

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ ТА
ЗАСОБІВ ПРОЄКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ

RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS AND
TOOLS FOR DESIGNING DIGITAL DEVICES

Виконав: студент 2 курсу, групи КНТ-211м
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

Троценко О.С.

(прізвище та ініціали)

Керівник Пархоменко А.В.

(прізвище та ініціали)

Рецензент Зеленьова І.Я.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет комп'ютерних наук та технологій

Кафедра програмних засобів

Ступінь вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма (спеціалізація) Системи штучного інтелекту

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

С.О. Субботін

“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Троценка Олексія Сергійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів та засобів проєктування цифрових пристроїв. Research and Software Implementation of Methods and Tools for Designing Digital Devices

керівник проєкту (роботи) Пархоменко Анжеліка Володимирівна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “04” листопада 2022 року №363

2. Строк подання студентом проєкту (роботи) 1 грудня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Дослідження методів та засобів автоматизованого проєктування цифрових пристроїв. 2. Розробка програмного забезпечення цифрових пристроїв засобами віртуальної та віддаленої інженерії. 3. Тестування програмного забезпечення цифрових пристроїв засобами віртуальної та віддаленої інженерії.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3 Основна частина	Пархоменко А.В., доцент		
Нормоконтроль	Камінська Ж.К., асистент		

7. Дата видачі завдання “5” вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Дослідження методів та засобів вирішення задачі.	4–5 тижні	Розділ 1
4	Розробка удосконаленого методу автоматизованого проєктування цифрових пристроїв	6-7 тиждень	Розділ 1
5	Розробка програмного забезпечення.	8–10 тижні	Розділ 2
6	Тестування програмного забезпечення.	11-12 тиждень	Розділ 3
7	Оформлення пояснювальної записки та документів до неї.	13-14 тижні	ПЗ, Додатки
8	Нормоконтроль та рецензування.	15 тиждень	
9	Захист роботи.	16 тиждень	

Студент

(підпис) Троценко О. С.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Пархоменко А.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
116 с., 108 рис., 3 табл., 2 додатки, 29 джерел.

АВТОМАТИЗОВАНЕ ПРОЄКТУВАННЯ, ВІДДАЛЕНА
ЛАБОРАТОРІЯ, ЦИФРОВИЙ ПРИСТРІЙ, ПЛАТА ШВИДКОГО
ПРОТОТИПУВАННЯ.

Об'єкт дослідження – процес автоматизованого проєктування цифрових пристроїв.

Предмет дослідження – методи та засоби автоматизованого проєктування цифрових пристроїв.

Мета роботи – дослідження та програмна реалізація методів та засобів віртуальної та віддаленої інженерії для розробки та тестування комбінаційних цифрових пристроїв.

Матеріали, методи та технічні засоби: середовище розробки Quartus II, мова опису апаратури VHDL, середовище моделювання ModelSim, плата прототипування віддаленої лабораторії GOLDi, персональний комп'ютер з процесором Intel Core 2 Duo під управлінням операційної системи Microsoft Windows 10.

Результати. Запропоновано удосконалений метод автоматизованого проєктування комбінаційних цифрових пристроїв на основі технологій віртуальної та віддаленої інженерії. Розроблено програмне забезпечення для бібліотеки комбінаційних цифрових пристроїв та файли тестування для них.

Висновки. Інтегроване середовище розробки на основі Quartus II та віддаленої лабораторії GOLDi надає зручний інструментарій для ефективного створення та тестування комбінаційних цифрових пристроїв.

Галузь використання – практика автоматизованого проєктування цифрових пристроїв.

ABSTRACT

Explanatory note to the master thesis: 116 pages, 108 figures, 3 tables, 2 appendixes, 29 sources.

COMPUTER AIDED DESIGN, REMOTE LABORATORY, DIGITAL DEVICE, RAPID PROTOTYPING BOARD.

The object of study is the process of computer aided design of digital devices.

The subject of the study are methods and tools of digital devices computer aided design.

The goals of the work are research and software implementation of virtual and remote engineering methods and tools for the developing and testing of digital combinational devices.

Materials, methods, and tools: Quartus II development environment, VHDL hardware description language, simulator ModelSim, GOLDi remote laboratory prototyping board, personal computer with an Intel Core 2 Duo processor running the Microsoft Windows 10 operating system.

Results. The improved method of computer aided design of digital combinational devices based on virtual and remote engineering technologies is proposed. Software for digital combiner devices library as well as test schemes for them have been developed.

Conclusions. The integrated development environment based on Quartus II and GOLDi remote laboratory provides a convenient toolkit for efficient creation and testing of digital combinational devices.

The field of application is the practice of digital devices computer aided design.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК.....	8
ВСТУП.....	9
1 ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ АВТОМАТИЗОВАНОГО ПРОЄКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ.....	10
1.1 Огляд компонентів цифрових комунікаційних систем	10
1.2 Дослідження особливостей процесу проєктування цифрових пристроїв .	13
1.2.1 Класифікація цифрових пристроїв	13
1.2.2 Базис розробки цифрових пристроїв	16
1.2.3 Алгоритм проєктування цифрових пристроїв	18
1.3 Засоби проєктування цифрових пристроїв	22
1.3.1 Вибір середовища розробки	22
1.3.2 Вибір мови опису апаратури	25
1.4 Метод автоматизованого проєктування цифрових комбінаційних пристроїв на основі технологій віртуальної та віддаленої інженерії.....	27
1.5 Постановка задач магістерської роботи	35
2 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЦИФРОВИХ ПРИСТРОЇВ ЗАСОБАМИ ВІРТУАЛЬНОЇ ТА ВІДДАЛЕНОЇ ІНЖЕНЕРІЇ.....	36
2.1 Опис особливостей реалізації бібліотеки КЦП	36
2.1.1 Структура бібліотеки та шаблон опису елементів.....	36
2.1.2 Алгоритм розробки елементів бібліотеки КЦП	38
2.2 Розробка дешифратора 7-сегментного індикатору	50
2.2.1 Розробка структурної схеми дешифратора у середовищі Quartus	54
2.2.2 Опис дешифратора на мові VHDL	57
2.3 Розробка арифметичних пристроїв.....	57
2.3.1 Розробка 1-бітного повного суматора.....	60
2.3.2 Розробка 4-бітного суматора.....	62
2.3.3 Розробка BCD-суматора	63

2.3.4 Розробка двійкового суматора/віднімача	66
2.3.5 Розробка перетворювача доповняльного коду	67
2.4 Розробка компаратору	69
2.4.1 Розробка 8-бітного компаратору	74
2.4.2 Розробка 4-бітного ітераційного компаратору	76
2.5 Розробка арифметико-логічного пристрою	79
2.5.1 Розробка структурної схеми АЛП	80
2.5.2 Розробка структурних блоків АЛП у середовищі Quartus	83
2.5.3 Розробка 4-бітного АЛП у середовищі Quartus	87
2.5.4 Опис 4-бітного АЛП на мові VHDL	88
2.6 Висновки до розділу 2	89
3 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЦИФРОВИХ ПРИСТРОЇВ ЗАСОБАМИ ВІРТУАЛЬНОЇ ТА ВІДДАЛЕНОЇ ІНЖЕНЕРІЇ	90
3.1 Початок роботи з віддаленою лабораторією GOLDi	90
3.2 Тестування розроблених цифрових пристроїв	93
3.2.1 Тестування дешифратору 7-сегментного індикатору	93
3.2.2 Тестування арифметичних пристроїв	93
3.2.3 Тестування компараторів	95
3.2.4 Тестування арифметико-логічного пристрою	96
3.3 Висновки до розділу 3	97
ВИСНОВКИ	98
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	99
ДОДАТОК А Текст програми	102
ДОДАТОК Б Слайди презентації	108

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BCD	– Binary Code Decimal;
BPU	– Bus Protection Unit;
CPLD	– Complex Programmable Logic Device
ECP	– Experimental Control Panel;
FPGA	– Field-Programmable Gate Array;
GOLDi	– Grid of Online Lab Devices Ilmenau;
HDL	– Hardware Description Language;
PAL	– Programmable Array Logic;
RIA	– Rich Internet Application;
VHDL	– VHSIC Hardware Description Language;
АЛП	– арифметико-логічний пристрій;
ВЛ	– віддалена лабораторія;
КЦП	– комбінаційний цифровий пристрій;
ЛБ	– логічний блок;
ЛЕ	– логічний елемент;
ПЛІС	– програмована логічна інтегральна схема;
ПП	– плата прототипування;
САПР	– система автоматизованого проєктування;
ЦП	– цифровий пристрій.

ВСТУП

Сучасний світ суттєво залежить від цифрових комунікацій. Усе, що людство використовує для будь-якого виду зв'язку, так чи інакше пов'язане з цифровими технологіями [1]. Саме тому важливим є розвиток різних аспектів цифрового зв'язку, процесів передачі цифрового сигналу та технологій автоматизованого проєктування компонентів цифрових комунікаційних систем.

Швидкий розвиток Інтернету та інформаційних технологій має великий вплив на методи та засоби комп'ютерного проєктування. Зокрема, нові можливості надають технології віртуальної та віддаленої інженерії.

Віддалені лабораторії дозволяють проводити дослідження фізичного прототипу проєктованого об'єкта в режимі 24/7 онлайн. Переваги такого підходу очевидні: не потрібно витрачати кошти на обладнання та його обслуговування; тестування розробки проводиться не у віртуальному середовищі, а з використанням реального обладнання. Одним з прикладів віддаленої онлайн лабораторії є GOLDi, що розроблена в Технічному університеті Ільменау, Німеччина, а частина її експериментального обладнання також знаходиться у Національному Університеті «Запорізька Політехніка» на базі кафедри програмних засобів.

В той же час, створення та дослідження віртуального прототипу проєктованого об'єкта шляхом моделювання процесів його роботи дозволяє виявити та виправити помилки проєкту на ранніх етапах та знизити витрати на подальше доопрацювання, коли створення фізичного прототипу є ускладненим.

Тому, подальший розвиток та впровадження технологій віртуальної та віддаленої інженерії в процес автоматизованого проєктування компонентів цифрових комунікаційних систем є актуальною задачею.

Мета роботи – дослідження та програмна реалізація методів та засобів віртуальної та віддаленої інженерії для розробки та тестування комбінаційних цифрових пристроїв.

1 ДОСЛІДЖЕННЯ МЕТОДІВ ТА ЗАСОБІВ АВТОМАТИЗОВАНОГО ПРОЄКТУВАННЯ ЦИФРОВИХ ПРИСТРОЇВ

1.1 Огляд компонентів цифрових комунікаційних систем

Цифровий зв'язок визначається як процес, за допомогою якого цифрові пристрої передають інформацію в цифровому вигляді. Цифровий зв'язок має багато переваг перед аналоговим [1]:

- конкретний рівень сигналу цифрового сигналу не дуже важливий. Завдяки цьому на цифрові сигнали майже не впливають недоліки електронних систем, які можуть зіпсувати аналогові сигнали;
- процес налаштування цифрових сигналів легший, ніж аналогових сигналів;
- шифрування за допомогою кодів краще працює в цифрових сигналах;
- цифрові схеми більш послідовні та надійні;
- цифрові схеми зазвичай простіше проєктувати;
- вартість виробництва цифрових схем нижча, ніж аналогових схем;
- цифрові сигнали не спотворюються шумом, перешкодами та викривленнями;
- перехресне спілкування дуже рідкісне явище в цифровому зв'язку;
- передача даних на великі відстані є простішою та дешевшою завдяки цифровим сигналам;
- апаратна реалізація в цифрових схемах набагато гнучкіша порівняно з аналоговими схемами;
- спосіб об'єднання цифрових сигналів за допомогою мультиплексування з поділом часу простіший, ніж метод об'єднання аналогових сигналів за допомогою мультиплексування з поділом частот;
- цифрові сигнали можна зберегти та витягнути легше, ніж аналогові сигнали;

- більшість цифрових схем мають майже загальні методи кодування, тому подібні пристрої можна використовувати для багатьох цілей;
- цифровий зв'язок підтримує багатовимірну передачу одночасно;
- можливості каналу ефективно використовуються цифровими сигналами;
- сигнал залишається незмінним, оскільки для зміни властивостей імпульсу потрібне велике переривання, що є дуже складним [1].

Загалом кожна система цифрового зв'язку складається з набору компонентів, що виконують певні функції (рис.1.1) [1].



Рисунок 1.1 – Основні компоненти цифрової системи зв'язку [1]

До основних компонентів відносяться наступні [1]:

- джерело аналогового сигналу, наприклад, звуковий сигнал;
- вхідний перетворювач – приймає фізичний вхідний сигнал і перетворює його на електричний сигнал, наприклад, мікрофон;
- аналого-цифровий перетворювач – додатково обробляє та перетворює електричний сигнал від вхідного перетворювача на цифровий сигнал;
- кодувальник джерела, або вхідний кодер стискає дані до найменшої кількості бітів, що сприяє ефективній роботі смуги пропускання;

- кодер каналів – виконує кодування для виправлення помилок, оскільки під час передачі сигналу він спотворюватися. Щоб уникнути цього, каналний кодер додає деякі біти виправлення помилок до переданих даних;
- цифровий модулятор – тут сигнал, який має бути переданий, модулюється несучою, що використовується для ефективної передачі даних на великі відстані;
- канал зв'язку забезпечує шлях для сигналу та дозволяє аналоговому сигналу передавати від передавача до приймача;
- цифровий демодулятор – це місце, звідки на стороні приймача починається процес отримання даних. Отриманий сигнал демодулюється і знову перетворюється з аналогового в цифровий, фактично відбувається відновлення сигналу;
- цифро-аналоговий перетворювач – цифровий сигнал, отриманий з несучої, потім знову перетворюється в аналоговий, щоб сигнал міг ефективно проходити через канал або середовище;
- декодер каналу виконує виправлення помилок після виявлення послідовності;
- декодер джерела – отриманий сигнал знову оцифровується шляхом дискретизації та квантування. Це робиться для отримання чистого цифрового виходу без втрати інформації. Декодер джерела знову створює вихідний сигнал джерела;
- вихідний перетворювач – це останній блок, який перетворює сигнал у вихідну форму (яка була на вході передавача). Він перетворює електричний сигнал у фізичний вихід, наприклад, спікер;
- вихідний сигнал – це результат, заради якого виконується весь процес, наприклад, отримано звуковий сигнал [1].

1.2 Дослідження особливостей процесу проектування цифрових пристроїв

1.2.1 Класифікація цифрових пристроїв

Цифровий пристрій (ЦП) – це електронний пристрій, який може створювати, генерувати, надсилати, обмінюватися, передавати, отримувати, зберігати, відображати або обробляти інформацію [2].

Таким чином, ЦП – це частина фізичного обладнання, яке використовує цифрові дані, наприклад надсилаючи, отримуючи, зберігаючи або обробляючи їх [3].

ЦП обробляє електронні сигнали, які представляють одиницю («увімкнено») або нуль («вимкнено»). Увімкнений стан представлений наявністю електронного сигналу, а вимкнений стан представлений відсутністю електронного сигналу. Група ЦП включає такі пристрої, як [4]:

- настільні комп'ютери;
- портативні комп'ютери;
- мобільні телефони;
- планшетні комп'ютери;
- електронні книги;
- запам'ятовуючі пристрої, такі як флеш-накопичувачі;
- пристрої введення, такі як клавіатури, миші та сканери;
- пристрої виведення, такі як принтери та динаміки [4].

ЦП – це технічний пристрій, призначений для отримання та обробки інформації у цифровій формі, використовуючи цифрові технології. ЦП поділяються на такі основні види (рис. 1.2) [5]:

а) за способом введення та виведення:

- 1) послідовні пристрої (інформація вводиться послідовно розряд за розрядом);

2) паралельні (інформація знімається одночасно по всіх розрядах);

3) послідовно-паралельні пристрої.

б) за принципом дії:

1) комбінаційні ЦП (автомати без пам'яті). В них стан вихідних сигналів однозначно визначається лише діючою нині комбінацією вхідних сигналів і не залежить від значень змінних, які діяли на входах раніше. До таких пристроїв належать кодери, декодери, мультиплексори, демультиплексори, суматори, компаратори;

2) послідовні ЦП (автомати з пам'яттю). Вихідні сигнали визначаються не тільки чинною наразі комбінацією вхідних сигналів, а й попереднім станом пристрою, залежним від вхідних змінних, що діяли на входах раніше. До них відносяться тригери та всі пристрої, побудовані на тригерах: регістри, лічильники, оперативні запам'ятовуючі пристрої, постійні запам'ятовуючі пристрої [5].

Фактично, цифрові логічні схеми поділяються на два класи:

- комбінаційні схеми, що не містять елементів пам'яті;
- послідовні схеми, які містять елементи пам'яті.

Таким чином, узагальнена схема класифікації ЦП може бути представлена як наведено на рисунку 1.2.

Комбінаційні цифрові пристрої (КЦП) є складовими частинами контролерів, комп'ютерів, а також інших цифрових пристроїв та виконують перетворення цифрових кодів з однієї системи в іншу. КЦП реалізуються у вигляді інтегральних мікросхем. При цьому застосовуються як окремі логічні елементи, що реалізують елементарні булеві функції, так і їх комбінації. На входи КЦП подають два типи сигналів: інформаційні логічні сигнали та сигнали управління, що визначають порядок передачі вхідних сигналів на вихід з синхронізацією цього процесу [6].

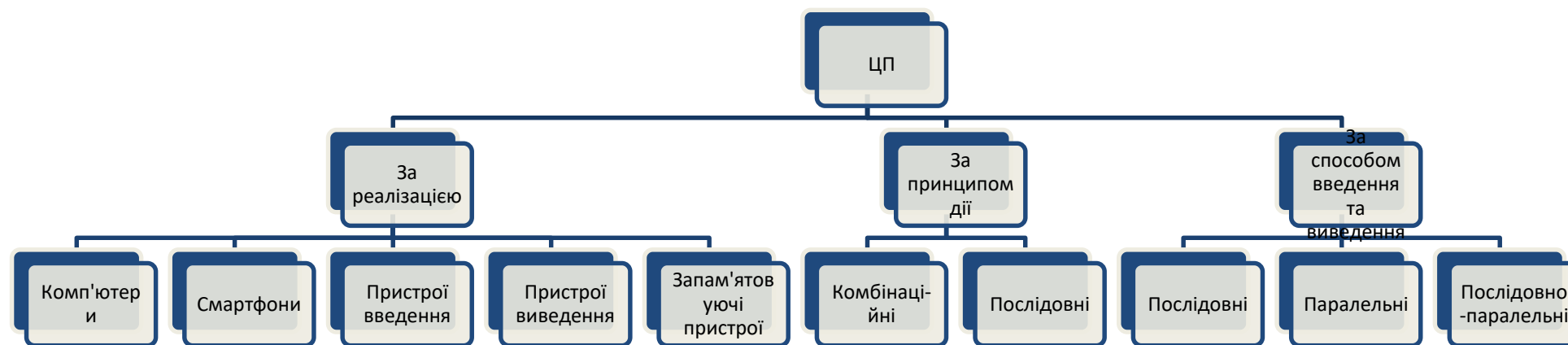


Рисунок 1.2 – Класифікація цифрових пристроїв

За функціональною ознакою КЦП діляться на такі класи:

- суматори;
- шифратори та дешифратори;
- цифрові компаратори;
- мультиплексори та демультимплексори (селектори);
- перетворювачі кодів;
- програмовані логічні матриці;
- перемножувачі;
- арифметико-логічні пристрої (АЛП) [7].

Таким чином, комбінаційні пристрої – електронні схеми, вихідний сигнал яких залежить від поєднання вхідних сигналів. Будь-яка зміна вхідного сигналу спричиняє зміну сигналу на виході.

1.2.2 Базис розробки цифрових пристроїв

Основним базисом, у якому виконується розробка КЦП є програмовані логічні інтегральні схеми (ПЛІС). ПЛІС є узагальненням різноманітних реалізацій інтегральних схем, структура яких задається не на стадії розробки, а під час програмування. Двома основними різновидами ПЛІС є Complex Programmable Logic Devices (CPLD) та Field Programmable Gate Array (FPGA).

CPLD являють собою так звані програмовані логічні схеми в цифровій техніці. Вони є технологічним наступником програмованих логічних матриць (Programmable Array Logic, PAL) і мають простішу структуру, ніж значно складніші програмовані вентильні матриці (FPGA) [8].

Таким чином, CPLD є програмованою інтегральною схемою, що знаходиться у діапазоні складності між програмованими логічними матрицями та FPGA, об'єднуючи їх особливості. Головним елементом CPLD є матриця макрокомірок, у якій містяться диз'юнктивні нормальні форми реалізованих функцій та більш складні логічні операції. CPLD складається з багатьох

простих програмованих логічних пристроїв, або макроклітин, що з'єднані через шини. Однорідна структура дозволяє точно визначати час пропускання, що є основною відмінністю від FPGA. Ще одна відмінність, полягає в тому, що після конфігурації програма залишається в CPLD і її не потрібно перезавантажувати при кожному запуску [8].

Завдяки великій кількості входів/логічних блоків CPLD особливо підходять для вирішення складної паралельної комбінаторної логіки І/АБО, яка вимагає великої кількості входів і виходів. У той же час, кількість пам'яті (тригерів), необхідних при використанні CPLD, має бути невеликою, оскільки зазвичай лише один тригер доступний як регістр для кожного вхідного або вихідного виводу. Цифрові схеми зазвичай потребують багатьох регістрів, таких як регістри зсуву або цифрові лічильники, тому можуть бути ефективно реалізовані в CPLD лише до певної міри [8].

Відомими виробниками є Xilinx, Altera, Lattice, Actel, Lucent, Cypress, Atmel або Quicklogic [8].

FPGA є подальшим розвитком PLD і використовуються в багатьох областях цифрової електроніки. FPGA - це цифрова технологічна інтегральна схема, у яку можна завантажити логічну схему. На відміну від програмування комп'ютерів, мікроконтролерів або елементів керування, термін програмування FPGA стосується не лише специфікації часових послідовностей, але, перш за все, визначення бажаної структури схеми. Це формулюється за допомогою мови опису апаратного забезпечення та перекладається у файл конфігурації за допомогою програмного забезпечення генератора, яке визначає, як фізичні елементи в FPGA мають бути з'єднані між собою. Різні схеми та функції можуть бути реалізовані в FPGA шляхом налаштування внутрішніх елементів. Вони варіюються від схем низької складності, таких як простий синхронний лічильник або інтерфейси для цифрових компонентів, до дуже складних схем, таких як контролери пам'яті та мікропроцесори [9].

FPGA використовуються у всіх сферах цифрових технологій, але особливо там, де важлива швидка обробка сигналів і гнучкі зміни в схемі, наприклад, щоб мати можливість робити подальші вдосконалення реалізованих функцій без необхідності безпосередньо змінювати апаратне забезпечення [9].

На відміну від CPLD, FPGA складаються з матриці програмованих логічних блоків та ієрархічної структури з'єднань між ними, що має властивість до реконфігурації. Головною відмінністю FPGA від CPLD є наявність у останньої постійної конфігураційної пам'яті, що зберігає конфігурацію схеми після припинення постачання живлення. Порівняльну характеристику FPGA та CPLD наведено в таблиці 1.1 [10]-[12].

Таблиця 1.1 – Порівняльна характеристика базисів розробки [10] - [12]

Характеристика	CPLD	FPGA
Гнучкість	низька	висока
Ціна	низька	висока
Безпека	висока	низька
Швидкість	висока	висока
Складність	низька	висока
Застосування	просте	складне

1.2.3 Алгоритм проєктування цифрових пристроїв

Схему алгоритму проєктування ЦП наведено на рисунку 1.3 [13].

Етап 1. Специфікація проєкту. Надає повну інформацію про функціональні можливості та складність проєкту, тип інтерфейсу, тактову частоту, деталі реалізації та вимоги. Визначає функціональність ЦП та характеристики потужності/площі/швидкості. Потім на основі аналізу вимог створюється архітектура дизайну, що зазвичай розділяється на підмодулі, які взаємодіють один з одним, утворюючи модуль системного рівня. Ця ієрархічна



Рисунок 1.3 – Узагальнений алгоритм проєктування ЦП [13]

організація полегшує командну роботу, дозволяє відстежувати помилки та повторно використовувати модулі для інших проєктів [13]. Етап 2. Розробка вхідного опису. Існують різні формати, у яких можуть бути представлені вхідні файли проєкту (VHDL/Verilog/SystemVerilog). Можна також завантажити дані в середовище CAD у схематичному форматі, який легше читати та в основному він більш ефективний для невеликих проєктів. Файли проєкту також можуть бути представлені з використанням апаратної мови (Hardware Description Language, HDL), яка більше підходить для складних систем. Також задаються файли обмежень проєкту.

Етап 3. Поведінкове моделювання. Далі робота коду перевіряється шляхом запуску моделювання поведінки, яке відображатиме форму сигналу входів і виходів. Щоб переконатися, що код працює належним чином, можна протестувати різні комбінації вводів. Цей етап призначений лише для того, щоб переконатися, що код не містить помилок [13].

Етап 4. Синтез проєкту. Вихідні коди та обмеження проєкту, включаючи часові та фізичні обмеження, передаються в інструмент синтезу. Обмеження додаються, щоб гарантувати, що функціональність і продуктивність проєкту відповідають вимогам проєктувальника. Інструмент перетворює код RTL в опис на рівні вентилів, який піддається логічній оптимізації та скороченню. Потім логічні вентиля зіставляються з існуючими технологічними бібліотеками, щоб зрештою отримати оптимізований список з'єднань на рівні шлюзів, який може зрозуміти наступний інструмент. На цьому етапі також створюються звіти, що містять інформацію про використання комірок, площу та час, і виводяться користувачеві, щоб їх можна було перевірити на наявність удосконалень або порушень щодо будь-яких обмежень проєкту. Результатом цього етапу є список з'єднань синтезованих модулів, який використовується для перевірки того, чи всі призначені блоки справді можна синтезувати [13].

Етап 5. Функціональна верифікація. Полягає в тому, щоб підтвердити, чи є введений проєкт логічно та функціонально правильним чи ні, і це

перевіряється за допомогою заданого стимулу та порівнюється з очікуваним результатом.

Етап 6. Реалізація проєкту. Здійснюється у три фази (рис. 1.4). Спочатку йде фаза «Переклад», яка об'єднує всі попередньо створені списки мереж і файли обмежень в один великий файл [13].

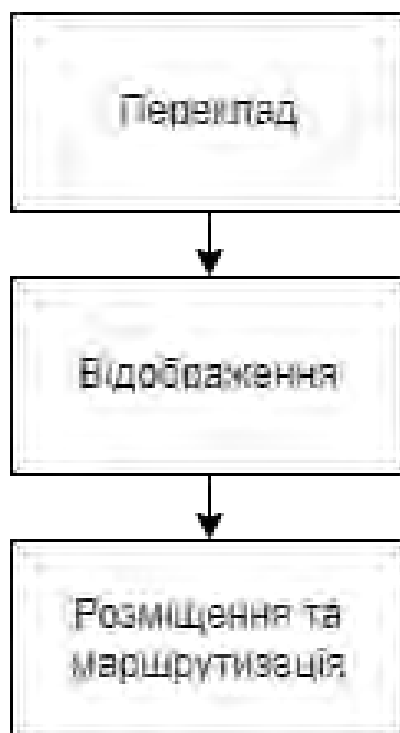


Рисунок 1.4 – Фази реалізації проєкту [13]

Також призначаються порти фактичним контактам, перемикачам і кнопкам FPGA, дотримуючись обмежень конструкції. Наступною є фаза «Відображення», яка розділяє список з'єднань на підблоки та відображає їх у фактичних блоках. І, нарешті, остання фаза – «Розміщення та маршрутизація». Виконується розміщення проєктних підблоків у призначених для них блоках, а потім будуються маршрути між блоками. Розміщення та маршрутизація мають здійснюватися з урахуванням часових обмежень, наприклад; розміщення поблизу контактів введення/виведення може допомогти заощадити час [13].

Етап 7. Статичний часовий аналіз часу. Виконується після реалізації проєкту, щоб перевірити, чи він відповідає часовим обмеженням. Він перевіряє

всі можливі шляхи поширення сигналу на наявність затримок. Потім він повідомляє про найдовшу можливу затримку, яка називається критичним шляхом, щоб переконатися, що вона не порушує бажану продуктивність синхронізації. Під час аналізу також перевіряється час утримання та час налаштування та надсилаються повідомлення про будь-які порушення.

Етап 8. Зворотна анотація. На цьому кроці перевіряються звіти, створені після етапу реалізації, на наявність порушень або покращень, які необхідно внести. У разі порушення часових або фізичних обмежень потрібно відредагувати або обмеження, або сам дизайн. Після цього проєкт повторно синтезується та повторно реалізується. Цей процес повторюється, доки не будуть виконані всі правила проєктування [13].

Етапи 9-10. Генерація файлів бітового потоку. Потім реалізований проєкт потрібно перетворити на бітовий потік за допомогою інструменту генерації бітів, щоб платформа могла зрозуміти дизайн. Потім файл Bitstream зберігається на карті пам'яті, тому його може завантажити плата. Плата повинна бути підключена до пристрою за допомогою USB. Потім файл Bitstream можна використовувати для програмування плати безпосередньо через апаратний менеджер.

Етап 11. Внутрішньосхемна перевірка. Після того, як файл Bitstream завантажується на плату, виконується внутрішньо-схемна перевірка, щоб переконатися, що відбулося правильне впровадження схеми. Це робиться за допомогою IP-адрес для налагодження обладнання, інтегрованих у плату [13].

1.3 Засоби проєктування цифрових пристроїв

1.3.1 Вибір середовища розробки

Altera Quartus Prime (Quartus II) – це середовище розробки програмного забезпечення для ПЛІС виробництва Intel.

Проєктування в середовищі Quartus II включає такі етапи [14]:

- введення проєкту (Design Entry) – бажана схема проєкту задається або графічним способом, або з використанням мов опису апаратних засобів, таких як Verilog HDL, VHDL та ін.;
- синтез (Synthesis) – проєкт, що вводиться, синтезується в схему, яка складається з (ЛЕ) логічних елементів і (ЛБ) логічних блоків в мікросхемі ПЛІС (програмованої логічної інтегральної схеми);
- функціональне моделювання (Functional Simulation) – синтезована схема тестується щодо коректності функціонування у вбудованому симуляторі, який моделює залежність стану (вибраних розробником) сигналів схеми від часу. Це моделювання не враховує тимчасових затримок сигналів (званих логічними перегонами) між ЛЕ/ЛБ мікросхеми ПЛІС;
- трасування (Fitting) – інструмент трасування САПР обчислює оптимальне розміщення та з'єднання ЛЕ/ЛБ, визначених у списку з'єднань реальної мікросхеми ПЛІС. Трасувальник також обирає дроти або «маршрут руху» у чіпі для реалізації необхідних зв'язків між заданими ЛЕ/ЛБ;
- часовий аналіз (Timing Analysis) – аналізує затримки розповсюдження сигналів уздовж різних шляхів у трасованій логічній схемі для обчислення наявності/відсутності логічних перегонів, щоб сигнали з різних ЛБ приходили одночасно в той чи інший кінцевий вузол схеми;
- часове моделювання (Timing Simulation) – схема тестується для перевірок функціональності та тимчасових обмежень, але на відміну від функціонального моделювання, тут для виявлення наявності чи відсутності логічних перегонів враховуються реальні затримки вибраних сигналів;
- програмування та конфігурація (Programming and Configuration) - розроблена схема розміщується в мікросхемі ПЛІС шляхом програмування електронних зв'язків між конфігурованими ЛЕ/ЛБ, що реалізується шляхом передачі конфігураційного файлу з комп'ютера або мікросхеми ПЛІС, або додаткову (не вбудовану в ПЛІС) пам'ять, якщо така є у налагоджувальному комплекті.

Vivado Design Suite — це середовище для розробки програмного забезпечення для ПЛІС виробництва Xilinx. Vivado призначений для синтезу та аналізу проєктів на мові опису апаратури (hardware description language, HDL).

Інтегроване середовище проєктування Vivado забезпечує користувача інтуїтивно зрозумілим графічним з потужними функціями. Усі інструменти та параметри інструментів написані на мові інструментів (Tcl), що дозволяє використовувати як оболонку Vivado IDE, так і Vivado Design Suite Tcl [15].

Користувач може підвищити продуктивність проєктування, використовуючи нові алгоритми, які надає Vivado IDE, зокрема:

- розробку на рівні регістрових передач (RTL) на мовах VHDL, Verilog та SystemVerilog;
- інтеграцію ядер інтелектуальної власності (ip cores);
- поведінкове, функціональне та часове моделювання за допомогою симулятора Vivado;
- синтез Vivado;
- трасування Vivado;
- логічний аналізатор для налагодження;
- статичний аналіз часу;
- розробку топології мікросхеми;
- детальне розміщення та модифікацію маршрутів;
- генерацію бітового потоку.

Оскільки у віддаленій лабораторії GOLDi використовується CPLD фірми Altera, то розробку доцільно провести у середовищі проєктування Quartus II.

1.3.2 Вибір мови опису апаратури

Як показали проведені дослідження, існуючі мови опису апаратури діляться на дві основні групи: Verilog і VHDL, кожна з яких має свої особливості застосування.

Мова VHSIC Hardware Description Language (VHDL) використовується у багатьох системах для моделювання цифрових схем, проєктування програмованих логічних інтегральних мікросхем, базових матричних кристалів, замовних інтегральних схем. З точки зору програмування, мова VHDL складається з двох компонентів: загальноалгоритмічної та проблемно-орієнтованої [16].

Загальноалгоритмічна складова VHDL – це мова, близька за синтаксисом та семантиці до сучасних мов програмування. Мова відноситься до класу строго типізованих. Окрім вбудованих типів (пакет standard), користувач може вводити свої типи даних. Окрім скалярних даних, можна використовувати агреговані, такі як масиви array, бітові вектори bit_vector, символьні строки string, записи record, файли file [15].

Оператори послідовного виконання VHDL можуть використовуватися у описі процесів, процедур та функцій. До таких операторів відносяться [16]:

- оператор привласнення змінної;
- послідовний оператор призначення сигналу;
- послідовний оператор ствердження, умовний оператор;
- оператор вибору;
- оператор циклу, пустий оператор;
- оператор повернення з процедури;
- оператор послідовного виклику процедури.

Фрагменти опису, які можуть незалежним чином аналізуватися компілятором і при відсутності помилок розміщуватися у бібліотеці проєкту, називаються пакетами (design unit). Такими пакетами можуть бути оголошення

інтерфейсу об'єкту, оголошення архітектури, оголошення конфігурації, оголошення інтерфейсу пакету та оголошення пакету [16].

Проблемно-орієнтована компонента дозволяє описувати цифрові системи у звичних для розробника поняттях і термінах. Сюди можна віднести:

- поняття модульного часу;
- дані типу, що дозволяють вказувати час затримки у фізичних одиницях;
- дані виду сигнал, значення яких змінюється не миттєво, як у звичайних змінних, а із вказаною затримкою, а також спеціальні операції та функції над ними;
- засоби оголошення об'єктів та їх архітектур [16].

Мова опису апаратури Verilog є однією з двох найбільш популярних мов опису, що використовуються розробниками інтегрованих мікросхем. Мова Verilog може використовуватися для розробки на чотирьох рівнях абстракції:

- алгоритмічний рівень (оператори вибору, циклів);
- рівень регістрових передач;
- рівень вентилів;
- транзисторний рівень.

Мова Verilog описує цифрову систему як набір модулів. Кожен з цих модулів має інтерфейс з іншими модулями, а також опис його вмісту. Модуль являє собою логічний блок, який можна описати шляхом вказівки його внутрішньої логічної структури, наприклад, опису логічних елементів, з яких він складається, або опису його поведінки програмним способом, зосереджуючись на тому, що модуль робить, а не на його логічній реалізації. Ці модулі потім з'єднуються мережами, що дозволяє їм спілкуватися між собою. На відміну від мови VHDL, Verilog не строго типізований та має синтаксис, схожий на мову програмування C [17].

Порівняльну характеристику двох мов наведено в таблиці 1.2 [18]- [19].

Таблиця 1.2 – Порівняльна характеристика мов опису апаратури

Verilog	VHDL
Для ACIS	Для FPGA
Не строго типізована	Строго типізована
Низька багатослівність	Висока багатослівність
Частково детермінована	Сильно детермінована
С-подібна	Не С-подібна

Оскільки обидві мови опису апаратури широко використовуються розробниками та мають взаємозамінні функції, то не можна віддати перевагу одній з них. У даній роботі використовується мова опису VHDL, але всі схеми можуть бути описані мовою Verilog без змін у функціонуванні схем.

1.4 Метод автоматизованого проєктування цифрових комбінаційних пристроїв на основі технологій віртуальної та віддаленої інженерії

Удосконалений метод автоматизованого проєктування КЦП (рис. 1.5) базується на спільному використанні методів та засобів віртуальної та віддаленої інженерії для створення та дослідження прототипу ЦП [20] -[21].

Основні етапи процесу автоматизованого проєктування передбачають виконання наступних видів робіт.

Етап 1. Введення проєкту - схема проєкту задається або графічним способом, або з використанням мов опису апаратних засобів, таких як Verilog HDL, VHDL та ін.

Етап 2. Синтез – проєкт, що вводиться, синтезується в схему, яка складається з логічних елементів (ЛЕ) і логічних блоків (ЛБ) в ПЛІС.

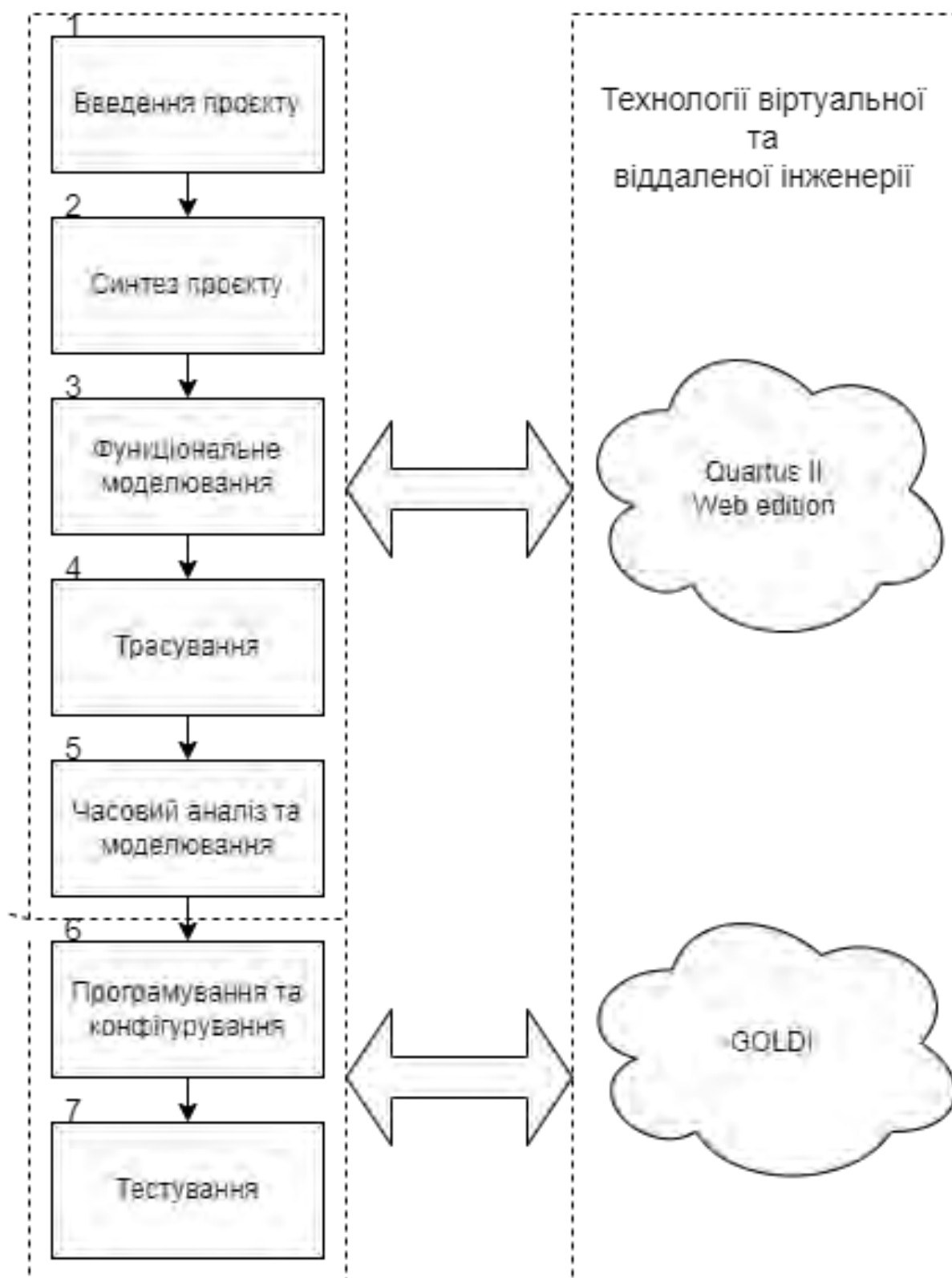


Рисунок 1.5 – Схема алгоритму удосконаленого методу автоматизованого проєктування комбінаційних цифрових пристроїв

Етап 3. Функціональне моделювання – синтезована схема тестується щодо коректності функціонування у вбудованому симуляторі, який моделює залежність стану (вибраних розробником) сигналів схеми від часу. Це моделювання не враховує часових затримок сигналів (між ЛЕ/ЛБ ПЛІС).

Етап 4. Трасування – інструмент трасування САПР обчислює оптимальне розміщення та з'єднання ЛЕ/ЛБ, визначених у списку з'єднань реальної ПЛІС. Трасувальник також обирає дроти або «маршрут руху» у чіпі для реалізації необхідних зв'язків між заданими ЛЕ/ЛБ.

Етап 5. Часовий аналіз та часове моделювання – аналізує затримки поширення сигналів вздовж різних шляхів у трасованій логічній схемі для обчислення наявності/відсутності логічних перегонів, щоб сигнали з різних ЛБ приходили одночасно в той чи інший кінцевий вузол схеми; схема тестується для перевірок функціональності та часових обмежень, але на відміну від функціонального моделювання, тут для виявлення наявності чи відсутності логічних перегонів враховуються реальні затримки вибраних сигналів.

Етап 6. Програмування та конфігурація – розроблена схема розміщується в мікросхемі ПЛІС шляхом програмування електронних зв'язків між конфігурованими ЛЕ/ЛБ, що реалізується шляхом передачі конфігураційного файлу з комп'ютера або в мікросхему ПЛІС, або додаткову (не вбудовану в ПЛІС) пам'ять, якщо така є у налагоджувальному комплекті. На рисунку 1.6 представлено програмування CPLD у ВЛ GOLDi.

Етап 7. Тестування – проводиться засобами інтерактивної гібридної онлайн лабораторії GOLDi (Grid of Online Lab Devices Ilmenau), що надає можливості як віддаленого тестування на реальній платі CPLD фірми Altera, так і тестування на віртуальній моделі [22]- [24].

Отже, засобами середовища Quartus II можна створити цифрову схему, виконати її моделювання та отримати файли для програмування CPLD. ВЛ GOLDi у свою чергу дозволяє виконати тестування розробленої цифрової схеми на реальному обладнанні.

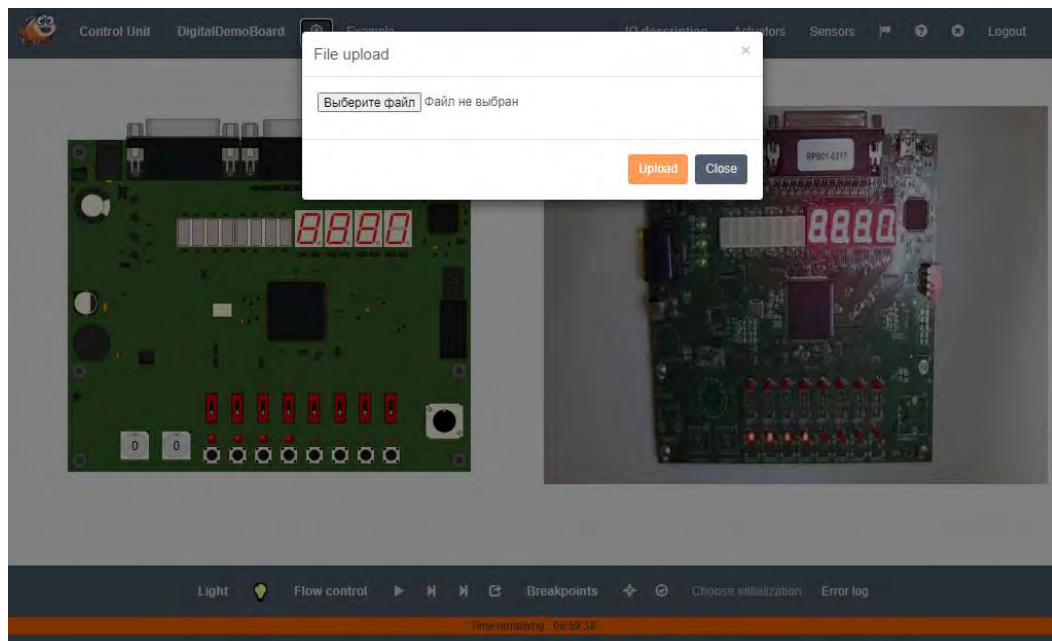


Рисунок 1.6 – Програмування CPLD у GOLDi

Quartus II та Modelsim можна завантажити на сторінці «FPGA Software Download Center», обравши зі списку Quartus II Web Edition версії 13.1 (рис.1.7). Можна завантажити Multiple Download чи Individual Files. Завантажуючи Individual Files, необхідно завантажити файли пристроїв MAX II, MAX V.

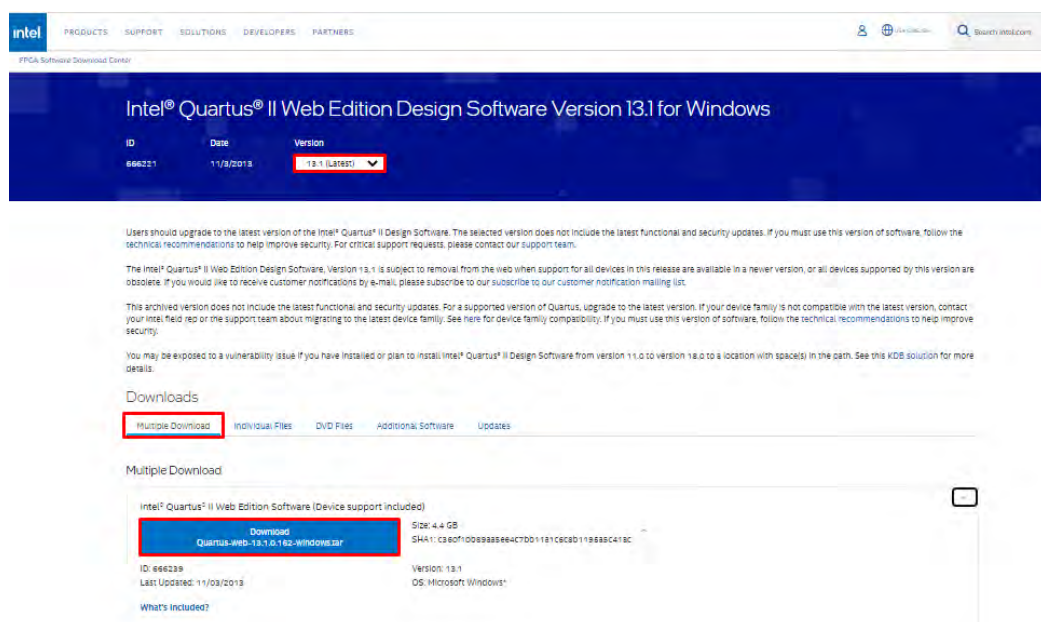


Рисунок 1.7 – Сторінка завантаження Quartus II Web Edition

Архітектура інфраструктури GOLDi складається з серверної та клієнтської частин (рис. 1.8) [25].

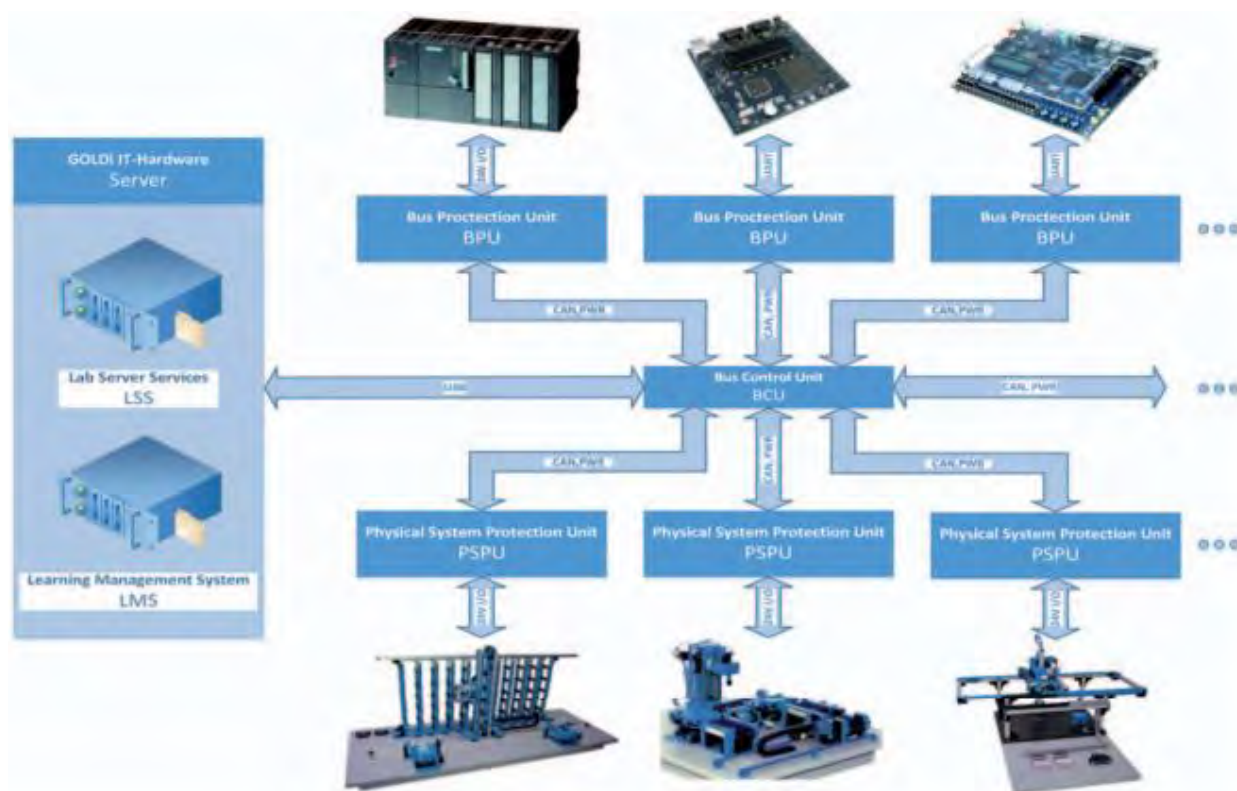


Рисунок 1.8 - Архітектура інфраструктури GOLDi [25]

Серверна частина інфраструктури GOLDi включає в себе наступні три складові:

- внутрішню послідовну шину віддаленої лабораторії (remote lab bus);
- модуль захисту шини (bus protection unit, BPU), що виступає у ролі інтерфейсу між модулем керування та шиною віддаленої лабораторії та захищає останню від неправильного використання та ушкоджень;
- модуль захисту фізичної системи (physical system protection unit, PSPU), що захищає фізичну систему (електромеханічні моделі) від навмисного або випадкового ушкодження неправильними командами та надає різноманітний доступ та механізми керування.

Клієнтська частина інфраструктури GOLDi складається з вебсередовища – дослідницької панелі керування (experimental control panel, ECP), що надає

можливості для генерації та виконання розробок за допомогою симуляції моделей [25].

У роботі [23] розглядаються основні можливості, які надає віддалена лабораторія GOLDi для роботи з CPLD. Зовнішній вигляд плати прототипування (ПП) наведений на рис. 1.9.

Режими роботи з ПП наступні:

а) віддалене (браузерне) прототипування цифрових систем передбачає наступні можливості:

1) проєктування (текстове, графічне або на основі діаграми переходів;

2) програмування CPLD за допомогою віддаленої інфраструктури лабораторії;

3) тестування віддаленої ПП, що виконується за допомогою інтерфейсу RIA (Rich Internet Application).

б) віддалена валідація цифрової системи:

1) аналіз та визначення функцій заданого пристрою, що представлений у вигляді моделі чорної скриньки, за допомогою маніпулювання вхідними сигналами;

2) визначення несправностей заданого пристрою, що представлений у вигляді моделі білої скриньки.

Основні компоненти плати прототипування наступні:

- чіп MAX V 5M1270Z (CPLD фірми Altera);
- компоненти введення (8 повзункових перемикачів, 8 натискних кнопок, 2 обертових шістнадцяти розрядних енкодерів);
- компоненти виводу (4 7-сегментні індикатори, 1 світлодіодна панель);
- інші компоненти (10МГц кварцовий генератор, генератор частоти, п'єзовипромінювач, інкрементальний енкодер, UART (через USB), 25-контактний SUB-D конектор) [23].

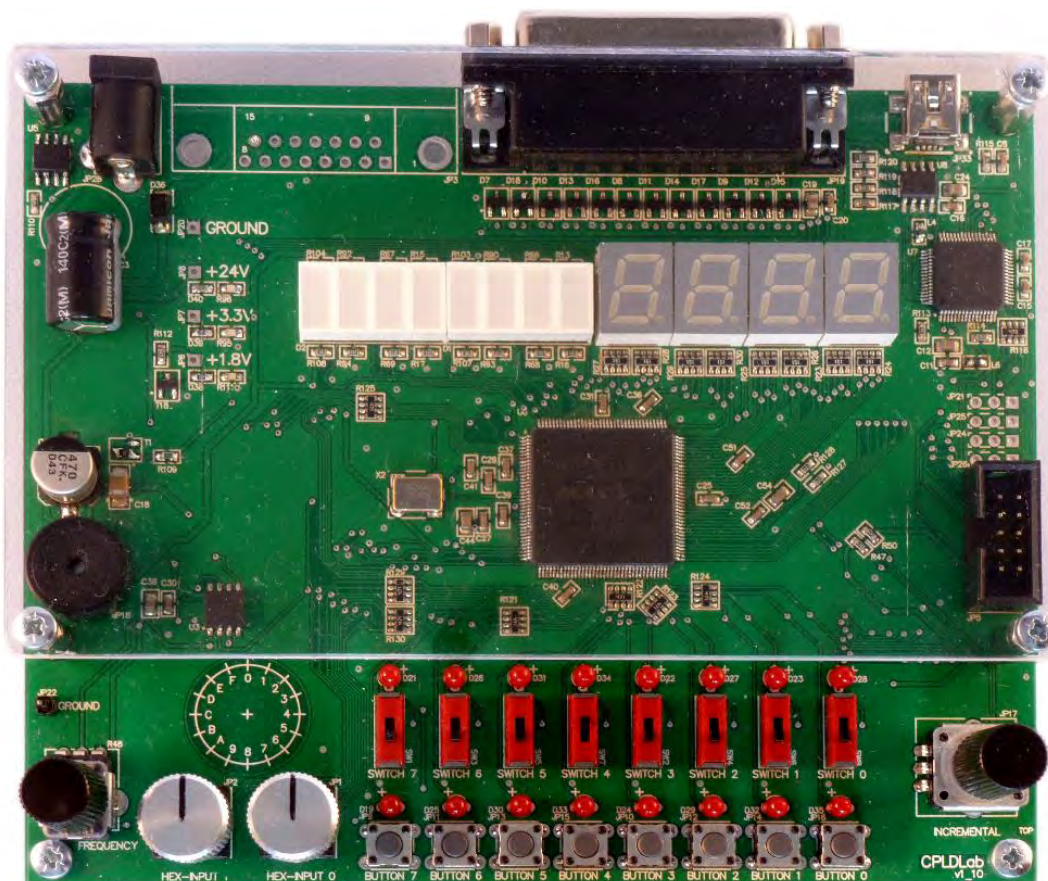


Рисунок 1.9 – Плата прототипування віддаленої лабораторії GOLDi [23]

Схема взаємодії ПП з інфраструктурою віддаленої лабораторії зображена на рис. 1.10. Комунікація з інфраструктурою віддаленої лабораторії відбувається за допомогою так званої з'єднувальної FPGA (Field-Programmable Gate Array), що встановлена на нижній стороні плати. Версія ПП для віддаленого програмування позбавлена таких компонентів введення, як повзункові перемикачі, кнопки, та використовує замість них пряме підключення до виходів з'єднувальної FPGA, що в свою чергу встановлює вхідні сигнали ПП відповідно до значень, введених через користувацький інтерфейс на персональному комп'ютері. Згенеровані ПП вихідні сигнали можуть бути напряму прочитані з'єднувальною FPGA без потреби у демонтажу світлодіодів. FPGA з'єднується з інфраструктурою віддаленої лабораторії через BPU [23].

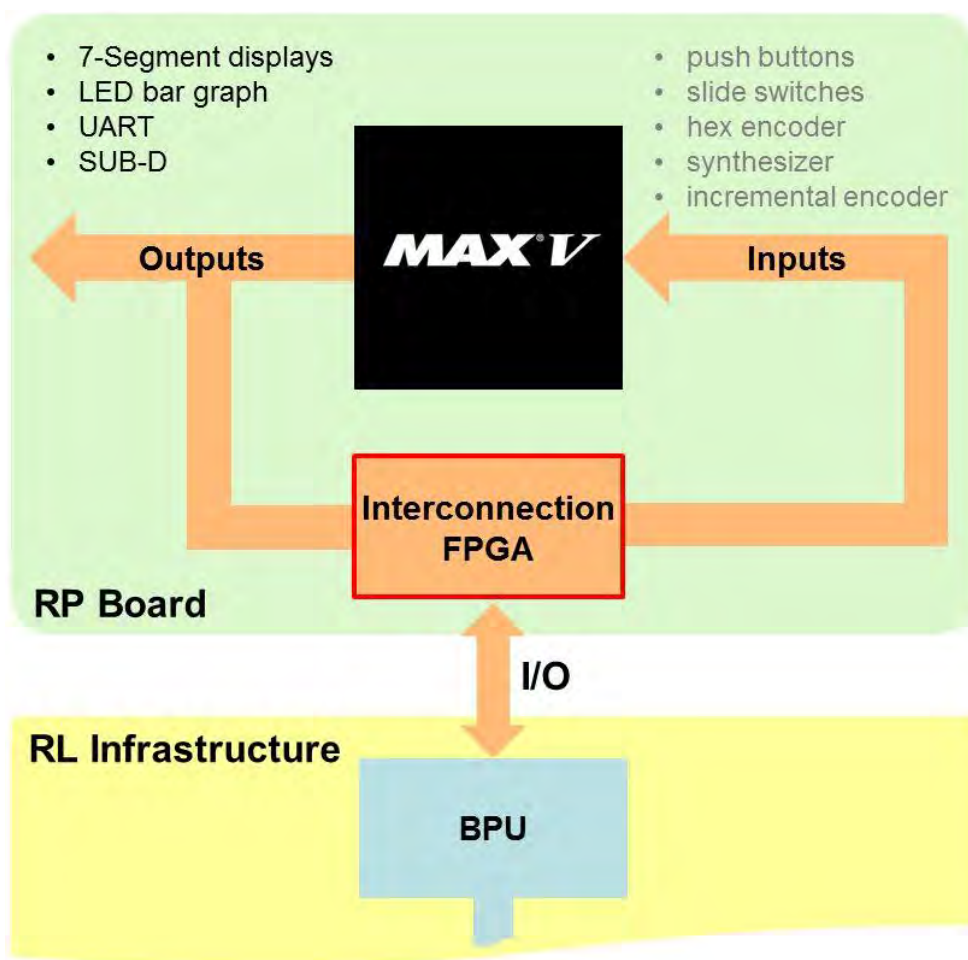


Рисунок 1.10 – Взаємодія з інфраструктурою GOLDi [23]

Головні складові інтерфейсу наступні:

- засоби завантаження синтезованого коду проєкту та автоматичного програмування CPLD у віддаленій лабораторії;
- візуальна модель ПП, що дозволяє маніпулювати входами та надає можливість спостереження за виходами ПП;
- засоби огляду через вебкамеру для спостереження за результатами дій користувача.

Таким чином, ВЛ надає широкі можливості використання обладнання, яке доступне онлайн 24/7, причому в різних університетах мережі GOLDi-labs.net [25] - [26].

1.5 Постановка задач магістерської роботи

Дослідження особливостей автоматизованого проєктування ЦП дозволило визначити наступні методи та технічні засоби, що будуть використовуватися у подальшій роботі: CPLD у якості базису розробки; Quartus II як середовище розробки; ModelSim як середовище моделювання; VHDL як мова опису апаратури; GOLDi як середовище для тестування.

Мета роботи – дослідження та програмна реалізація методів та засобів віртуальної та віддаленої інженерії для розробки та тестування КЦП.

Завдання роботи:

- розробити удосконалений метод автоматизованого проєктування КЦП з використанням технологій віртуальної та віддаленої інженерії;
- розробити структуру бібліотеки основних комбінаційних цифрових пристроїв;
- розробити описи бібліотечних елементів у вигляді діаграм блоків та на мові опису апаратури VHDL в середовищі Quartus II;
- виконати схемотехнічне моделювання бібліотечних елементів в середовищі ModelSim;
- розробити схеми тестування розроблених бібліотечних елементів;
- провести функціональне тестування бібліотечних елементів засобами віддаленої лабораторії GOLDi.

2 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЦИФРОВИХ ПРИСТРОЇВ ЗАСОБАМИ ВІРТУАЛЬНОЇ ТА ВІДДАЛЕНОЇ ІНЖЕНЕРІЇ

2.1 Опис особливостей реалізації бібліотеки КЦП

2.1.1 Структура бібліотеки та шаблон опису елементів

Бібліотека ЦКП (рис. 2.1) може використовуватись в практиці автоматизованого проєктування комунікаційних цифрових систем для реалізації методології повторного використання (reuse) [27].

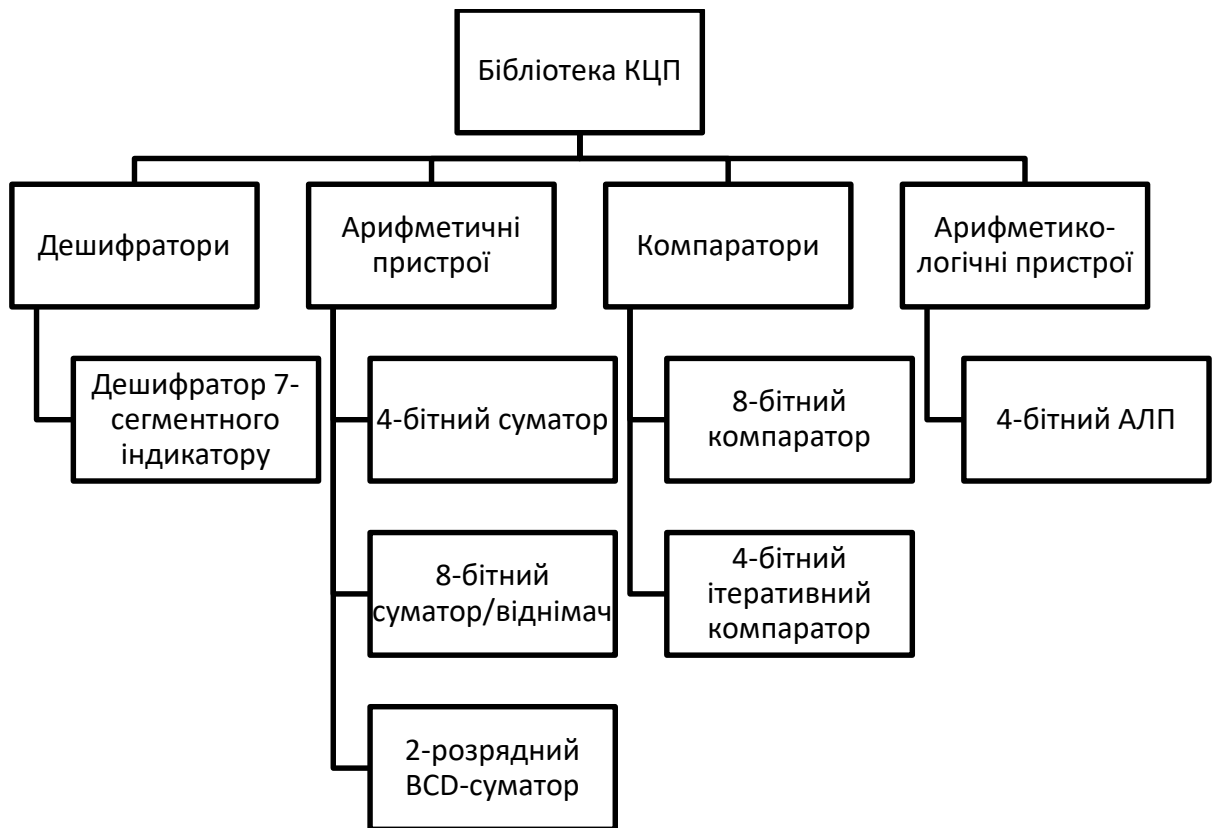


Рисунок 2.1 – Структура бібліотеки КЦП

Бібліотека об'єднує чотири групи найбільш розповсюджених КЦП: дешифратори, арифметичні пристрої, компаратори та арифметико-логічні пристрої. До кожної з груп входять один або декілька об'єктів проєктування.

Модульну структуру опису об'єкта наведено на рисунку 2.2.

Модуль 1. Опис інтерфейсу об'єкта (вхідних/вихідних сигналів та параметрів проєкту).

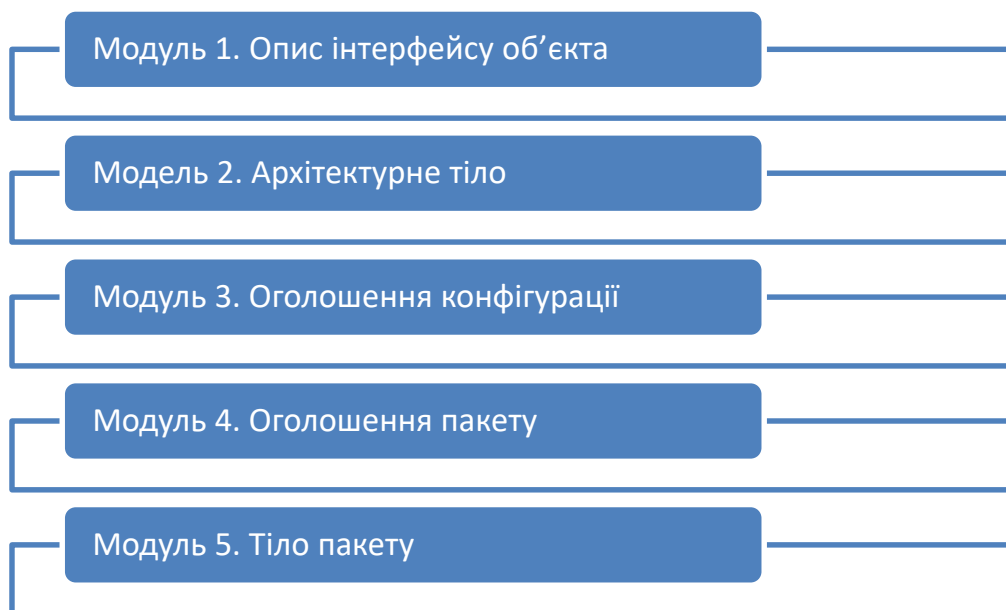


Рисунок 2.2 - Структура шаблону опису об'єкта на мові VHDL

Модуль 2. Архітектурне тіло – опис функції, логічної організації та/або структурної схеми об'єкта проекту.

Модуль 3. Оголошення конфігурації.

Модуль 4. Оголошення пакету – опис користувальницьких типів даних, констант, інтерфейсів процедур та функцій.

Модуль 5. Тіло пакету – опис тіл процедур та функцій.

Розроблений шаблон опису об'єкта на мові VHDL наведено у Додатку А.1. В таблиці 2.1 наведено базові оператори мови VHDL, що використовувались при розробці. Описати об'єкт на мові VHDL можна багатьма способами. Наприклад, у Додатку А.2 – А.5 представлено варіанти опису дешифратора за допомогою різних конструкцій мови VHDL, а саме, з використанням:

- найпростіших логічних операторів;
- умовного привласнення;
- оператору SELECT;
- оператору PROCESS.

Таблиця 2.1 – Базові оператори мові VHDL

Тип оператора	Приклад
Оператори привласнення значень змінним та сигналам	<code>:= ; <=</code>
Логічні оператори	<code>not; and; or; xor</code>
Арифметичні оператори	<code>++; --; /; rem; mod; abs</code>
Послідовні оператори	<code>IF, CASE, NEXT, LOOP</code>
Паралельні оператори	<code>PROCESS, GENERATE, BLOCK, SELECT</code>

2.1.2 Алгоритм розробки елементів бібліотеки КЦП

Алгоритм розробки бібліотечних елементів (рис. 2.3) в середовищі Quartus II може бути продемонстровано на прикладі розробки схеми 4-бітного суматора.

Етап 1. Початок. Для того, щоб створити проєкт, необхідно запустити середовище Quartus II. Обраємо «New Project Wizard» та у вікні, що відкрилося, натискаємо «Next». У наступному вікні обираємо бажане місце розташування майбутнього проєкту та вказуємо його ім'я та ім'я головної сутності (top-level entity). Важливо, щоб файл сутності з таким ім'ям, як у проєкту, був обов'язково створений, інакше при компіляції виникне помилка про його відсутність. Після створення проєкту, головну сутність завжди можна змінити. Слід також відзначити, що назва файлу будь-якої сутності повинна співпадати з назвою самої сутності. Називаємо проєкт «FourBitFullAdder» та двічі натискаємо на «Next».

Етап 2. Налаштування проєкту. Після переходу відкриється вікно вибору пристрою для програмування. Необхідно обрати бажану CPLD.



Рисунок 2.3 – Схема алгоритму розробки бібліотечних елементів

У даному випадку потрібна MAX V 5M1270ZT144C5 (рис. 2.4). Пристрій також може бути змінений після створення проєкту. Після цього двічі натискаємо на «Next», а потім на «Finish».

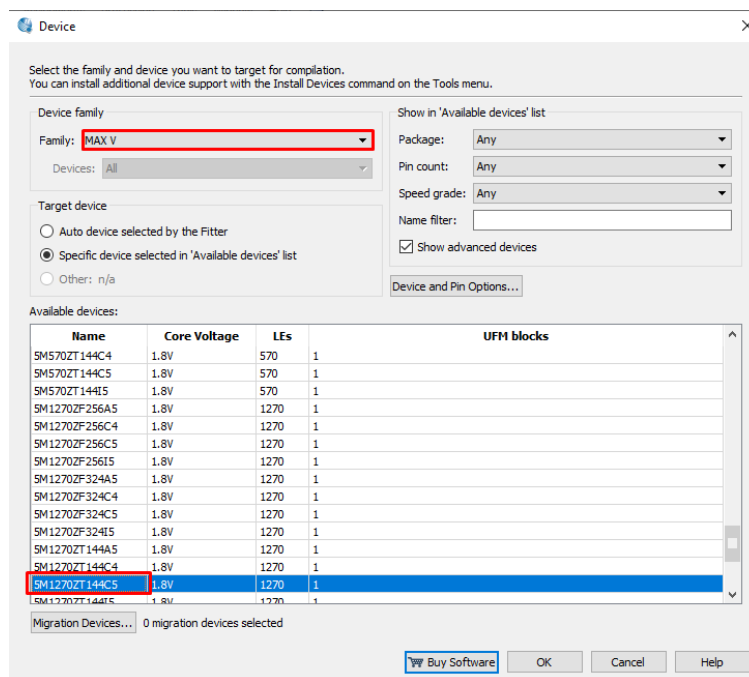


Рисунок 2.4 – Вибір пристрою для програмування

Етап 3. Складання рівнянь функціонування виходів. Оскільки у даному прикладі створюється 4-бітний суматор, то варто ознайомитися з принципами його роботи і розробки. 4-бітний суматор складається з 4-х 1-бітних повних суматорів, кожен з яких, як слідує з назви, виконує операцію складання двох бітів. Схему 1-бітного повного суматору зображено на рисунку 2.5.

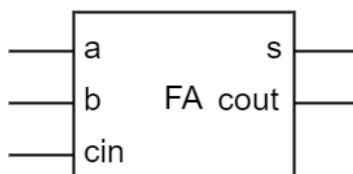


Рисунок 2.5 – Схеми 1-бітного повного суматору

При двійковому складанні, як і у випадку з десятковим, утворюється сума та перенесення у наступний розряд, якщо сума розрядів більша або дорівнює основі системи числення. У процесі двійкового складання можуть виникнути наступні ситуації (a , b – відповідні біти доданків, s – біт суми, c_{in} , c_{out} – біти перенесення з попереднього та до наступного розрядів):

- $a=0, b=0, c_{in}=0 \rightarrow s=0, c_{out}=0$;
- $a=1, b=0$ ($a=0, b=1$), $c_{in}=0 \rightarrow s=1, c_{out}=0$;
- $a=1, b=0$ ($a=0, b=1$), $c_{in}=1 \rightarrow s=0, c_{out}=1$;
- $a=1, b=1, c_{in}=0 \rightarrow s=0, c_{out}=1$;
- $a=1, b=1, c_{in}=1 \rightarrow s=1, c_{out}=1$.

Дослідивши принципи функціонування 1-бітного повного суматора, можна побудувати його таблицю дійсності, скласти рівняння виходів та мінімізувати їх.

На рисунку 2.6 наведено таблицю дійсності 1-бітного повного суматора, що зображує принципи його функціонування.

За допомогою таблиці дійсності можна скласти рівняння функціонування виходів. Але у такому випадку рівняння не будуть мінімальними, що при проектуванні великих та складних пристроїв може призвести до зайвих витрат. Тому рівняння зазвичай мінімізуються. Існує багато методів мінімізації логічних рівнянь, одним з яких є графічний метод карт Карно.

Побудуємо карти Карно для бітів суми s та перенесення у наступний розряд c_{out} . У клітинках карти, що знаходяться на перехресті вхідних сигналів (у даному випадку бітів a , b та c_{in}), проставляються значення одного вихідного сигналу, для якого була зроблена карта (у даному або біт s або c_{out}). Потім будуються так звані імпліканти – прямокутні області, що покривають певну кількість одиниць у клітинках карти. Кількість одиниць, що можуть бути покриті однією імплікантою, має дорівнювати степеням двійки. Карти Карно для бітів s та c_{out} наведені на рис. 2.7 та 2.8.

a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Рисунок 2.6 – Таблиця дійсності 1-бітного повного суматору

		$b \ c_{in}$			
		00	01	11	10
a	0	0	1	0	1
	1	1	0	1	0

Рисунок 2.7 – Карта Карно для біту суми s

		$b \ c_{in}$			
		00	01	11	10
	0	0	0	1	0
	1	0	1	1	1

Рисунок 2.8 – Карта Карно для біту перенесення c_{out}

Знайдемо рівняння бітів суми та перенесення з відповідних карт Карно. Для цього для кожної імпліканти продивляємося, які біти не змінюють свого значення у її межах. Об'єднуємо ці біти за допомогою кон'юнкції (логічного «І»). Якщо біт має нульове значення, то ставимо над ним знак інверсії. Отриманні кон'юнкції об'єднуємо диз'юнкцією (логічним «АБО»).

Таким чином, отримані рівняння для бітів суми та перенесення мають наступний вигляд (формули 2.1 та 2.2):

$$s = a\bar{b}\bar{c}_{in} + \bar{a}\bar{b}c_{in} + abc_{in} + \bar{a}b\bar{c}_{in} = a(\bar{b}\bar{c}_{in} + bc_{in}) + \bar{a}(\bar{b}c_{in} + b\bar{c}_{in}) = \overline{ab} \oplus c_{in} + \bar{a}b \oplus c_{in} = a \oplus b \oplus c_{in} \quad (2.1)$$

$$c_{out} = ac_{in} + bc_{in} + ab \quad (2.2)$$

Етап 4. Створення структурної схеми у Quartus II. Для створення файлів проекту оберемо «File → New». У списку можливих типів файлів оберемо «Block Diagram File» (рис. 2.9).

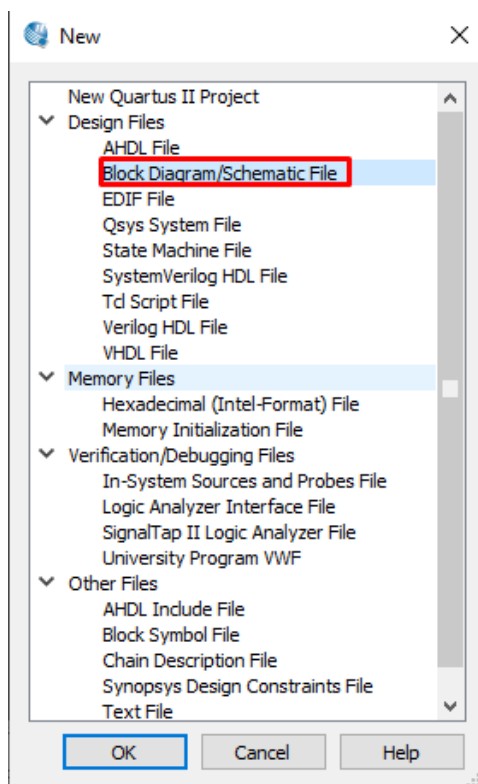


Рисунок 2.9 – Список файлів для створення

Спочатку зберігаємо файл. Якщо це не головна сутність (top-level design), то оберемо назву, що не співпадає з назвою проєкту (проєкт ми назвали «FourBitFullAdder»). 1-бітний повний суматор буде зібрано з логічних елементів так, як це представлено рівняннями 2.1 та 2.2. Зберігаємо файл під ім'ям «FullAdder».

На панелі інструментів необхідно знайти іконку вставки символу, «Symbol Tool» (рис. 2.10). У вікні, що відкрилося, обираємо вентилі XOR, AND та OR (Libraries → primitives → logic).

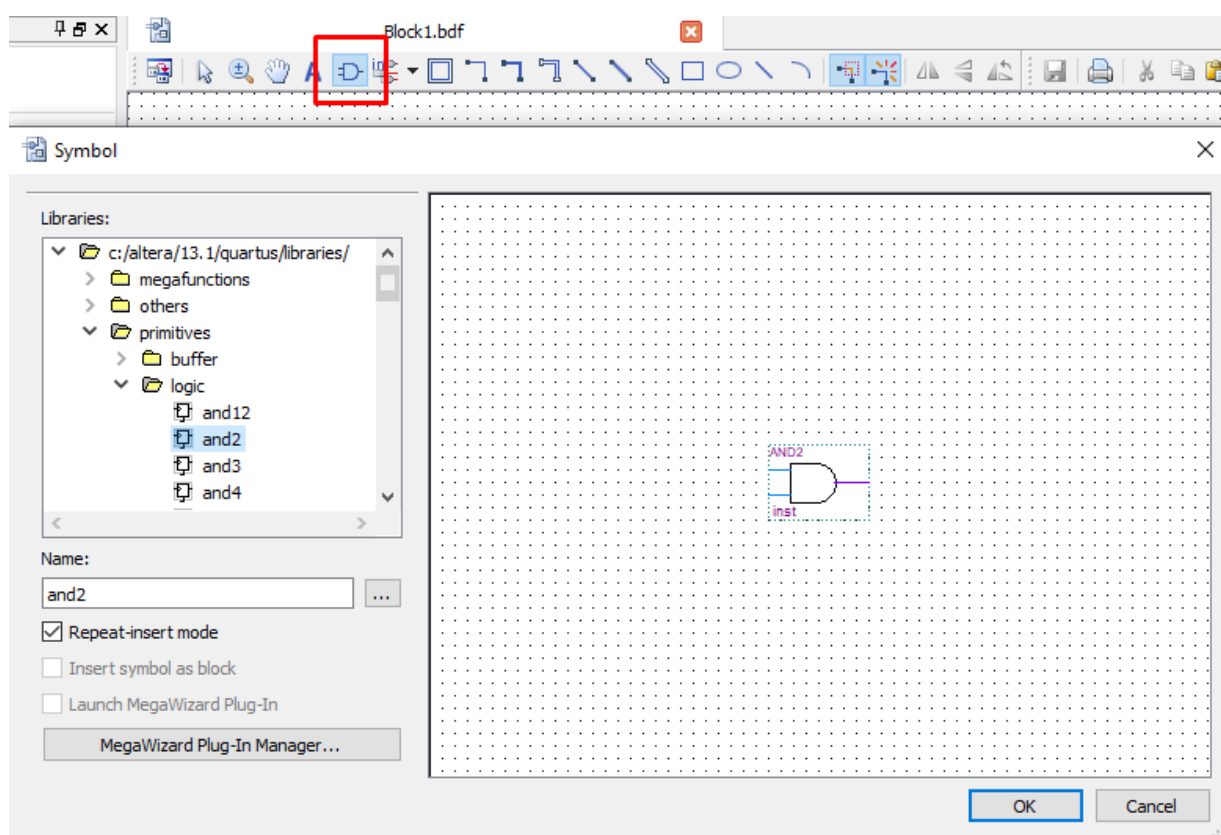


Рисунок 2.10 – Вікно вибору символів

Поряд з іконкою «Symbol Tool» знаходимо іконку «Pin Tool», що використовується для додавання вхідних та вихідних контактів (пінів). Додаємо два вхідних контакти та два вихідних контакти. Отримана схема має бути такою, як показано на рис. 2.11.

Таким чином, створено 1-бітний повний суматор з вхідними сигналами a , b та c_{in} , що позначають відповідні біти доданків та біт перенесення, та вихідними сигналами s та c_{out} , що позначають біт суми та біт переносу у наступний розряд відповідно.

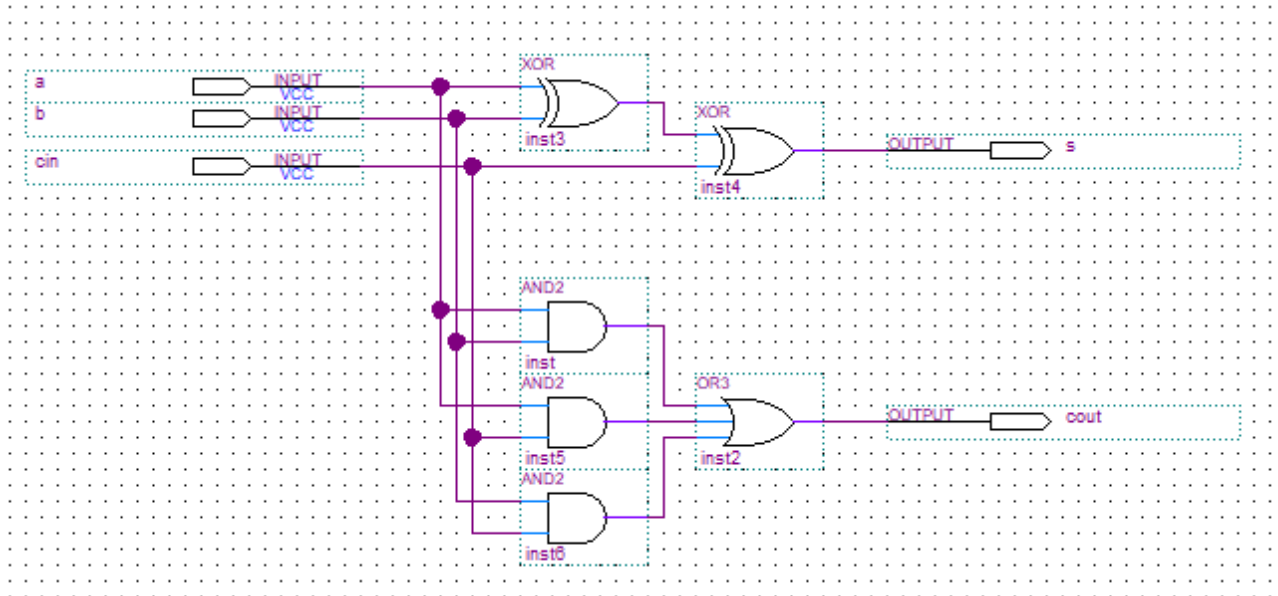


Рисунок 2.11 – Схема 1-бітного повного суматора

Для того, щоб використати напівсуматор як будівельний блок для 1-бітного повного суматора створимо символний файл (Symbol File) для напівсуматора. Таким чином, ми зможемо у подальшому додати напівсуматор до структурної схеми як окремий символ (за аналогією з вентилями XOR, AND та OR). Для цього обираємо «File → Create/Update → Create Symbol Files for Current File», як показано на рис. 2.12.

Тепер створимо 4-бітний суматор з чотирьох 1-бітний повних суматорів у новій структурній схемі (Block Diagram). Створений раніше символ повного суматора можна знайти, обравши «Symbol Tool».

У вікні, що відкрилося, обираємо Project → FullAdder, як показано на рис. 2.13.

Розміщуємо та з'єднуємо елементи так, як показано на рис. 2.14. Оскільки це головна сутність проекту, зберігаємо файл під «FourBitFullAdder» (це ім'я ми вибрали для головної сутності при створенні проекту).

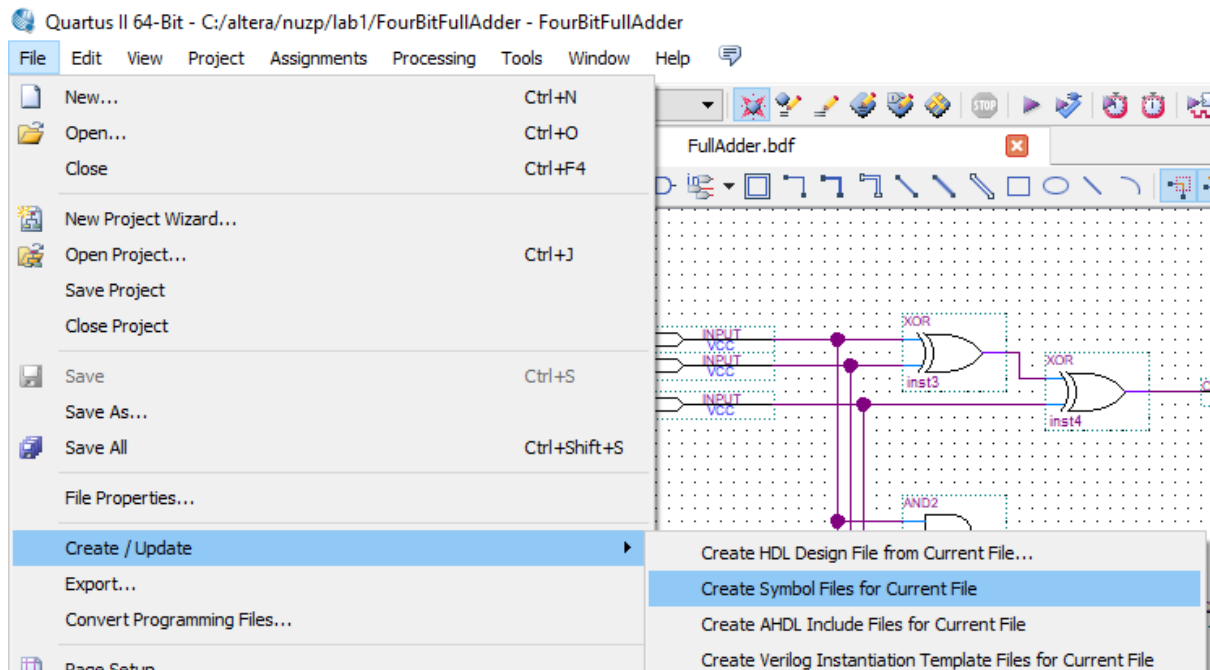


Рисунок 2.12 – Створення символного файлу

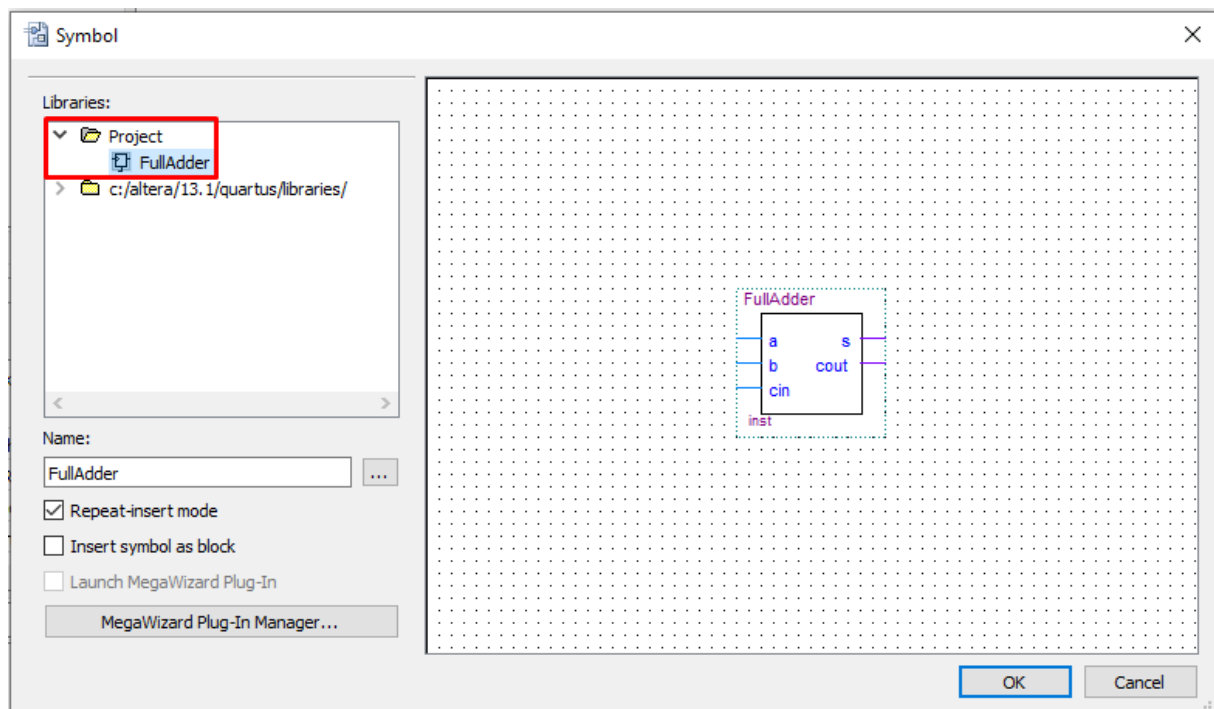


Рисунок 2.13 – Створений символ повного суматора

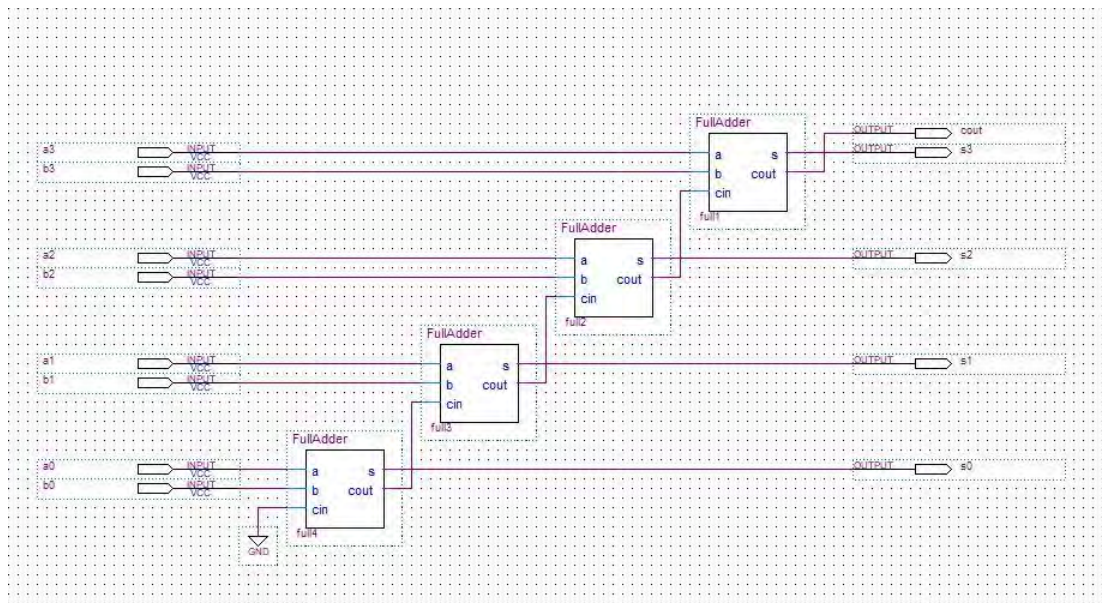


Рисунок 2.14 – Схема 4-бітного суматора

Етап 5. Компіляція проєкту. Перед компіляцією необхідно обрати файл головної сутності. Для прикладу оберемо файл кінцевого автомату як головну сутність. Встановлення головної сутності зображено на рис. 2.15.

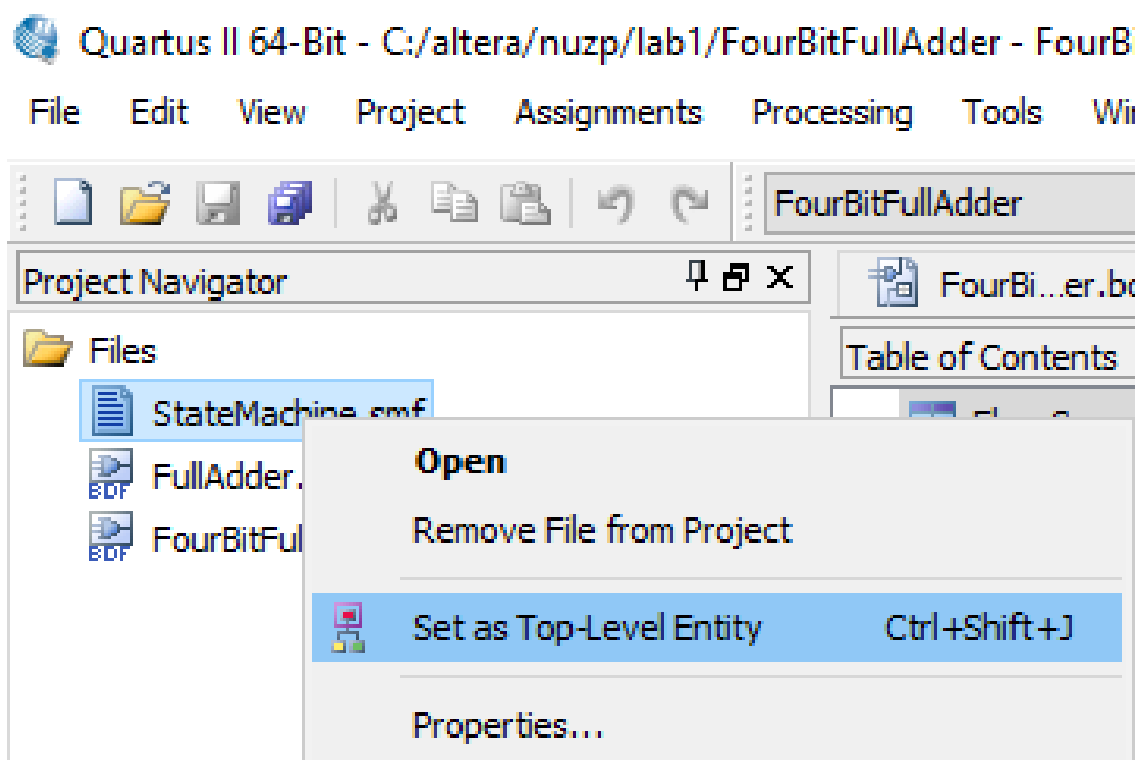


Рисунок 2.15 – Встановлення головної сутності

Якщо виникнуть помилки, вони відобразяться у звіті компіляції (Compilation Report).

Етап 6. Призначення виводів. Для розпіновки створеного пристрою використовується Pin Planner: Assignments → Pin Planner, вікно якого представлено на рис. 2.16.

Якщо ви до цього не проводили компіляцію проєкту, то список можливих контактів у вікні «Pin Planner» буде порожнім. У такому випадку необхідно буде визначити вхідні та вихідні сигнали власноруч. У випадку з компіляцією цей процес відбувається автоматично.

Тепер необхідно тільки визначити контакти, або піни («Locations»). Приклад встановлення пінів зображений на рис. 2.17.

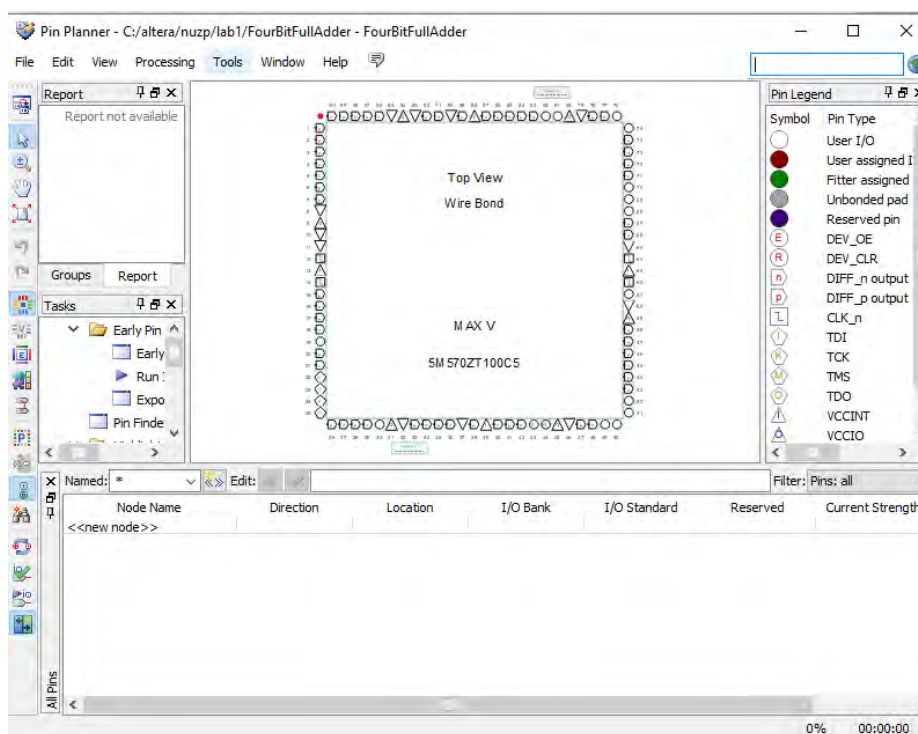


Рисунок 2.16 – Вікно Pin Planner

Етап 7. Моделювання проєкту. Для цього необхідно перейти до Tools → Options. Далі обрати General → EDA Tool Options та у полі ModelSim-Altera вказати шлях: C : \altera\14.1\modelsim ase\win32aloem.

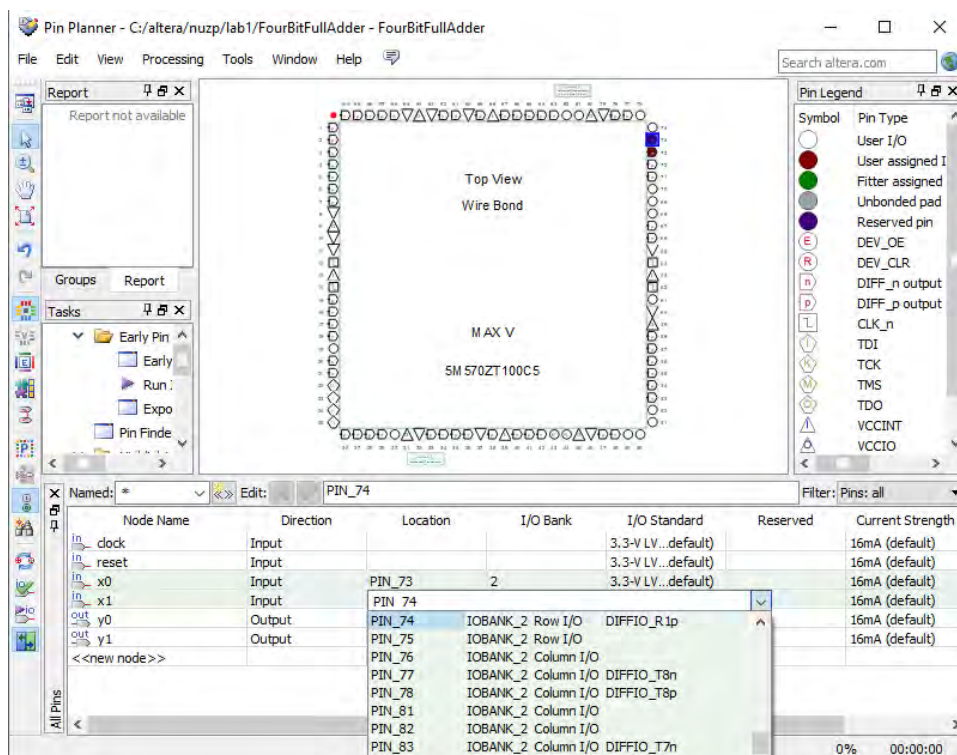


Рисунок 2.17 – Встановлення пінів

Проводити моделювання можна лише головної сутності (top-level entity). Якщо необхідно провести моделювання іншого файлу, то необхідно визначити цей файл як головну сутність. Для цього у вкладці «Project Navigator», що зазвичай розташована з лівої сторони екрану необхідно перейти на вкладку «Files», натиснути правою кнопкою миші на файл, що бажаєте зробити головною сутністю та у контекстному меню обрати «Set as Top-Level Entity». Або відкрити файл та виконати «Project → Set as Top-Level Entity». Після того, як змінили головну сутність, необхідно скопіювати проєкт знову.

Після моделювання отримується файл тільки для читання з результатами моделювання (рис. 2.18). Цей файл схожий з вхідним файлом, але містить значення вихідних сигналів. Тут можна переконатись, що все працює як слід або повернутись для внесення змін в проєкт.

Етап 8. Програмування CPLD. Для цього в лабораторії GOLDi вже створено шаблон зі всіма входами та виходами. Необхідно вставити розроблену логіку та під'єднати її до входів та виходів.

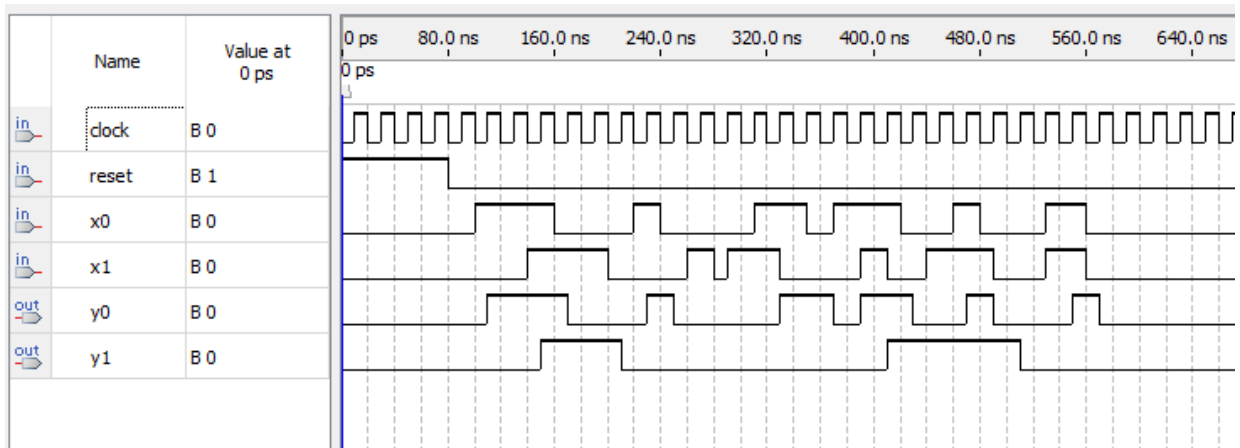


Рисунок 2.18 – Результати моделювання

Для того, щоб запрограмувати CPLD використовується програматор IUT RemoteLab CPLD Programmer. Необхідно переконатися, що плата під'єднана до комп'ютера. Після цього запустити програматор, виконавши «Tools → Programmer», та оберіть роф-файл проєкту та натиснути «Program». Процес програмування займає декілька секунд.

2.2 Розробка дешифратора 7-сегментного індикатору

7-сегментний індикатор головним елементом виведення ПП віддаленої лабораторії GOLDi. Робота з 7-сегментним індикатором ускладнюється кодуванням чисел та літер. Тому для спрощення виводу числових результатів необхідно розробити дешифратор 7-сегментного індикатору.

7-сегментний індикатор – це пристрій, призначений для відображення цифр та деяких латинських літер. За допомогою 7-сегментного індикатору можна представити числа від «0» до «15» у шістнадцятковому вигляді, при цьому числа від «10» до «15» будуть представлені літерами від «A» до «F». На рис. 2.19 зображено схему 7-сегментного індикатору, на якій видно, що кожному з семи сегментів (восьмим сегментом іноді вважається сегмент крапки) для простоти поставлена у відповідність літера. Так наприклад, щоб відобразити цифру «3» необхідно увімкнути сегменти a, b, g, c, d. А якщо

представити кожний сегмент як біт, то двійкове слово закодованої цифри «3» буде мати вигляд «01111001» (схема кодування «*abcdefg*», де «*p*» – біт крапки).

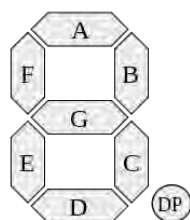


Рисунок 2.19 – Позначення сегментів індикатору

Підключення 7-сегментного відрізняється в залежності від типу індикатору. Існують два типи індикаторів: зі загальним анодом та загальним катодом (рис. 2.20). Кожен сегмент в індикаторі це LED-світлодіод, який має анод та катод. Загальний анод у схемі під'єднаний до живлення, а загальний катод – до землі.

Якщо сегменти з'єднані за схемою з загальним анодом, то керуючий сигнал кожного сегмента буде мати низький рівень, оскільки потенціал окремого катода повинен бути нижчим за потенціал загального аноду. Якщо сегменти з'єднані за схемою з загальним катодом, то керуючий сигнал кожного сегмента буде мати високий рівень. Таким чином, цифра «3» у схемі з загальним катодом буде мати вигляд «01111001», а у схемі з загальним анодом – «10000110» (схема кодування «*abcdefg*»). Тобто різниця в кодуванні для двох схем полягає у інвертуванні сигналу.

Метою побудови дешифратора для 7-сегментного індикатору є спрощення виводу чисел. Оскільки двійкове представлення числа, очевидно, не підходить для відображення його на індикаторі, для цього необхідно використовувати дешифратор для переводу двійкового вигляду числа у вигляд за схемою «*abcdefg*». Дешифратор 7-сегментного індикатору представлений на рис. 2.21.

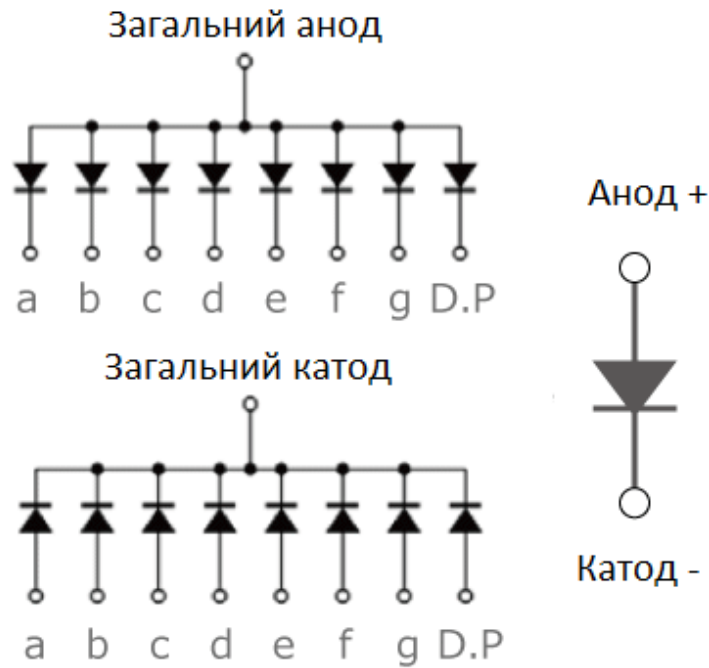


Рисунок 2.20 – Схеми підключення

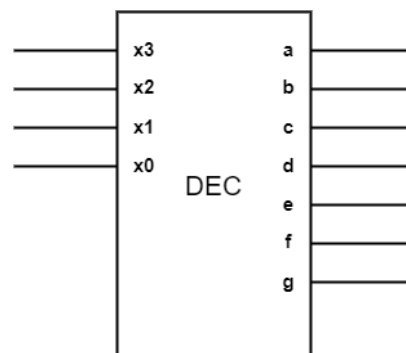


Рисунок 2.21 – Схема дешифратора

На схемі на рисунку 2.21 відсутній сигнал керування сегментом крапки, оскільки у даній роботі він не знадобиться.

Побудова структурної схеми потребує наявності таблиці істинності. Таблиця істинності дешифратора 7-сегментного індикатора відрізняється простотою, оскільки кожній двійковій комбінації чисел від 0 до 15 ставиться у відповідність закодоване представлення числа. Таблицю істинності наведено на рисунку 2.22.

x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

Рисунок 2.22 – Таблиця істинності дешифратора 7-сегментного індикатору

Для створення рівнянь керуючих сигналів для кожного індикатору використовуються карти Карно, що дозволяють також провести мінімізацію отриманих виразів. Для прикладу карти Карно для кожного сегментів a та b та їх логічні рівняння представлені на рис. 2.23, 2.24 та у формулах 2.3, 2.4:

$$a = \bar{x}_3\bar{x}_2\bar{x}_0 + \bar{x}_3x_1 + x_2x_1 + \bar{x}_3x_2x_0 + x_3\bar{x}_0 + x_3\bar{x}_2\bar{x}_1 \quad (2.3)$$

$$b = \bar{x}_3\bar{x}_2 + \bar{x}_3\bar{x}_1\bar{x}_0 + \bar{x}_3x_1x_0 + \bar{x}_2\bar{x}_1\bar{x}_0 + x_3\bar{x}_1x_0 + \bar{x}_2x_1\bar{x}_0 \quad (2.4)$$

		$x_1 x_0$			
		00	01	11	10
$x_3 x_2$	00	1	0	1	1
	01	0	1	1	1
	11	1	0	1	1
	10	1	1	0	1

Рисунок 2.23 – Карта Карно для сегменту «а»

		$x_1 x_0$			
		00	01	11	10
$x_3 x_2$	00	1	1	1	1
	01	1	0	1	0
	11	0	1	0	0
	10	1	1	0	1

Рисунок 2.24 – Карта Карно для сегменту «b»

2.2.1 Розробка структурної схеми дешифратора у середовищі Quartus

Для розробки схеми дешифратора використовується проєкт, що використовується в якості шаблону для тестування. Даний проєкт містить вхідні та вихідні сигнали ПП з визначеними для кожного з них пінів.

За допомогою таблиці дійсності (рис. 2.22) та карт Карно складаємо логічні рівняння для кожного сегменту за прикладом сегментів а та b. Після

цього у проєкті створюємо файл `decoder_logic.bdf` зі схемою дешифратору (рис. 2.25).

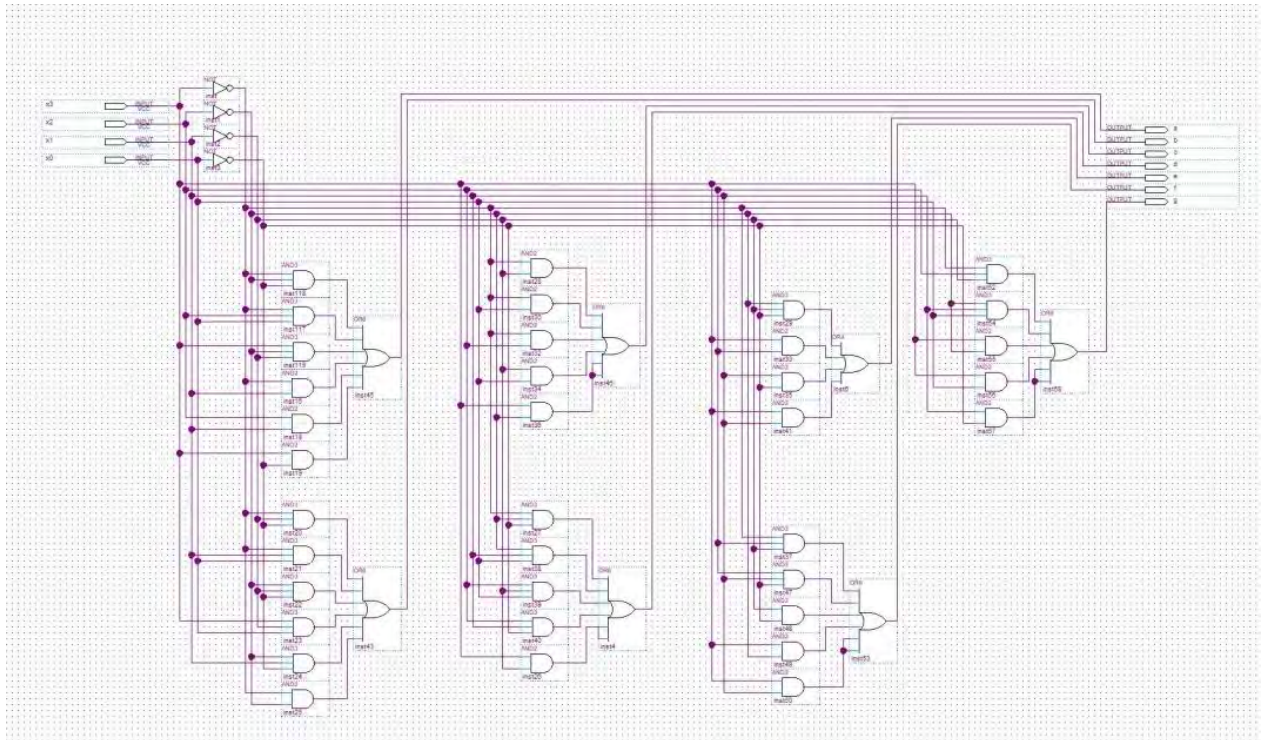


Рисунок 2.25 – Структурна схема дешифратору

Встановлюємо файл `decoder_logic.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання дешифратору (рис. 2.26). Для цього створюємо файл моделювання «University Program VWF» та додаємо до нього входи та виходи схеми. Входом схеми є число від 0 до 15, тому для спрощення моделювання та аналізу результатів варто об'єднати вхідні сигнали у групу, виділивши їх та обравши `Edit → Grouping → Group`, та застосувати до групи «Count value». Таким чином, входи будуть поводити себе як лічильник. Також можна виділити ім'я групи та обрати представлення сигналу «Unsigned Decimal» (`Edit → Radix → Unsigned Decimal`). З виходами схеми можна повторити те ж саме.

Для тестування створюємо символ з дешифратору (рис. 2.27) та поміщаємо його у файл `seven_seg_test.bdf`. З'єднуємо входи дешифратору з 16-розрядним декодером, а вихід з 7-сегментним індикатором (рис. 2.28).

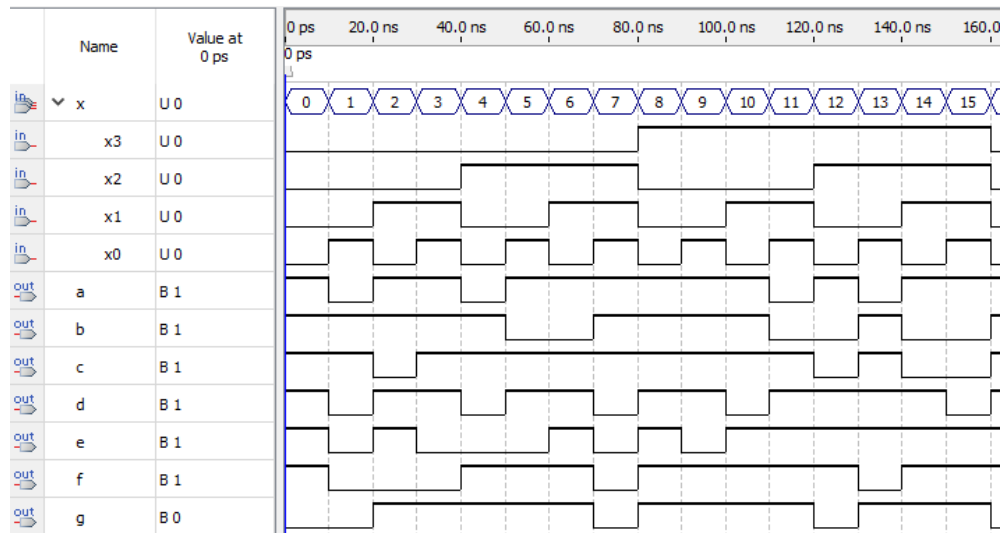


Рисунок 2.26 – Часова діаграма дешифратора

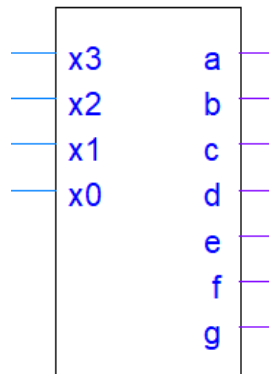


Рисунок 2.27 – Символ дешифратора

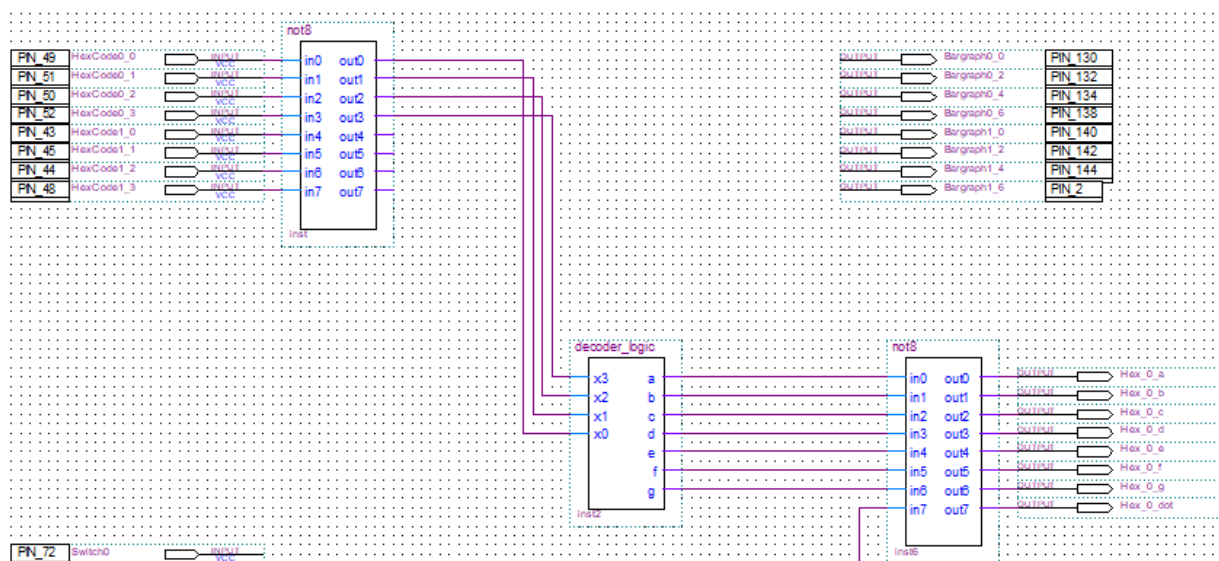


Рисунок 2.28 – Схема для тестування дешифратора

2.2.2 Опис дешифратора на мові VHDL

Описати дешифратор 7-сегментного індикатору на мові VHDL можна багатьма способами. У Додатку А.2 – А.5 представлено варіанти опису дешифратора за допомогою різних конструкцій мови VHDL.

Далі встановлюємо файл `decoder_vhdl.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання.

Таким чином, для спрощення подальшої роботи з 7-сегментними індикаторами на ПП віддаленої лабораторії GOLDi можна створити засобами середовища розробки Quartus II дешифратор 7-сегментного індикатору.

2.3 Розробка арифметичних пристроїв

Двійкова арифметика підпорядковується таким же правилам, як і десяткова. Так, наприклад, при додаванні двох одиниць у відповідних бітах числа у біті суми отримується нуль, а одиниця переносу розповсюджується у наступний розряд. На рисунку 2.29 наведено таблицю дійсності повного суматора, що також представляє собою правила додавання двійкових чисел.

Для операції віднімання можна побудувати схожу таблицю, за якою згодом можна буде створити пристрій віднімання двох чисел. Але пристрої віднімання використовуються рідко, оскільки вони складні та малоефективні у порівнянні з суматорами. Для віднімання використовується так званий доповняльний код (2's complement), що дозволяє замінити операцію віднімання операцією додавання.

Розглянемо правила переведення числа у доповняльний код та принципи його роботи. Правила переведення двійкового числа у доповняльний код наступні:

a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Рисунок 2.29 – Таблиця дійсності повного суматора

– якщо число додатне, то його доповняльний код відповідає звичайному двійковому коду (якщо у записі числа використовується окремий знаковий біт, то такий запис називається прямим кодом);

– якщо число від’ємне, то для представлення його у доповняльному необхідно інвертувати всі розряди та додати до отриманого числа одиницю.

Розглянемо приклад представлення від’ємного числа -5_{10} у доповняльному коді. Двійкове представлення числа додатного числа 5_{10} наступне: 0101_2 . Спочатку інвертуємо всі розряди числа: 1010_2 . Після цього додамо одиницю до отриманого числа: $1010_2 + 0001_2 = 1011_2$. Отримане число є представленням числа -5_{10} у доповняльному коді.

На рис. 2.30 зображено розподіл чисел, що представлені у доповняльному коді.

Видно, що всі від’ємні числа мають одиницю у старшому розряді, а додатні – нуль. Таким чином, старший розряд можна вважати знаковим. Також варто відзначити, що у випадку 4-розрядних чисел діапазон їх значень $[-8; 7]$ (порівняйте з діапазоном $[0; 15]$ для 4-розрядних чисел у звичайному двійковому представленні).

Тепер розглянемо арифметику з числами у доповняльному коді. Для простоти візьмемо 4-х розрядні числа.

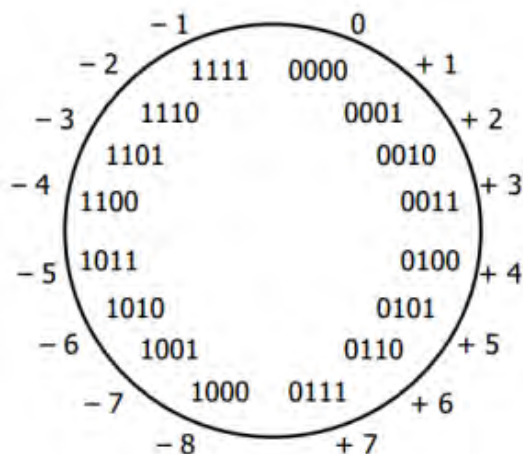


Рисунок 2.30 – Розподіл чисел у доповняльному коді

– $a+b$, $a=3_{10}=0011_2$, $b=2_{10}=0010_2$. Оскільки два числа додатні, то їх представлення у доповняльному коді відповідають представленню у звичайному коді. $a+b=0011_2+0010_2=0101_2=5_{10}$. Отримане число відповідає числу 5_{10} .

– $a-b$, $a=3_{10}=0011_2$, $b=2_{10}=0010_2$. Операція віднімання від числа від’ємника може бути замінена операцією додавання від’ємника, взятого з протилежним знаком, тобто $a-b=a+(-b)$. У даному випадку замість віднімання числа 2_{10} ми додаємо число -2_{10} . Оскільки $-2_{10}=1110_2$, то маємо $a+(-b)=0011_2+1110_2=10001_2$. Одиницею переносу нехтуємо і у результаті отримуємо число 0001_2 , тобто 1_{10} .

– $a-b$, $a=3_{10}=0011_2$, $b=7_{10}=0111_2$. Правила виконання віднімання ті ж самі, тому $a-b=a+(-b)=0011_2+1001_2=1100_2$. Отримане число має одиницю у старшому розряді, що свідчить про те, що воно від’ємне та записане у доповняльному коді. Результат відповідає числу -4_{10} .

– $a+b$, $a=-3_{10}=0011_2$, $b=-7_{10}=0111_2$. У даному випадку додаються два від’ємні числа. Переведемо їх у доповняльний код та виконаємо операцію складання. $a+b=1101_2+1001_2=10110_2$. Одиницею переносу нехтуємо, оскільки розрядів всього 4, і отримуємо число 0110_2 , тобто 6_{10} . Отриманий результат, очевидно, невірний. Така ситуація, що виникає при додаванні чисел у доповняльному коді і полягає у неправильному результаті, називається

переповненням (overflow). Іншими словами, значення результату вийшло за межі дозволеного діапазону. Також можна сказати, що при додаванні чисел одного знаку результат отримав протилежний знак.

Окрім арифметики, що використовує двійковий код, існує арифметика, що використовує так званий BCD (binary-coded decimal). У коді BCD кожен розряд десятичного числа кодується окремо у двійковий код. Наприклад, число 527_{10} буде мати наступний вигляд: $0101\ 0010\ 0111_2$, де кожен розряд представлений окремо (порівняйте зі звичайним представленням цього числа у двійковому коді $0010\ 0000\ 1111_2$).

При додаванні чисел у BCD необхідно враховувати, що кожен розряд має максимальне значення 9_{10} . Тому, якщо сума відповідних розрядів більша за 9_{10} , то необхідно скоригувати суму та сформувати біт перенесення у наступний розряд. Корекція суми відбувається додаванням до неї числа 6_{10} . Дійсно, якщо результат вийшов більшим за 9_{10} , то розряд десятків результату відкидається за допомогою віднімання 10_{10} від значення результату. Оскільки віднімання відбувається у доповняльному коді, то число 10_{10} представляється як 10110_2 . Старший розряд двійкового числа 10110_2 не враховується, тому що у коді BCD кожен десятичний розряд кодується за допомогою 4-х бітного двійкового числа. Таким чином залишається число 0110_2 , що відповідає числу 6_{10} .

2.3.1 Розробка 1-бітного повного суматору

За допомогою таблиці дійсності (рис. 2.29) створюємо у файлі ha.bdf схему напівсуматору (рис. 2.31). а відміну від повного суматору, не враховує біт переносу з попереднього розряду.

Встановлюємо файл ha.bdf як головну сутність, скопіюємо проєкт та виконаємо моделювання напівсуматору (рис. 2.32).

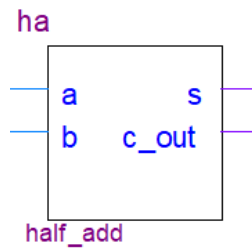


Рисунок 2.31 – Загальний вигляд напівсуматору

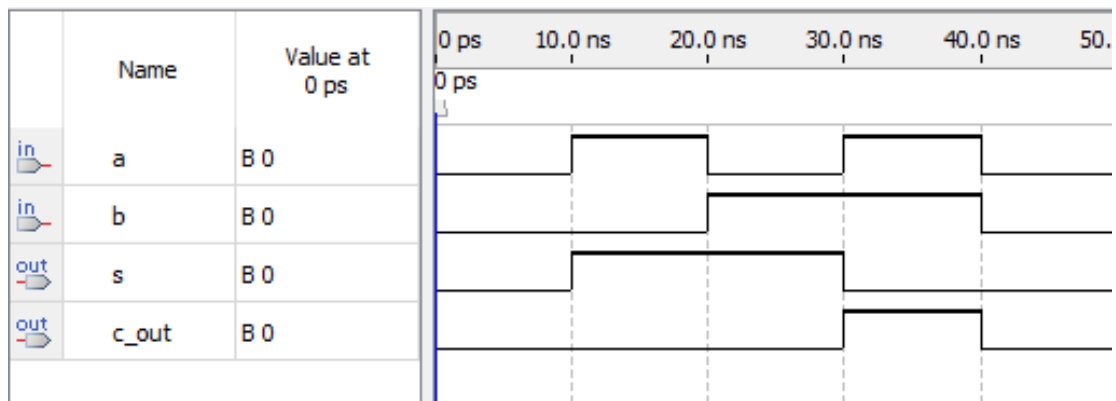


Рисунок 2.32 – Часова діаграма напівсуматору

На основі створеного символу напівсуматору створюємо у файлі add1.bdf 1-бітний повний суматор (рис. 2.33). Повний суматор використовує два напівсуматори, один з яких складає біти доданків, а інший – суму та біт перенесення з попереднього розряду. о також схему перенесення у наступний розряд.

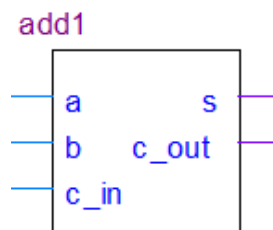


Рисунок 2.33 – Загальний вигляд повного суматору

Встановлюємо файл add1.bdf як головну сутність, компілюємо проєкт та виконуємо моделювання напівсуматору (рис. 2.34).

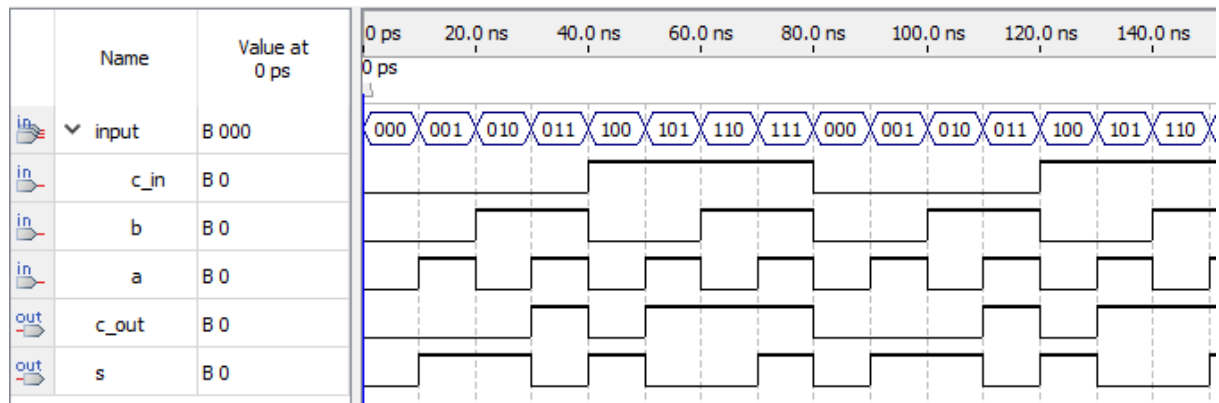


Рисунок 2.34 – Часова діаграма повного суматору

2.3.2 Розробка 4-бітного суматору

За допомогою символів повного суматору створюємо у файлі `add4x1.bdf` схему 4-бітного суматору (рис. 2.35), з'єднавши повні суматору між собою.

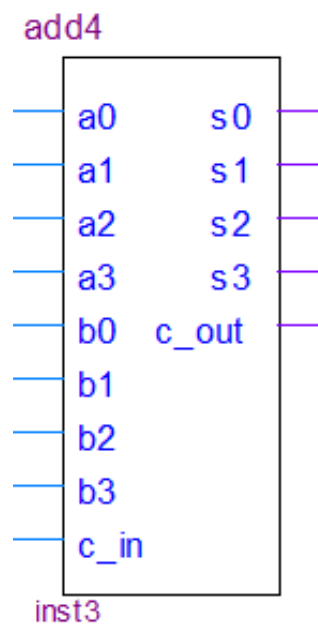


Рисунок 2.35 – Загальний вигляд 4-бітного суматору

Встановлюємо файл `add4x1.bdf` як головну сутність, компілюємо проект та виконуємо моделювання 4-бітного суматору (рис. 2.36).

Тестування 4-бітного суматору проводимо у файлі-шаблоні, як показано на рисунку 2.37.

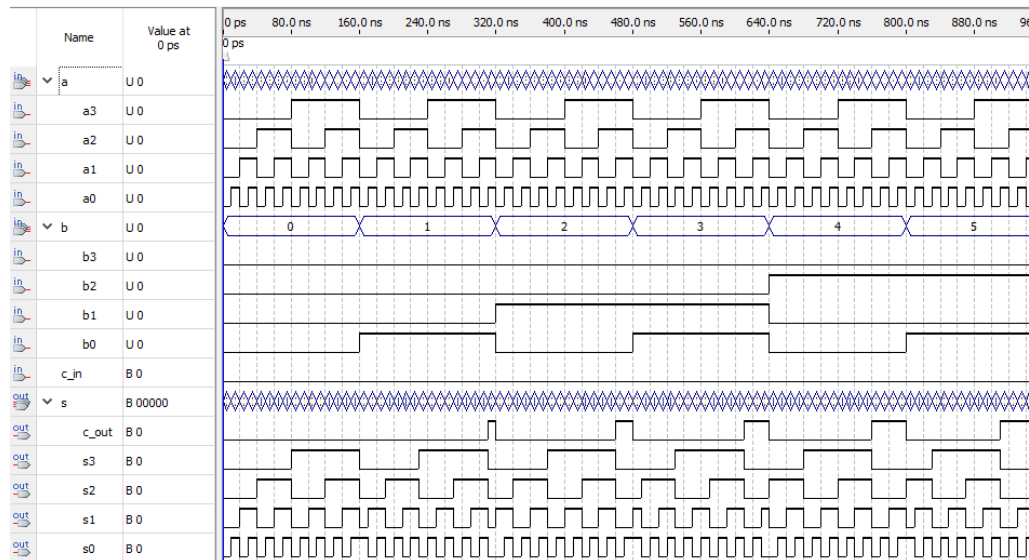


Рисунок 2.36 – Часова діаграма 4-бітного суматора

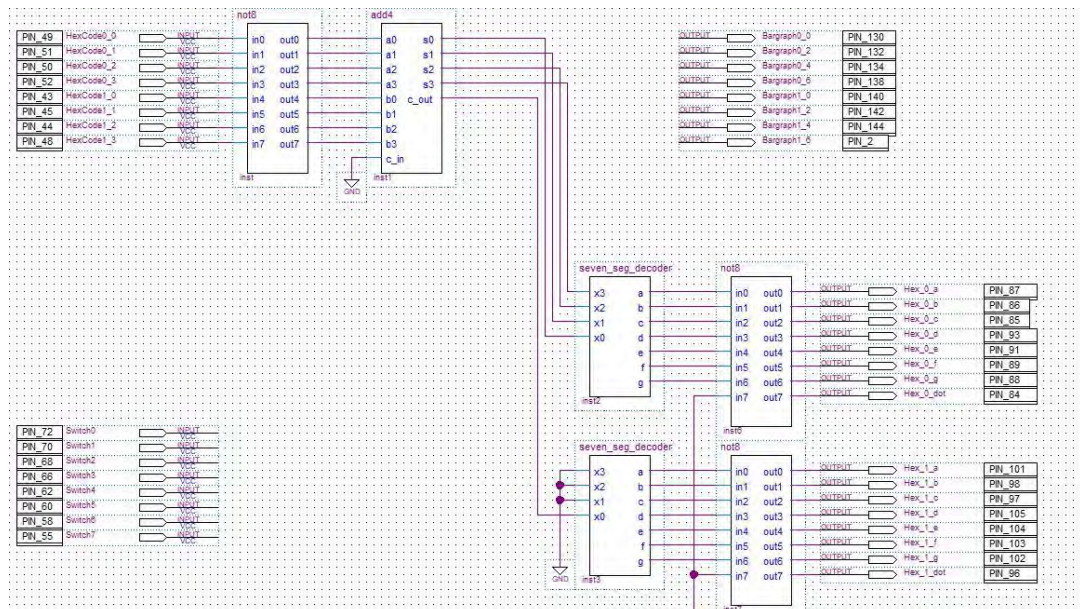


Рисунок 2.37 – Схема для тестування 4-бітного суматора

2.3.3 Розробка BCD-суматора

У файлі `bcd_add_1x4.bdf` створюємо схему суматора BCD-чисел (рис. 2.38). Суматор BCD-чисел складається з двох 4-бітних суматорів, з яких один складає розряди двох чисел, а інший виконує корекцію результату відповідно до сигналу корекції. Сигнал про необхідність корекції встановлюється при виявленні так званих «заборонених» комбінацій, тобто

таких, що не можуть з’явитися на виході схеми. Сигнал про необхідність переносу відповідає сигналу корекції результату.

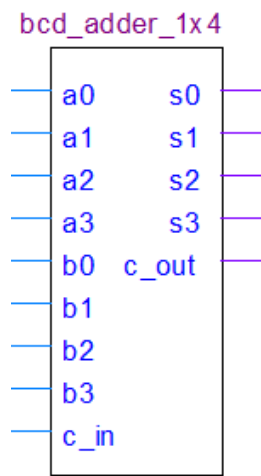


Рисунок 2.38 – Загальний вигляд 1-розрядного BCD-суматору

Встановлюємо файл `bcd_add_1x4.bdf` як головну сутність, компілюємо проект та виконуємо моделювання повного суматору (рис. 2.39).

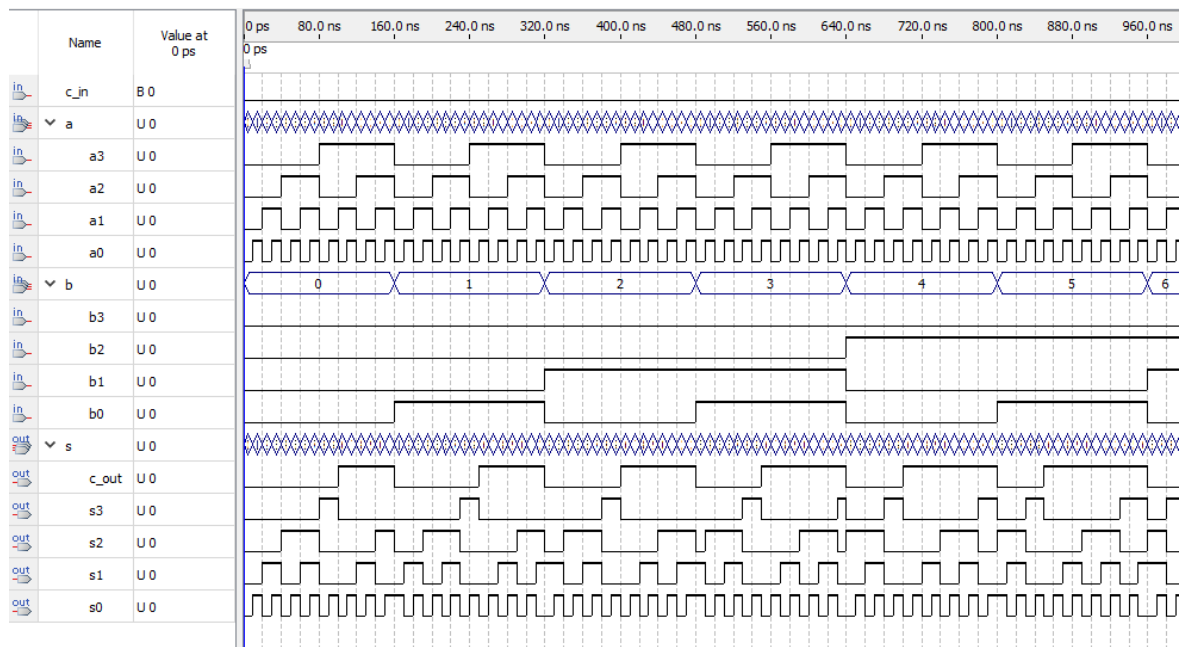


Рисунок 2.39 – Часова діаграма 1-розрядного BCD-суматору

У створеному файлі bcd_add_2x4.bdf на основі двох символів 1-розрядного VCD-суматору створюємо схему 2-розрядного VCD-суматору (рис. 2.40).

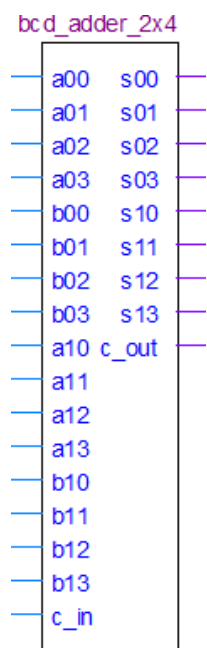


Рисунок 2.40 – Загальний вигляд 2-розрядного VCD-суматору

На рис. 2.41 наведено результати моделювання 2-розрядного VCD-суматору

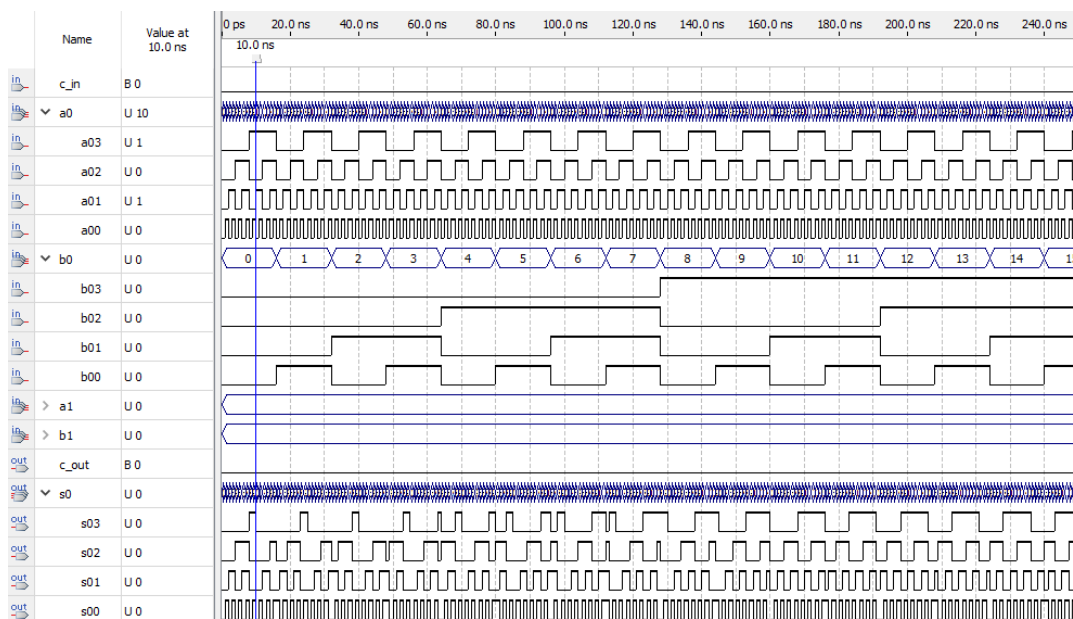


Рисунок 2.41 – Часова діаграма 2-розрядного VCD-суматору

На рисунку 2.42 наведено схему тестування 2-розрядного BCD-суматору.

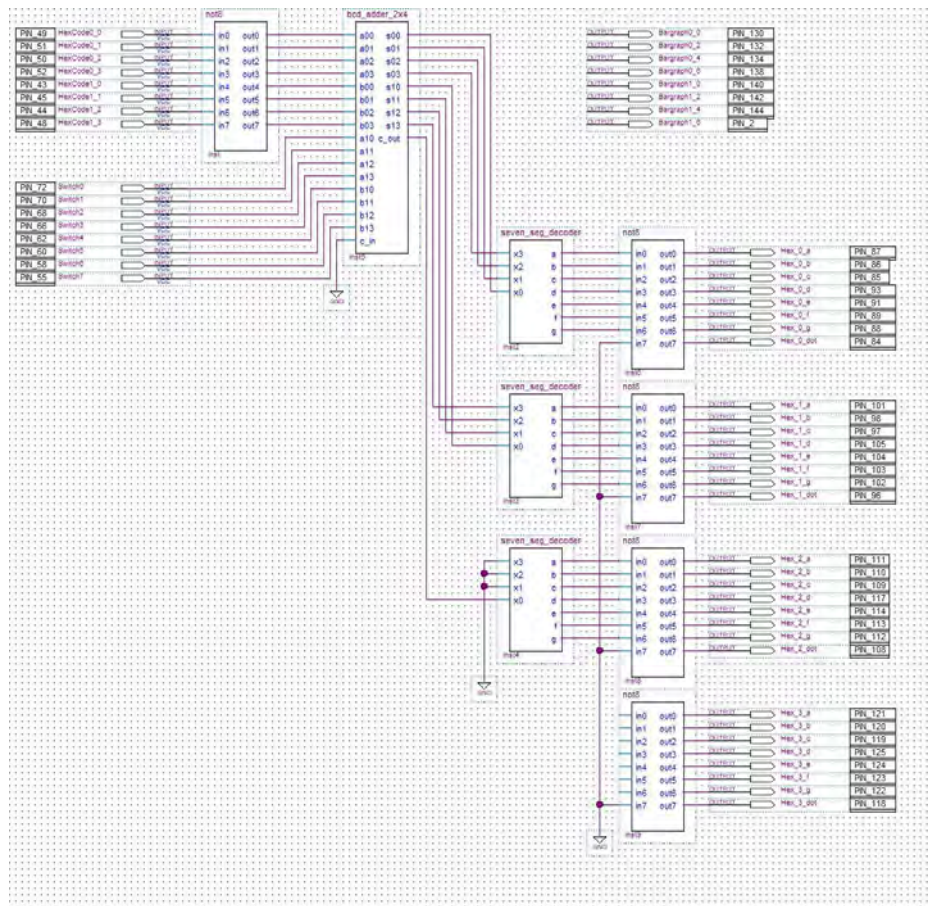


Рисунок 2.42 – Схема для тестування 2-розрядного BCD-суматору

2.3.4 Розробка двійкового суматору/віднімача

На основі 4-бітного суматору та правил складання двійкових чисел у доповняльному коді можна створити схему двійкового суматору/віднімача з сигналом вибору режиму роботи.

У файлі `add_sub_2x4.bdf` створюємо схему 8-бітного суматору/віднімача. Враховуємо сигнал вибору режиму `sel` (`sel=0` – режим складання, `sel=1` режим віднімання), сигнал переповнення `overflow` та перенесення `c_out` (рис. 2.43).

Встановлюємо файл `add_sub_2x4.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання повного суматору (рис. 2.44).

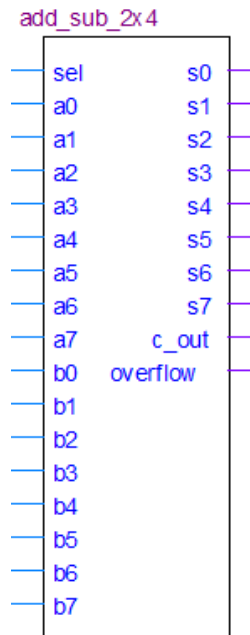


Рисунок 2.43 – Загальний вигляд 8-бітного суматора/віднімача

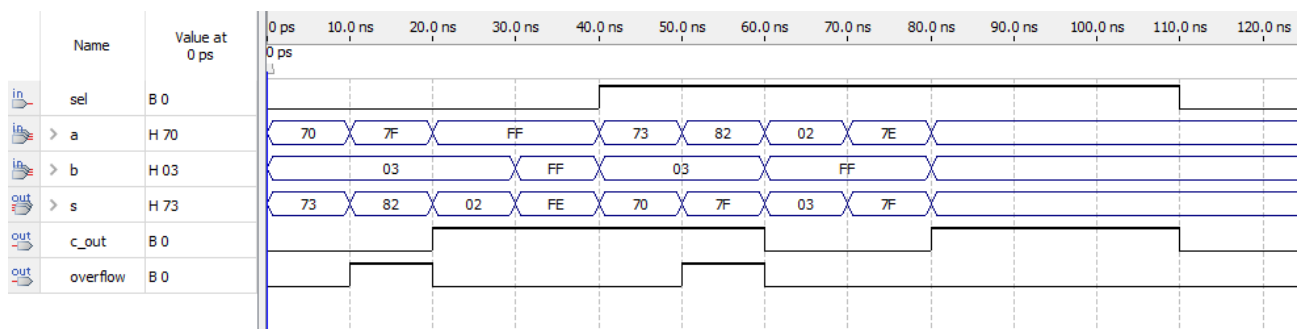


Рисунок 2.44 – Часова діаграма 8-бітного суматора/віднімача

2.3.5 Розробка перетворювача доповняльного коду

Перед тим, як перейти до тестування суматора/віднімача, необхідно створити схем перетворювача з доповняльного коду у прямий.

У файлі `dec_2K_8bit.bdf` створюємо 8-бітний перетворювач доповняльного коду. Схema перетворювача повинна мати сигнал `sign`, що позначає від'ємні числа (рис. 2.45).

Встановлюємо файл `dec_2K_8bit.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання повного суматора (рис. 2.46).

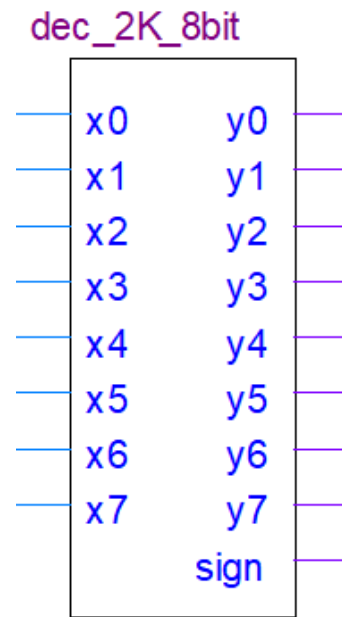


Рисунок 2.45 – Загальний вигляд 8-бітного перетворювача доповняльного коду

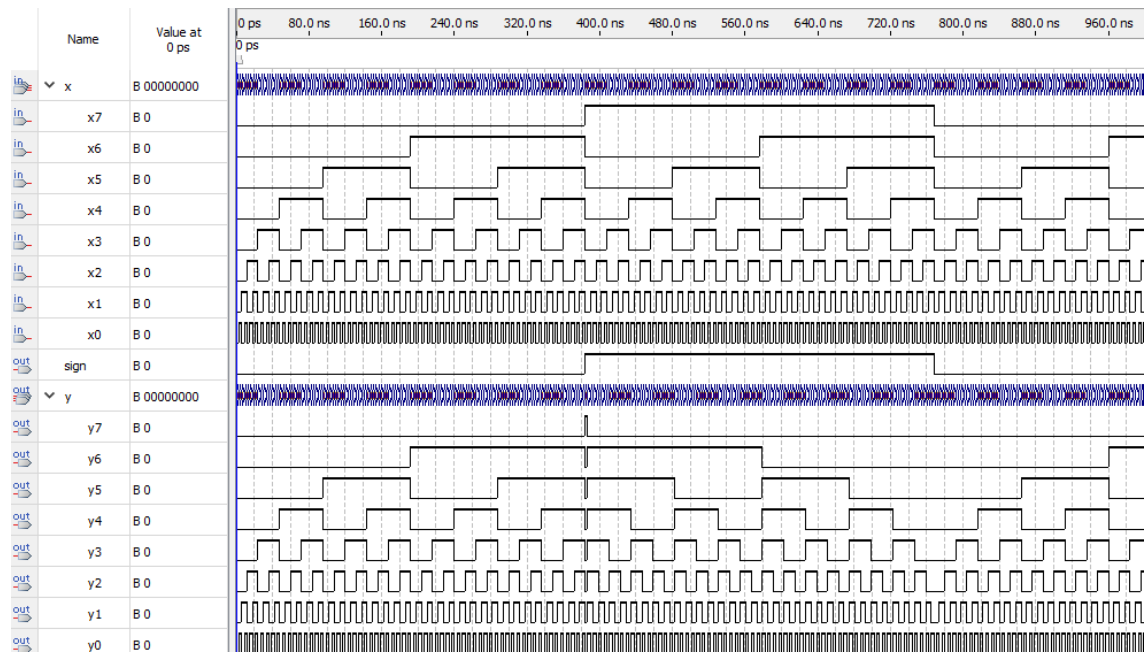


Рисунок 2.46 – Часова діаграма 8-бітного перетворювача доповняльного коду

Схема тестування 8-бітного перетворювача доповняльного коду наведена на рис. 2.47.

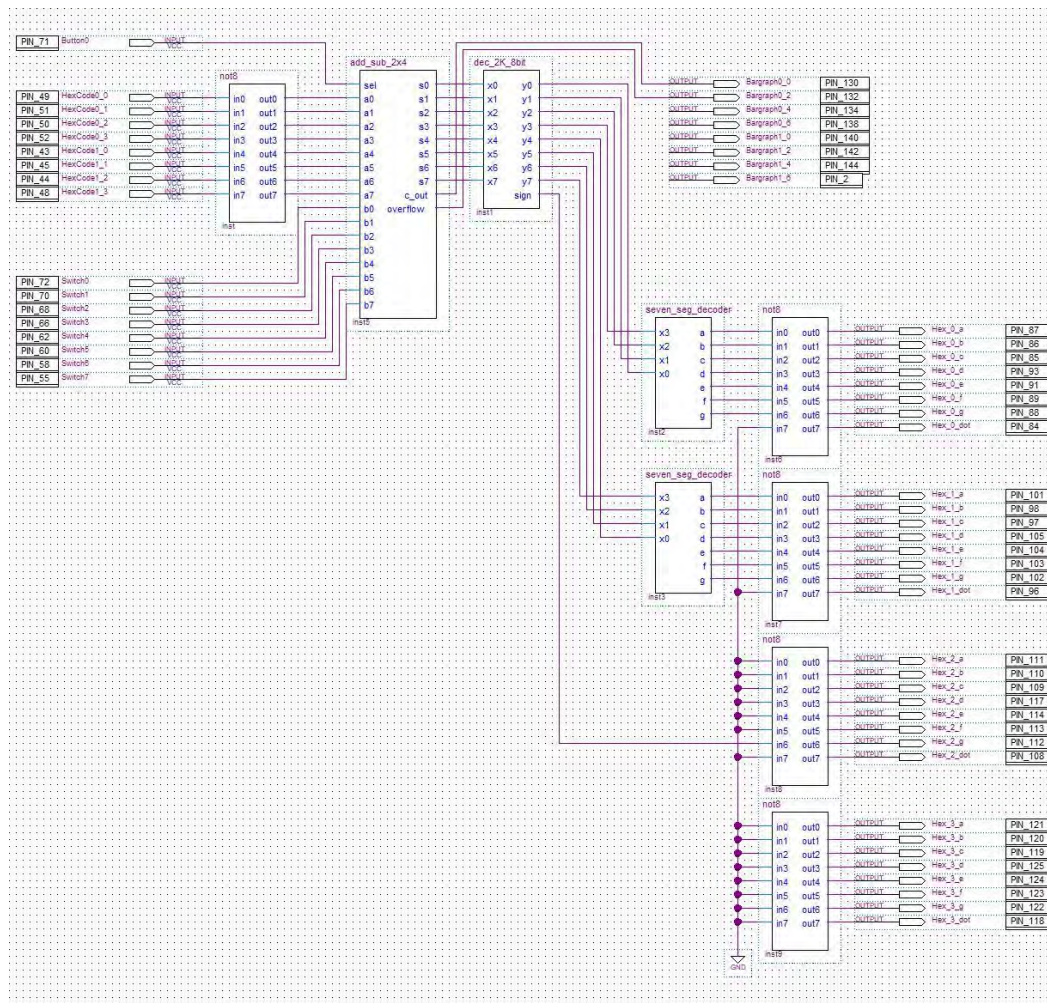


Рисунок 2.47 – Схема для тестування 8-бітного суматору/віднімача

2.4 Розробка компаратору

Пристрої, які порівнюють два числа і показують, рівні вони чи ні, називаються компараторами. Деякі компаратори інтерпретують вхідні слова як числа зі знаком або без знаку, а також видають арифметичне відношення між числами (більше або менше). Такі пристрої називаються компараторами значень (magnitude comparators) [28].

На рисунку 2.48 представлено загальний вигляд 4-бітного компаратору.

Дослідити принципи роботи компаратору можна проаналізувавши його таблицю дійсності. Для простоти розглянемо таблицю дійсності 2-бітного компаратору, що представлена на рисунку 2.49. Сигнал y_2 позначає, що двійкове число a більше за b , сигнал y_1 – дорівнює b , а сигнал y_0 – менше b .

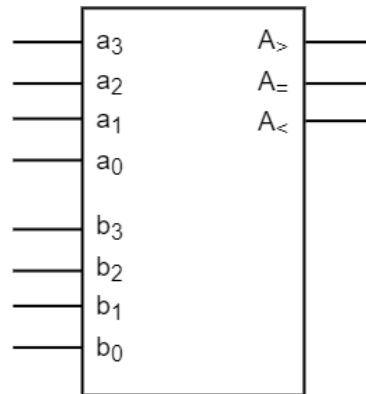


Рисунок 2.48 – Загальний вигляд 4-бітного компаратору

a_1	a_0	b_1	b_0	y_2	y_1	y_0
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Рисунок 2.49 – Таблиця дійсності 2-бітного компаратору

За допомогою таблиці дійсності компаратору можна побудувати логічні рівняння його виходів та мінімізувати їх за допомогою карт Карно.

На основі малорозрядних комбінаційних схем зазвичай будуються багаторозрядні. У даному випадку можна на основі 2-розрядного компаратору побудувати 8-розрядний.

Проаналізуємо принцип роботи багаторозрядних компараторів. Пристрої порівняння на рівність будуються на основі порозрядних операцій над однойменними розрядами обох чисел. Числа рівні, якщо рівні всі їх однойменні розряди, тобто якщо в обох нулі або одиниці [29]. Позначимо через r_i ознаку

рівності двох i -х розрядів. Тоді ознака рівності двох чисел буде виражатися наступною формулою:

$$R = r_{n-1}r_{n-2} \dots r_0 \quad (2.5)$$

Позначимо через m_i ознаку того, що розряд a_i більший за розряд b_i . Тоді ознаку того, що число a більше за число b можна вивести на основі наступних роздумів. Якщо старші розряди не дорівнюють один одному, то результат відомий незалежно від значень молодших розрядів: при $a_{n-1}=1$ $b_{n-1}=0$ ($m_{n-1}=1$) маємо $a>b$, а при $a_{n-1}=0$ $b_{n-1}=1$ ($m_{n-1}=0$) маємо $a<b$. Якщо $a_{n-1}=b_{n-1}$, то результат ще невідомий і необхідний аналіз наступного розряду за цим же алгоритмом. Даний підхід можна використовувати для чисел будь-якої розрядності – до аналізу наступного розряду можна переходити лише за рівності попередніх. Таким чином, формула для порівняння на ознаку більшості n -розрядних чисел наступна [29]:

$$F_{A>B} = m_{n-1} + r_{n-1}m_{n-2} + \dots + r_{n-1}r_{n-2} \dots r_1m_0 \quad (2.6)$$

Формула для порівняння на ознаку меншості n -розрядних чисел ідентична.

Тому 2-бітний компаратор може слугувати будівельним блоком для 8-бітного компаратору, оскільки він вже має окремі сигнали ознак рівності, більшості або меншості.

Ще одним способом побудови компараторів та комбінаційних схем взагалі є ітераційний підхід. Ітераційна схема представляє собою спеціальний тип комбінаційної схеми, структура якої показана на рис. 2.50. Схема складається з n ідентичних модулів, кожен з яких має як первинні входи та виходи (primary inputs and outputs), так і міжкаскадні входи та виходи (cascading inputs and outputs). У більшості ітераційних схем у крайнього лівого модуля є

окремі входи, що називаються граничними входами (boundary inputs) та на них подаються фіксовані логічні значення. Спеціально виокремленні виходи крайнього правого модуля називаються граничними виходами (boundary outputs), і зазвичай сигнали на цих виходах несуть важливу інформацію [28].

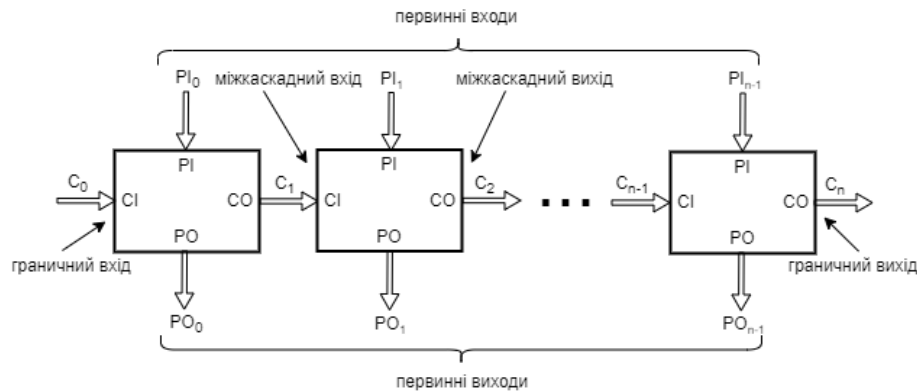


Рисунок 2.50 – Структура ітераційної схеми

Ітераційні схеми дуже добре підходять для задач, що можна вирішити, застосувавши простий ітераційний алгоритм [28].

Крок 1. Встановити початкові значення C_0 та $i=0$.

Крок 2. Використовуємо C_i та PI_i для визначення значень PO_i та C_{i+1} .

Крок 3. Збільшуємо значення i на 1.

Крок 4. Якщо $i < n$, то переходимо до кроку 2.

Розробити ітераційну схему компаратору можна наступним чином. Для цього створюється модуль, який порівнює окремі розряди чисел та граничний модуль, який приймає рішення про відношення між двома числами. Модулі з'єднуються за схемою, зображеною на рис. 2.51.

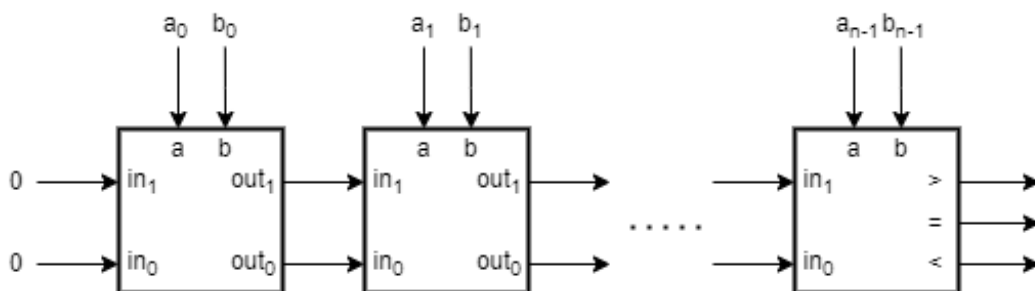


Рисунок 2.51 – Ітераційна схема компаратору

На рисунку 2.52 наведено таблицю дійсності окремого модулю порівняння двох розрядів чисел.

in_1	in_0	a_i	b_i	out_1	out_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	x	x
1	1	0	1	x	x
1	1	1	0	x	x
1	1	1	1	x	x

Рисунок 2.52 – Таблиця дійсності модуля порівняння розрядів

Виходячи з таблиці дійсності окремого модуля порівняння розрядів, можна дослідити роботу ітераційного компаратору.

Сигнали in_1 та in_0 позначають відношення двох чисел на даному етапі порівняння. Комбінація сигналів $in_1in_0=00$ свідчить про рівність двох чисел на даному етапі, 01 – про те, що число a менше за b , а 10 – про те, що більше. Комбінація 11 є забороненою і не використовується і в таблиці дійсності для неї виходи не визначені (don't care value).

Якщо два числа на момент порівняння i -х розрядів були рівні, а в i -у розряді $a_i > b_i$, то число a більше за b і на виході буде комбінація сигналів $out_1out_0=10$. Якщо в i -у розряді $a_i < b_i$, то число a менше за b і на виході буде комбінація сигналів $out_1out_0=01$. Інакше, числа рівні і на виході отримаємо комбінацію сигналів $out_1out_0=00$. Таким же чином відбувається порівняння

розрядів, якщо на момент порівняння одне з чисел було більше (менше) за інше.

Граничний модуль видає сигнал про відношення двох чисел на основі вхідної комбінації сигналів in_1in_0 .

2.4.1 Розробка 8-бітного компаратору

За допомогою таблиці дійсності (рис. 2.52) створюємо у файлі `comparator_2bit.bdf` схему 2-бітного компаратору (рис. 2.53). Сигнал y_2 позначає те, що a більше за b , сигнал y_1 позначає рівність чисел, а сигнал y_0 – те, що a менше за b . Встановлюємо файл `comparator_2bit.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання 2-бітного компаратору (рис. 2.54).

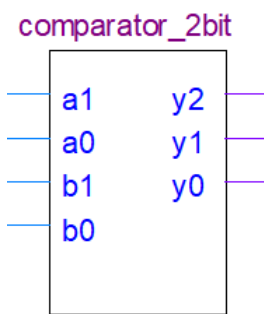


Рисунок 2.53 – Загальний вигляд 2-бітного компаратору

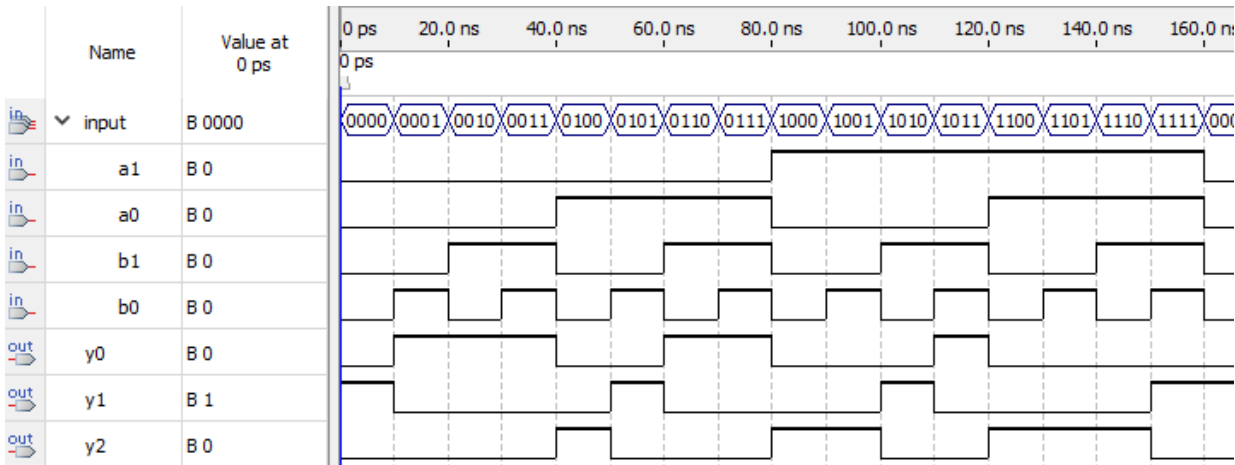


Рисунок 2.54 – Часова діаграма 2-бітного компаратору

На основі створеного символу 2-бітного компаратору створюємо у фалі comparator_8bit.bdf 8-бітний компаратор (рис. 2.55). Для цього користуємося ознаками відношення чисел, представленими формулами 2.5 та 2.6.

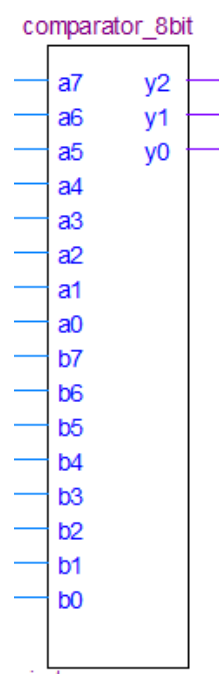


Рисунок 2.55 – Загальний вигляд 8-бітного компаратору

Встановлюємо файл comparator_8bit.bdf як головну сутність, компілюємо проєкт та виконуємо моделювання 8-бітного компаратору (рис. 2.56).

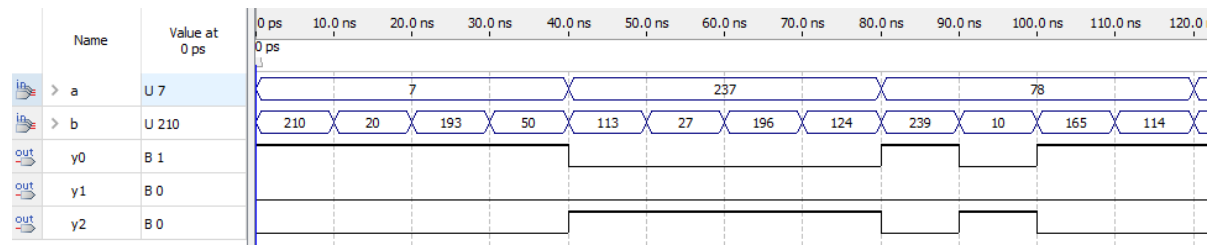


Рисунок 2.56 – Часова діаграма 8-бітного компаратору

На рисунку 2.57 наведено схему тестування 8-бітного компаратору.

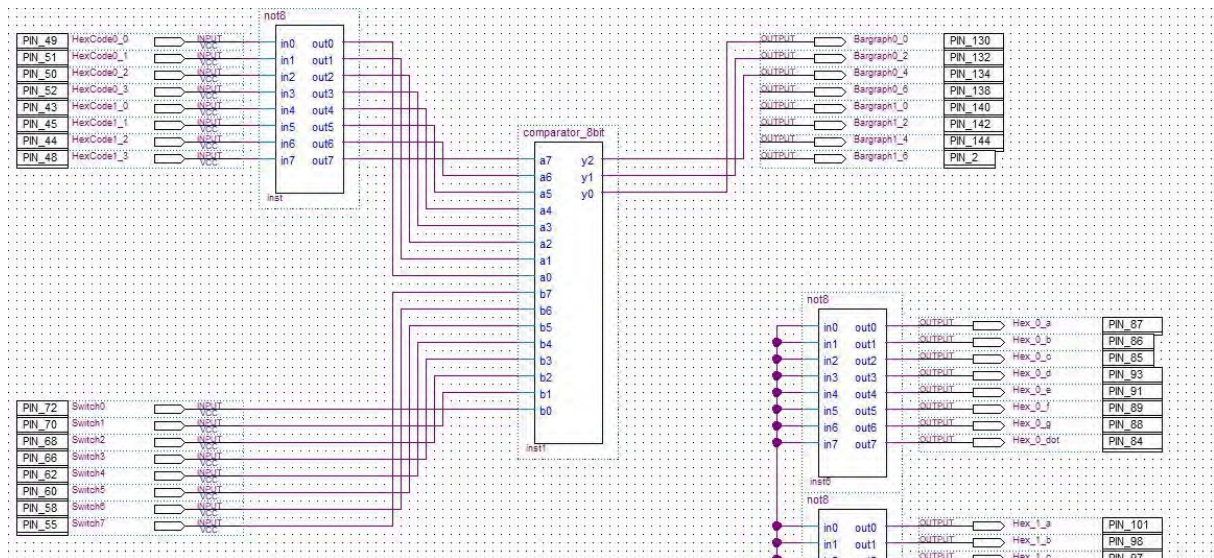


Рисунок 2.57 – Схема для тестування 8-бітного компаратору

2.4.2 Розробка 4-бітного ітераційного компаратору

За допомогою таблиці дійсності (рис. 2.52) створюємо у файлі `comparator_iterative_cell.bdf` схему модуля порівняння розрядів (рис. 2.58).

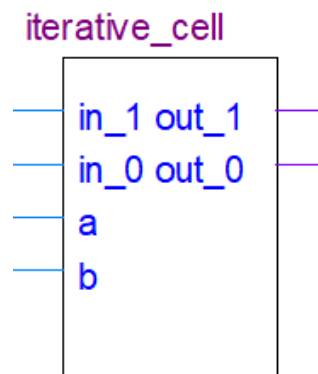


Рисунок 2.58 – Загальний вигляд модуля порівняння розрядів

Встановлюємо файл `comparator_iterative_cell.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання (рис. 2.59).

На основі значень комбінацій сигналів in_1 та in_0 розроблюємо у файлі `comparator_iterative_boundary.bdf` схему переведення їх у сигнали y_2 , y_1 та y_0 (рис. 2.60).

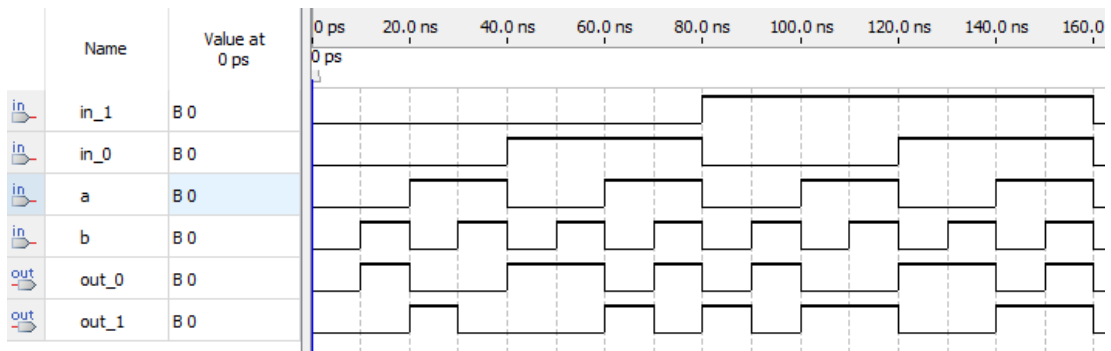


Рисунок 2.59 – Часова діаграма модуля порівняння розрядів

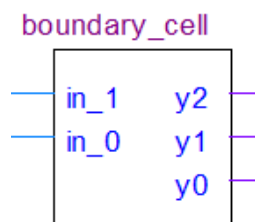


Рисунок 2.60 – Загальний вигляд граничного модуля

Встановлюємо файл `comparator_iterative_boundary.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання граничного модуля (рис. 2.61).

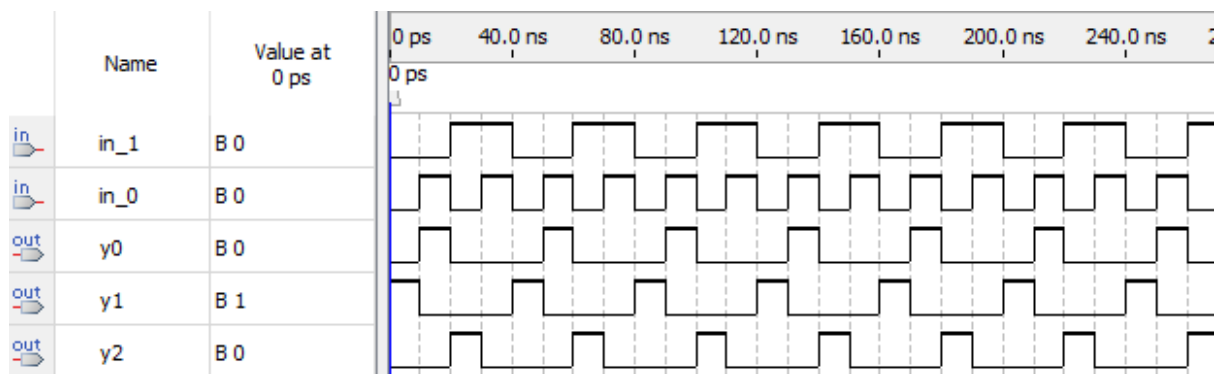


Рисунок 2.61 – Часова діаграма граничного модуля

На основі символів модулів порівняння розрядів та граничного модуля розроблюємо у файлі `comparator_iterative_4bit.bdf` схему 4-бітного ітеративного компаратору (рис. 2.62). Встановлюємо файл `comparator_iterative_4bit.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання ітеративного компаратору (рис. 2.63).

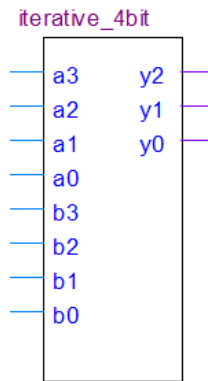


Рисунок 2.62 – Загальний вигляд 4-бітного ітеративного компаратору

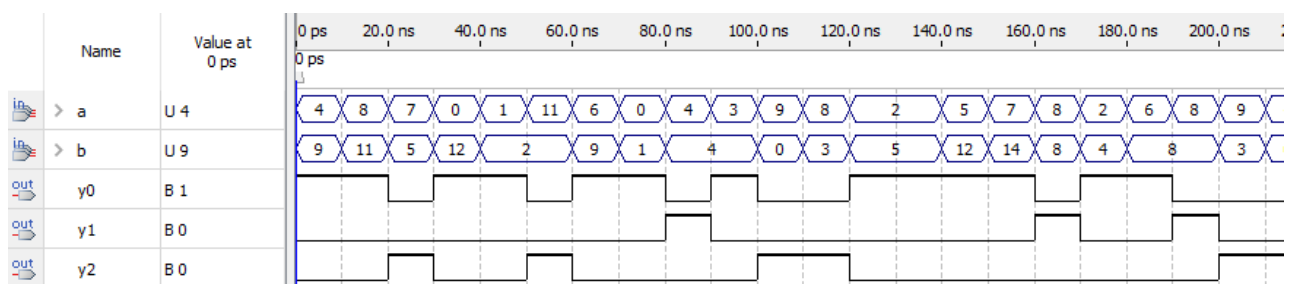


Рисунок 2.63 – Часова діаграма 4-бітного ітеративного компаратору

На рисунку 2.64 наведено схему тестування 4-бітного ітеративного компаратору.

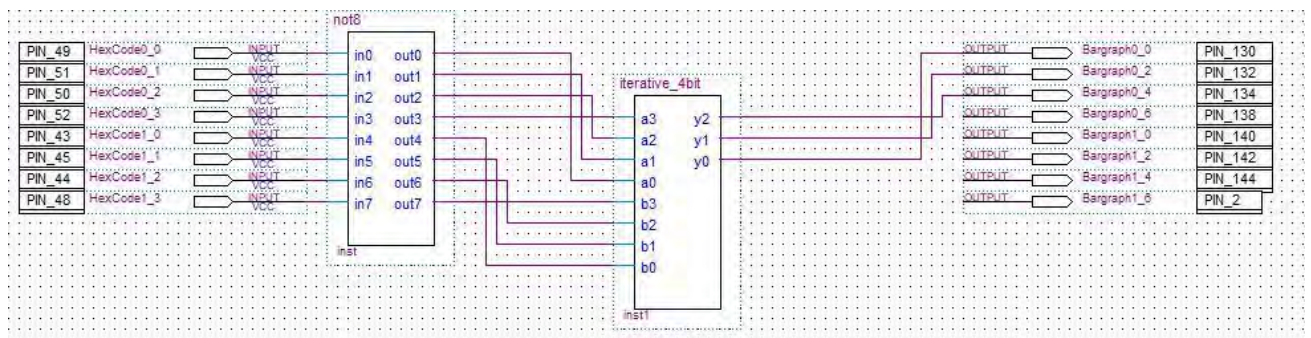


Рисунок 2.64 – Схема для тестування 4-бітного ітеративного компаратору

Отже, створити та провести моделювання компаратору можна за допомогою можливостей середовища проєктування Quartus II.

2.5 Розробка арифметико-логічного пристрою

Арифметико-логічний пристрій (АЛП, arithmetic and logic unit, ALU) є комбінаційною схемою, що виконує цілий ряд різноманітних арифметичних і логічних операцій з парою операндів однакової розрядності. Операція, що виконується у даний момент визначається комбінацією сигналів на входах вибору функції. Типові АЛП, що виконані у вигляді ІС середнього ступеня інтеграції, працюють з 4-розрядними операндами та мають від трьох до п'яти входів вибору функції, що дозволяє виконувати їм до 32 різноманітних функцій.

На рисунку 2.65 наведено умовне позначення 4-бітного АЛП 74181, що був першим АЛП, виконаним у мікросхемі.

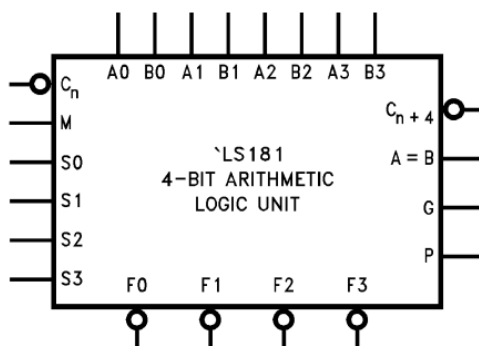


Рисунок 2.65 – Умовне позначення АЛП 74181

На рисунку 2.66 наведено перелік функцій, які може виконувати даний АЛП в залежності від сигналів вибору операції. АЛП 74181 може виконувати 32 функції, з яких 16 логічних та 16 арифметичних. Також у АЛП присутні виходи G та P (carry generate та carry propagate), що дозволяють комбінувати АЛП у структури більшої розрядності, використовуючи схеми прискореного переносу.

MODE SELECT INPUTS				ACTIVE HIGH INPUTS AND OUTPUTS	
S_3	S_2	S_1	S_0	LOGIC (M=H)	ARITHMETIC ⁽²⁾ (M=L; C _n =H)
L	L	L	L	$\bar{A}$	A
L	L	L	H	$\overline{A+B}$	A + B
L	L	H	L	$\overline{AB}$	A + $\bar{B}$
L	L	H	H	logical 0	minus 1
L	H	L	L	$\overline{AB}$	A plus $\overline{AB}$
L	H	L	H	$\bar{B}$	(A + B) plus $\overline{AB}$
L	H	H	L	$A \oplus B$	A minus B minus 1
L	H	H	H	$\overline{AB}$	$\overline{AB}$ minus 1
H	L	L	L	$\bar{A} + B$	A plus AB
H	L	L	H	$\overline{A \oplus B}$	A plus B
H	L	H	L	B	(A + $\bar{B}$) plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	logical 1	A plus A ⁽¹⁾
H	H	L	H	$A + \bar{B}$	(A + B) plus A
H	H	H	L	$A + B$	(A + $\bar{B}$) plus A
H	H	H	H	A	A minus 1

Рисунок 2.66 – Функції АЛП 74181

2.5.1 Розробка структурної схеми АЛП

Початковим кроком в розробці АЛП є визначення функцій, які можна виконати над операндами (рис. 2.67).

S_2	S_1	S_0	Function
0	0	0	AB
0	0	1	$A + B$
0	1	0	$A \oplus B$
0	1	1	$\bar{A}$
1	0	0	A plus B
1	0	1	A minus B
1	1	0	A minus 1
1	1	1	A plus 1

Рисунок 2.67 – Функції АЛП

Таким чином, АЛП може виконувати 4 логічні операції та 4 арифметичні операції. Можна також побачити, що логічні операції виконуються при $S_2=0$, а арифметичні – при $S_2=1$.

На рис. 2.68 наведено структурну схему так званої 1-бітної секції (1-bit slice) АЛП, яку згодом можна об'єднати у ланцюг з іншими секціями, збільшивши таким чином розрядність. Видно, що сигнал контролює S_2 мультиплексор, що направляє до виходу сигнал арифметичного або логічного модуля.

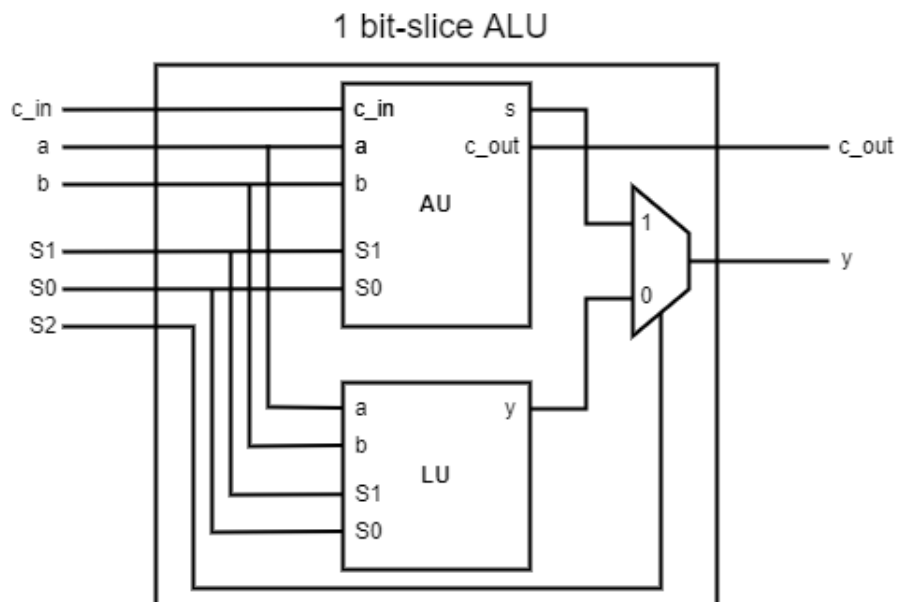


Рисунок 2.68 – Структурна схема секції АЛП

На рис. 2.69 та 2.70 наведено структурні схеми арифметичного блоку та блоку операцій над операндом b (B_GEN).

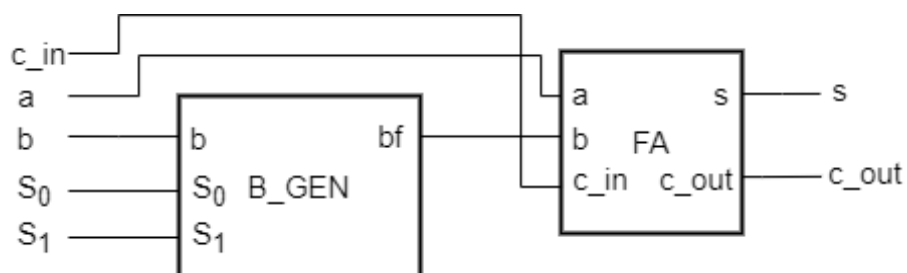


Рисунок 2.69 – Структурна схема арифметичного блоку

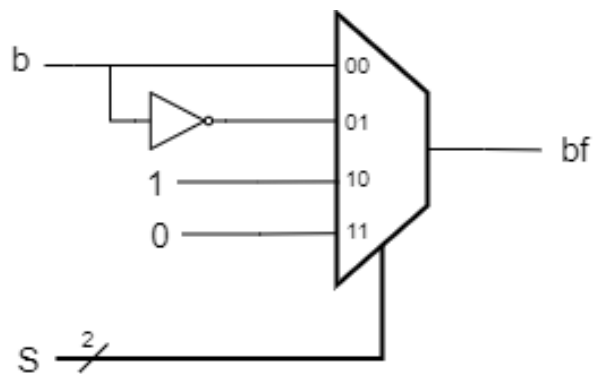


Рисунок 2.70 – Структурна схема блоку B_GEN

На рис. 2.71 наведено структурну схему 4-бітного АЛП, зібраного з чотирьох секцій та блоку статусу.

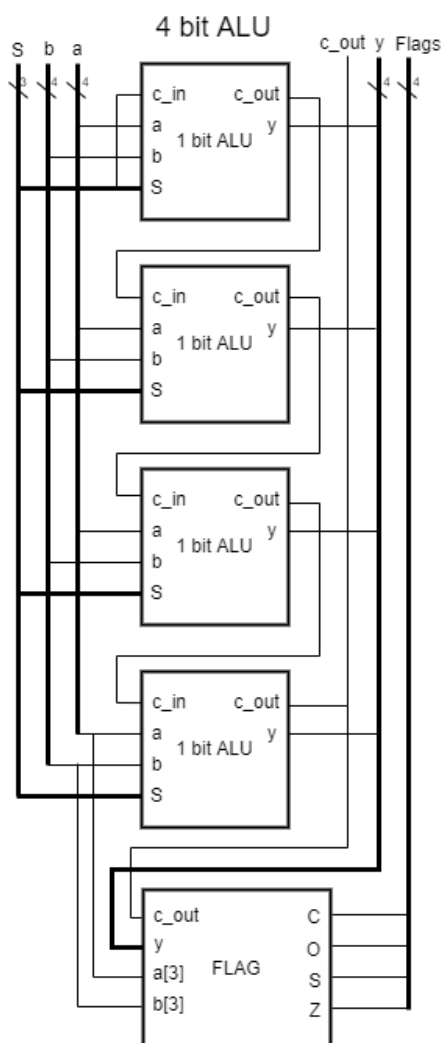


Рисунок 2.71 – Структурна схема 4-бітного АЛП

Слід зазначити, що у схемі 4-бітного АЛП сигнал S_0 з'єднаний зі входом переносу c_in першої секції. Це необхідно для коректного виконання операції віднімання. Дійсно, при комбінації сигналів $S_2S_1S_0=101$ виконується операція $A-B$. Сигнали S_1S_0 в свою чергу виконують перетворення числа B всередині блоку B_GEN . У даному випадку, при $S_1S_0=01$, число B інвертується. Інвертування числа та додавання до молодшого розряду одиниці є операцією переведення до доповняльного коду. При аналізі команд додавання та віднімання одиниці виконуються ті ж самі роздуми.

2.5.2 Розробка структурних блоків АЛП у середовищі Quartus

За допомогою таблиці 2.6 функцій АЛП створимо у файлі `lu.bdf` схему логічного блоку (рис. 2.72). Для створення логічного блоку використовується символ мультиплексора так, як показано на рис. 2.73. Зазначте у параметрах мультиплексора ширину входу та під'єднайте до нього шину (Orthogonal Bus Tool). Назвіть шину, зазначивши у квадратних дужках індекси з максимального до мінімального. З'єднувати сигнали з шиною необхідно називаючи їх іменами, що вказують індекс сигналу у шині.

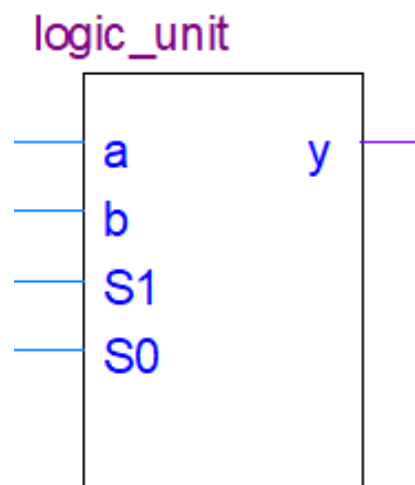


Рисунок 2.72 – Символ логічного блоку

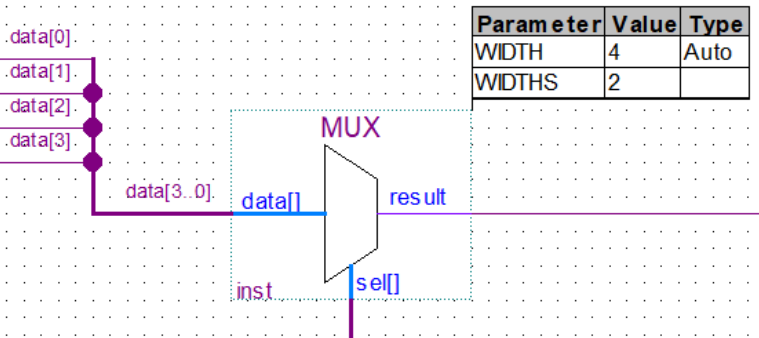


Рисунок 2.73 – Використання мультиплексору

Встановлюємо файл lu.bdf як головну сутність, компілюємо проєкт та виконуємо моделювання (рис. 2.74).

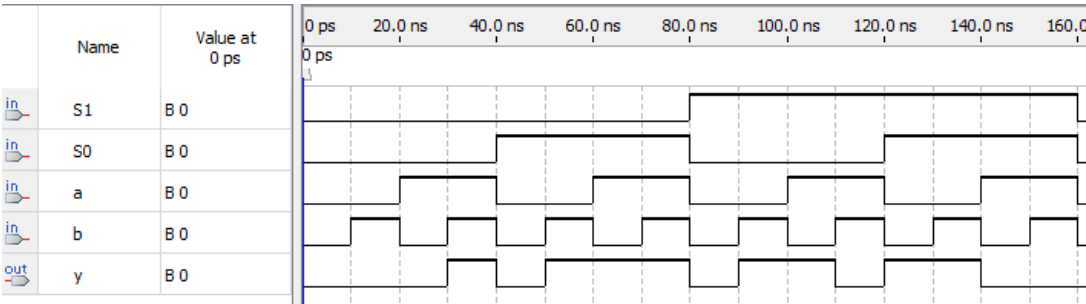


Рисунок 2.74 – Часова діаграма логічного блоку

На основі структурної схеми на рис. 2.70 зберемо схему блоку B_GEN (рис. 2.75) у файлі b_gen.bdf.

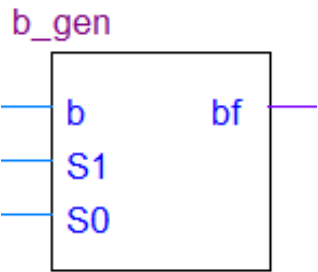


Рисунок 2.75 – Символ блоку B_GEN

Встановлюємо файл b_gen.bdf як головну сутність, компілюємо проєкт та виконуємо моделювання (рис. 2.76).

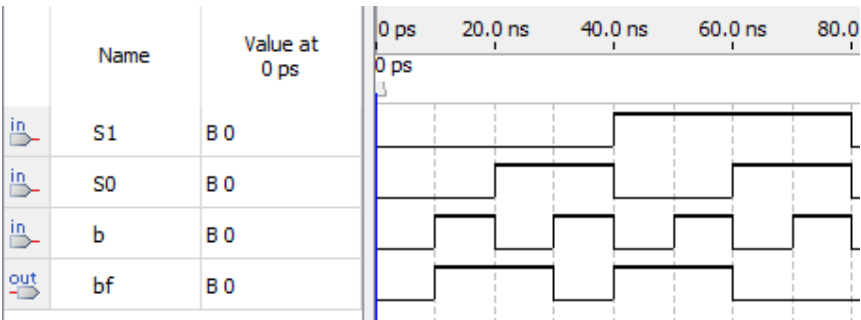


Рисунок 2.76 – Часова діаграма блоку B_GEN

Користуючись схемою на рис. 2.69 збираємо арифметичний блок (рис. 2.77).

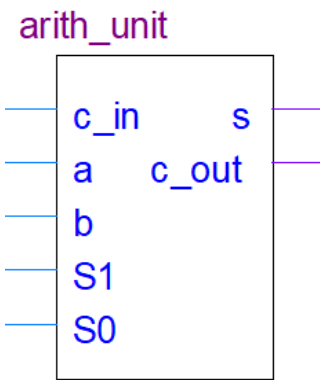


Рисунок 2.77 – Символ арифметичного блоку

Встановлюємо файл au.bdf як головну сутність, компілюємо проєкт та виконуємо моделювання (рис. 2.78).

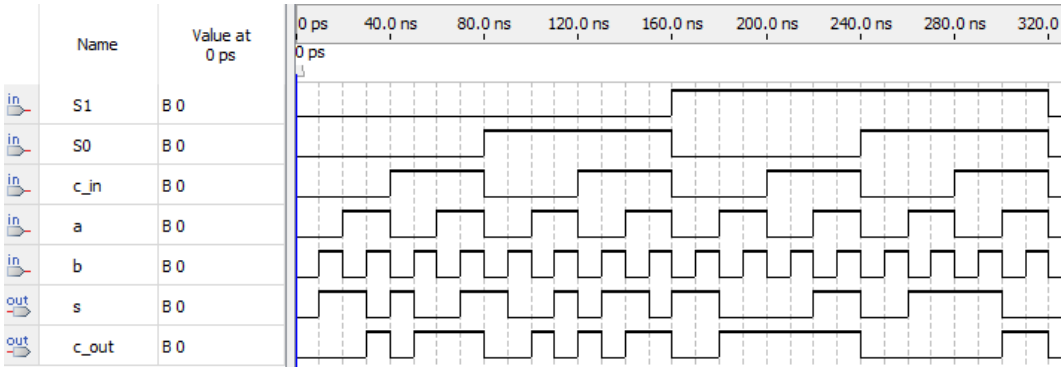


Рисунок 2.78 – Часова діаграма арифметичного блоку

Користуючись схемою на рис. 2.68 збираємо 1-бітну секцію АЛП (рис. 2.79).

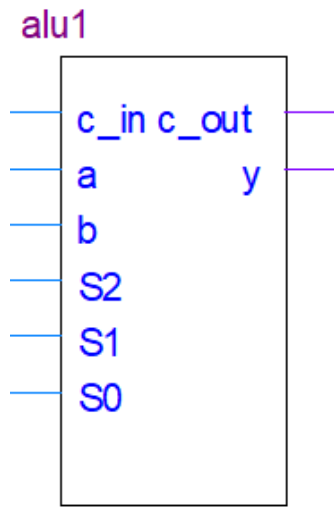


Рисунок 2.79 – Символ секції АЛП

Встановлюємо файл alu1.bdf як головну сутність, компілюємо проєкт та виконуємо моделювання (рис. 2.80).

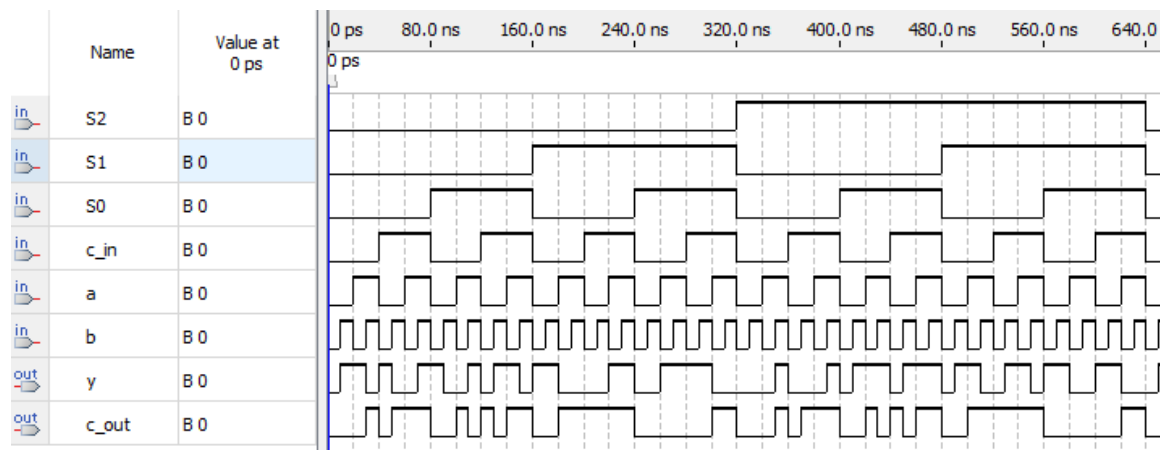


Рисунок 2.80 – Часова діаграма секції АЛП

Користуючись правилами перевірки числа на нуль, наявності переповнення та від’ємного знаку збираємо схему блоку статусу (рис. 2.81).

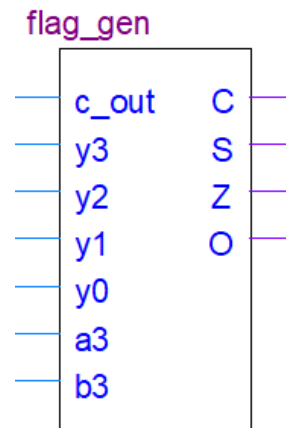


Рисунок 2.81 – Символ блоку статусу

Встановлюємо файл `flag_gen.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання (рис. 2.82).

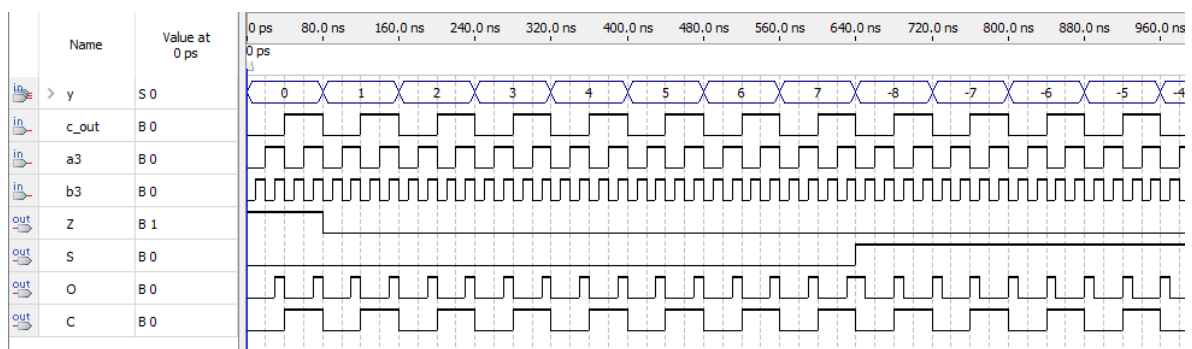


Рисунок 2.82 – Часова діаграма блоку статусу

2.5.3 Розробка 4-бітного АЛП у середовищі Quartus

Створюємо файл `alu4x1.bdf` та додаємо до нього символи блоку статусу та 1-бітних секцій. Користуючись структурною схемою, наведеною на рис. 2.71 збираємо схему 4-бітного АЛП (рис. 2.83).

Встановлюємо файл `alu4x1.bdf` як головну сутність, компілюємо проєкт та виконуємо моделювання (рис. 2.84).

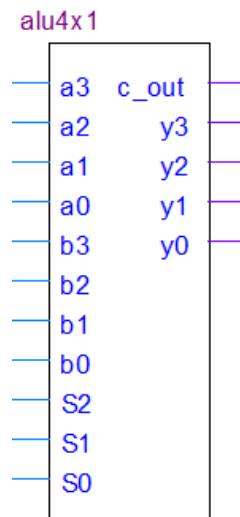


Рисунок 2.83 – Символ 4-бітного АЛП

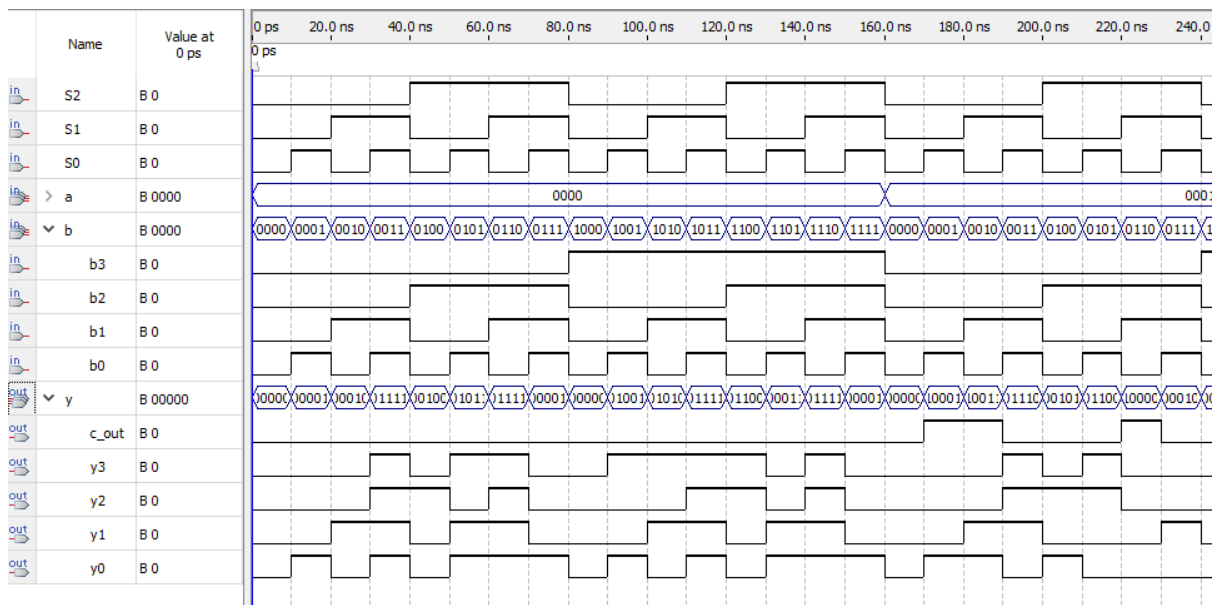


Рисунок 2.84 – Часова діаграма 4-бітного АЛП

На рисунку 2.85 наведено схему тестування 4-бітного АЛП.

2.5.4 Опис 4-бітного АЛП на мові VHDL

Для опису 4-бітного АЛП на мові VHDL створюємо файл alu_vhdl.vhd та на основі шаблону створюємо опис АЛП на мові VHDL (Додаток А.6).



Рисунок 2.85 – Тестування 4-бітного АЛП

Таким чином, у середовищі Quartus II можна створювати такі складні комбінаційні схеми, як АЛП з подальшим моделюванням та тестуванням у віддаленій лабораторії GOLDi.

2.6 Висновки до розділу 2

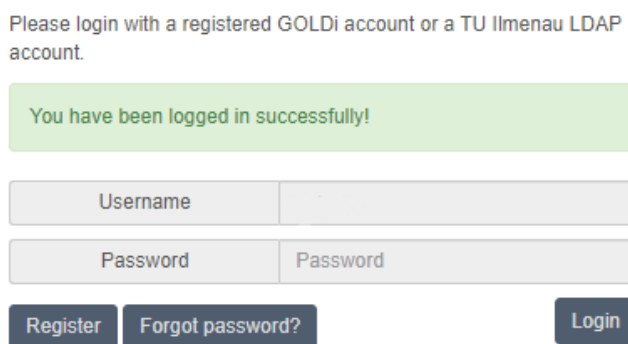
В цьому розділі, у відповідності до мети та задач магістерської роботи представлено структуру розробленої бібліотеки КЦП та алгоритм розробки бібліотечних елементів засобами віртуальної та віддаленої інженерії. Наведено результати розробки описів бібліотечних елементів на мові VHDL та моделювання роботи схем на їх основі.

3 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЦИФРОВИХ ПРИСТРОЇВ ЗАСОБАМИ ВІРТУАЛЬНОЇ ТА ВІДДАЛЕНОЇ ІНЖЕНЕРІЇ

3.1 Початок роботи з віддаленою лабораторією GOLDi

Розглянемо приклад використання віддаленої лабораторії GOLDi для тестування цифрових пристроїв на прикладі розробленого у розділі 2 АЛП.

Перед початком роботи з віддаленою лабораторією GOLDi необхідно увійти до акаунту Технічного університету Ільменау (рис. 3.1).



Please login with a registered GOLDi account or a TU Ilmenau LDAP account.

You have been logged in successfully!

Username

Password Password

Register Forgot password? Login

Рисунок 3.1 – Вхід до лабораторії

Далі на верхній панелі меню необхідно перейти на вкладку Experiment (рис. 3.2).

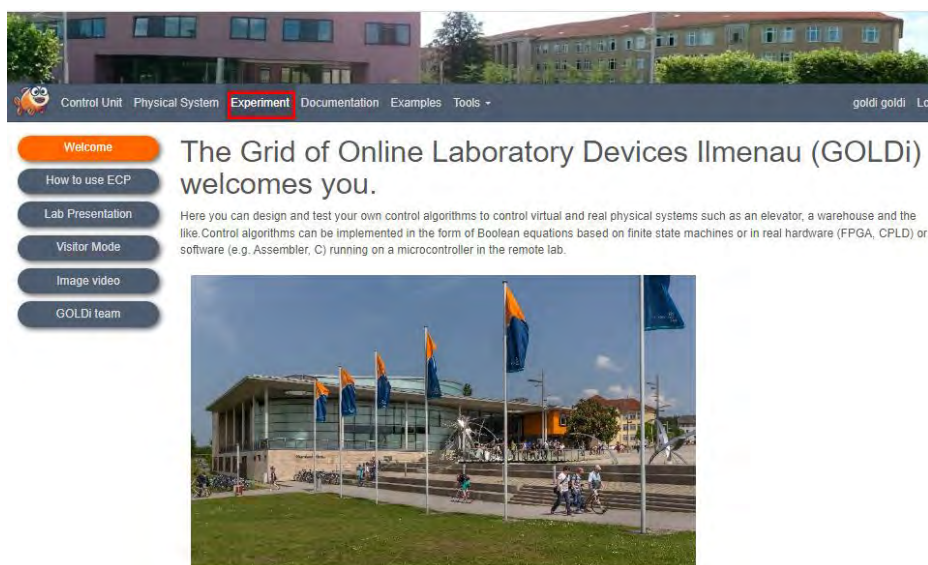


Рисунок 3.2 – Вибір вкладки Experiment

На сторінці з переліком лабораторій обираємо ту, яка в даний момент працює (рис. 3.3).

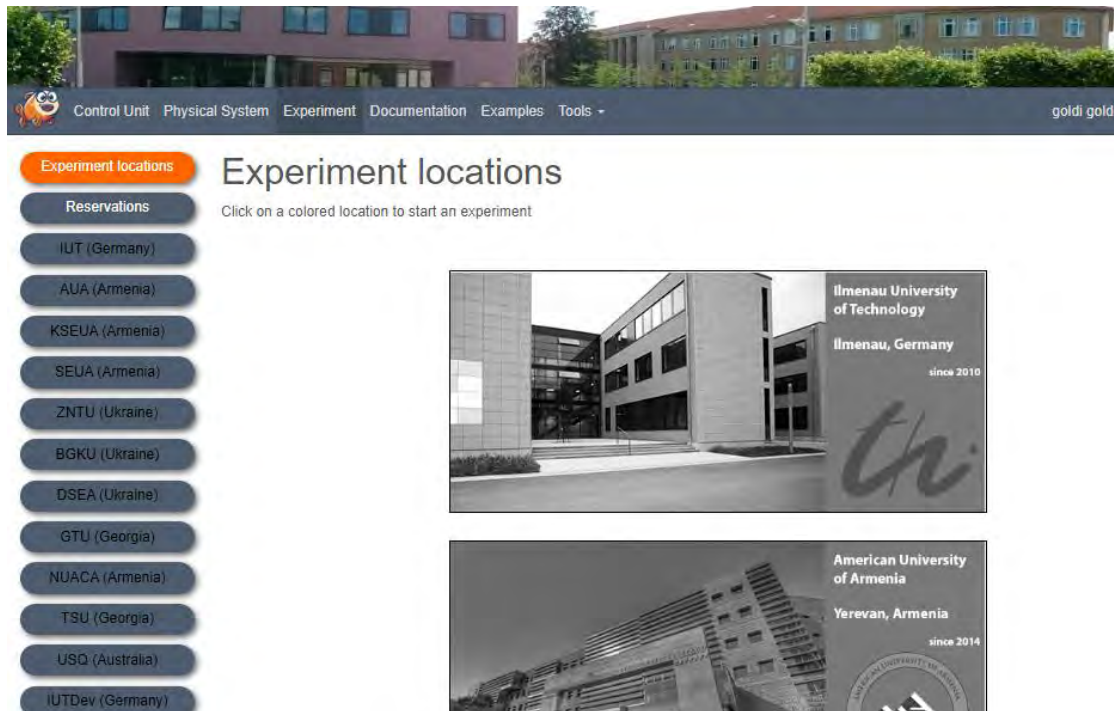


Рисунок 3.3 – Сторінка з лабораторіями GOLDi

На сторінці університету з-поміж модулів керування (Control unit) обираємо Digital Demo Board. Після цього натискаємо Start experiment (рис. 3.4).

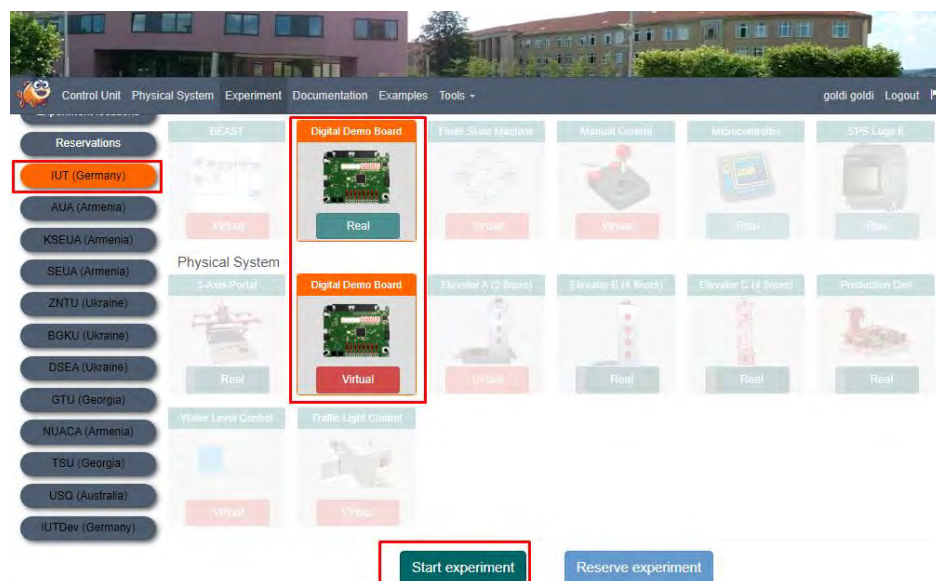


Рисунок 3.4 – Сторінка вибору фізичної системи

На сторінці з ПП присутні віртуальна та реальні плати (рис. 3.5). Для завантаження файлу проєкту для тестування обираємо на верхній панелі меню Upload file та завантажуюмо необхідний файл (рис. 3.6).

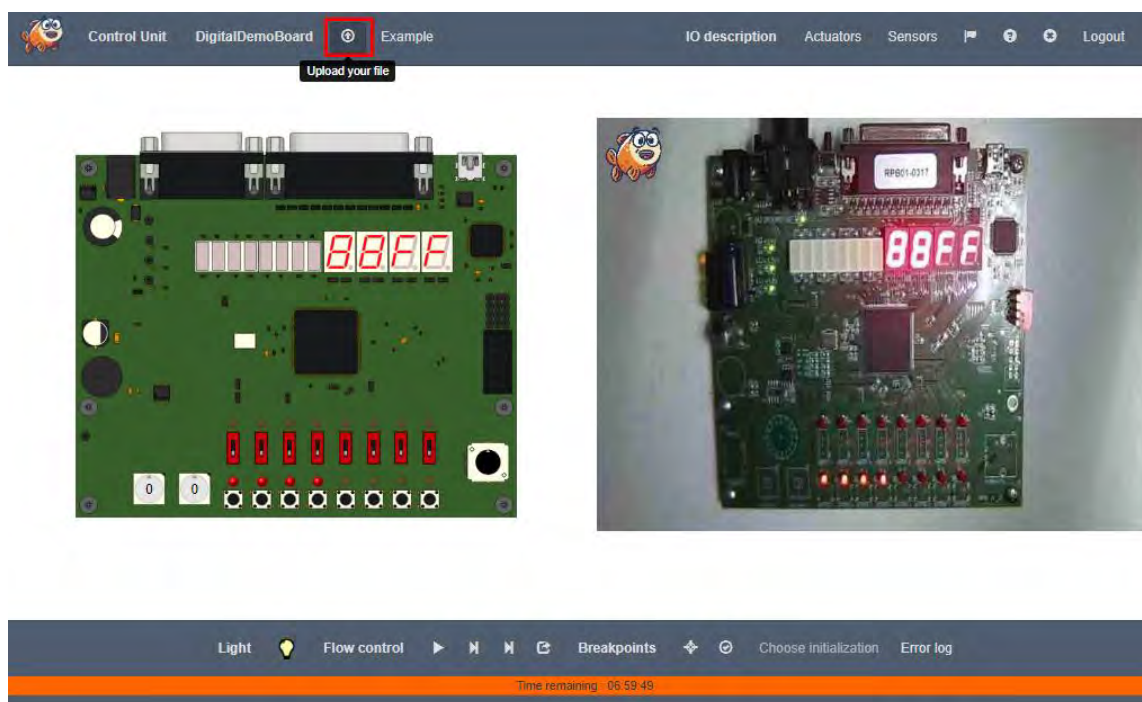


Рисунок 3.5 – Сторінка плати прототипування

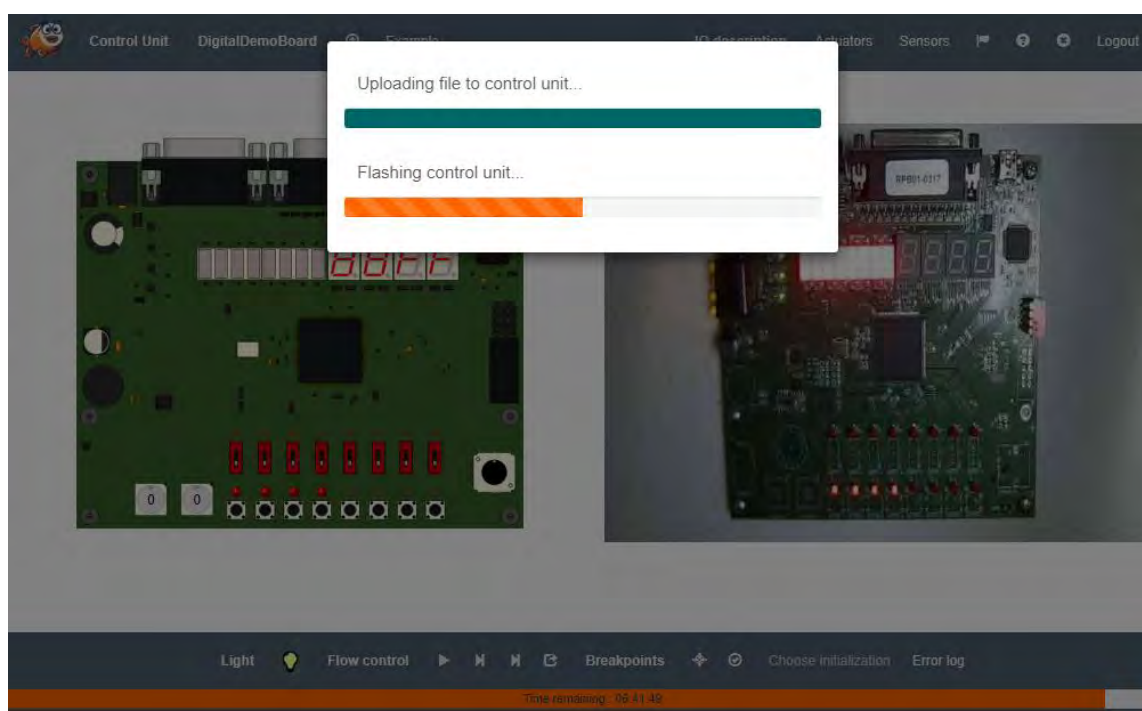


Рисунок 3.6 – Завантаження проєкту

3.2 Тестування розроблених цифрових пристроїв

3.2.1 Тестування дешифратора 7-сегментного індикатору

Для тестування дешифратора 7-сегментного індикатору завантажуюємо схему, яка наведена на рис. 2.28. На рис. 3.7 наведено результати тестування 7-сегментного індикатору у віддаленій лабораторії GODLi. З рисунку видно, що встановлені значення шістнадцяти розрядних енкодерів відповідають значенням, що відображаються на 7-сегментних індикаторах.

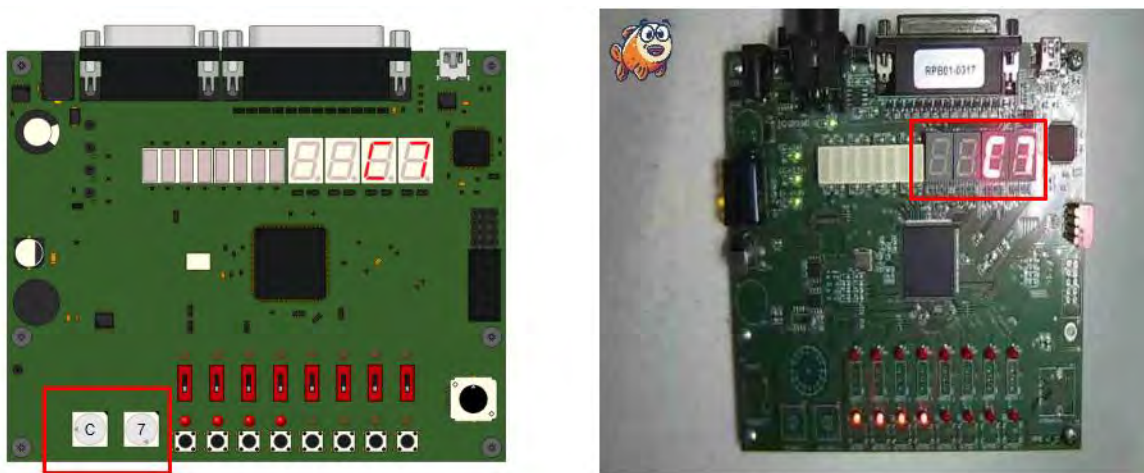


Рисунок 3.7 – Тестування дешифратора 7-сегментного індикатору у віддаленій лабораторії GOLDi

3.2.2 Тестування арифметичних пристроїв

Для тестування 4-бітного суматора завантажуюємо схему, яка наведена на рис. 2.37. На рис. 3.8 наведено результати тестування 4-бітного суматора. У даному прикладі складаються два числа 9 та В. З рисунку видно, що сума дорівнює 4, а біт переносу дорівнює одиниці. Тому сума дорівнює 14_{16} , або 20_{10} .

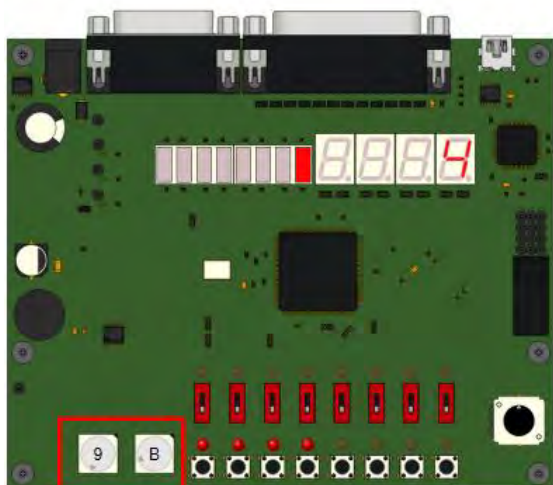


Рисунок 3.8 – Тестування 4-бітного суматора у віддаленій лабораторії GOLDi

Для тестування 2-розрядного BCD-суматора завантажуюємо схему, яка наведена на рис. 2.42. На рис. 3.9 наведено результати тестування 2-розрядного BCD-суматора. Два розряди першого числа представлені відповідно двома шістнадцяти розрядними енкодерами, а два розряди другого – перемикачами, по чотири на кожен розряд. З рисунку видно, що перше число 57_{10} , а друге 73_{10} . Сума чисел, що відображається на 7-сегментних індикаторах дорівнює 30_{10} , а з урахуванням біту переносу, що позначений світлодіодом на панелі світлодіодів, – 130_{10} .

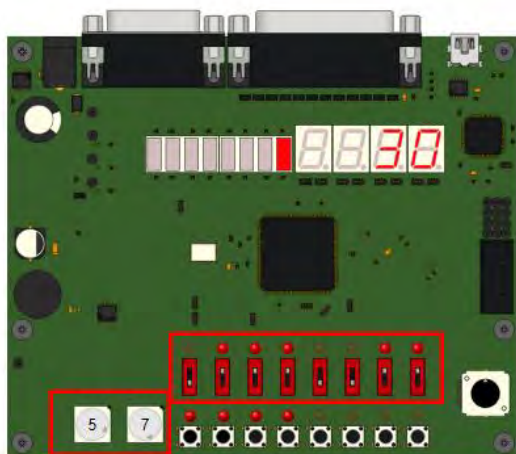


Рисунок 3.9 – Тестування 2-розрядного BCD-суматора у віддаленій лабораторії GOLDi

Для тестування 8-бітного суматору/віднімача завантажуюємо схему, яка наведена на рис. 2.47. На рис. 3.10 наведено результати тестування 8-бітного суматору/віднімача. Перше число представлене двома енкодерами, а друге – перемикачами. З рисунку видно, що перше число дорівнює 68_{16} (104_{10}), а друге – $0D_{16}$ (13_{10}). У даному прикладі тестується операція віднімання. Це видно з того, що натиснута перша кнопка. Різниця двох чисел дорівнює $5B_{16}$ (91_{10}).

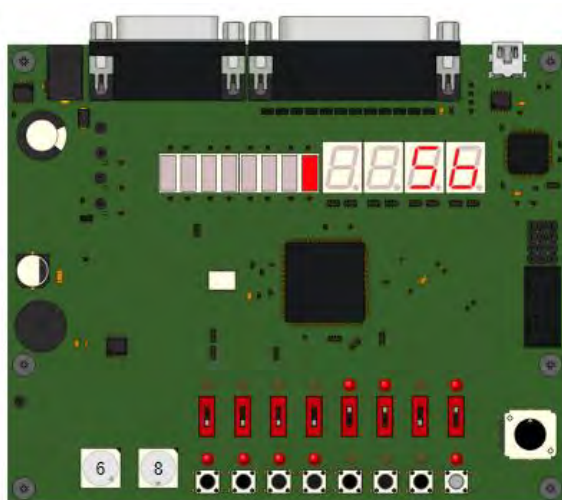


Рисунок 3.10 – Тестування 8-бітного суматору/віднімача у віддаленій лабораторії GOLDi

3.2.3 Тестування компараторів

Для тестування 8-бітного компаратору завантажуюємо схему, яка наведена на рис. 2.57. На рис. 3.11 наведено результати тестування 8-бітного компаратору. У даному прикладі перше і друге числа дорівнюють $9B_{16}$. Два числа рівні, про що свідчить свічення другого світлодіоду.

Для тестування 4-бітного ітеративного компаратору завантажуюємо схему, яка наведена на рис. 2.64. На рис. 3.12 наведено результати тестування 4-бітного ітеративного компаратору. Як і у попередньому прикладі обидва числа, що представлені двома енкодерами, обрані однаковими. Про рівність двох чисел свідчить свічення другого світлодіоду.

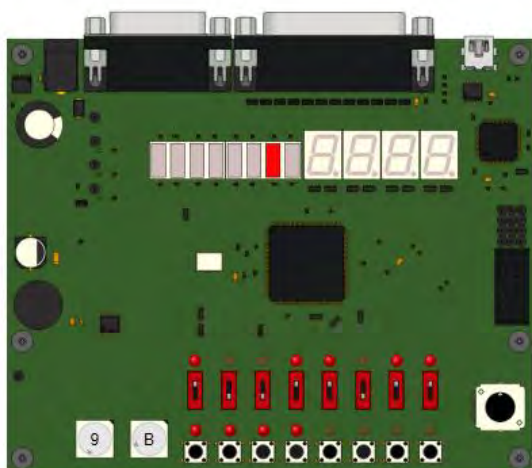


Рисунок 3.11 – Тестування 8-бітного комаратору у віддаленій лабораторії GOLDi

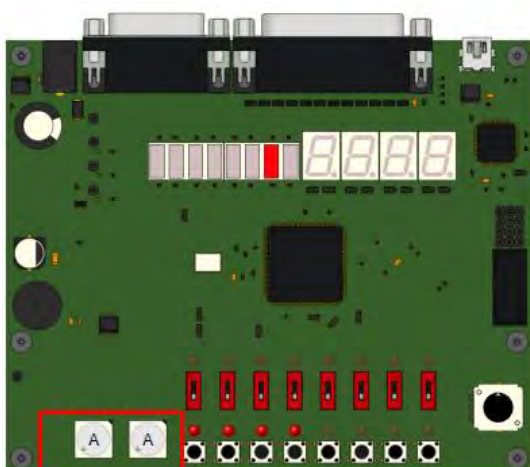


Рисунок 3.12 – Тестування 4-бітного ітеративного комаратору у віддаленій лабораторії GOLDi

3.2.4 Тестування арифметико-логічного пристрою

Для тестування АЛП завантажуюємо схему, яка наведена на рис. 2.84. На рис. 3.13 наведено результати тестування АЛП у віддаленій лабораторії GODLi. У даному прикладі проводиться тестування функції складання (встановлений 3-й перемикач, що відповідає комбінації «100»). Вхідними операндами є два шістнадцяти розрядних енкодери.

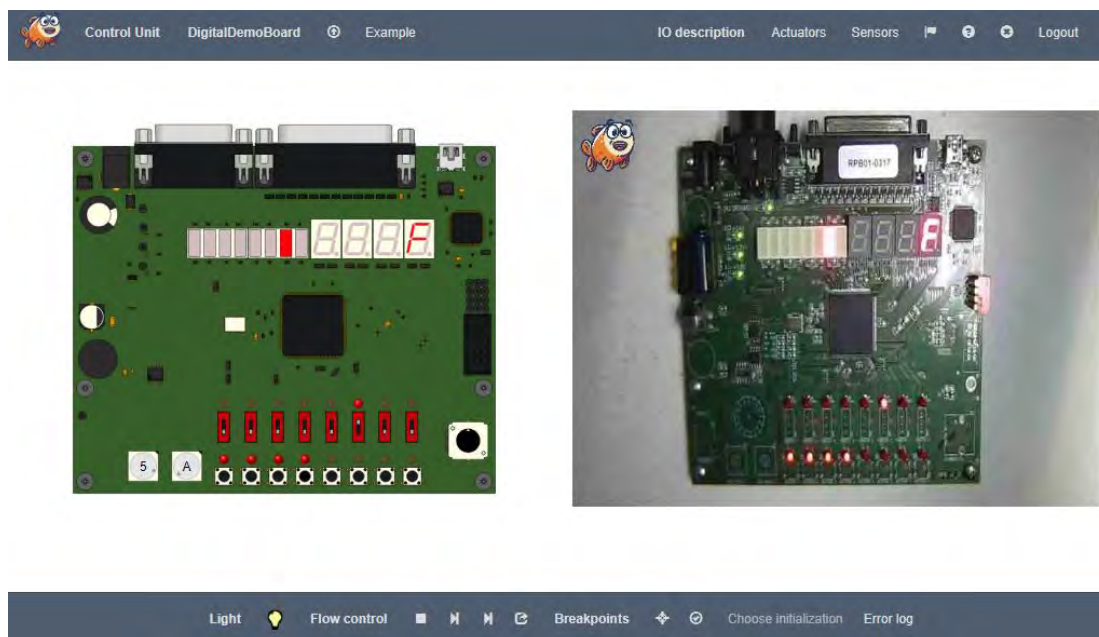


Рисунок 3.13 – Тестування АЛП у віддаленій лабораторії GODLi

3.3 Висновки до розділу 3

В даному розділі представлено результати функціонального тестування бібліотечних елементів КЦП на основі дослідження роботи віртуальної та реальної плат віддаленої лабораторії GODLi.

Отримані результати свідчать, що розроблені у попередньому розділі КЦП працюють коректно у відповідності до заданих вимог, що підтверджується відображенням результату роботи на віртуальній моделі та на реальній платі, за якою можливо спостерігати через вебкамеру онлайн.

ВИСНОВКИ

Як показали проведені дослідження, технології віртуальної та віддаленої інженерії суттєво впливають на сучасні методи та засоби автоматизованого проектування, оскільки надають безліч можливостей онлайн використання не лише програмних, але й апаратних засобів розробки та тестування.

У даній роботі розглянуто можливості віддаленої лабораторії GOLDi щодо прототипування цифрових пристроїв на прикладі різноманітних арифметичних та логічних схем. Досліджено середовище розробки програмного забезпечення Quartus II, що є основним інструментом для програмування плати прототипування на базі CPLD фірми Altera в лабораторії GOLDi.

В результаті виконання роботи створено бібліотеку комбінаційних цифрових пристроїв (суматори, віднімачі, компаратори, арифметико-логічні пристрої), побудовано їх схеми та створено описи на мові VHDL. Виконано тестування програмного забезпечення у віддаленій лабораторії GOLDi, що підтвердило працездатність розроблених КЦП та доцільність запропонованого алгоритму розробки.

Наукова новизна роботи полягає в тому, що запропоновано удосконалений метод автоматизованого проектування комбінаційних цифрових пристроїв, що відрізняється від існуючих спільним використанням технологій віртуальної та віддаленої інженерії для створення та дослідження прототипу цифрового пристрою.

Практична цінність роботи полягає в тому, що розроблене програмне забезпечення для бібліотеки комбінаційних цифрових пристроїв та файли тестування для них можуть використовуватись в практиці автоматизованого проектування цифрових комунікаційних систем як повторно використовувані компоненти, що дозволить знизити терміни розробки та підвищити ефективність процесу проектування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Digital communication – introduction, basic components, how signal process works and advantages [Electronic resource]. – Access mode: <https://electricalfundablog.com/digital-communication-introduction-basic-components-how-signal-process-works-and-advantages/>.
2. Digital device definition [Electronic resource]. – Access mode: <https://www.lawinsider.com/dictionary/digital-device>
3. Digital devices 1 [Electronic resource]. – Access mode: <https://www.knowitallninja.com/lessons/digital-devices-1/>.
4. Information systems. Module 2 - information system hardware. Digital devices [Electronic resource]. – Access mode: <https://www.coursehero.com/study-guides/santaana-informationsystems/digital-devices/>.
5. Проектирование цифрового устройства [Электрон. ресурс]. – Режим доступа: <http://www.generallytech.ru/gentec-721.html>.
6. Цифрові комбінаційні пристрої [Електрон. ресурс]. – Режим доступу: https://studwood.net/2556447/tehnika/tsifrovi_kombinatsiyni_pristroyi
7. Электроника. Комбинационные устройства устройства [Электрон. ресурс]. – Режим доступа: <https://jak.waykun.com/articles/kombinacijni-cifrovi-pristroi.html>
8. Complex programmable logic device [Electronic resource]. – Access mode: https://de.wikipedia.org/wiki/Complex_Programmable_Logic_Device.
9. Field programmable gate array [Electronic resource]. – Access mode: https://de.wikipedia.org/wiki/Field_Programmable_Gate_Array#cite_note-4.
10. CPLD vs FPGA, What is the difference? [Electronic resource]. – Access mode: <https://hardwarebee.com/cpld-vs-fpga-difference/>.
11. FPGA Vs CPLD and microcontrollers [Electronic resource]. – Access mode: <https://www.electronicsforu.com/technology-trends/fpga-vs-cpld-microcontrollers>.

12. Difference between CPLD and FPGA [Electronic resource]. – Access mode: <https://electronics-club.com/difference-between-cpld-and-fpga/>.
13. Understanding FPGA Programming and design flow [Electronic resource]. – Access mode: <https://hardwarebee.com/understanding-fpga-programming-and-design-flow/>.
14. Акчурин, А. Д. Основы работы в среде Quartus II / А. Д. Акчурин, К. М. Юсупов, А. А. Колчев. - Казань: КФУ, 2017. - 49 с.
15. Vivado design suite user guide, using the vivado ide [Electronic resource]. – Access mode: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_3/ug893-vivado-ide.pdf.
16. Стащенко, В.Б. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов / В.Б. Стащенко. – М.: ДОДЭКА, 2000. – 128 с.
17. Thomas, D.E. The Verilog hardware description language / D.E. Thomas, P.R. Moorby. – Springer Science & Business Media, 2008. – 404 с.
18. Verilog vs. VHDL – What to Choose? [Electronic resource]. – Access mode: <https://hardwarebee.com/verilog-vs-vhdl-what-to-choose/>
19. Is it true Verilog is used more in the industry rather than VHDL? [Electronic resource]. – Access mode: <https://www.quora.com/Is-it-true-Verilog-is-used-more-in-the-industry-rather-than-VHDL>.
20. Development and application of remote laboratory for embedded systems design / [A. Parkhomenko, O. Gladkova, E. Ivanov et al.] // International journal of online engineering (iJOE). - 2015. – Vol.11. – P. 27-31.
21. Internet-based technologies for design of embedded systems / [A. Parkhomenko, O. Gladkova, S. Kurson et al.] // Journal of control science and engineering. - 2015. – Vol.3(2). – P. 55–63.
22. Henke, K. Web-based rapid prototyping of digital systems / K. Henke, S. Krug] // Interactive computer-aided blended learning: 4th International conference,

Antigua, Guatemala, 2-4 November 2011: proceedings. – Los Alamitos: IEEE, 2011. – P. 22-27.

23. Using interactive hybrid online labs for rapid prototyping of digital systems / [K. Henke, G. Tabunshchyk, H.-D. Wuttke et al.] // Remote Engineering and Virtual Instrumentation: 11th International conference, Porto, Portugal, 26 - 28 February 2014: proceedings. – Los Alamitos: IEEE, 2014. – P. 61-66.

24. Fields of applications for hybrid online labs / [K. Henke, S. Ostendorff, H.-D. Wuttke et al.] // International journal of online engineering (iJOE). – 2013. – Vol. 9. – P.20-30.

25. Grid concept for reliable, flexible and robust remote engineering laboratories / [K. Henke, S. Ostendorff, H.- D. Wuttke, S. Vogel // International journal of online engineering (iJOE). – 2012. – Vol. 8, Issue 3 – P.42-49.

26. The remote lab cloud "GOLDi-labs.net" / [K. Henke, T. Vietzke, R. Hutschenreuter, H. -D. Wuttke] // Remote Engineering and Virtual Instrumentation: 13th International conference, Madrid, Spain, 24 - 26 February 2016: proceedings. – Los Alamitos: IEEE, 2016. –P. 37-42.

27. Reusable solutions for embedded systems' design / [A. Parkhomenko, O. Gladkova, A. Sokolyanskii et al.] // Remote engineering and virtual instrumentation: 13th International conference, Madrid, Spain, 24 - 26 February 2016: proceedings. – Los Alamitos: IEEE, 2016. –P. 313-317.

28. Уэйкерли, Дж. Ф. Проектирование цифровых устройств / Дж. Ф. Уэйкерли. - М.: ИНФРА-М, 2012. – 768 с.

29. Угрюмов, Е.П. Цифровая схемотехника / Е.П. Угрюмов. – СПб.: БХВ – Санкт-Петербург, 2004. – 800 с.

ДОДАТОК А
Текст програми

A.1 Шаблон опису об'єкту на мові VHDL

```
--оголошення стандартних бібліотек
library ieee;
use ieee.std_logic_1164.all;

--оголошення об'єкту
entity simple_design is
    --порти об'єкту (входи та виходи)
    port
    (
        a, b, c : in std_logic;
        x, y: out std_logic
    );
end simple_design;

--архітектура об'єкту
architecture behave of simple_design is
begin
    -- присвоєння виходам значень
    x <= (a and b) or not c;
    y <= (a xor b) and c;
end behave;
```

A.2 Опис дешифратора 7-сегментного індикатору з використанням найпростіших логічних елементів

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder_vhdl is
    port
    (
        x:  in std_logic_vector(3 downto 0);
        seg:out std_logic_vector(6 downto 0)
    );
end decoder_vhdl;

architecture behave of decoder_vhdl is
begin
```

```
-- segment "a"
seg(6) <=      (not x(3) and not x(2) and not x(0)) or
               (not x(3) and x(1)) or
               (x(2) and x(1)) or
               (not x(3) and x(2) and x(0)) or
               (x(3) and not x(0)) or
               (x(3) and not x(2) and not x(1));

-- інші сегменти також описуються
-- згідно з їх логічними рівняннями
end behave;
```

A.3 Опис дешифратора 7-сегментного індикатору з використанням умовного привласнення

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder_vhdl is
    port
    (
        x:  in std_logic_vector(3 downto 0);
        seg:out std_logic_vector(6 downto 0)
    );
end decoder_vhdl;

architecture behave of decoder_vhdl is
begin
    seg <=  "1111110" when x=X"0" else
           "0110000" when x=X"1" else
           "1101101" when x=X"2" else
           "1111001" when x=X"3" else
           "0110011" when x=X"4" else
           "1011011" when x=X"5" else
           "1011111" when x=X"6" else
           "1110000" when x=X"7" else
           "1111111" when x=X"8" else
           "1111011" when x=X"9" else
           "1110111" when x=X"a" else
           "0011111" when x=X"b" else
           "1001110" when x=X"c" else
           "0111101" when x=X"d" else
```

```

        "1001111" when x=X"e" else
        "1000111";
end behave;

```

A.4 Опис дешифратора 7-сегментного індикатора з використанням оператора «select»

```

library ieee;
use ieee.std_logic_1164.all;

entity decoder_vhdl is
    port
    (
        x: in std_logic_vector(3 downto 0);
        seg: out std_logic_vector(6 downto 0)
    );
end decoder_vhdl;

architecture behave of decoder_vhdl is
begin
    with x select
    seg <=  "1111110" when X"0",
            "0110000" when X"1",
            "1101101" when X"2",
            "1111001" when X"3",
            "0110011" when X"4",
            "1011011" when X"5",
            "1011111" when X"6",
            "1110000" when X"7",
            "1111111" when X"8",
            "1111011" when X"9",
            "1110111" when X"a",
            "0011111" when X"b",
            "1001110" when X"c",
            "0111101" when X"d",
            "1001111" when X"e",
            "1000111" when others;
end behave;

```

A.5 Опис дешифратора 7-сегментного індикатору з використанням оператора «process»

```

library ieee;
use ieee.std_logic_1164.all;

entity decoder_vhdl is
    port
    (
        x: in std_logic_vector(3 downto 0);
        seg: out std_logic_vector(6 downto 0)
    );
end decoder_vhdl;

architecture behave of decoder_vhdl is
begin
    process(x) is
    begin
        case x is
            when X"0" => seg <= "1111110";
            when X"1" => seg <= "0110000";
            when X"2" => seg <= "1101101";
            when X"3" => seg <= "1111001";
            when X"4" => seg <= "0110011";
            when X"5" => seg <= "1011011";
            when X"6" => seg <= "1011111";
            when X"7" => seg <= "1110000";
            when X"8" => seg <= "1111111";
            when X"9" => seg <= "1111011";
            when X"a" => seg <= "1110111";
            when X"b" => seg <= "0011111";
            when X"c" => seg <= "1001110";
            when X"d" => seg <= "0111101";
            when X"e" => seg <= "1001111";
            when others => seg <= "1000111";
        end case;
    end process;
end behave;

```

A.6 Опис арифметико-логічного пристрою

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu_vhdl is
    port
    (
        a, b      : in std_logic_vector(3 downto 0);
        s          : in std_logic_vector(2 downto 0);
        y          : out std_logic_vector(3 downto 0);
        CFlag, OFlag, SFlag, ZFlag : out std_logic
    );
end alu_vhdl;

architecture behave of alu_vhdl is
    signal op_a, op_b: unsigned(4 downto 0);
    signal y_res: unsigned(4 downto 0);
begin

    op_a <= unsigned('0' & a);
    op_b <= unsigned('0' & b);

    with s(2 downto 0) select y_res <=
        op_a and op_b when "000",
        op_a or op_b when "001",
        op_a xor op_b when "010",
        not op_a when "011",
        op_a + op_b when "100",
        op_a - op_b when "101",
        op_a - 1 when "110",
        op_a + 1 when others;

    CFlag <= y_res(4);
    OFlag <= (op_a(3) and op_b(3) and not y_res(3)) or
              (not op_a(3) and not op_b(3) and
y_res(3));
    SFlag <= y_res(3);
    ZFlag <= '1' when y_res(3 downto 0)="0000" else '0';

    y <= std_logic_vector(y_res(3 downto 0));
end behave;

```

ДОДАТОК Б
Слайди презентації

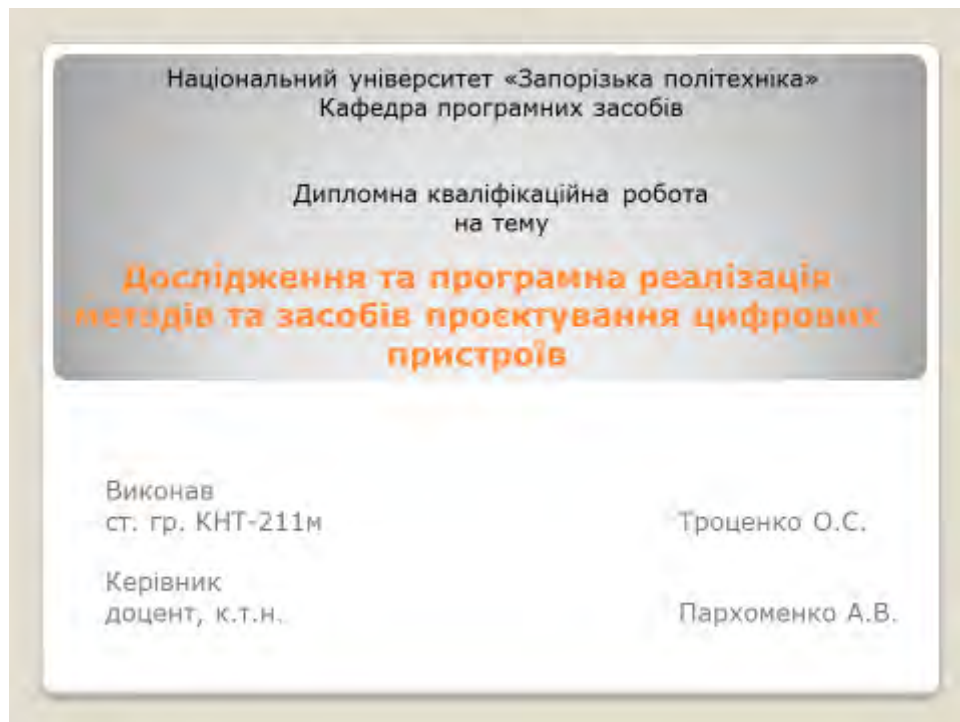


Рисунок Б.1 – Слайд презентації №1



Рисунок Б.2 – Слайд презентації №2

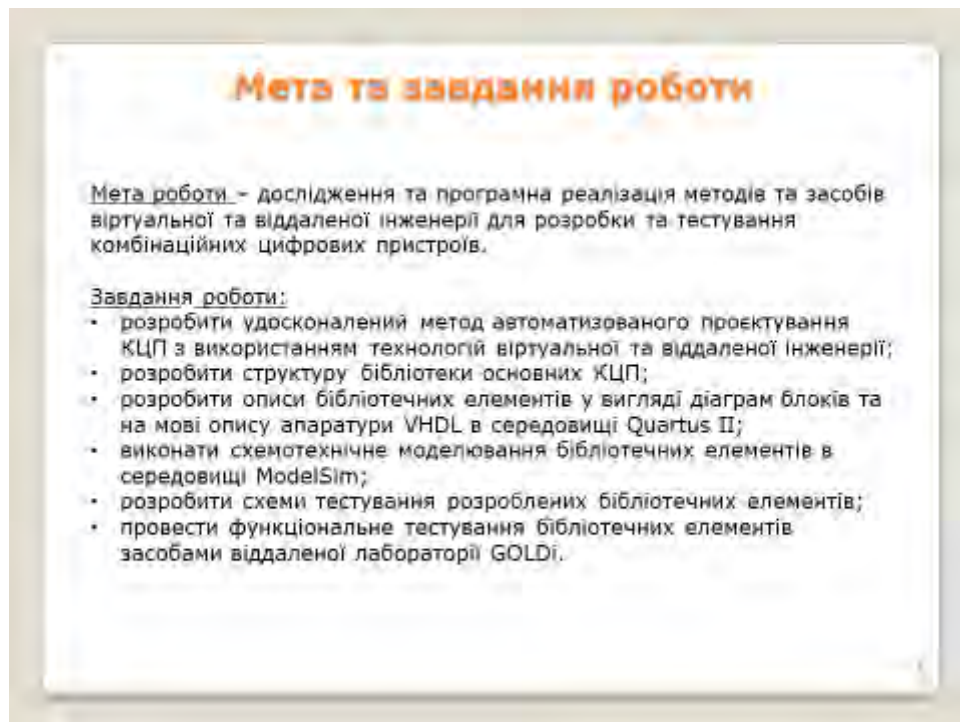


Рисунок Б.3 – Слайд презентації №3



Рисунок Б.4 – Слайд презентації №4

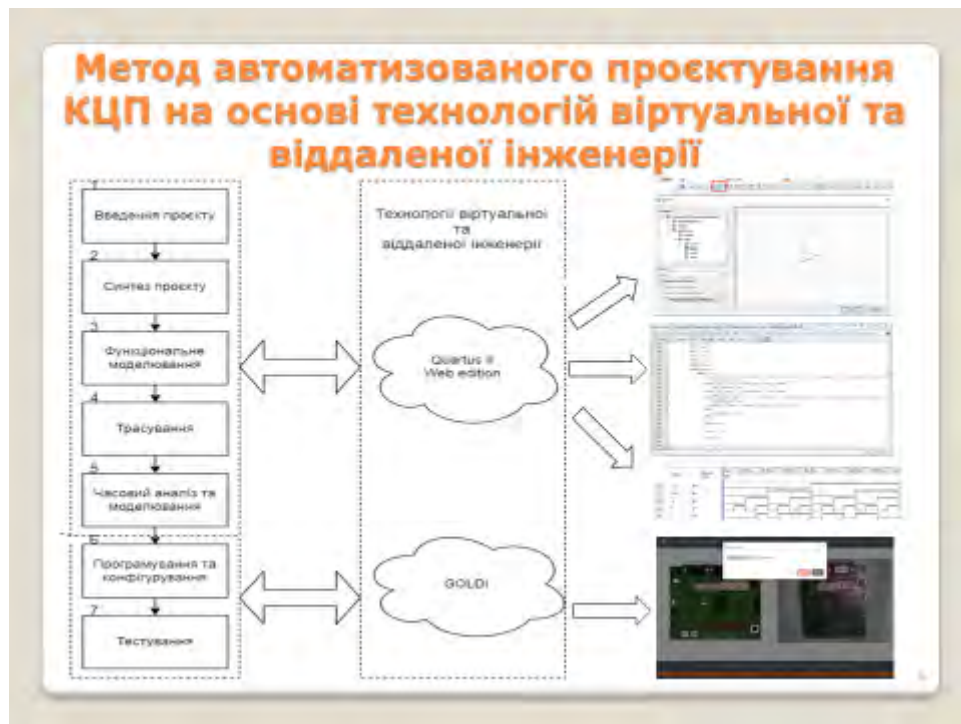


Рисунок Б.5 – Слайд презентації №5



Рисунок Б.6 – Слайд презентації №6

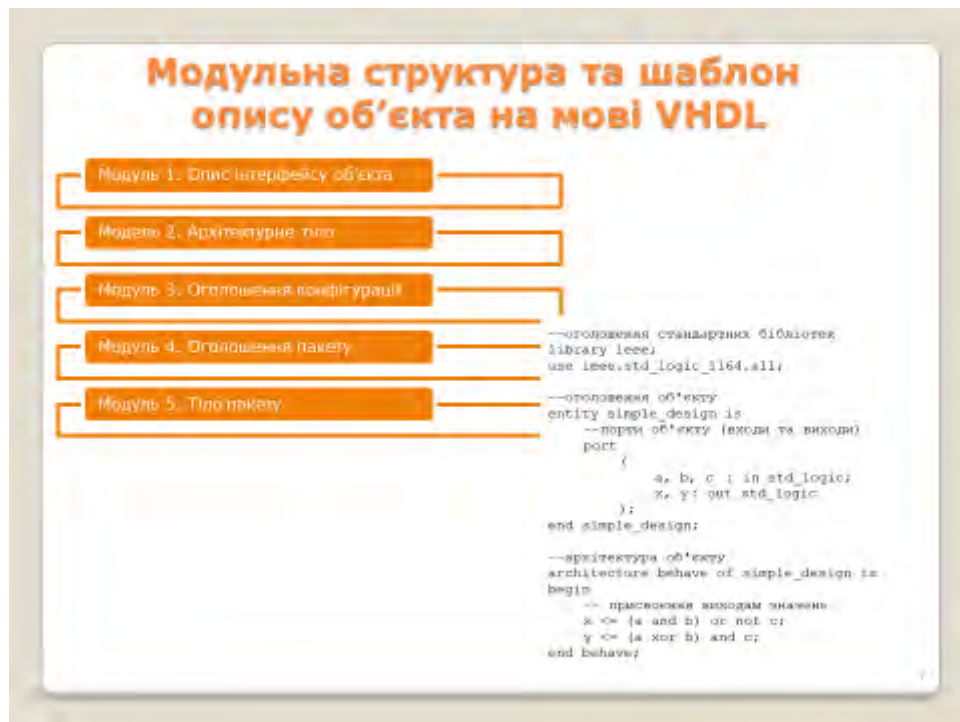


Рисунок Б.7 – Слайд презентації №7



Рисунок Б.8 – Слайд презентації №8

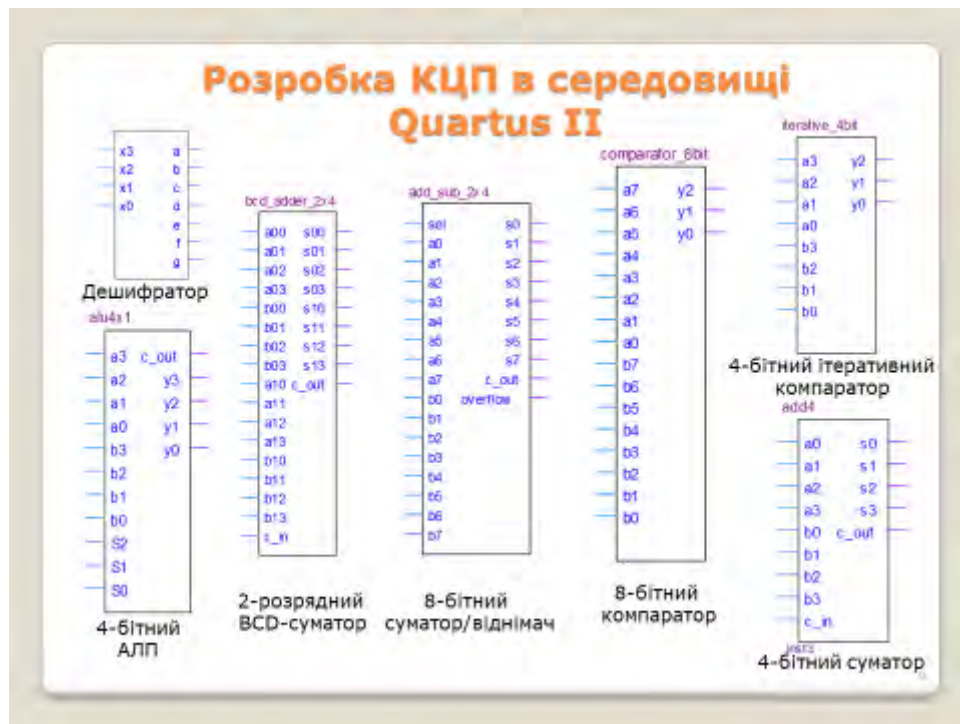


Рисунок Б.9 – Слайд презентації №9



Рисунок Б.10 – Слайд презентації №10

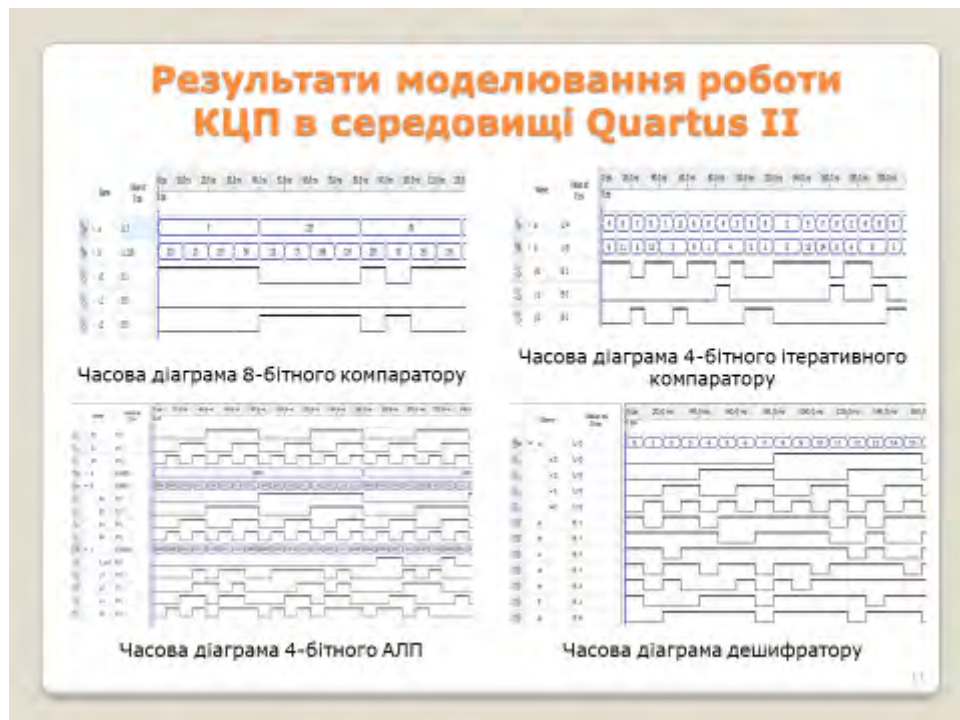


Рисунок Б.11 – Слайд презентації №11



Рисунок Б.12 – Слайд презентації №12



Рисунок Б.13 – Слайд презентації №13



Рисунок Б.14 – Слайд презентації №14

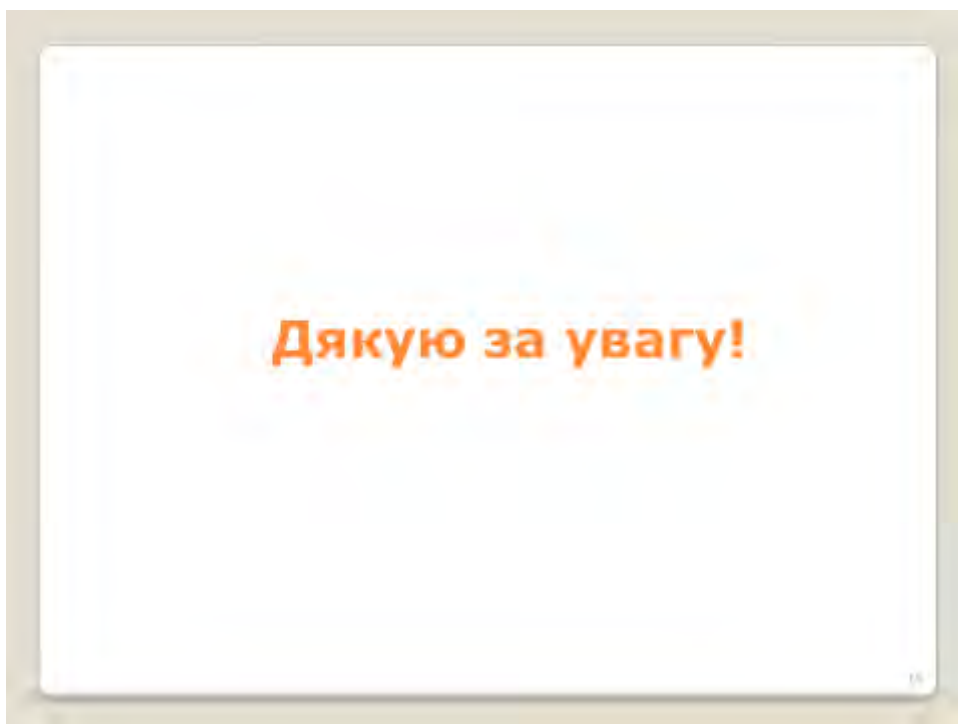


Рисунок Б.15 – Слайд презентації №15