline{Q_0}$	T_{16}	$\overline{Q_3}\ \overline{Q_2}\ \overline{Q_1}\ Q_0$	T_{26}	$Q_3\ Q_2\ Q_1\ \overline{Q_0}$
T_7	$Q_3\ \overline{Q_2}\ Q_1\ \overline{Q_0}$	T_{17}	$\overline{Q_3}\ \overline{Q_2}\ Q_1\ \overline{Q_0}$		
T_8	$\overline{Q_3}\ Q_2$	T_{18}	$\overline{Q_3}\ \overline{Q_2}\ Q_1\ Q_0$		
T_9	$Q_2\ \overline{Q_1}$	T_{19}	$Q_3\ \overline{Q_2}\ \overline{Q_1}\ \overline{Q_0}$		

Отже, з урахуванням термів рівняння збудження тригерів матимуть наступний вигляд:

$$\begin{cases} D_0 = T_0\ \text{v}\ T_1\ \text{v}\ T_2\ \text{v}\ T_3\ \text{v}\ T_4 \\ D_1 = T_5\ \text{v}\ T_6\ \text{v}\ T_7\ \text{v}\ T_4 \\ D_2 = T_8\ \text{v}\ T_9\ \text{v}\ T_{10}\ \text{v}\ T_{11} \\ D_3 = T_{12}\ \text{v}\ T_{13}\ \text{v}\ T_{14}\ \text{v}\ T_{15} \end{cases}, \tag{2.11}$$

Майже усі рівняння які відповідають за формування мікрокоманд складаються з одного терму, тож особливого сенсу у тому щоб так само розписувати і їх нема.

2.3 Розробка компараторів

Враховуючи те, що у розділі 2.1.1 було обрано здвоєну логічну умову, ми можемо реалізувати це у вигляді двох компараторів об’єднаних в один. Тобто в такому випадку виходи двох компараторів будуть йти у оператор And, оскільки для виконання поставленої умови обидва компаратори мають видати на вихід істину. Самі ж компаратори всередині схеми будуть працювати незалежно один від одного і порівнювати кожен своє вхідне число зі своєю константою.

Отже, кожен компаратор буде генерувати результат порівняння вхідного числа з константою, після чого ці два сигнали будуть проходити через оператор And і вже цей сигнал буде передано керуючому автомату я якості сигналу умови. Щоб не заплутатись при проєктуванні назвемо сигнали результатів компараторів cmp_1 та cmp_2. Тоді умови цих компараторів можемо записати таким чином:

- для того, щоб умова cmp_1 дала результат «істина» необхідно реалізувати нерівність $A < 20_{10}$ (00010100_2);
- для того, щоб умова cmp_2 дала результат “істина” необхідно реалізувати нерівність $B > 26_{10}$ (00011010_2).

2.3.1 Визначення базових функцій

Першим кроком визначимо базові функції і-тих розрядів чисел обох компараторів. Для цього спочатку складемо для кожного з них ТІ. Ці ТІ будуть відображені у табл. 2.5 та табл. 2.6.

Таблиця 2.5 – Таблиця істинності для компаратора cmp_1

a_i	b_i	l_i
0	0	0
0	1	1
1	0	0
1	1	0

Проаналізувавши ТІ зображену у табл. 2.5 сформуємо базову функцію для цього компаратора cmp_1. Вона матиме наступний вигляд: $l_i = b_i * not(a_i)$.

Таблиця 2.6 – Таблиця істинності для компаратора cmp_2

a_i	b_i	m_i
0	0	0
0	1	0
1	0	1
1	1	0

Проаналізувавши ТІ зображену у табл. 2.6 сформуємо базову функцію для цього компаратора cmp_2. Вона матиме наступний вигляд: $m_i = a_i * not(b_i)$.

2.3.2 Формування повних функцій компараторів

Наступним кроком сформуємо тепер вже повні функції для компараторів за допомогою яких надалі буде відбуватись їх моделювання. У загальному вигляді формула для формування повної функції записується таким чином:

$$Y = y_i \vee (R_i * y_{i-1}) \vee (R_i * R_{i-1} * y_{i-2}) \vee \ldots \vee (R_i * \ldots * R_1 * y_0), \qquad (2.12)$$

де y_i – базова функція;

R_i – функція рівнозначності, яка визначається за формулою $R_i = a_i * b_i \vee not(a_i) * not(b_i)$.

Для компаратора cmp_1 повна функція буде мати наступний вигляд:

$$L = l_7 \vee R_7 l_6 \vee R_7 R_6 l_5 \vee R_7 R_6 R_5 l_4 \vee R_7 R_6 R_5 R_4 l_3 \vee R_7 R_6 R_5 R_4 R_3 l_2 \vee \vee R_7 R_6 R_5 R_4 R_3 R_2 l_1 \vee R_7 R_6 R_5 R_4 R_3 R_2 R_1 l_0, \qquad (2.13)$$

Для компаратора cmp_2 повна функція буде мати наступний вигляд:

$$M = m_7 \vee R_7 m_6 \vee R_7 R_6 m_5 \vee R_7 R_6 R_5 m_4 \vee R_7 R_6 R_5 R_4 m_3 \vee R_7 R_6 R_5 R_4 R_3 m_2 \vee \vee R_7 R_6 R_5 R_4 R_3 R_2 m_1 \vee R_7 R_6 R_5 R_4 R_3 R_2 R_1 m_0, \qquad (2.14)$$

2.3.3 Підстановка константних значень у попередньо визначені формули

Оскільки обидва компаратори порівнюють вхідне значення із константою можемо підставити значення цієї константи у формули базових функцій та функцій рівнозначності. Це дозволить нам значно скоротити та спростити запис повної функції після підстановки у неї значень y_i та R_i.

Підстановка константних значень у базові функції компаратора cmp_1:

$$\begin{gathered} l_7 = b_7 * not(a_7) = 0 * not(a_7) = 0; \\ l_6 = b_6 * not(a_6) = 0 * not(a_6) = 0; \\ l_5 = b_5 * not(a_5) = 0 * not(a_5) = 0; \\ l_4 = b_4 * not(a_4) = 1 * not(a_4) = not(a_4); \\ l_3 = b_3 * not(a_3) = 0 * not(a_3) = 0; \\ l_2 = b_2 * not(a_2) = 1 * not(a_2) = not(a_2); \\ l_1 = b_1 * not(a_1) = 0 * not(a_1) = 0; \\ l_0 = b_0 * not(a_0) = 0 * not(a_0) = 0. \end{gathered} \tag{2.15}$$

Підстановка константних значень у рівняння рівнозначності компаратора cmp_1:

$$\begin{gathered} R_7 = a_7 * b_7 \vee not(a_7) * not(b_7) = a_7 * 0 \vee not(a_7) * 1 = not(a_7); \\ R_6 = a_6 * b_6 \vee not(a_6) * not(b_6) = a_6 * 0 \vee not(a_6) * 1 = not(a_6); \\ R_5 = a_5 * b_5 \vee not(a_5) * not(b_5) = a_5 * 0 \vee not(a_5) * 1 = not(a_5); \\ R_4 = a_4 * b_4 \vee not(a_4) * not(b_4) = a_4 * 1 \vee not(a_4) * 0 = a_4; \\ R_3 = a_3 * b_3 \vee not(a_3) * not(b_3) = a_3 * 0 \vee not(a_3) * 1 = not(a_3); \\ R_2 = a_2 * b_2 \vee not(a_2) * not(b_2) = a_2 * 1 \vee not(a_2) * 0 = a_2; \\ R_1 = a_1 * b_1 \vee not(a_1) * not(b_1) = a_1 * 0 \vee not(a_1) * 1 = not(a_1); \\ R_0 = a_0 * b_0 \vee not(a_0) * not(b_0) = a_0 * 0 \vee not(a_0) * 1 = not(a_0). \end{gathered} \tag{2.16}$$

Підстановка константних значень у базові функції компаратора cmp_2:

$$\begin{gathered} m_7 = a_7 * not(b_7) = a_7 * 1 = a_7; \\ m_6 = a_6 * not(b_6) = a_6 * 1 = a_6; \\ m_5 = a_5 * not(b_5) = a_5 * 1 = a_5; \\ m_4 = a_4 * not(b_4) = a_4 * 0 = 0; \\ m_3 = a_3 * not(b_3) = a_3 * 0 = 0; \\ m_2 = a_2 * not(b_2) = a_2 * 1 = a_2; \\ m_1 = a_1 * not(b_1) = a_1 * 0 = 0; \\ m_0 = a_0 * not(b_0) = a_0 * 1 = a_0. \end{gathered} \tag{2.17}$$

Підстановка константних значень у рівняння рівнозначності компаратора cmp_2:

$$\begin{gathered}
R_7 = a_7 * b_7 \vee not(a_7) * not(b_7) = a_7 * 0 \vee not(a_7) * 1 = not(a_7); \\
R_6 = a_6 * b_6 \vee not(a_6) * not(b_6) = a_6 * 0 \vee not(a_6) * 1 = not(a_6); \\
R_5 = a_5 * b_5 \vee not(a_5) * not(b_5) = a_5 * 0 \vee not(a_5) * 1 = not(a_5); \\
R_4 = a_4 * b_4 \vee not(a_4) * not(b_4) = a_4 * 1 \vee not(a_4) * 0 = a_4; \\
R_3 = a_3 * b_3 \vee not(a_3) * not(b_3) = a_3 * 1 \vee not(a_3) * 0 = a_3; \\
R_2 = a_2 * b_2 \vee not(a_2) * not(b_2) = a_2 * 0 \vee not(a_2) * 1 = not(a_2); \\
R_1 = a_1 * b_1 \vee not(a_1) * not(b_1) = a_1 * 1 \vee not(a_1) * 0 = a_1; \\
R_0 = a_0 * b_0 \vee not(a_0) * not(b_0) = a_0 * 0 \vee not(a_0) * 1 = not(a_0).
\end{gathered} \tag{2.18}$$

Останнім кроком буде підстановка отриманих значень у попередньо визначені повні функції компараторів.

Повна функція для компаратора cmp_1 матиме наступний вигляд:

$$L = \bar{a}_7\bar{a}_6\bar{a}_5\bar{a}_4 \vee \bar{a}_7\bar{a}_6\bar{a}_5a_4\bar{a}_3\bar{a}_2, \tag{2.19}$$

Повна функція для компаратора cmp_2 буде мати такий вигляд:

$$M = a_7 \vee \bar{a}_7a_6 \vee \bar{a}_7\bar{a}_6a_5 \vee \bar{a}_7\bar{a}_6\bar{a}_5a_4a_3a_2 \vee \bar{a}_7\bar{a}_6\bar{a}_5a_4a_3\bar{a}_2a_1a_0, \tag{2.20}$$

У другому розділі було проведено аналіз поставленої задачі, утворено формули згідно з якими буде відбуватись обчислення вхідних значень і на основі всього цього утворено алгоритм роботи спецобчислювача. Далі, відповідно до створеного алгоритму, було спроєктовано усі структурні складові цифрового пристрою, а саме, операційний автомат, керуючий автомат та два компаратори. Було проведено усі необхідні розрахунки для подальшого синтезу моделей кожного зі структурних елементів цифрового пристрою.

3 МОДЕЛЮВАННЯ І ТЕСТУВАННЯ ЦИФРОВОГО ПРИСТРОЮ У ALDEC ACTIVE-HDL 13

3.1 Моделювання операційного автомата

Перед тим як почати моделювання самого операційного автомата спочатку змоделюємо усі його функціональні блоки та модулі контролю.

Необхідно також врахувати що у схемі операційного автомата усі дані будуть передаватись восьмибітними шинами. Відповідно щоб проводити маніпуляції над конкретним бітом числа треба його діставати з шини. Також, враховуючи те як у Aldec Active-HDL прописуються шини даних, необхідно заздалегідь визначити у якому напрямку буде іти нумерація розрядів. У програмі є два варіанти, “7 downto 0” або “0 to 7”. Інформаційно різниці в них небагато, оскільки відрізняються лише напрямком читання. Однак якщо у одній схемі обрати перший вид, а у іншій другий то підключити такі схеми або буе неможливо або сигнали в результаті буде переплутано. Тож під час проєктування в усіх схемах шини будуть виду “7 downto 0”, бо такий вид найбільш наближений до написання двійкових чисел.

У такому випадку, завдяки можливостям програмного продукту Aldec Active-HDL ми матимемо можливість зробити щось, схоже на бібліотеку з мов програмування. Коли ми створюємо яку-небудь схему вона одразу додається у дизайн, який є аналогом бібліотеки для схем пристроїв, і надалі ми зможемо використовувати збережені в ньому схеми у побудові інших.

3.1.1 Моделювання функціональних блоків

Оскільки в нас будуть обчислюватись восьмибітні числа нам необхідно щоб логічні операції відбувались між відповідними розрядами обох чисел. В такому разі ми можемо створити схему у яку будуть входити два числа, всередині вони будуть розбиватись на розряди й кожен відповідний розряд буде йти на звичайний логічний вентиль. На рис. 3.1 зображена схема модуля для обчислення операції AND.

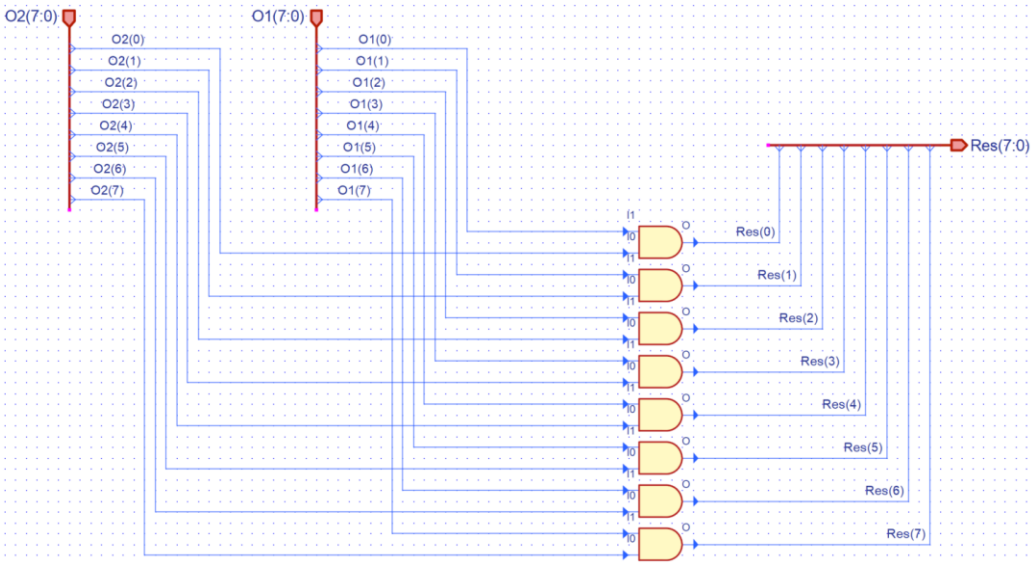

Рисунок 3.1 – Схема модуля операції AND для обчислення двох восьмибітних чисел

За тим самим принципом моделюються модулі для операцій XOR та OR. Також за схожим принципом моделюється і модуль для операції інверсії, лише з поправкою на те що вхідне число лише одне. Цей модуль продемонстровано на рис. 3.2.

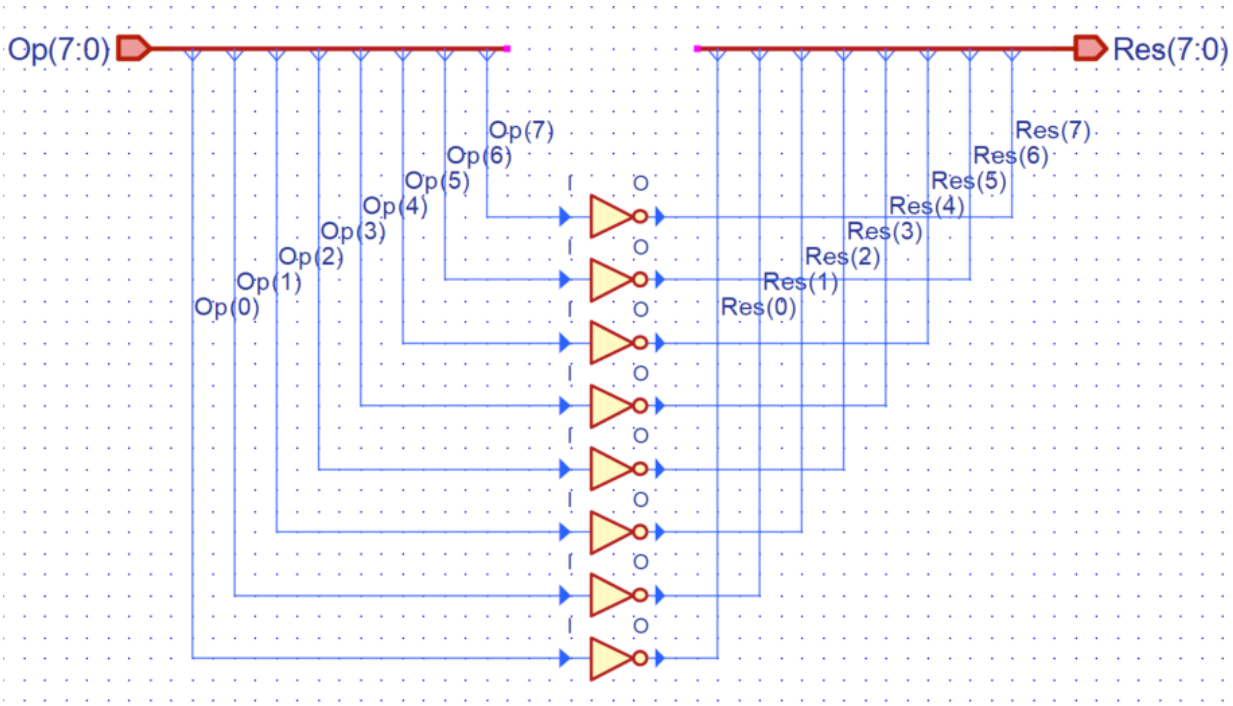

Рисунок 3.2 – Схема модуля операції NOT для обчислення одного восьмибітного числа

Усі схеми для модулів зсувів також моделюються за схожим між собою принципом. Схему модуля зсуву в ліво на три біти відображено на рис. 3.3. Логічні вентилі на схемі поставлені через те, що підключення напряму вхідного і вихідного сигналів заборонено програмою, але якщо між ними поставити OR тоді все спрацює як потрібно. Також, враховуючи те, що зсуви логічні, ті біти які було зміщено скидаються в нуль.

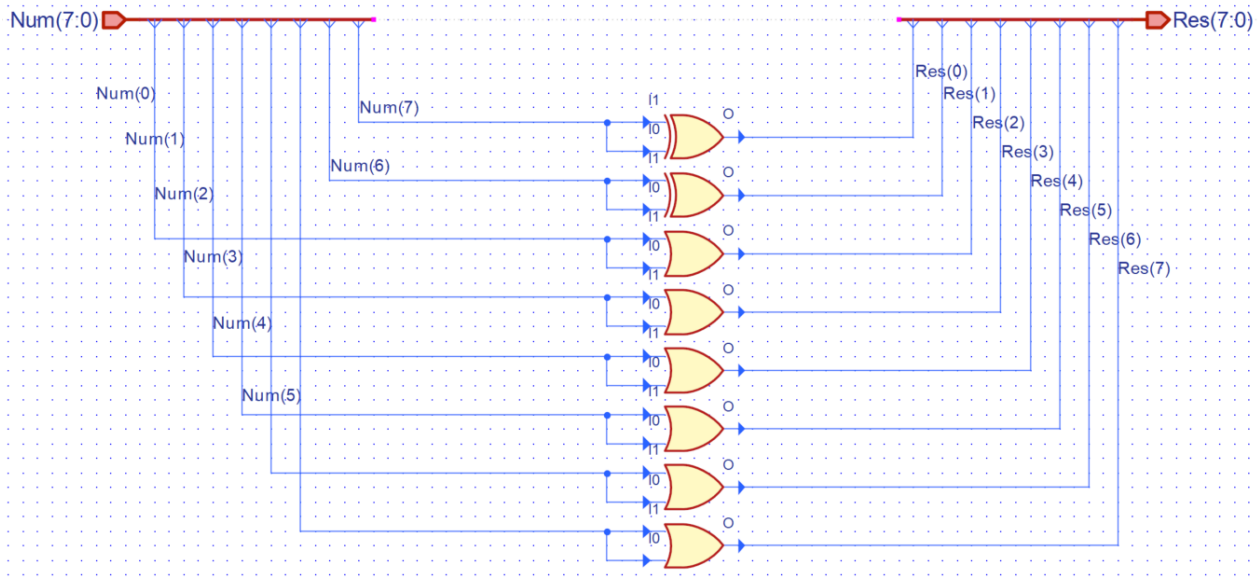

Рисунок 3.3 – Схема модуля операції зсуву вліво на два біти

Наступним змоделюємо суматор. Спочатку створимо схему суматора на два біти, який буде обчислювати відповідні біти обох операндів. Оскільки кінцевою ціллю є схема суматора на вісім біт двохбітний суматор повинен враховувати перенесення які можуть утворюватись при обчисленні. Зібравши схему суматора на два біти розмістимо сім таких на схемі майбутнього восьмибітного суматора. Для наймолодшого біту нема потреби враховувати перенесення, тому реалізуємо суму простим XOR та за допомогою AND будемо генерувати перенесення. Схема двухбітного суматора відображено на рис. 3.4, а схема восьмибітного суматора відображено на рис. 3.5.

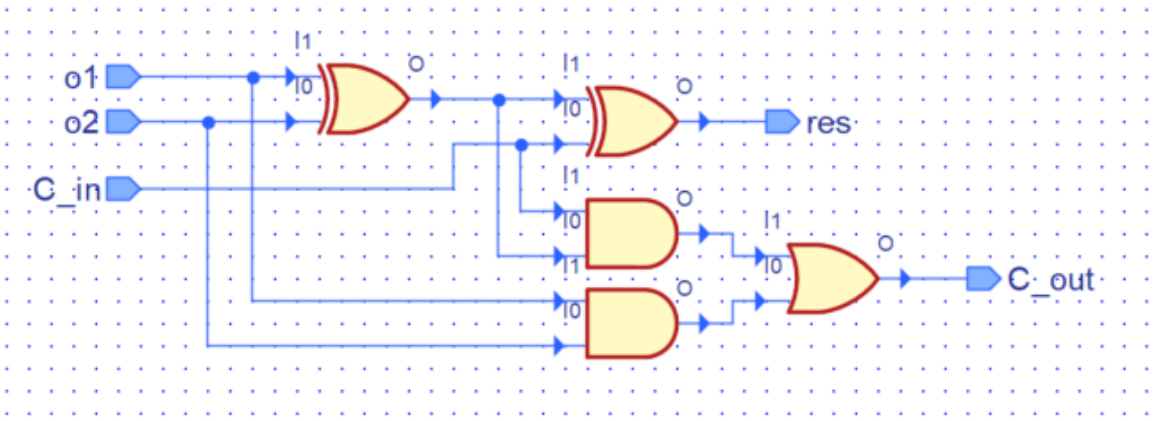

Рисунок 3.4 – Схема двохбітного суматора з урахуванням переносу

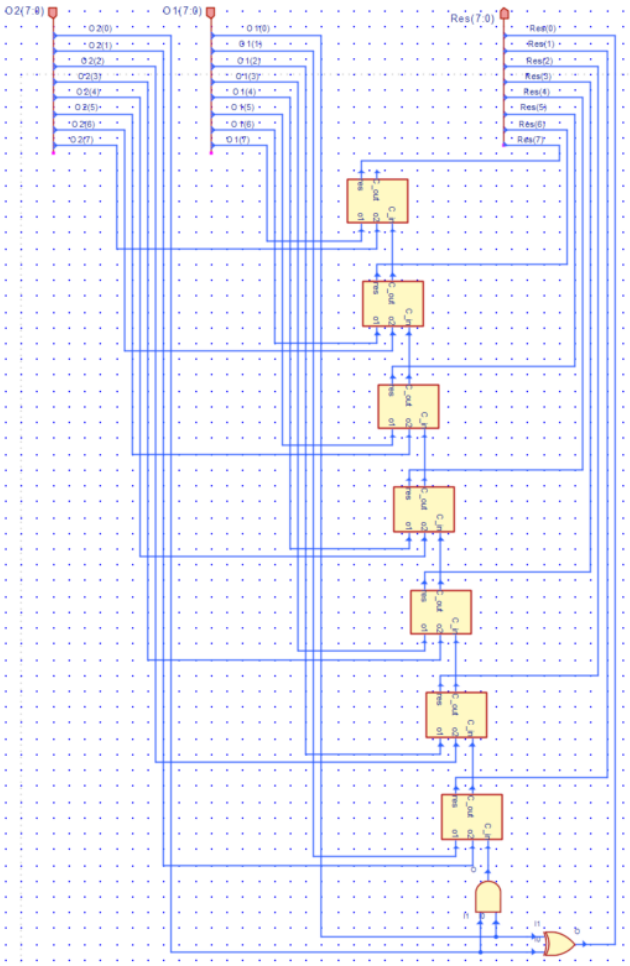

Рисунок 3.5 – Схема суматора для обчислення двох восьмибітних чисел

Далі змоделюємо схему модуля Enable. Це буде схема контролю, яка буде пропускати передавати на вихід схеми восьмибітне число тільки якщо вхідна умова істина. Схема модуля Enable зображена на рис. 3.6.

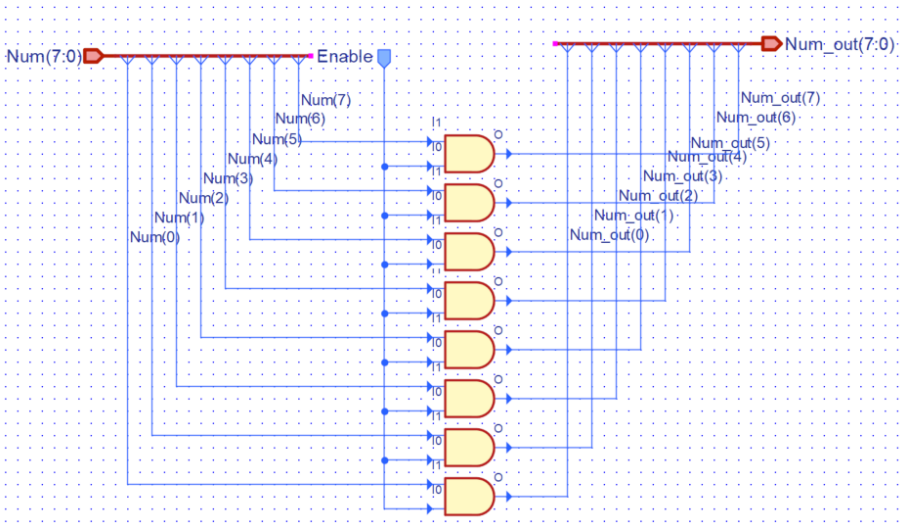

Рисунок 3.6 – Схема модуля Enable

Останнім залишилось змоделювати схему восьмибітного регістра який буде зберігати дані у операційному автоматі. Побудуємо регістр на основі D-тригерів, схему якого зображено на рис. 3.7. Окрім вхідного восьмибітного числа, синхросигналу та сигналу скидання на вхід схеми також було додано сигнал clk_control. Він фактично виконує функцію сигналу Enable деяких мікросхем, тобто тільки коли clk_control активний дозволяється запис у регістр. Це було зроблено, щоб регістри не перезаписували в себе нулі в моменти коли на їх вхід нічого не має поступати. Цей контролюючий сигнал було вирішено об’єднати разом з синхросигналом через логічний вентиль AND. Схема регістра зображена на рис. 3.8.

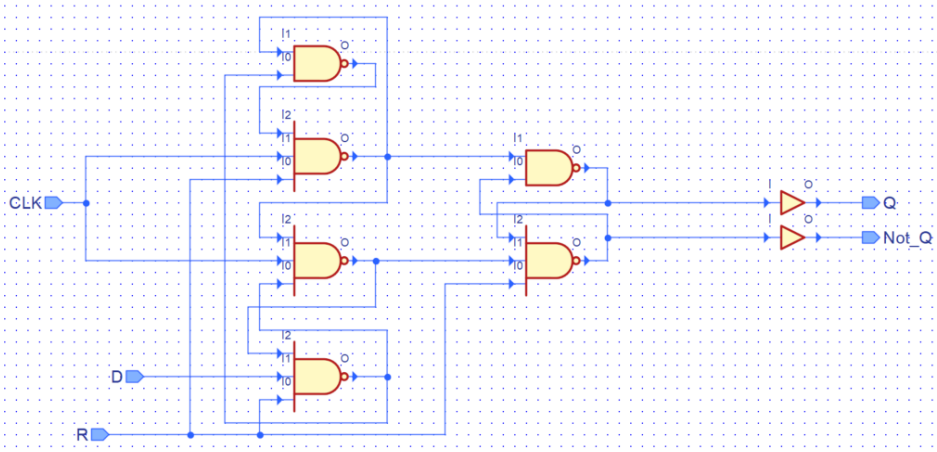

Рисунок 3.7 – Схема D-тригера

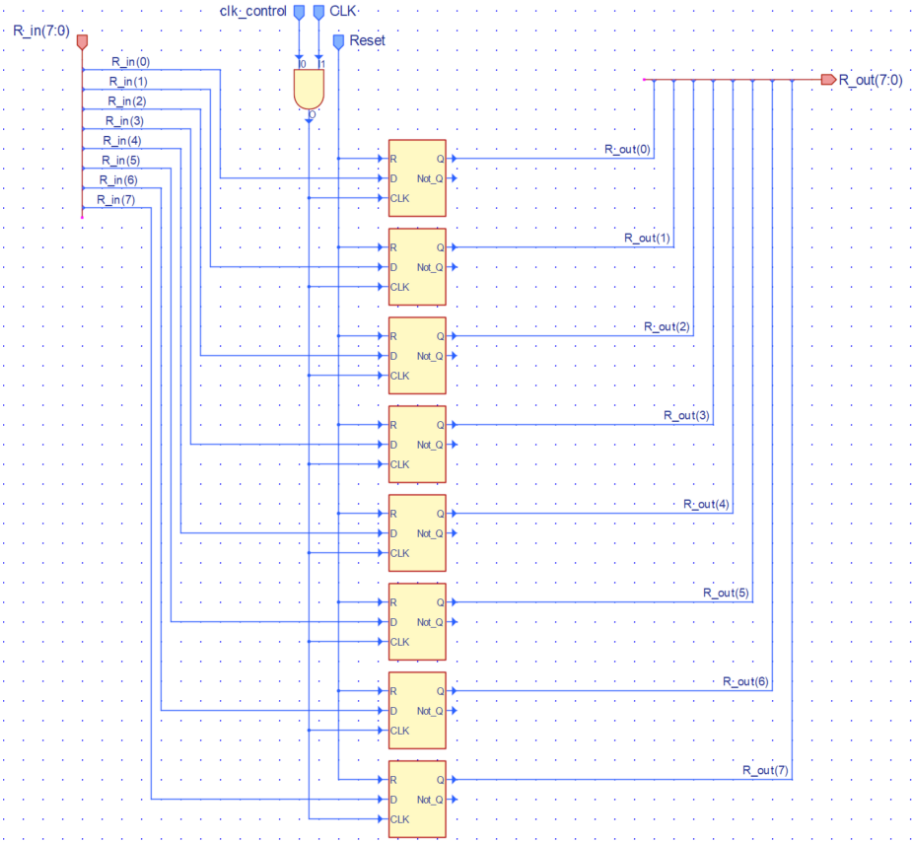

Рисунок 3.8 – Схема восьмибітного регістра на D-тригерах

3.1.2 Моделювання схем мультиплексорів

У схемі операційного автомата на рис. 2.4 можемо бачити дев’ять мультиплексорів призначення яких полягає у тому, щоб визначати які дані передати у функціональний блок або на шину даних. Три мультиплексори визначають операнд для функціонального блоку і чотири вивантажують дані на шини регістрів з функціонального блока який використовувався в поточній операції. Усі мультиплексори моделюються за однаковим принципом. Кожне вхідне число, як і у модулі Enable, побітово проходить через логічний вентиль And в якому другим операндом виступає умова пропускання числа. У нашому випадку у ролі таких умов використовуються сигнали мікрокоманд відповідної операції. Необхідно також враховувати, що для кожного числа буде своя умова і в один момент на вихід мультиплексора може виходити тільки одне число. Схему мультиплексорів для модулів зображено на рис. 3.9-3.11. Схеми мультиплексорів для шин регістрів зображено на рис. 3.12-3.15.

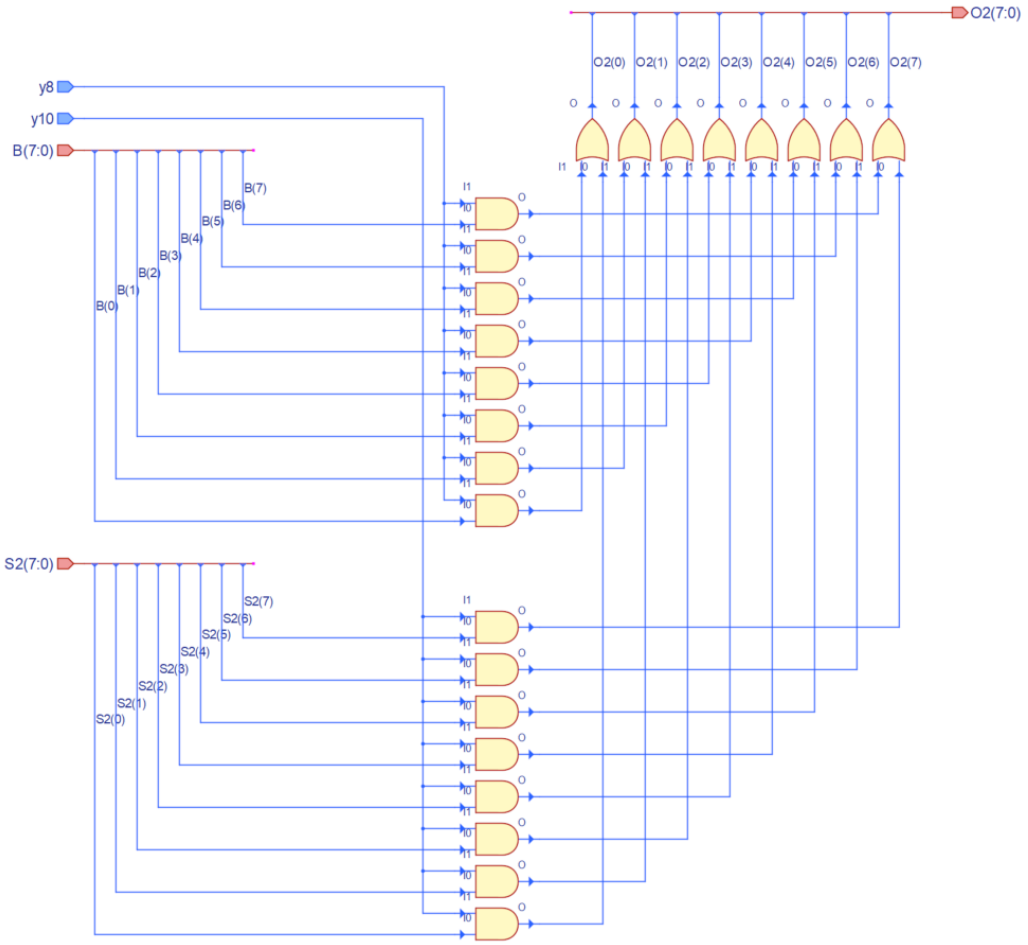

Рисунок 3.9 – Схема мультиплексора для суматора шини A

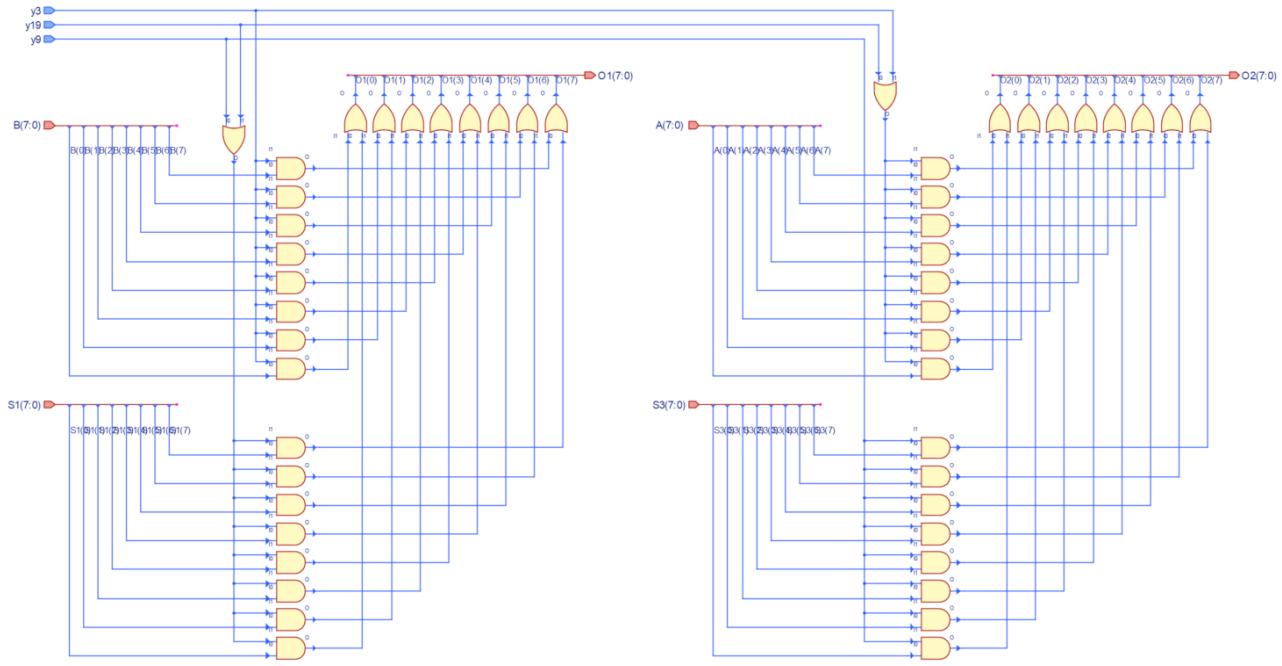

Рисунок 3.10 – Схема мультиплексора для суматора шини S_1

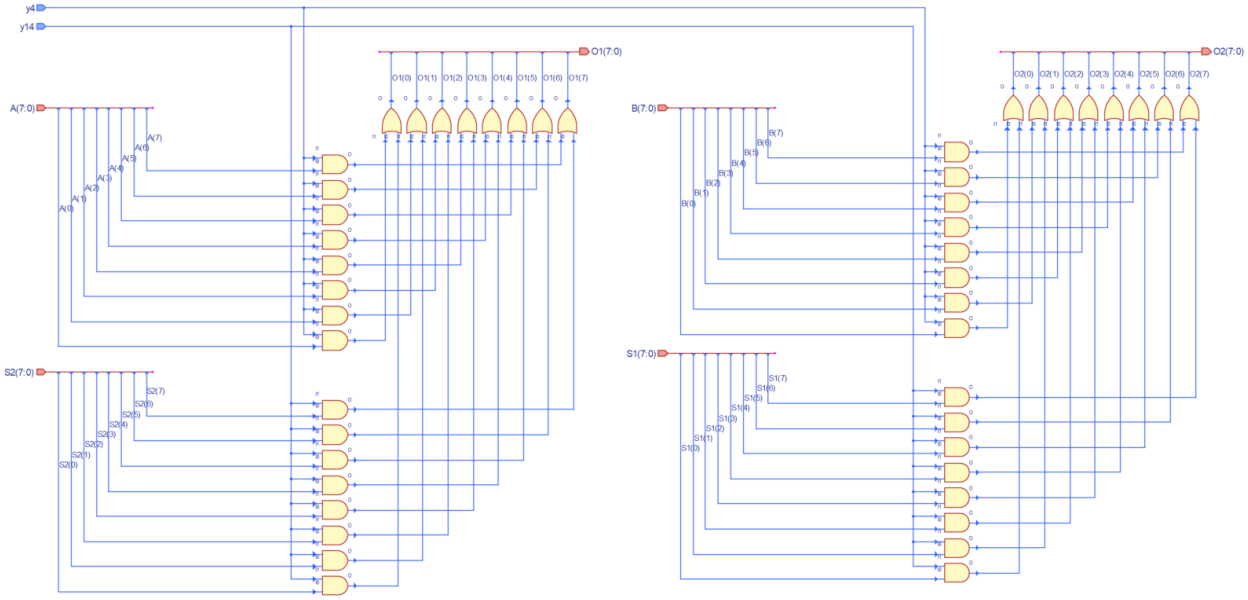

Рисунок 3.11 – Схема мультиплексора для модуля XOR шини S_2

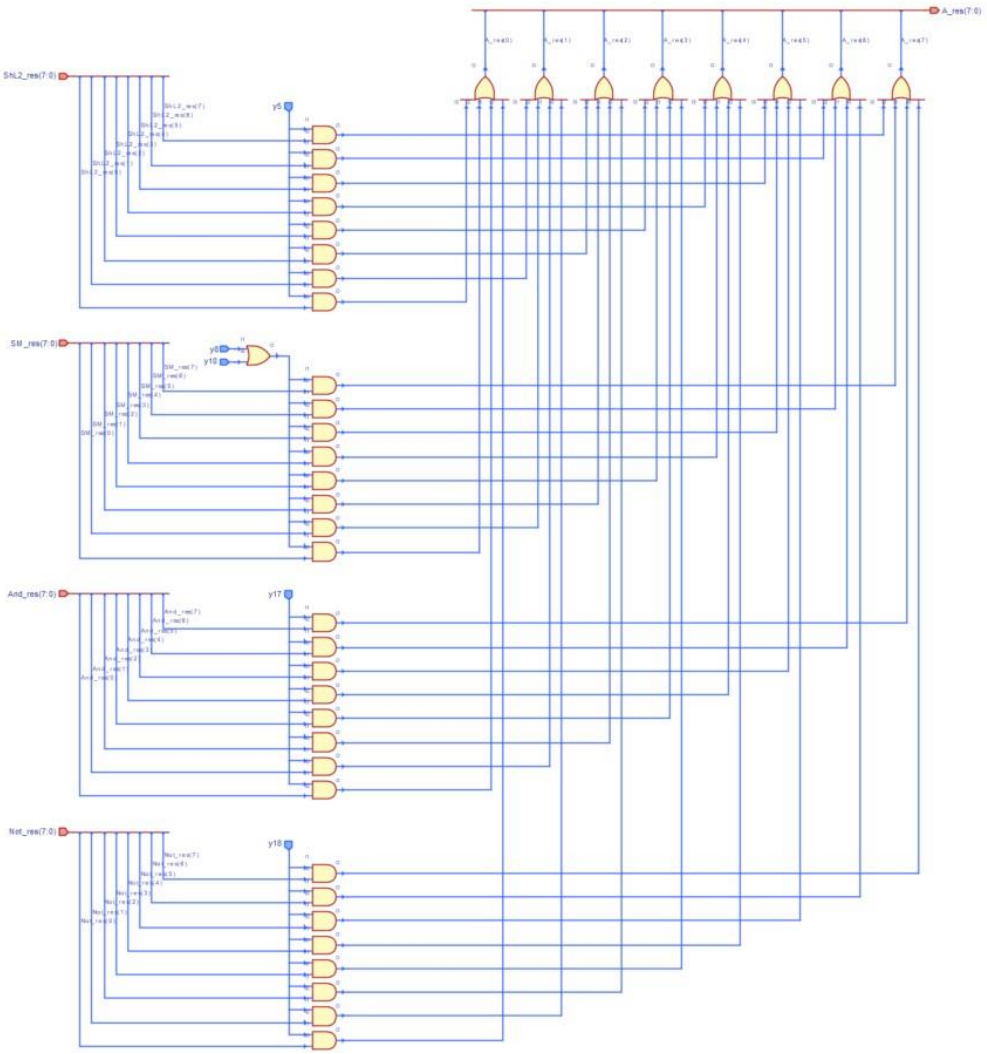

Рисунок 3.12 – Схема мультиплексора для шини A

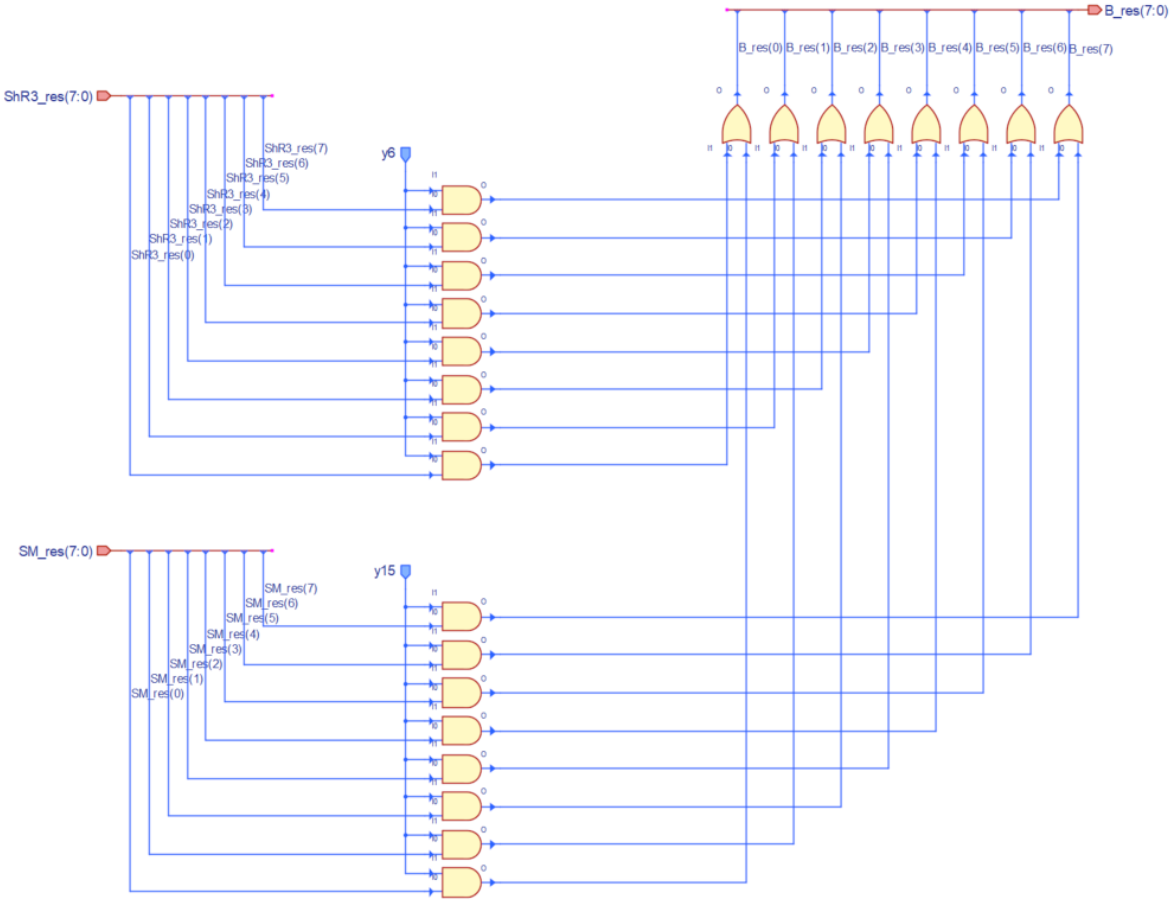

Рисунок 3.13 – Схема мультиплексора для шини B

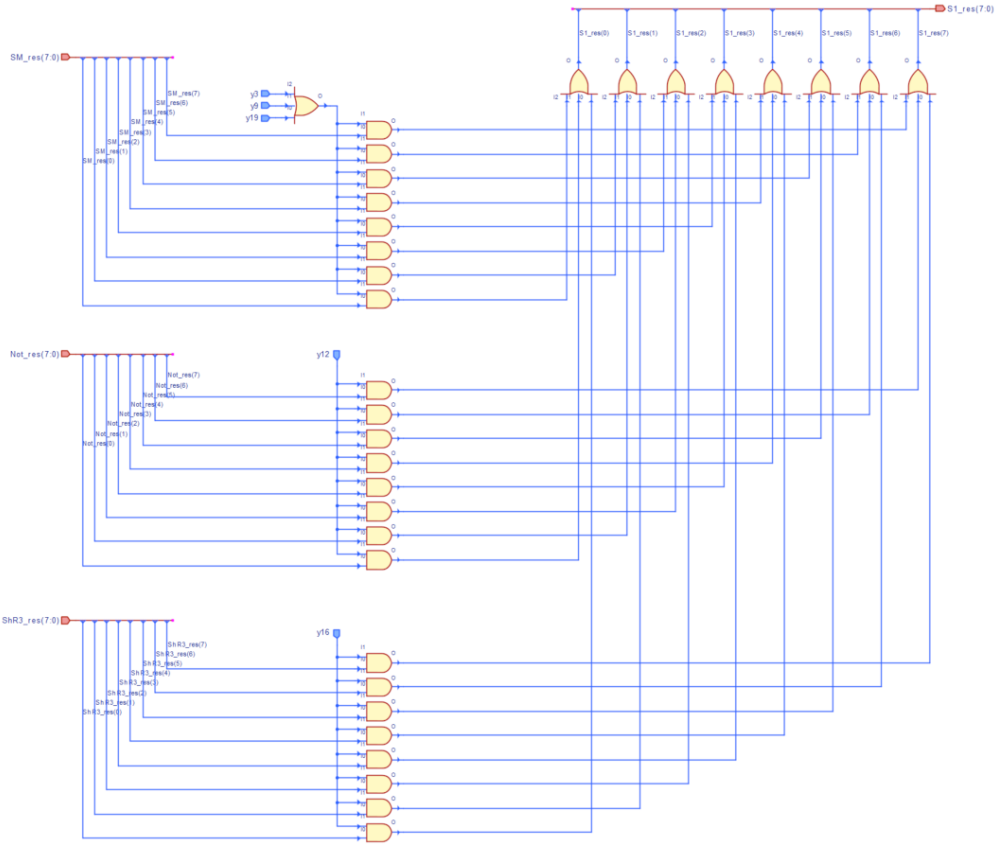

Рисунок 3.14 – Схема мультиплексора для шини S_1

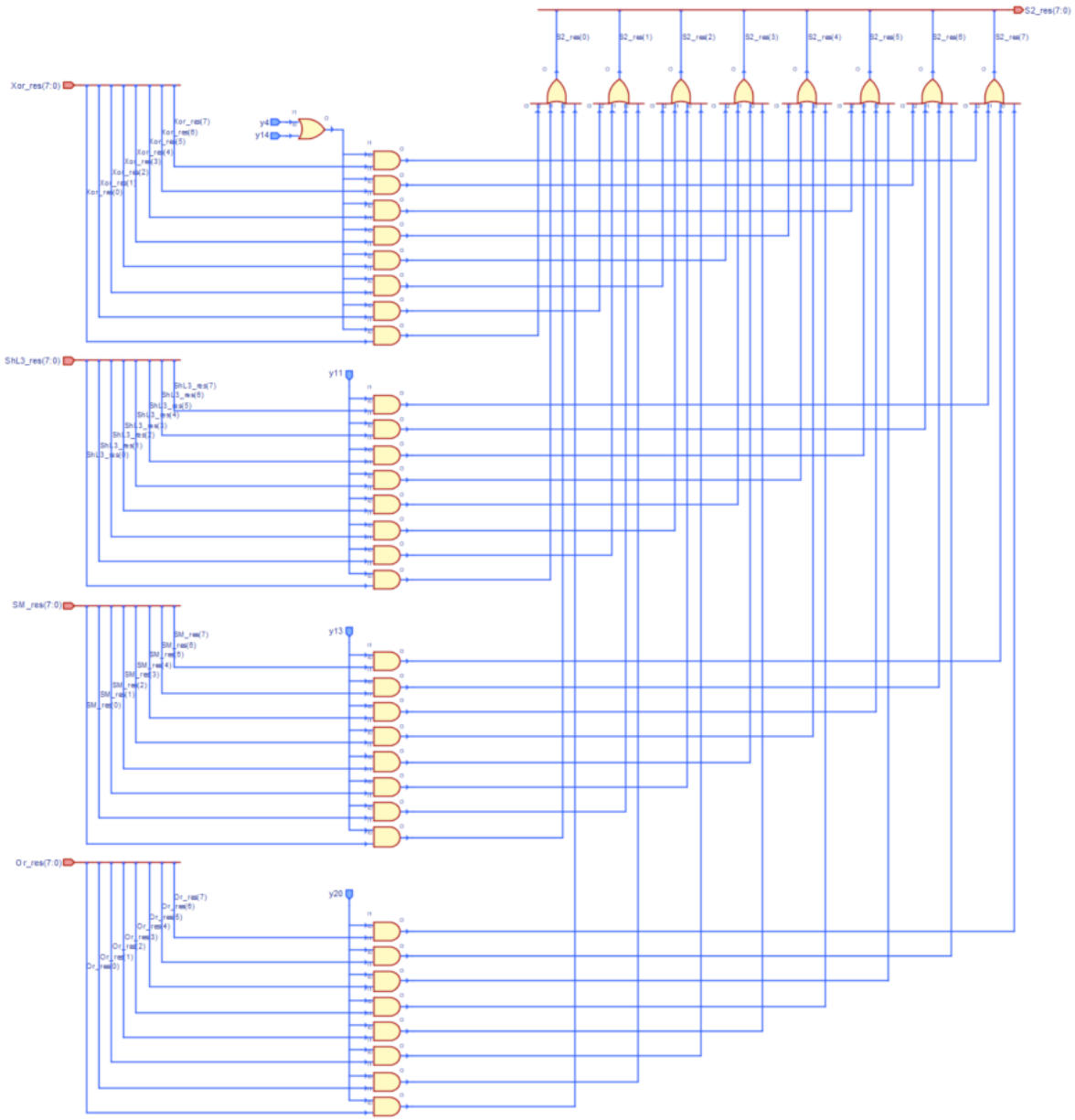

Рисунок 3.15 – Схема мультиплексора для шини S_2

3.1.3 Моделювання схем демультиплексорів

Для того, щоб дані з регістрів надходили лише на потрібні схеми у потрібний такт під кожен з регістрів змоделюємо демультиплексори. Виключенням буде лише регістр S_3 вивантаження даних з якого відбувається лише в один функціональний блок. Демультиплексор фактично виконує операцію зворотну мультиплексору, тобто він за певної умови пропускає сигнал однією з вихідних ліній. За цим принципом будуються усі демультиплексори. Схеми демультиплексорів для регістрів зображено на рис. 3.16-3.19.

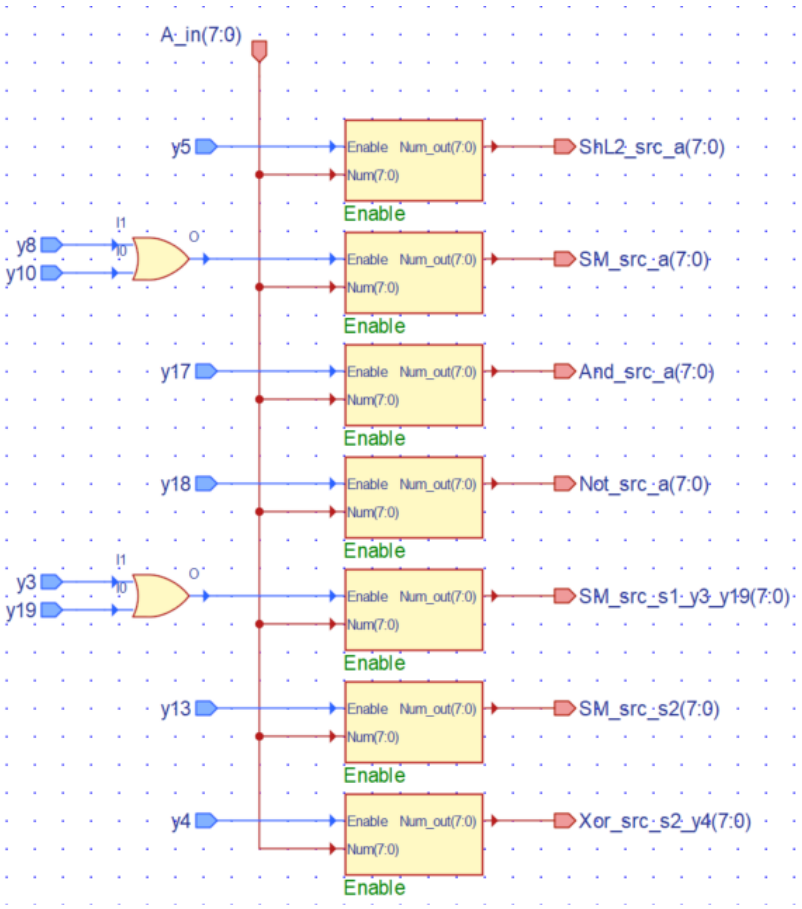

Рисунок 3.16 – Схема демультиплексора для регістру A

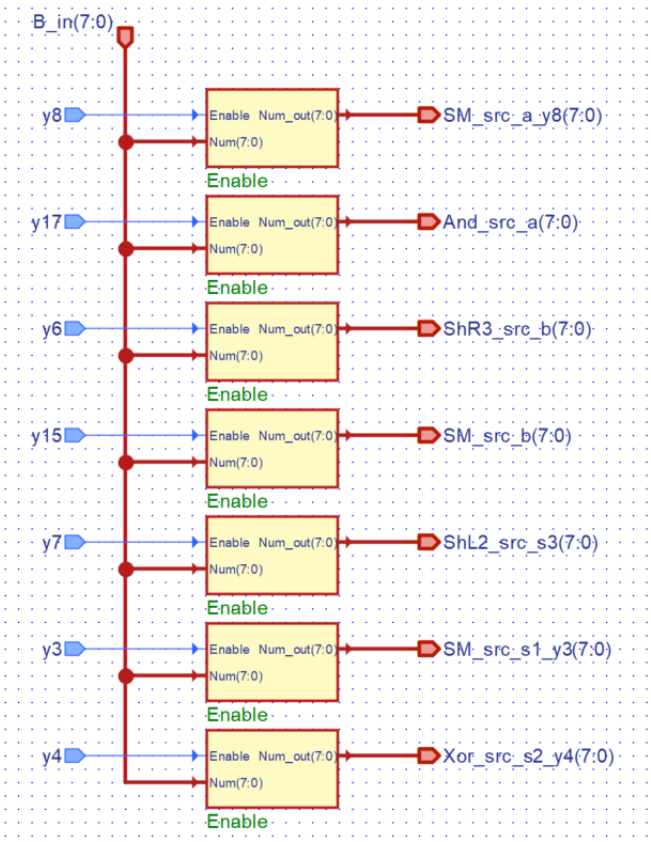

Рисунок 3.17 – Схема демультиплексора для регістру B

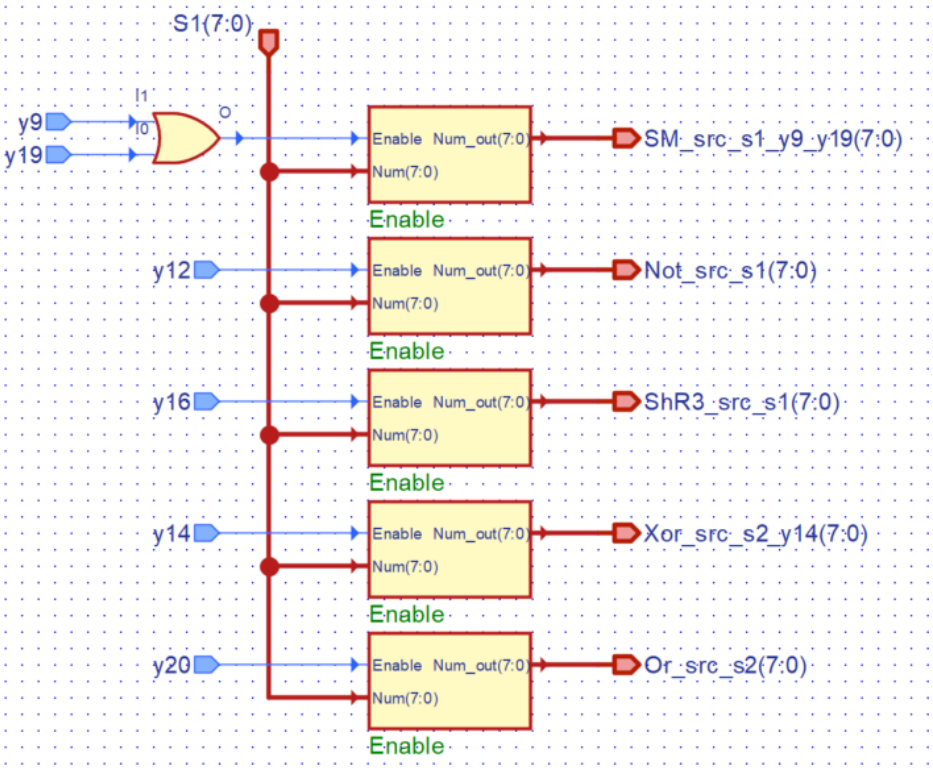

Рисунок 3.18 – Схема демультиплексора для регістру S_1

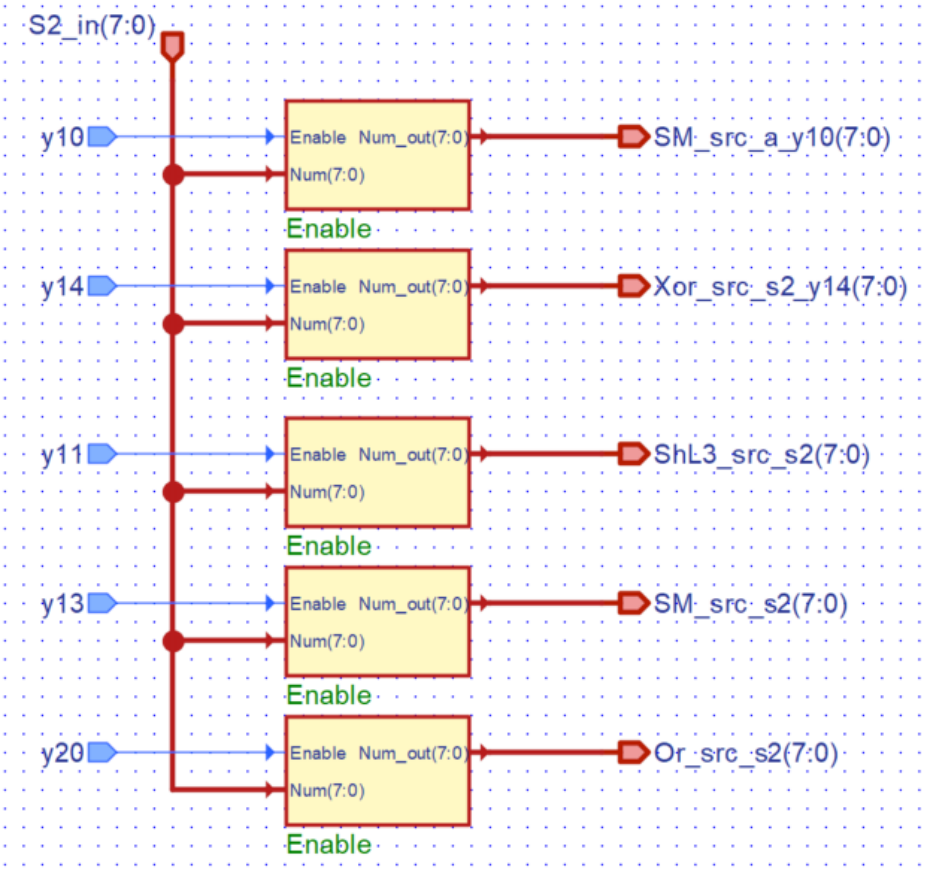

Рисунок 3.19 – Схема демультиплексора для регістру S_2

3.1.4 Збірка схеми операційного автомата

Тепер, зібравши у бібліотеках схеми усіх потрібних нам модулів можемо, спираючись на схему на рис. 2.4, можемо розпочати моделювання операційної частини автомата. Через те що моделювання цілого операційного автомата доволі складне і включає в себе комбінування великої кількості схем зробимо докладний опис процесу цього моделювання.

Спочатку при утворенні схеми визначаємо її вхідні та вихідні сигнали. На вхід до операційного автомата має надходити два восьмибітних числа, які позначимо d_1 та d_2 а на вихід має йти одне восьмибітне число результату обробки, його позначимо як res. Також оскільки кодування керуючих сигналів з керуючого автомата не закладалось при проєктуванні, то на вхід операційного автомата необхідно додати усі керуючі сигнали. Залишається ще додати синхросигнал та сигнал скидання(Reset) для регістрів. Приблизний вигляд опису сигналів схеми у програмі зображено на рис. 3.20.

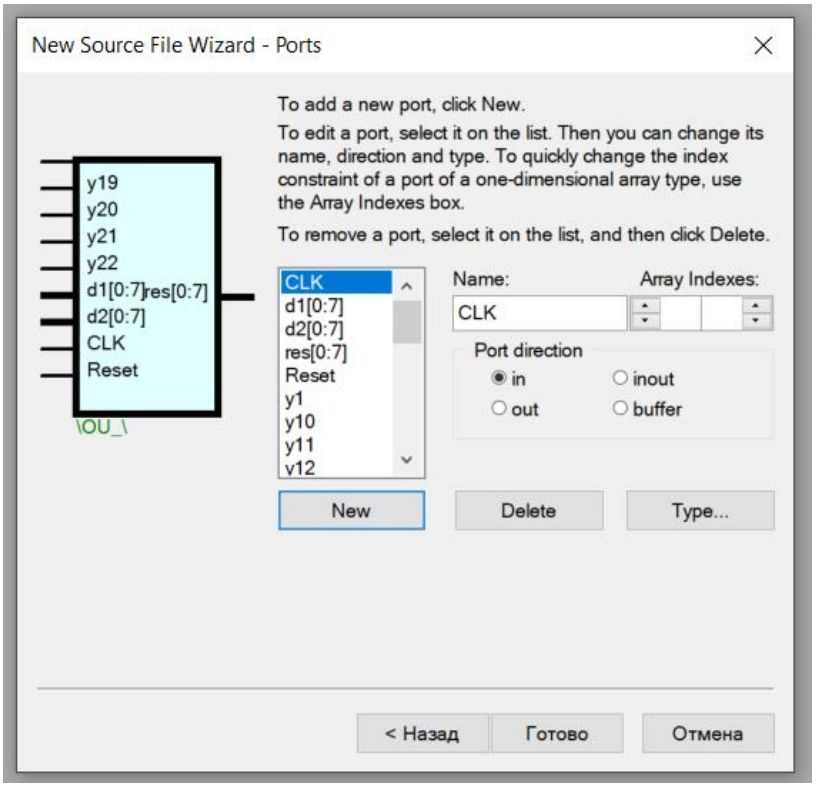

Рисунок 3.20 – Опис вхідних і вихідних сигналів схеми

Після утворення самої схеми розміщуємо на ній спершу п’ять регістрів. До регістрів A та B необхідно підключити шини вхідних значень d_1 та d_2, це зробимо за допомогою схеми Enable. Однак через те що ці регістри використовуються також і для зберігання проміжних результатів, і відповідно в них протягом операції відбуватиметься запис цих результатів, між Enable та регістром треба буде підключити схему OR для виключення конфліктів між шинами зовнішніх даних та шинами запису результатів обчислень функціональних модулів. Також можемо підключити до всіх регістрів одразу синхросигнал та сигнал скидання. Оскільки сигнал скидання використовується в роботі схеми для стирання внутрішніх даних регістрів після завершення обчислень то цей сигнал треба об’єднати з відповідним керуючим сигналом y_{22}, зробимо це за допомогою схеми XOR. На схемі це буде виглядати як показано на рис. 3.21.

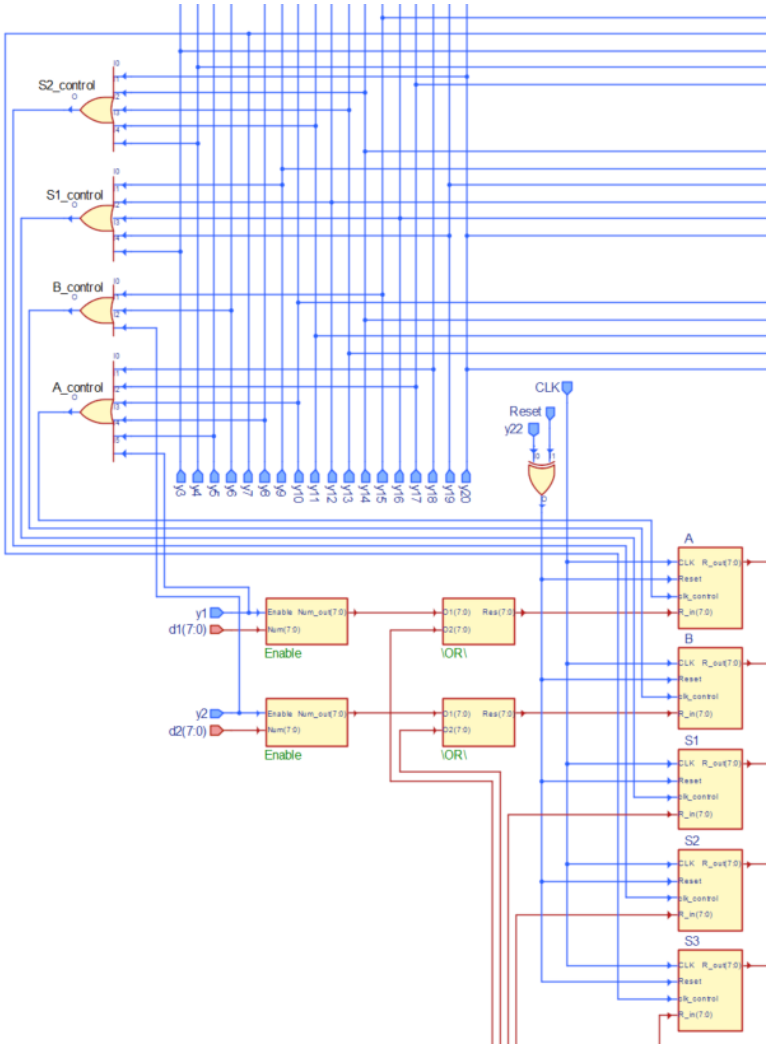

Рисунок 3.21 – Розміщення та підключення регістрів

Окрім синхросигналу і сигналу скидання до регістрів також підведено сигнал контролю запису, щоб уникнути небажаного перезапису внутрішніх даних регістрів. Цей сигнал контролю утворимо з об’єднаних між собою контролюючих сигналів у які надходить з керуючого автомату. Тобто такий сигнал буде утворюватись з об’єднаних функцією OR керуючих сигналів під час яких має відбуватись запис у відповідний регістр. Це ми можемо побачити у верхній лівій частині рис. 3.21.

Наступними на схемі розмістимо демультиплексори. Разом з цим розмістимо обчислювальні модулі які будуть безпосередньо оброблювати дані.

При розміщенні обчислювальних модулів варто їх групувати відповідно до того на шину якого регістру вони будуть виводити результат, це значно спростить підключення модулів до мультиплексорів шин та покращить зрозумілість і без того великої схеми. Там де це потрібно підключимо мультиплексори до модулів у яких вони передбачені згідно зі схемою на рис. 2.4. Після розміщення необхідно підключити усі відповідні сигнали до демультиплексорів, мультиплексорів та самих обчислювальних модулів.

Для початку підключимо демультиплексори, до них мають підключатись виходи регістрів та відповідні керуючі сигнали які будуть сигналізувати по якій лінії пропускати дані з регістру. Кожен сигнал підписано на умовному позначенні схеми, що спрощує підключення. Щоб ще більше спростити підключення ми можемо розмістити лінії керуючих сигналів у ряд масивом так буде і красивіше і зрозуміліше підключати, до того ж так чи інакше доведеться тягнути підключення до мультиплексорів шин.

Після того як підключили демультиплексори від них підключаємо усі відповідні обчислювальні модулі. Якщо один або два входи модуля мають підключатись до мультиплексора то спочатку підключаємо його аналогічно до демультиплексора, після виходи мультиплексора підключаємо до потрібного модуля.

Частина схеми з розміщенням демультиплексорів та обчислювальних модулів відображена на рис. 3.22.

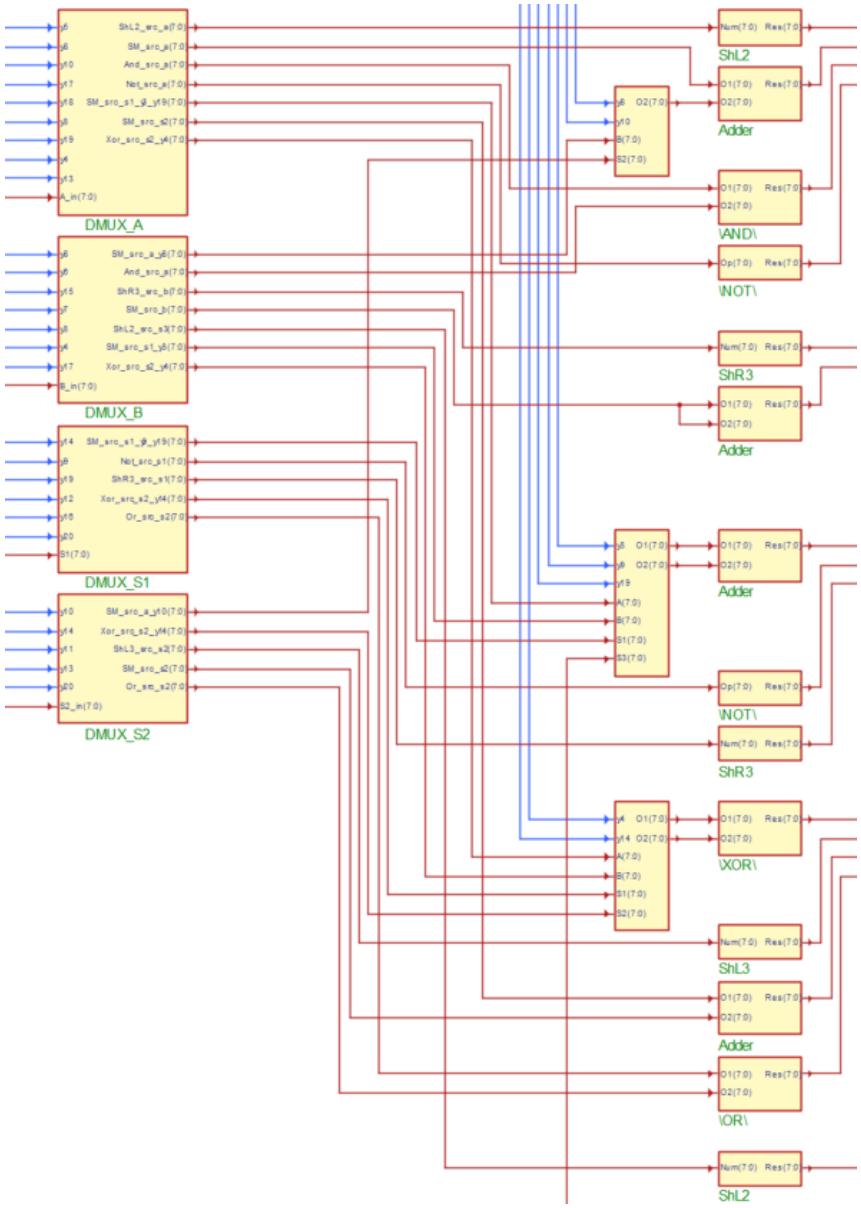

Рисунок 3.22 – Схема підключення демультиплексорів та обчислювальних модулів

Більшою мірою залишилось лише підключити виходи обчислювальних модулів до відповідних мультиплексорів які будуть виводити дані на шину для запису у певний регістр. Як і у минулому кроці розміщуємо на схемі мультиплексори та підключаємо усі його вхідні сигнали та шини, після чого можемо одразу підключити виходи мультиплексорів до відповідних їм регістрів. Окрім мультиплексорів для регістрів A та B, бо їх виходи підключаються через функцію OR до регістрів. Розміщення та підключення мультиплексорів на схемі зображено на рис. 3.23.

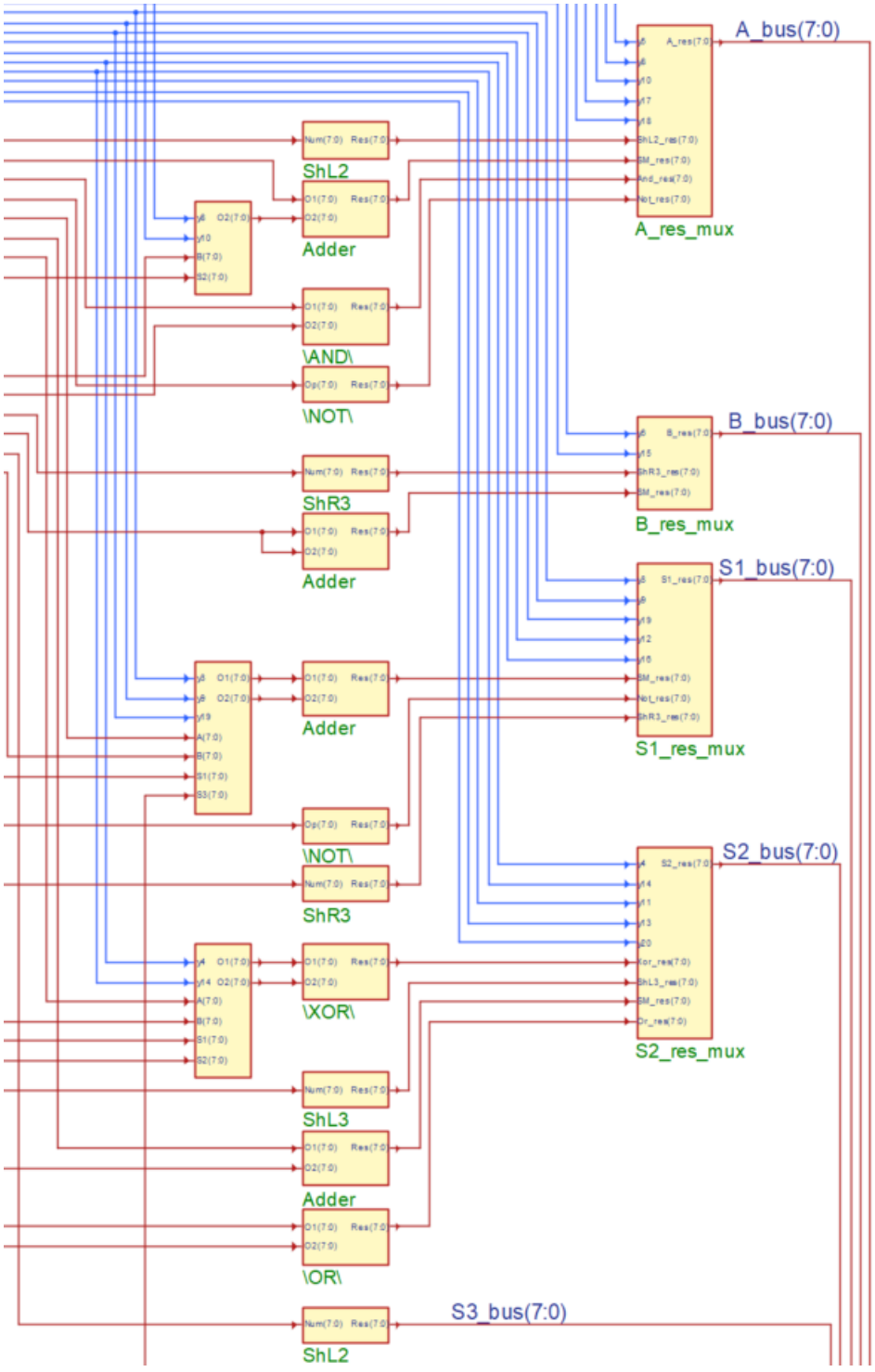

Рисунок 3.23 – Схема підключення мультиплексорів шин даних регістрів

Останнім кроком додамо на схемі виведення результату. Зробимо це завдяки звичайній схемці Enable у яку заведемо шину з регістру S_2, де у кінці буде записано результат, та керуючий сигнал y_{21} який відповідає за виведення результату. Як це буде виглядати відображено на рис. 3.24.

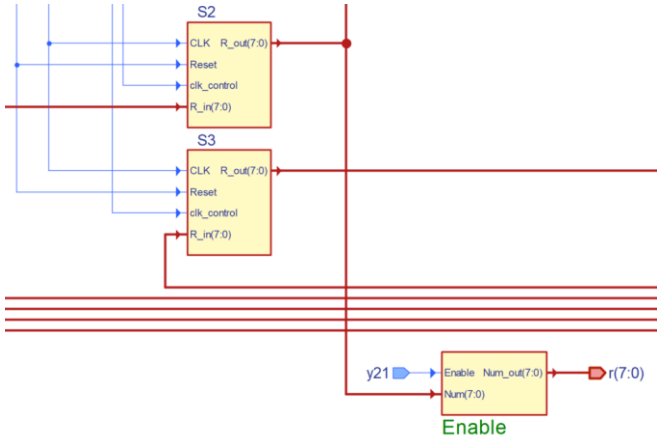

Рисунок 3.24 – Частина схеми для виведення результату

Отже, після розміщення усіх структурних елементів згідно з схемою зображеної на рис. 2.4 та підключення їх між собою отримаємо схему операційного автомата яку відображено на рис. 3.25.

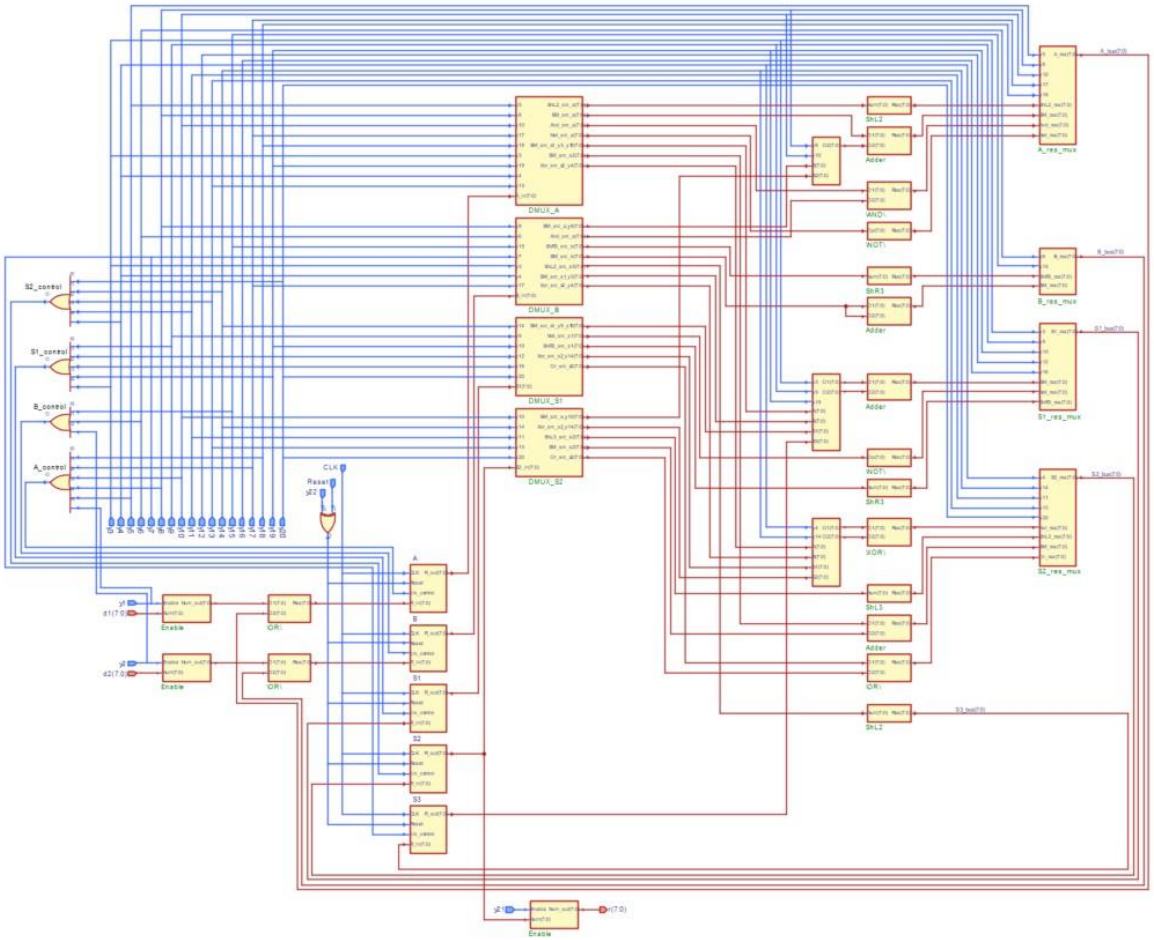

Рисунок 3.25 – Схема повністю зібраного операційного автомата

3.1.5 Тестування операційного автомата

Проведемо тестування операційного автомата, щоб впевнитись у коректності значень які він видає у результаті. Для цього виконаємо симуляцію роботи схеми. Оскільки керуючий автомат ще не змодельовано керуючі сигнали пропишемо вручну у налаштуваннях вхідних значень симуляції. Оберемо ліву гілку алгоритму який зображено на рис. 2.3 і відповідно до нього пропишемо формули зміни стану для усіх потрібних керуючих сигналів. Керуючі сигнали які не будуть використовуватись під час даного тестування також необхідно ініціалізувати, бо інакше ми отримаємо некоректну відповідь, або не отримаємо її взагалі. Зайві сигнали проініціалізуємо логічним “0”. Необхідно також врахувати, що регістри працюють лише коли сигнал Reset активний, але при запуску він має бути хоча б кілька наносекунд неактивним, щоб була можливість ініціалізувати початкові значення регістрів. Синхросигнал також доведеться налаштувати, для цього зменшимо період коли він активний не змінюючи загальну тривалість циклу. Це дозволяє зробити налаштування “Duty cycle” яке поставимо на 60, що означатиме що 60% циклу сигнал неактивний, усю іншу частину активний. Лишилось тільки записати значення вхідних даних, врахуємо також щоб вони вміщувались і діапазон умов для лівої гілки алгоритму. Тоді візьмемо наступні вхідні дані: A = 19 і B = 29. Результат проведеної симуляції зображено на рис. 3.26.

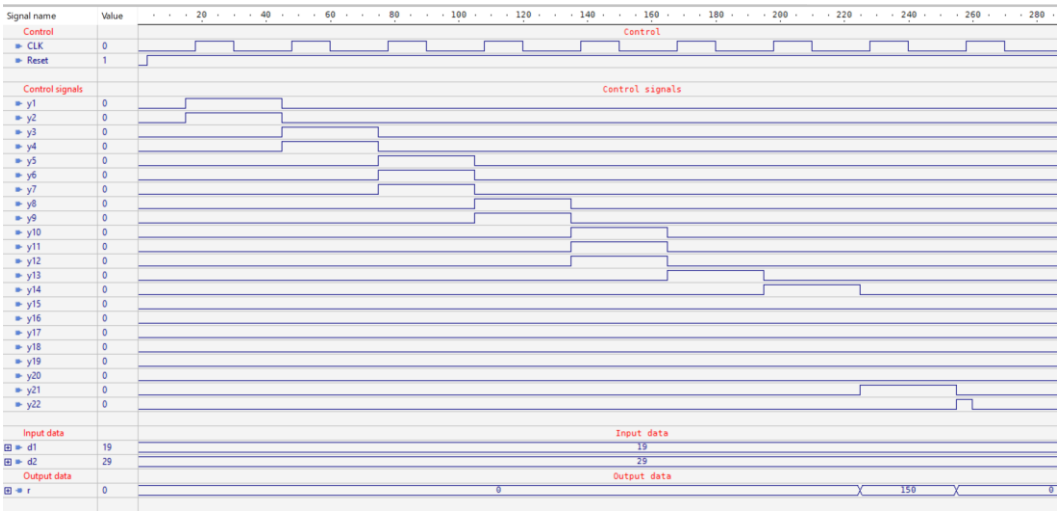

Рисунок 3.26 – Діаграма з результатом симуляції роботи операційного автомата

Перевіримо правильність вирахуваних значень. Для цього власноруч обчислимо обране рівняння у яке підставимо значення, подані на вхід операційного автомата. Ці розрахунки слід робити за тим самим алгоритмом за яким рахує і сам операційний автомат. Обчислення запишемо у вигляді таблиці де буде у строку буде записано оператор, операнди які приймають в ній участь і результат обчислень. Розрахунки записано у табл. 3.1.

Таблиця 3.1 – Таблиця розрахунків для перевірки правильності результату операційного автомату.

Оператор	Операнд 1	Операнд 2	Результат
A + B	00010011	00011101	S_1 <= 00110000
A ⊕ B	00010011	00011101	S_2 <= 00001110
ShL_2(A)	00010011	-	A <= 01001100
ShR_3(B)	00011101	-	B <= 00000011
ShL_2(B)	00011101	-	S_3 <= 01110100
A + B	01001100	00000011	A <= 01001111
S_1 + S_3	00110000	01110100	S_1 <= 10100100
A + S_2	01001111	00001110	A <= 01011101
ShL_3(S_2)	00001110	-	S_2 <= 01110000
Not(S_1)	10100100	-	S_1 <= 01011011
S_2 + A	01110000	01011101	S_2 <= 11001101
S_2 xor S_1	11001101	01011011	S_2 <= 10010110
Результат на виході	$10010110_2 == 150_{10}$		

Як і на діаграмі з рис. 3.26 при рочному обчисленні при вхідних даних A = 19 та B = 29 ми отримали відповідь 150. Відповідно робимо висновок що операційний автомат працює коректно.

3.2 Моделювання керуючого автомата

У процесі моделюванні операційного автомата ми використовували лише, так би мовити, візуальне моделювання. Однак програмний пакет Aldec Active-HDL дозволяє моделювати пристрої також за допомогою мов опису апаратури VHDL та Verilog. І враховуючи кількість термів у керуючому автоматі доцільніше буде описати їх на якійсь з цих мов опису.

Загалом є можливість описати повністю весь керуючий автомат, однак було вирішено виконати моделювання змішаним способом. Тобто усі перетворення термів винести у окремий модуль, назвемо його “генератор термів”, який буде описано на VHDL після чого залишиться підключити до нього на вхід чотири D-тригери та змінну X, а його виходи до усіх виходів керуючих сигналів.

3.2.1 Опис “генератора термів”

Для того, щоб описати цей модуль для початку визначимо які вхідні та вихідні сигнали нам потрібні. На вхід достатньо буде подати прямі виходи тригерів та сигнал умови X. А на вихід зі схеми будуть йти сигнали збудження тригерів та усі керуючі сигнали.

Отже, виконаємо опис “генератора термів” згідно таблиці термів яка відображена у табл. 2.4. Після чого реалізуємо рівняння з формул 2.7 та 2.8 підставивши в них потрібні терми. Код опису модуля на мові опису VHDL наведено у додатку А.

3.2.2 Збірка схеми керуючого автомата

Отже, тепер маючи “генератор термів” для збірки керуючого автомата нам залишається тільки розмістити генератор термів разом з тригерами на схемі та підключити всі сигнали. Схема керуючого автомата зображена на рис. 3.27.

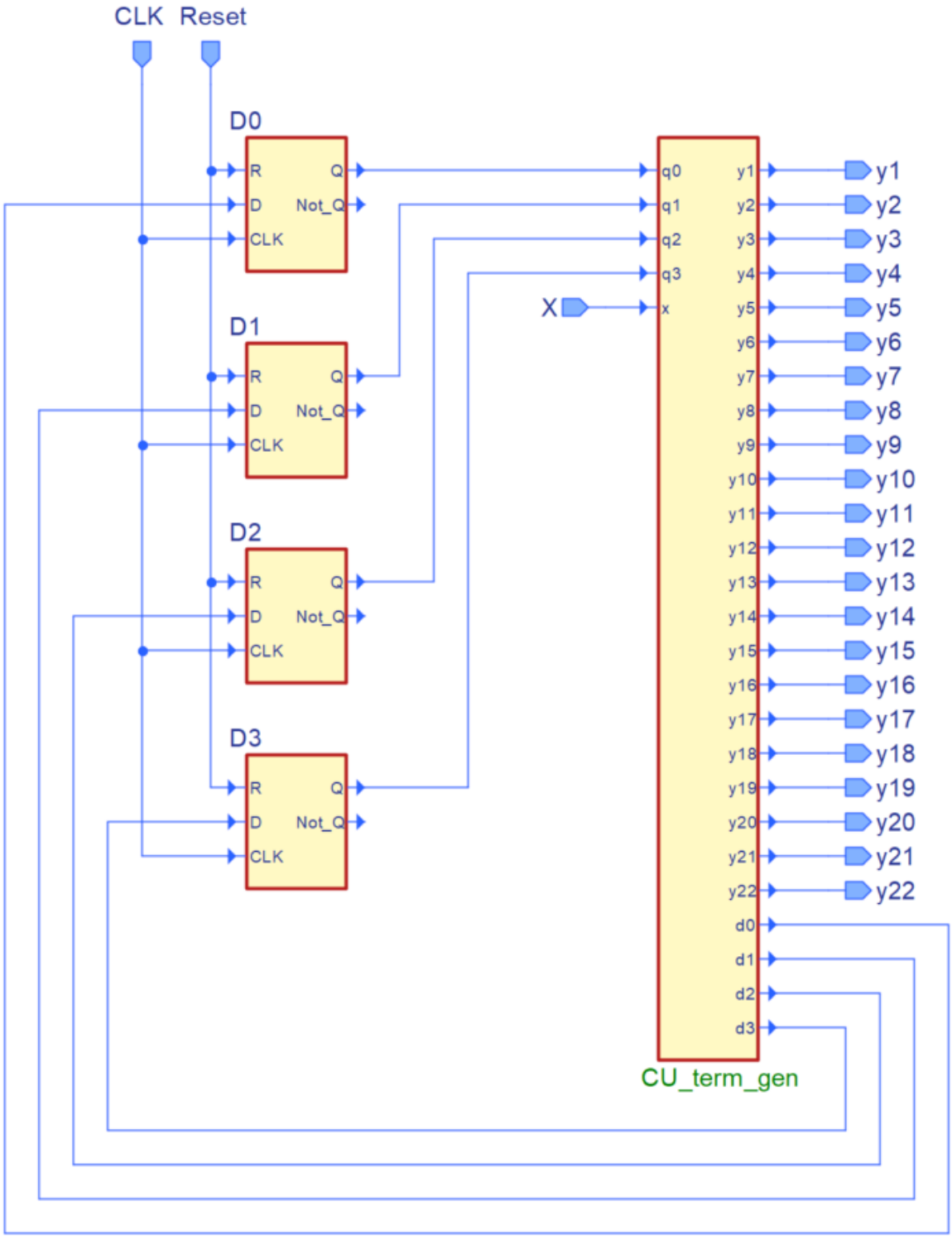

Рисунок 3.27 – Схема керуючого автомата

3.2.3 Тестування керуючого автомата

Тепер проведемо тестування керуючого автомата, щоб впевнитись у коректності його роботи. Синхросигнал та Reset налаштовуємо так само як і при тестуванні операційного автомата. Після перевірки лівої гілки алгоритму визначив, що алгоритм закінчує свою роботу приблизно на 300 наносекунді, тож щоб на одному графіку відобразити перевірку обох гілок алгоритму у формулі для X пропишемо зміну з логічної “1” на логічний “0” на 300-тій наносекунді. Тоді, завдяки тому, що автомат циклічно змінює стани, після завершення першого циклу при X = 1 після 300-тої наносекунди почнеться цикл вже з X = 0 який переводить на праву гілку алгоритму. Результат тестування керуючого автомата зображено на рис. 3.28.

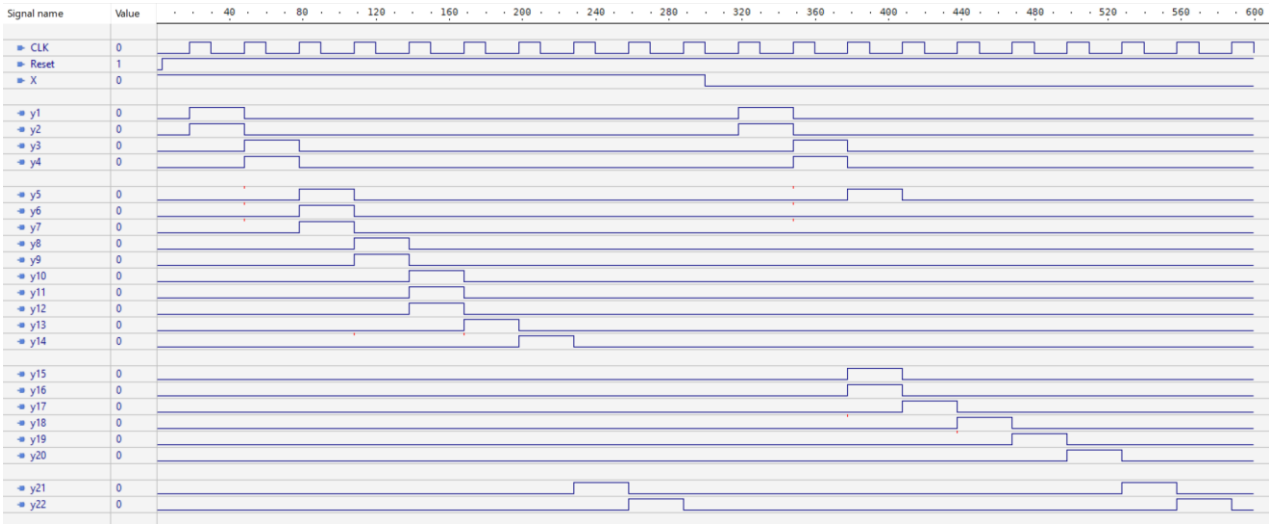

Рисунок 3.28 – Діаграма з результатом симуляції керуючого автомата

Порівняймо результати, зображені на діаграмі з рис. 3.28, із послідовністю керуючих сигналів відображеною на рис. 2.5. Для цього потактово запишемо усі керуючі сигнали які генерує керуючий автомат. Там де це потрібно також треба врахувати значення умови X, оскільки він впливає на те якою гілкою алгоритму буде йти керуючий автомат. Результат порівняння занесемо у табл. 3.2.

Таблиця 3.2 – Таблиця порівняння результату роботи керуючого автомата з еталонними значеннями граф-схеми алгоритму

Номер такту	Керуючі сигнали на граф-схемі		Керуючі сигнали на виходах керуючого автомату	
	X == 1	X == 0	X == 1	X == 0
1	y_1 y_2		y_1 y_2	
2	y_3 y_4		y_3 y_4	
3	y_5 y_6 y_7	y_5 y_{15} y_{16}	y_5 y_6 y_7	y_5 y_{15} y_{16}
4	y_8 y_9	y_{17}	y_8 y_9	y_{17}
5	y_{10} y_{11} y_{12}	y_{18}	y_{10} y_{11} y_{12}	y_{18}
6	y_{13}	y_{19}	y_{13}	y_{19}
7	y_{14}	y_{20}	y_{14}	y_{20}
8	y_{21}		y_{21}	
9	y_{22}		y_{22}	

Отже, як бачимо з рис. 3.28 та табл. 3.2 керуючий автомат працює коректно і генерую усі потрібні керуючі сигнали у потрібні такти.

3.3 Моделювання компараторів

3.3.1 Опис компараторів

Останніми елементами які залишилось змоделювати є компаратори. Враховуючи спосіб синтезу компараторів та спираючись на вигляд формул які ці компаратори описують бачимо що вони складаються з великої кількості простих логічних вершин AND та OR. Отже, дивлячись на це, можемо дійти висновку що

зручніше за все буде змоделювати такі компаратори за допомогою мови опису апаратури VHDL.

Скористаємось для цього формулами 2.19 та 2.20. Згідно з тим, що на алгоритмі, зображеному на рис. 2.3, обидві умови, і відповідно їх компаратори, входять в одну вершину умови ці дві формули необхідно буде у кінці підключити до логічного вентиля AND після чого ми отримаємо потрібний нам сигнал умови. Код опису обох компараторів наведено у лістингу 3.1.

Лістинг 3.1 – Код опису компараторів на мові опису апаратури VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
entity CMP is
  port(
        A  : in STD_LOGIC_VECTOR(7 downto 0);
        B  : in STD_LOGIC_VECTOR(7 downto 0);
        L  : out STD_LOGIC;
        M  : out STD_LOGIC;
        x  : out STD_LOGIC
  );
end CMP;
architecture CMP of CMP is
signal L_tmp : STD_LOGIC;
signal M_tmp : STD_LOGIC;

begin

        L_tmp <= ( not(a(7)) and not(a(6)) and not(a(5)) and
not(a(4)) )
        or ( not(a(7)) and not(a(6)) and not(a(5)) and a(4) and
not(a(3)) and not(a(2)) );

        M_tmp <= b(7) or ( not(b(7)) and b(6) ) or ( not(b(7)) and
not(b(6)) and b(5) )
        or ( not(b(7)) and not(b(6)) and not(b(5)) and b(4) and
b(3) and b(2) )
        or ( not(b(7)) and not(b(6)) and not(b(5)) and b(4) and
b(3) and not(b(2)) and b(1) and b(0) );

        x <= L_tmp and M_tmp;
        L <= L_tmp;
        M <= M_tmp;

end CMP;
```

3.3.2 Тестування компараторів

Проведемо тестування, щоб впевнитись у тому, що компаратори працюють правильно та видають коректний сигнал відповідно до вхідних значень. Для полегшення процесу тестування створимо таблицю де запишемо усі варіанти вхідних значень для компараторів, результати кожного компаратору окремо та загальної умови яку будемо передавати у керуючий автомат. Очікувані дані тестування занесені у табл. 3.3.

Таблиця 3.3 – Таблиця тестування компараторів

A	B	Вихід першого компаратора (L)	Вихід другого компаратора (M)	X
21	11	0	0	0
42	42	0	1	0
19	23	1	0	0
18	28	1	1	1

Тепер введемо значення з перших двох стовпців таблиці 3.3 та перевіримо чи збігаються отримані значення зі значеннями у таблиці.

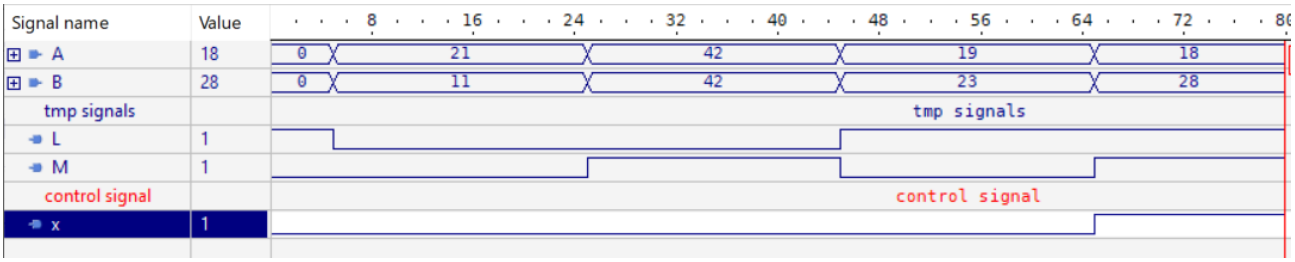

Рисунок 3.29 – Діаграма результату симуляції роботи компараторів

Передивившись результати на рис. 3.29 та порівнявши їх з цільовими значеннями у табл. 3.3 можемо дійти висновку що вони збігаються. В такому разі можемо стверджувати що компаратори працюють коректно.

3.4 Повна збірка і тестування спецобчислювача

Отже, зараз у нас є усі структурні елементи необхідні для того, щоб зібрати схему вже самого спецобчислювача. Усі структурні елементи було протестовано окремо і всі вони працюють коректно. Тепер нам залишається тільки підключити між собою їх усі, після чого перевірити коректність роботи повністю зібраного спецобчислювача. Його схему зображено на рис. 3.30.

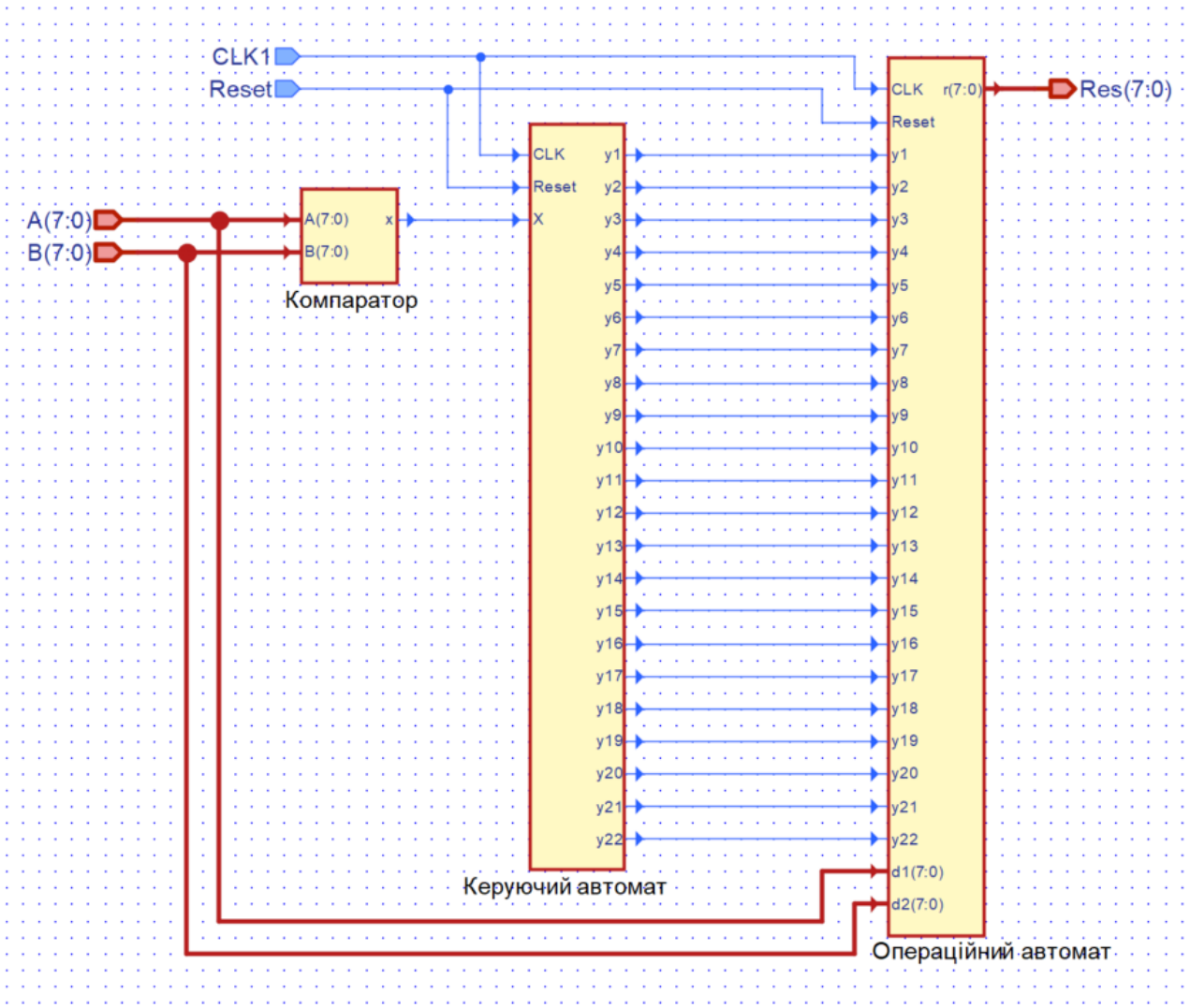

Рисунок 3.30 – Схема спецобчислювача

Отже, єдине що залишилось, це провести симуляцію роботи спецобчислювача та перевірити коректність отриманого результату. Візьмемо ті самі дані що і до цього. Це дозволить нам точно знати яким має бути результат оскільки вже власноруч за цими даними прораховували дані у табл. 3.1. Діаграма симуляції роботи спецобчислювача наведена на рис. 3.31.

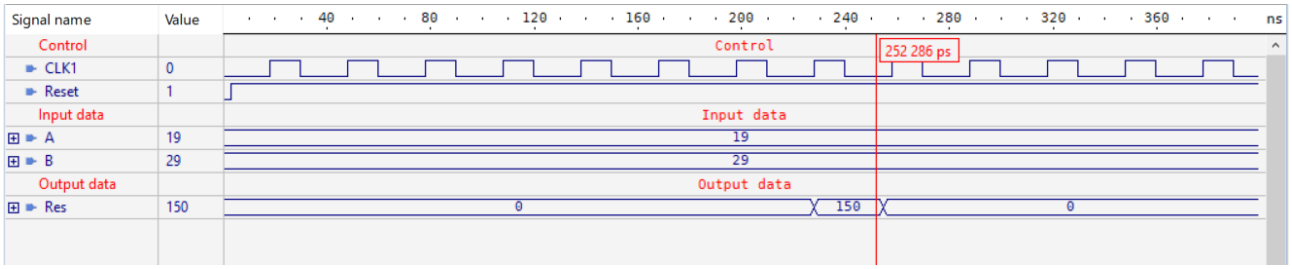

Рисунок 3.31 – Діаграма симуляції роботи спецобчислювача

В результаті, проаналізувавши діаграму на рис. 3.31 можемо зробити висновок що зібрані в один пристрій модулі працюють так само коректно як і коли ми тестували їх окремо один від одного.

У підсумку ми отримали восьмибітний спецобчислювач який обчислює два числа згідно з рівняннями зазначеними у формулі 2.1. Обчислення цих вхідних значень відбувається за десять тактів синхросигналу. Причому на дев’ятому такті відбувається вивантаження результату з пристрою на шину і наступним тактом відбувається очищення регістрів від тимчасово збережених даних які залишились у пам’яті.

ВИСНОВКИ

Під час роботи над дипломним проєктом було проведено аналіз архітектур ядер центральних процесорів різних поколінь та виробників. У ході цього аналізу було виявлено відмінності та особливості як між архітектурами різних поколінь в одного виробника, так і між архітектурами ядер конкурентних процесорів.

Для дослідження було визначено, який пристрій і на якій архітектурі буде доцільно розробити. У якості такого пристрою було обрано спецобчислювач із паралельною структурою операційної частини, для якого було сформовано рівняння, згідно яких він повинен виконувати обчислення. Наступним було розроблено алгоритм та проведено усі розрахунки, необхідні для моделювання цього спецобчислювача.

Виконано моделювання розроблених структурних елементів пристрою у середовищі програмного пакета. Після їх перевірки пристрій було зібрано у цільну систему.

Виконано тестування усіх структурних елементів пристрою засобами, наявними у програмному середовищі Aldec Active-HDL 13. Проведено тестування працездатності та коректності роботи спецобчислючвача зібраного вже у цільний пристрій. Процес тестування цих структурних елементів було докладно описано. За результатами тестування, додаткових перевірок та звірки із даними з етапу розробки можемо дійти висновку що пристрій працює цілком коректно. Підсумовуючи усе вище зазначене можемо дійти висновку, що розробку спецобчислювача для його подальшого використання у дослідженнях відмінностей архітектур завершено. Паралельна стурктура комбінаційної частини дозволяє прискорити обробку інформації, що сприяє підвищенню швидкодії пристрою в цілому, таким чином, мета роботи досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 AMD Zen 5 Core Architecture Breakdown At Hot Chips URL: https://wccftech.com/amd-zen-5-core-architecture-breakdown-hot-chips-new-chapter-high-performance-computing/

2 AMD Details ZEN Microarchitecture IPC Gains. URL: https://www.techpowerup.com/225271/amd-details-zen-microarchitecture-ipc-gains?cp

3 High Detail 'Zen 2' CPU core layout (With Zen 1 for reference). URL: https://www.eridonia-archives.com/post/high-detail-zen-2-cpucore-layout-with-zen-1-for-reference

4 Cannon Lake: Intel’s Forgotten Generation URL: https://chipsandcheese.com/p/cannon-lake-intels-forgotten-generation

5 AMD’s Zen 4 Part 1: Frontend and Execution Engine URL: https://chipsandcheese.com/p/amds-zen-4-part-1-frontend-and-execution-engine

6 ForgeFPGA Low-density FPGAs. URL: https://www.renesas.com/us/en/products/programmable-mixed-signal-asic-ip-products/forgefpga-low-density-fpgas

7 Sunny Cove: Intel’s Lost Generation URL: https://en.wikichip.org/wiki/intel/microarchitectures/sunny_cove

8 Intel’s Redwood Cove: Baby Steps are Still Steps URL: https://chipsandcheese.com/p/intels-redwood-cove-baby-steps-are-still-steps

9 Intel Core i9-13900K and i5-13600K Review: Raptor Lake Brings More Bite URL: https://www.anandtech.com/show/17601/intel-core-i9-13900k-and-i5-13600k-review

10 AMD "Zen" Core Architecture URL: https://www.amd.com/en/technologies/zen-core.html

11 AMD Zen 3 Ryzen Deep Dive Review URL: https://www.anandtech.com/show/16214/amd-zen-3-ryzen-deep-dive-review-5950x-5900x-5800x-and-5700x-tested/2

12 Intel Cove Microarchitecture URL: https://en.namu.wiki/w/%EC%9D%B8%ED%85%94%20%EC%BD%94%EB%B8%8C%20%EB%A7%88%EC%9D%B4%ED%81%AC%EB%A1%9C%EC%95%84%ED%82%A4%ED%85%8D%EC%B2%98

13 AMD Processor Specifications URL: https://www.amd.com/en/products/specifications/processors.html

14 Intel® Core™ Processors URL: https://www.intel.com/content/www/us/en/ark/products/series/122139/intel-core-processors.html

15 Рябенький В. М., Жуйков В. Я., Гулий В. Д. Цифрова схемотехніка: Навч. посібник. — Львів: "Новий Світ-2000", 2009. — 736 с.

16 Кочубей О.О. Прикладна теорія цифрових автоматів. Логічні основи : Навч. посіб./ О.О. Кочубей, О.В. Сопільник. - Д.: РВВ ДНУ; Вид-во ДНУ, 2009. - 264 с.

17 Бабич М. П., Жуков І. А. Комп’ютерна схемотехніка: Навчальний посібник. – К.: «МК-Прес», 2004. – 412 с.

18 Baranov S. Logic and System Desing of Digital Systems. – Tallinn: TUT Press, 2008. – 267 pp.

19 Sklyarov V., Sklyarova I., Barkalov A., Titarenko L. Synthesis and Optimization of FPGA – based Systems. – Berlin: Springer, 2014. – 432 pp.

Додаток А

Лістинг 1 – Код опису “генератора термів” на мові опису апаратури VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;

entity CU_term_gen is
  port(
       q0  : in STD_LOGIC;
       q1  : in STD_LOGIC;
       q2  : in STD_LOGIC;
       q3  : in STD_LOGIC;
       x   : in STD_LOGIC;
       y1  : out STD_LOGIC;
       y2  : out STD_LOGIC;
       y3  : out STD_LOGIC;
       y4  : out STD_LOGIC;
       y5  : out STD_LOGIC;
       y6  : out STD_LOGIC;
       y7  : out STD_LOGIC;
       y8  : out STD_LOGIC;
       y9  : out STD_LOGIC;
       y10 : out STD_LOGIC;
       y11 : out STD_LOGIC;
       y12 : out STD_LOGIC;
       y13 : out STD_LOGIC;
       y14 : out STD_LOGIC;
       y15 : out STD_LOGIC;
       y16 : out STD_LOGIC;
       y17 : out STD_LOGIC;
       y18 : out STD_LOGIC;
       y19 : out STD_LOGIC;
       y20 : out STD_LOGIC;
       y21 : out STD_LOGIC;
       y22 : out STD_LOGIC;
       d0  : out STD_LOGIC;
       d1  : out STD_LOGIC;
       d2  : out STD_LOGIC;
       d3  : out STD_LOGIC
  );
end CU_term_gen;

architecture CU_term_gen of CU_term_gen is

signal T : STD_LOGIC_VECTOR(27 downto 0);
```

```
begin

        --Terms--
        t(0)  <= not(q1) and not(q0);
        t(1)  <= not(q3) and q2 and q1;
        t(2)  <= q3 and not(q2) and not(q0);
        t(3)  <= not(q3) and q2 and not(q0);
        t(4)  <= not(q3) and not(q2) and q1 and not(q0) and x;
        t(5)  <= not(q1) and q0;
        t(6)  <= not(q3) and q2 and q1 and not(q0);
        t(7)  <= q3 and not(q2) and q1 and not(q0);
        t(8)  <= not(q3) and q2;
        t(9)  <= q2 and not(q1);
        t(10) <= not(q3) and q1 and q0;
        t(11) <= not(q2) and q1 and q0;
        t(12) <= q3 and not(q2);
        t(13) <= q3 and not(q1);
        t(14) <= not(q3) and q2 and q1 and q0;
        t(15) <=not(q3) and not(q2) and q1 and not(q0) and not(x);
        t(16) <= not(q3) and not(q2) and not(q1) and q0;
        t(17) <= not(q3) and not(q2) and q1 and not(q0);
        t(18) <= not(q3) and not(q2) and q1 and q0;
        t(19) <= q3 and not(q2) and not(q1) and not(q0);
        t(20) <= not(q3) and q2 and not(q1) and not(q0);
        t(21) <= not(q3) and q2 and not(q1) and q0;
        t(22) <= q3 and not(q2) and not(q1) and q0;
        t(23) <= q3 and not(q2) and q1 and q0;
        t(24) <= q3 and q2 and not(q1) and not(q0);
        t(25) <= q3 and q2 and not(q1) and q0;
        t(26) <= q3 and q2 and q1 and not(q0);


        --trigers funcs--
        d0 <= t(0) or t(1) or t(2) or t(3) or t(4);
        d1 <= t(5) or t(6) or t(7) or t(4);
        d2 <= t(8) or t(9) or t(10) or t(11);
        d3 <= t(12) or t(13) or t(14) or t(15);


        --control signals--
        y1  <= t(16);
        y2  <= t(16);
        y3  <= t(17);
        y4  <= t(17);
        y5  <= t(18) or t(19);
        y6  <= t(18);
        y7  <= t(18);
        y8  <= t(20);
        y9  <= t(20);
        y10 <= t(21);
        y11 <= t(21);
        y12 <= t(21);
        y13 <= t(6);
```

```
        y14 <= t(14);
        y15 <= t(19);
        y16 <= t(19);
        y17 <= t(22);
        y18 <= t(7);
        y19 <= t(23);
        y20 <= t(24);
        y21 <= t(25);
        y22 <= t(26);

end CU_term_gen
```