МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

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

на тему **СИСТЕМА РЕВЕРС-ІНЖИНІРИНГУ VERILOG HDL ПРОЄКТІВ ДЛЯ МІКРОСХЕМ INTEL FPGA**

Виконав: студент 2 курсу, групи КНТ-614м
спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Спеціалізовані комп'ютерні системи

БУТКО В.О.
(ПРІЗВИЩЕ та ініціали)
Керівник ЗЕЛЕНЬОВА І.Я.
(ПРІЗВИЩЕ та ініціали)
Рецензент РОМАНОВСЬКИЙ О.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій

Кафедра «Комп'ютерні системи та мережі»

Ступінь вищої освіти магістерський

Спеціальність 123 Комп'ютерна інженерія
(код і найменування)

Освітня програма (спеціалізація) Спеціалізовані комп'ютерні системи
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Зав. кафедри Кудерметов Р.К.
_____
"_15_" __жовтня___ 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

БУТКО Владислава Олексійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Система реверс-інжинірингу Verilog HDL проєктів для мікросхем Intel FPGA

керівник проєкту (роботи) кан. техн. наук, доцент, ЗЕЛЕНЬОВА Ірина Яківна
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від "_30_" _жовтня_ 2025 року №491

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

3. Вихідні дані до проєкту (роботи) математична формула, алгоритм виконання формули, HDL опис, який реалізує даний алгоритм, та додаткові вимоги до його функціонування, результати тестування HDL опису в програмному середовищі Vivado, документація до HDL опису, друкована плата, мультиметер, Quartus, Vivado, Simulink, Proteus

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

1) Огляд області реінжинірингу FPGA
2) Розробка моделі
3) Реінжиніринг друкованої плати

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	ЗЕЛЕНЬОВА І. Я., доцент		
нормоконтроль	ПОЛЬСЬКА В.О., ст. викл.		

7. Дата видачі завдання 01.10.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз попередніх наукових робіт	01.10.2025 р.	
2	Огляд інструментів реінжинірингу	04.10.2025 р.	
3	Розробка методу реінжинірингу FPGA проєкта	11.10.2025 р.	
4	Створення моделі FPGA проєкта	21.10.2025 р.	
5	Тестування моделі	08.11.2025 р.	
6	Реінжиніринг друкованої плати	13.11.2025 р.	
7	Оформлення ПЗ	10.12.2025 р.	
8	Оформлення графічного та додаткових матеріалів	16.12.2025 р.	

Студент _____ **Владислав БУТКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

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

РЕФЕРАТ

ПЗ: 91 с., 39 рис., 14 джерел, 2 додатки

РЕВЕРС-ІНЖИНІРИНГ, INTEL, ALTERA, FPGA, VERILOG HDL, МОДЕЛЮВАННЯ, SIMULINK, ДРУКОВАНІ ПЛАТИ

Об’єкт дослідження – метод реверс-інжинірингу (далі – реінжиніринг) Intel FPGA проєкта.

Предмет дослідження – метод створення Simulink моделі на основі Verilog HDL опису, призначеного для програмування Intel FPGA.

Мета роботи – пришвидшити процес аналізу функції вже готових Intel FPGA проєктів на основі Verilog HDL опису.

Методи дослідження – теоретичний аналіз Verilog HDL опису, верифікація результатів аналізу шляхом створення Simulink моделі, візуальний огляд друкованої плати, тестування доріжок на обрив, верифікація результатів аналізу друкованої плати шляхом створення моделі.

Робота містить наукову новизну, яка полягає в розробці методу ручного створення наочної для людського сприйняття Simulink моделі на основі Verilog HDL опису Intel FPGA проєкта.

Результати: визначено недоліки інструментів автогенерації моделі. Розроблено метод ручного створення Simulink моделі FPGA проєкта, яка в повній мірі відтворює функціональність FPGA проєкта, але має спрощену структуру, що робить модель легкою для сприйняття людиною. Сформовано рекомендації з реінжинірингу друкованої плати.

Галузь використання – автоматизація розробки FPGA проєктів.

ABSTRACT

Explanatory note to the master's work: 91 pages, 39 figures, 14 sources, 2 applications.

REVERSE ENGINEERING, INTEL, ALTERA, FPGA, VERILOG HDL, MODELING, SIMULINK, PRINTED CIRCUIT BOARD

The object of research is the method of the Intel FPGA project reverse engineering.

The subject of research is the method of creating a Simulink model based on a Verilog HDL description intended for programming Intel FPGA.

The purpose of the work is to speed up the process of analyzing Intel FPGA projects based on a Verilog HDL description.

Research methods: theoretical analysis of the Verilog HDL description, verification of the analysis results by creating a Simulink model, visual inspection of a printed circuit board (PCB), testing the tracks for breaks using a multimeter to create a netlist, verification of the PCB analysis results by creating a model in the Proteus.

The work contains a scientific novelty, which consists in developing the method for manually creating a Simulink model that is clear for human perception based on the Verilog HDL description of an Intel FPGA project.

Results: the shortcomings of model autogeneration tools have been identified. The method for manually creating a Simulink model of an FPGA project has been developed, which fully reproduces the functionality of the FPGA project, but has a simplified structure, which makes the model easy for human perception. Recommendations for reengineering of a PCB have been formed.

Field of application: automation of FPGA project development.

ЗМІСТ

Вступ……7
1 Огляд області ренжинірингу FPGA……9
1.1 Аналіз попередніх робіт……9
1.2 Сфери застосування реінжинірингу ……9
1.3 Інструменти реінжинірингу……10
2 Розробка моделі……12
2.1 Постановка задачі……12
2.2 Розробка методу реінжинірингу……14
2.3 Реалізація Simulink моделі……18
2.4 Порівняння Simulink моделі та HDL опису……42
3 Реінжиніринг друкованої плати……44
3.1 Огляд друкованої плати……44
3.2 Створення списку з'єднань……46
Висновки……53
Перелік джерел посилання……55
Додаток А. HDL опис для створення моделі……57
Додаток Б. Модель та тести……69
Додаток В. Слайди презентації……87

ВСТУП

Наразі, на ринку обчислювальної техніки спостерігається зростання в потребі адаптивних рішень, які можна швидко адаптувати до швидкоплинних умов використання. Особливо, це спостерігається у військовій галузі, де відбувається активна апробація нових способів ведення воєнних дій та швидко впроваджуються інновації [1]. В той же час, рішення повинно задовольняти потреби в енергоефективності, компактності та швидкодії. Всі ці переваги поєднуються в класі обчислювальних пристроїв FPGA (англ. Field-Programmable Gate Array або програмовані логічні матриці). На базі FPGA можна реалізувати власну інтегральну схему, використовуючи лише настільний комп'ютер та програматор. Таким чином, FPGA є компромісом між замовними схемами (мають відносно велику швидкодію, енергоефективність, компактність) та мікроконтролерами (швидкість зміни логіки поведінки пристрою).

Логіка поведінки FPGA задається за допомогою мов опису апаратури (англ. HDL або Hardware Description Language). Ці мови дозволяють задати цифрову схему, яка буде реалізуватися на базі FPGA, в текстовому представлені за допомогою високорівневих мовних конструкцій (наприклад, «for», «if») та нагадують інтерфейси (англ. API або Application Programming Interface) для паралельного програмування на мові C++ (наприклад, OpenMP, MPI). Проте, якщо транслятори OpenMP та MPI формують, на основі високорівневих конструкцій, програму на нижчому рівні абстракції, то транслятори HDL мов формують список з'єднань (англ. netlist), що відображає з'єднання між електричними компонентами технологічної бібліотеки (в даному випадку, технологічної бібліотеки FPGA), який далі використовується в процесі розташування та маршрутизації (англ. P&R або Place and Route) на базі FPGA. По суті, FPGA це готова інтегральна схема, яка дозволяє, використовуючи певні стандартні комірки (ресурси FPGA) та маршрути між ними, реалізувати власну логіку, інформація про яку зберігається в зовнішній, по відношенню до FPGA, пам'яті або внутрішній пам'яті FPGA.

Проте, може виникнути ситуація, коли існує готовий HDL опис, але разом з цим виконується одна або більше умов з переліку:

– персонал, відповідальний за супроводження вихідного HDL коду, відсутній;

– не має супровідної документації до вихідного HDL коду;

– вихідний HDL код є складним для сприйняття для не фахового персоналу.

У всіх цих випадках є необхідність в реінжинірингу вихідного HDL коду FPGA проєкта. Результатом реінжинірингу є модель HDL опису. Модель може мати різний вигляд: опис на природній мові, опис на мові програмування (наприклад, MATLAB або C++), система реального часу (наприклад, Simulink-модель), математична модель.

1 ОГЛЯД ОБЛАСТІ РЕІНЖИНІРИНГУ FPGA

1.1 Аналіз попередніх робіт

Часто під поняттям реінжинірингу FPGA проєкта мається на увазі процес отримання високорівневого HDL опису на основі потоку бітів (англ. bitstream) [2], де потік бітів – це конфігураційна інформація про логіку, що закладається в FPGA. Проте, в даній роботі поняття реінжинірингу має інше значення та означає процес створення Simulink моделі на основі Verilog HDL опису, призначеного для Intel FPGA проєкта. Огляд попередніх робіт показав, що наразі відсутні роботи присвячені процесу реінжинірингу в даному сенсі цього слова. Припускається, що причиною відсутності робіт є наявність автоматизованого способу генерації такої моделі.

1.2 Сфери застосування реінжинірингу

Може виникнути ситуація, коли існує готовий HDL опис, але разом з цим виконується одна або більше умов з переліку:

– персонал, відповідальний за супроводження вихідного HDL коду, відсутній, наприклад, по причині загибелі, арешту, мобілізацій, самовільного звільнення через відсутність бронювання на підприємстві або перевантажений графік, що є особливо актуальним під час воєнного стану в країні;

– не має супровідної документації до вихідного HDL коду, що пояснює його поведінку;

– вихідний HDL код є складним для сприйняття для не фахового персоналу, що, наприклад, сповільнює або не уможливлює проведення вдосконалень в апаратній частині приладу.

У всіх цих випадках є необхідність в реінжинірингу вихідного HDL коду. Результатом реінжинірингу є модель HDL опису. Модель може мати різний вигляд: опис на природній мові, опис на мові програмування (наприклад, MATLAB або C++), система реального часу (наприклад, Simulink-модель), математична модель.

В процесі створення моделі можуть використовуватися різні типи моделей на етапах її створення. Наприклад, спочатку виконується опис поведінки HDL коду на природній мові та/або з використанням математичних лексем (математичної моделі). Тобто, висувається теоретичне припущення про закладену функцію та алгоритм в FPGA проєкті. Далі, опис формалізується за допомогою, наприклад, високорівневої мови програмування MATLAB, що необхідно для перевірки теоретичних припущень щодо роботи закладеного функціоналу. Потім створюється Simulink-модель на основі MATLAB програм, що необхідно для наочної візуалізації структури досліджуваної системи (в даному випадку, FPGA проєкта). Отже, в процесі створення моделі можуть бути використані всі типи моделей на проміжних етапах.

1.3 Інструменти реінжинірингу

MATLAB-функція «importhdl» [3] виконує генерацію Simulink моделі на основі HDL опису на мовах Verilog та VHDL. Проте, даний спосіб отримання моделі має обмежений перелік призначень. Наприклад, це може бути корисним, якщо створюється Simulink модель друкованої плати з FPGA. Тоді, FPGA проєкт є лише частиною цієї моделі. Тому, для запуску моделі друкованої плати, достатньо моделі FPGA проєкта у вигляді чорного ящику, коли структура моделі є не доступною для людини через складність її представлення. Отже, автозгенеровані моделі мають стандартизовану структуру, яку складно сприймати людині. Тому, якщо модель призначена для наочного представлення структури FPGA проєкта, то

інструменти автогенерації Simulink моделі не підходять. В такому випадку, лише ручний процес створення моделі є доцільним.

Проте, існують також інші недоліки використання інструментів автогенерації. Для генерації моделі вхідний HDL опису повинен відповідати основним вимогам [4]:

а) HDL модуль не повинен мати більше одного синхросигналу;

б) не повинні використовуватися наступні шаблони HDL опису [5]:

1) сигнали тактування, скидання та сигнали дозволу не можуть використовуватися в логічних, арифметичних та інших операціях;

2) в умовних конструкціях «if» та «case» повинні бути присутні блоки «else» та «default», відповідно;

3) не може використовуватися передній та задній фронти для одного і того ж сигналу в одному списку чутливості;

4) не можуть бути наявні блоки, в середині одного HDL модуля, одні з яких мають хоча б один асинхронний сигнал, а інші не мають жодного асихронного сигналу;

5) в одному HDL модулі не може бути використано більше одного шаблону, призначеного для синтезу блоку RAM (англ. Random Access Memory) пам’яті;

6) в блоці «initial» не підтримується використання умовних блоків «if» та «case», а також не підтримується присвоєння не константних значень;

7) необхідно індексуватися до всіх вимірів багатовимірного сигналу в момент присвоєння йому значення;

в) не повинні використовуватися стандартні модулі, що реалізують елементарні логічні функції, наприклад, «АБО», «НІ» та інші.

Наявність не підтримуваних шаблонів в HDL описі не унеможливлюють генерацію моделі, а лише призводить до необхідності адаптації HDL опису. Для цього необхідно використати інший стиль опису. Натомість, логіка від цього не зміниться. Задля уникнення додаткових часових витрат при створені моделі можна

уникнути етапу адаптації, якщо враховувати вимоги інструменту автогенерації ще під час створення HDL опису.

Зроблено спробу застосування інструменту importhdl для автогенерації моделі. Натомість, HDL опис, який використовується в якості прикладу, не відповідає вимогам інструмента автогенерації. HDL модуль містить дільник частоти синхросигналу, реалізований на базі користувацької логіки FPGA. Сигнал на виході цього дільника інтерпретується інструментом як ще один синхросигнал. Тому, як, згідно з вимогами, в одному HDL модулі не може використовуватися більше одного синхросигналу, то модель згенерована не може бути згенерована. Проте, головною причиною не можливості використання інструменту автогенерації є відсутність можливості створення наочної моделі HDL опису за його допомогою.

2 РОЗРОБКА МОДЕЛІ

2.1 Постановка задачі

В попередніх роботах реалізовано Verilog HDL опис (лістинг в додатку А.1), призначений для програмування Intel FPGA. В HDL описі реалізовано автомат, що виконує формулу (1).

$$S = \begin{cases} (9(A \oplus B) + (4A + \frac{B}{8})) \oplus \overline{(5B + A)}, & A \leq 9 \\ \frac{A+B}{8} + \overline{(4A \wedge 2B)} \vee (A \oplus B) \end{cases} \quad (1)$$

Формула (1) виконується згідно з алгоритмом як на рис. 2.1.

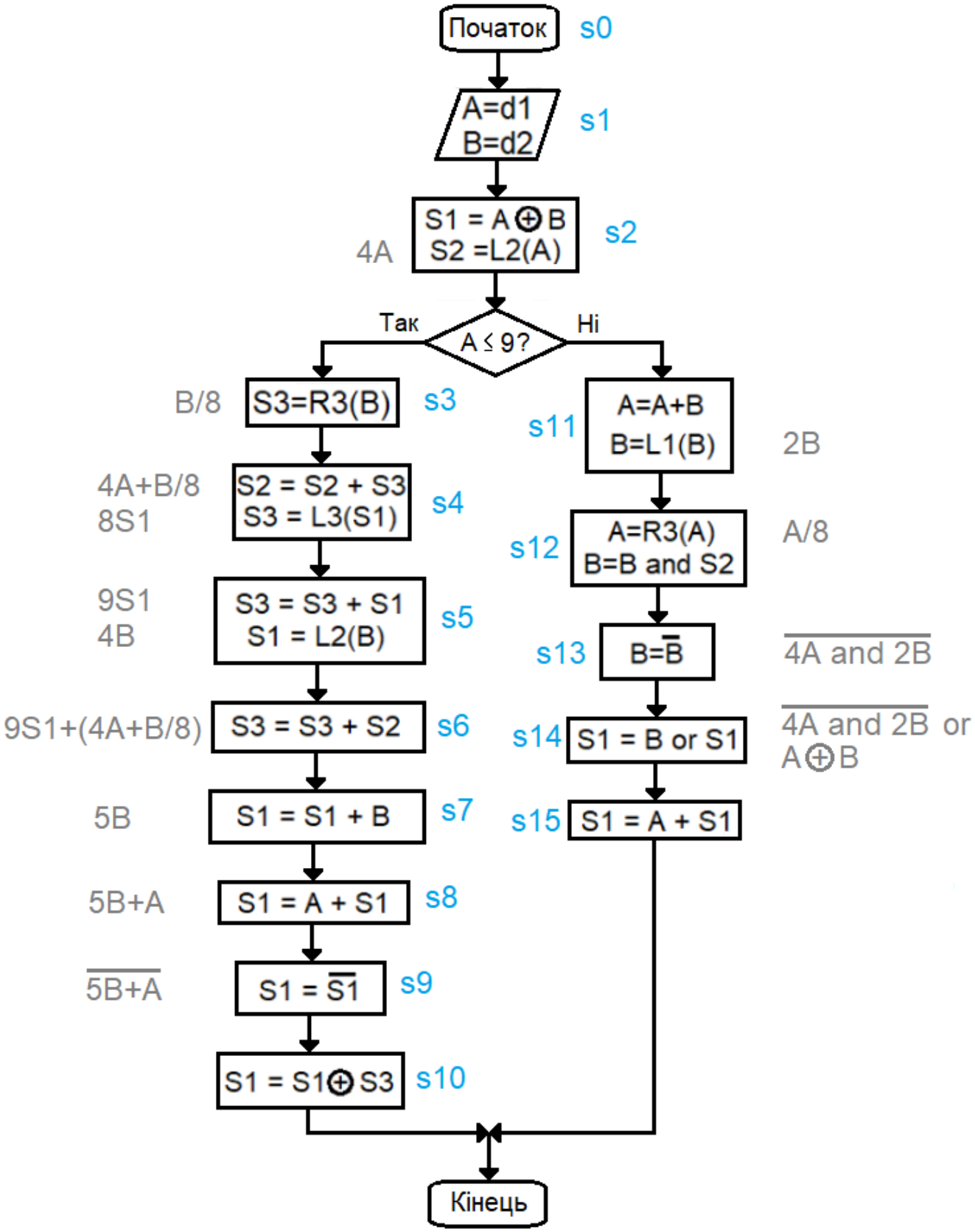


Рисунок 2.1 – Алгоритм виконання формули (1), що закладений в HDL опис

Додаткові вимоги до функціонування HDL опису:

– після завершення гілки алгоритму, повинен з’явитися активний сигнал на виході готовності;

– після зміни значення хоча б одного операнда, виконання алгоритму повинно перерватися та продовжитися з початку;

– повинен бути наявний зовнішній сигнал скидання, що перезапускає виконання алгоритму;

– повинен бути наявний зовнішній сигнал дозволу, що призупиняє виконання алгоритму.

Необхідно реалізувати Simulink модель на основі даного HDL опису. Модель повинна у повній мірі відтворювала функціональність HDL опису. Проте, структура моделі може відрізнятися. Таким чином, модель буде лише функціональним відображенням HDL опису, а структура буде спрощеною, що полегшить сприйняття людиною закладеного функціоналу в HDL описі.

2.2 Розробка методу реінжинірингу

Тема реінжинірингу схем на FPGA широко представлена в минулих наукових дослідженнях [6, 7, 8]. Проте, ці дослідження спрямовані на реінжиніринг бітового потоку (англ. bitstream). Бітовий потік містить інформацію про конфігурацію користувацької схеми на FPGA. Результатом такого реінжинірингу є список з’єднань. Інформація про процес формування бітового потому є закритою. Тому, більшість робіт сконцентровані на пошук вразливостей та слабких місць при реінжинірингу саме бітового потоку для уникнення або ускладнення можливості несанкціонованого доступу в майбутньому. Адже, власники FPGA проєктів не хочуть, щоб логіка їх продукт була відтворена в процесі реінжинірингу і, таким чином, була розкрита комерційна тайна щодо, наприклад, алгоритмів обробки, що реалізує цей FPGA проєкт. Натомість, в даній роботі метою є реінжиніринг не

бітового потоку, а HDL опису з ціллю спрощення його сприйняття людиною шляхом створення високорівневої його моделі.

2.2.1 Відтворення тракту обробки даних

Перед початком реінжинірингу необхідно визначити послідовність цього процесу для забезпечення найбільшої його оптимальності по часу та якості. Якість залежиться від наочності та простоти користування отриманою моделлю. Задля визначення порядку реінжинірингу необхідно відтворити тракт обробки даних, що реалізує схема на FPGA (далі – тракт даних).

FPGA проєкт складається з екземплярів HDL модулів (далі – екземпляри), поєднаних між собою в певній послідовності. Цю послідовність можна відслідковувати за допомогою інструмента RTL Viewer в програмному середовищі Quartus. Будемо називати цю послідовність трактом даних. Тракт даних має декілька етапів. Кожному етапу відповідає каскад з екземплярів, що можуть працювати паралельно. Проте, паралельно не означає одночасно. Адже, екземпляри можуть працювати на різній частоті, будучи під’єднаними до синхросигналів з різною частотою.

Алгоритм визначення каскаду для певного екземпляра.

Крок 1. Екземпляр відноситься до 1-го каскаду, якщо екземпляр має хоча б один вхід, перед яким не має жодного екземпляра на шляху до зовнішнього вхідного порту FPGA проєкта.

Крок 2. Екземпляр відноситься до 2-го каскаду, якщо екземпляр має хоча б один вхід, перед яким є екземпляр 1-го каскаду.

Крок 3. Екземпляр відноситься до 3-го каскаду, якщо екземпляр має хоча б один вхід, перед яким є екземпляр 2-го каскаду і так далі.

Результатом відтворення тракту даних будуть списки HDL модулів, екземпляри яких відносяться до певного каскаду. Ці списки можна представити у вигляді як на рис. 2.2. Згідно з рис. 2.2, спочатку необхідно виконати реінжиніринг HDL модулів, екземпляри яких відносяться до першого каскаду тракту даних. Першим модулем для реінжинірингу буде такий модуль, що найбільш простим для

цього процесу. Потім, необхідно виконати реінжиніринг HDL модулів, екземпляри яких відносяться до другого каскаду і так далі.

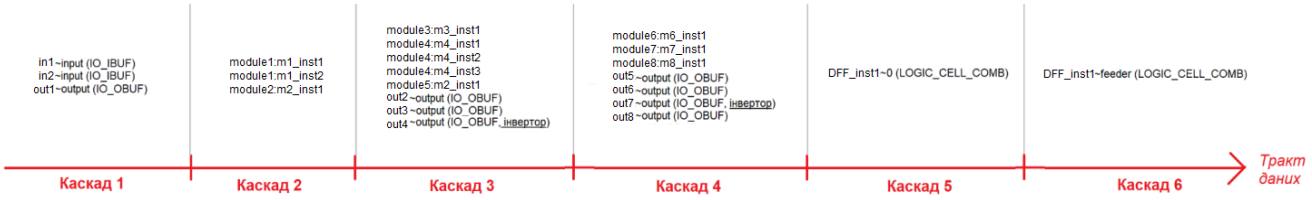


Рисунок 2.2 – Приклад вигляду списків HDL модулів, екземпляри яких відносяться до певних каскадів тракту даних

Зовнішні вхідні/вихідні порти та буфери після/перед ними, що створюються автоматично та використовуються через особливості архітектури FPGA (наприклад, зовнішній вхід, що під'єднується FLASH-пам'яті, яка є вбудованою в FPGA та зберігає конфігурацію FPGA проєкта), не враховуються в процесі реінжинірингу. Тому, що не впливають на логіку роботу FPGA проєкта.

2.2.2 Створення блок-діаграми

В процесі реінжинірингу будується блок-діаграма схеми на FPGA, де відображаються зовнішні вхідні/вихідні порти проєкта, екземпляри модулів, їх порти та зв'язки між ними за допомогою шини або ланцюга (рис. 2.3). На зв'язках можуть бути стрілки, що вказують напрямок передачі сигналу, з ціллю спрощення розрізнення прямих та зворотніх зв'язків на блок-діаграмі.

В процесі створення блок-діаграми, в першу чергу додаються зовнішні вхідні та вихідні порти схеми. Зовнішні порти, які має схема, можна визначати за допомогою вікна RTL Viewer. Далі, в додатку до блок-діаграми (текстовий документ) виконується опис кожного зовнішнього порту (його призначення, форма сигналу, що надходить або формується на порту, та інша інформації).

Далі, додаються HDL модулі першого каскаду тракту даних. Першим додається модуль з найбільш простою логікою для процесі реінжинірингу модулі. Відбувається реінжиніринг цього модуля, в процесі якого визначається приблизна

форма його вихідних сигналів, на основі закладеної в ньому логіки перетворення вхідних сигналів.

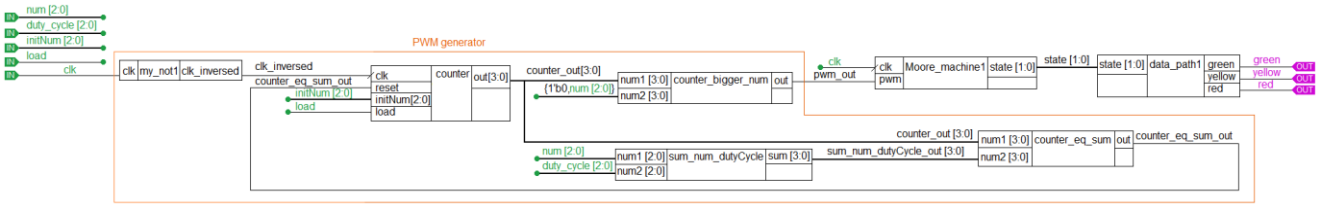


Рисунок 2.3 – Приклад блок-діаграми схеми на FPGA

В додатку до блок-діаграми описується форма вхідних/вихідних сигналів цього модуля. Таким чином, на основі форми вхідного та вихідного сигналів модуля читач даного додатку можне приблизно уявляти реалізований алгоритм в модулі та його призначення, про що також можна вказати в додатку. Наприклад, опис вихідного порту певного модуля може бути наступним: результат фільтрації FIR-фільтром, що має певні параметри, вхідного сигналу x.

Після цього, на блок-діаграму додається наступний модуль першого каскаду. Після додання модулів першого каскаду, відбувається додання модулів наступних каскадів.

2.2.3 Порядок реалізації Simulink-моделі

Після створення блок-діаграми, відбувається створення Simulink-моделі HDL опису. Спочатку, логіка HDL модулів описується за допомогою високорівневої мови програмування MATLAB. Модулі описуються в такому ж порядку, в якому вони були додані до блок-діаграми.

Після створення MATLAB-моделі модуля першого каскаду, відбувається його симуляція. Якщо в результаті симуляції, форма сигналу відповідає очікуваній, то ця MATLAB-модель додається до Simulink-моделі і створюється MATLAB-модель наступного HDL модуля. Якщо ця модель створена на основі HDL модуля, що належить до також до першого каскаду, то вона може бути просимульована окремо. Якщо HDL модуль належить до наступного каскаду, то ця MATLAB-модель додається до Simulink-моделі, відбувається поєднання цих моделей між

собою і відбувається симуляція двох MATLAB-моделей, поєднаних між собою. Таким чином, MATLAB-моделі послідовно додають до Simulink-моделі. Після додання всіх MATLAB-моделей, буде отримано Simulink-модель схеми на FPGA.

2.3 Реалізація Simulink моделі

2.3.1 Перевірка правої та лівої гілок алгоритму в першій версії моделі (без виходу готовності ready)

Згідно з рис. 2.4, проведено тестування виконання правої та лівої гілок алгоритму HDL опису в програмному середовищі Vivado. Не зважаючи, що HDL опис призначений для програмування Intel FPGA, поведінкове моделювання може проводитися в будь-якому середовищі незалежно від фірми виробника. В результаті виконання тестування, в опосередкованому вигляді (відстежуючи оновлення вмісту регістра проміжного та фінального результату S1) бачимо послідовність зміни станів автомата для правої та лівої гілок. Тобто, під час виконання для лівої гілки послідовність вмісту регістра S1 є 0, E, 0, 8, E, 1, 6 (в 15-й системі числення), а під час виконання правої гілки – 0, 6, f.

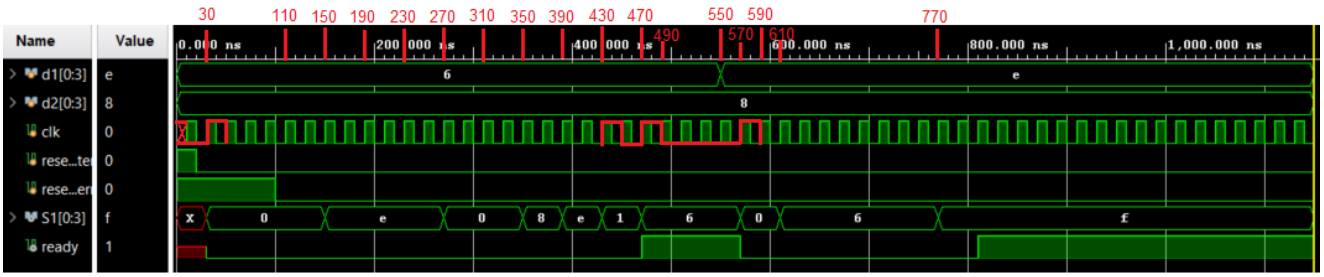


Рисунок 2.4 – Періврка виконання гілок алгоритму в HDL опису

Натомість, для опосередкованого визначення стану автомата необхідно спиратися не лише на вміст регістру S1, але також на тривалість збереження певного значення в цьому регістрі. Тривалість залежить від того як часто

змінюється реігстр S1 в процесі виконання алгоритму. Якщо, наприклад, вміст регістру змінюється в першому та третьому станах, то значення S1 буде незмінним протягом двох тактів синхросигналу або двох симуляційних кроків. Таким чином, для лівої гілки тривалість значень (в 15-й системи числення) 0, E, 0, 8, E, 1, 6 становить 1, 3, 2, 1, 1, 1, 1 тактів, відповідно. Для правої гілки тривалість значень 0, 6, F становить 1, 4, 1 тактів, відповідно.

Після завершення виконання алгоритму, в регістрі S1 перебує результат виконання алгоритму, а на виході готовності результату виконання алгориму ready встановлюється активний сигнал. Якщо хоча б один операнд (d1 або d2) алгоритму змінює значення, то алгоритм виконується заново з новими значеннями операндів.

При реалізації автомата мовою MATLAB також необхідно враховувати порядок виконання операцій, розташованих у вершинах алгоритму (див. рис. 2.1). В даному випадку, порядок виконання операцій змінювати не треба. Проте, може бути ситуація, коли цей порядок необхідно змінити. Нприклад, це необхідно у випадку, якщо б в стані 4 порядок операцій був би S3 = fi(bitsll(S1,3), bit4, wrap), S2 = fi(S2 + S3, bit4, wrap). Через те, що в операторній вершині припускається паралельне виконання операцій, і в HDL описі також можливе паралельне виконання операцій, але в програмі на мові MATLAB операцій виконується послідовно, регістр S3 в другій операцій містив би не значення, записане в попередньому стані автомата, а значення, записане в поточному стані в попредній операції. Тому, необхідно змінити операції місцями.

Проте, в ситуації наявності у вершині операцій S3 = S2 + 1 та S2 = S3 + 1, поміняти операції місцями не достатньо. Необхідно створити тимчасову зміну для збереження значення одного з регістрів, отримане в одному з попередніх станів автомата.

В першій версії моделі реалізовано спрощену модель головного автомата «mainAutomat» або «spec_dig_dev_model» (в додатоку Б.1 наведно вже завершену модель головного автомата). На рис. 2.5 наведено перший варіант (версію) моделі HDL опису, де присутній лише Simulink блок головного автомата та додаткові блокі необхідні для його тестування.

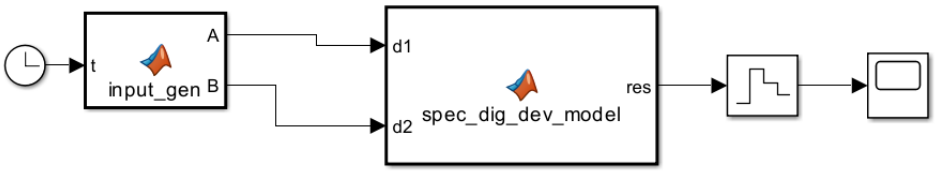


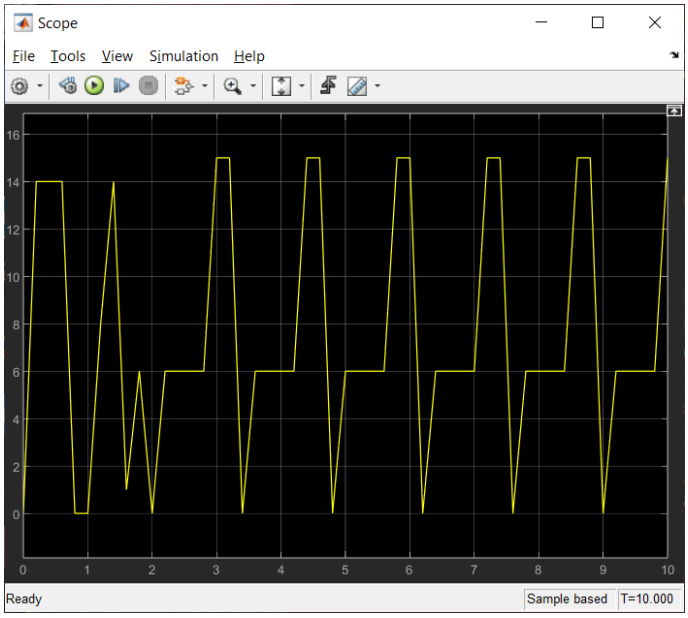
Рисунок 2.5 – Перша версія моделі HDL опису

Крайній лівий блок «годинник» використовується для часової затримки при генерації вхідних сигналів в наступному блоці «input_gen». В першій версії моделі тестується виконання правої та лівої гілок алгоритму (лістинг в додатку Б.4). Згідно з рис. 2.6, графік відображає вміст регістру S1, що містить проміжні та фінальний результат виконання алгоритму. Спочатку виконується ліва, а потім права гілки алгоритму в циклі. Передостаній блок справа «Zero-hold block» використовується для перетворення аналогового (безперервного) сигналу в цифровий.

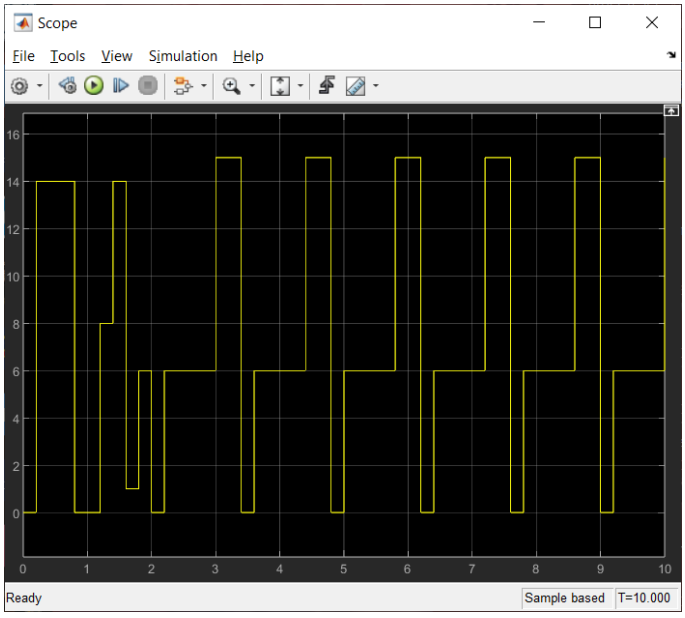
В даній версії моделі відсутній сигнал готовності ready. Тому, після переходу в останній стан, головний автомат переходить в перший стан автомату. Тобто, алгоритм почанає виконуватися заново. Таким чином, алгоритм виконується в циклі. Гілка алгоритму змінюється при виконання умови, що залежить від значення операнда A. В разі зміни операнда (згідно з рис. 2.6, на другій секунді) в момент виконання однієї з гілок можливий перехід на іншу гілку в результаті перевірки умови в наступному циклі виконання алгоритму. Таким чином, згідно з рис. 2.6, на другій секунді змінюється операнд A зі значення 6 на 14 і умова $A \leq 9$ не виконується, через що виконується права гілка в наступному циклі виконання алгоритму. Згідно з рис. 2.4 та рис. 2.6 послідовності та тривалості проміжних результатів для обох гілок співпадають. Тому, тест 1 пройдений успішно.

Всі блоки (зокрема, блоки, що реалізують MATLAB функції) в Simulink моделі запускаються періодично через певний проміжок часу, що називається симуляційним кроком (англ. simulation step). В даному випадку, симуялційний крок складає 200 мілісекунд. Цей крок регулює періодичінсть оновнення результату на виходах Simulink блоків. Тому, стани автомата оновлюються кожні 200 мілісекунд. На виході блоку «годинник» формується поточний симуляційний час, що

оновлюється також кожні 200 мілісекунд. Отже, в Simulink моделі не має потреби в синхросигналі clk, що використовується в HDL описі. Блоки запускаються кожний симуляційний крок.



а)



б)

а – аналоговий сигнал; б – цифровий сигнал

Рисунок 2.6 – Результат тесту 1

Щоб значення зміних функції не втрачались після завершення функції кожного разу в певний симуляційний крок, то ці зміні визначаються за допомогою ключого слова «persistent». Якщо зміна порожня, то вона ініціалізується початковим значеням, що є аналогом до застосування зовнішнього сигналу скидання reset_divider_extern в HDL описі.

Згідно з лістингом Б.1, зміна «bit4» використовується для скороченого запису «numerictype(0,4,0)», що повертає 4-бітний беззнаковий тип цілого числа з фіксованою комою. Далі, цей тип використовується з конструкторі створення числа з фіксованою комом. Наприклад, всі регістри автомата ініціалізуються нульовим значенням вигляду «fi(0, bit4, wrap)», де «wrap» використовується для скороченого запису «fimath('OverflowAction','Wrap', 'RoundingMethod','Zero')», що визначає поведінку при переповнені та виникнені дробової частини в результаті виконання операції. Коли в результаті виконанні операції з’являється дробова частина, то вона відкдається. У разі переповнення надлишкові біти відкидаються.

В мові MATLAB відсутня побітова операція інверсії. Операція «~» є логічною інверсіює, реальтатом якої є логічне значення «true» або «false». Лише з однорозрядними операндами ця операція працює як побітова інверсія. Якщо операнд є багаторозрядним, то результатом операції є інверсія результату побітового «АБО» над бітами багаторозрядного операнда. Отже, для роботи з багаторозрядними операндами необхідно застосовувати операцію «XOR» з маскою. Маскою є максмальне 4-бітні число «mask = fi(15, bit4, wrap)», бо всі операції в алгоритмі виконуються над 4-бітними числами. Виконання операції «XOR» над заданим числом та маскою в результаті дає інверсію заданого числа, наприклад, «B = fi(bitxor(mask, B), bit4, wrap)».

2.3.2 Перевірка правої та лівої гілок алгоритму в другій версії моделі (з виходом готовності ready)

Згідно з рис. 2.7, до моделі додано вихід готовності ready в блоці «mainAutomat» або «spec_dig_dev_model».

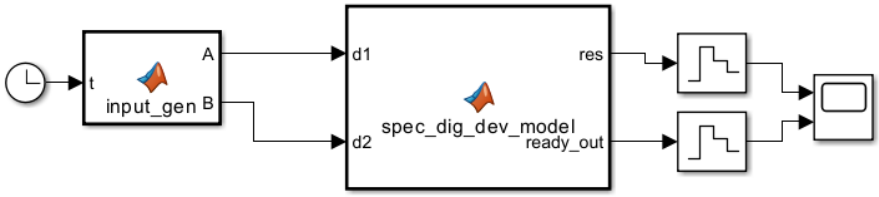


Рисунок 2.7 – Друга версія моделі HDL опису

Для тестування оновленої моделі використовується також тест 1, згідно з лістингом в додатку Б.4 (рис. 2.8). Наявність сигналу готовності (синій графік) в даній моделі не впливає на вміст регістру S1 (жовтий графік). Сигнал готовності встановлюється в момент формування фінального результату виконання алгоритму в регістрі S1. Тобто, в останньому стані виконання певної гілки. Якщо виконується ліва гілка, то останнім станом виконання алгоритму є стан 10, якщо права – стан 15. Сигнал готовності встановлюється протягом одного такту та скидується в першому стані автомата. Проте, згідно з вимогами до HDL опису, поява сигналу готовності повинна призупиняти виконання алгоритму (див. рис. 2.4). Натомість, механізм призупинення виконання алгоритму буде реалізований лише в четвертій версії моделі.

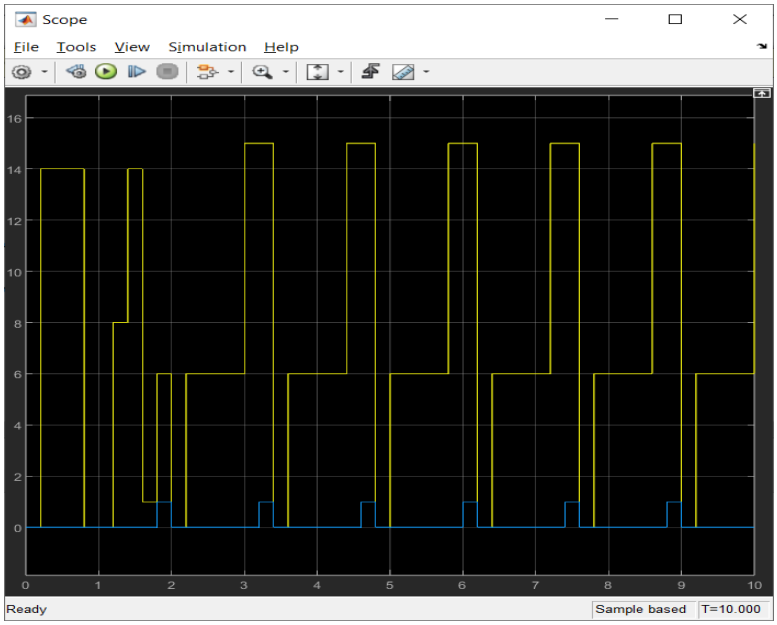


Рисунок 2.8 – Тестування правої та лівої гілок алгоритму з виходом готовності ready в моделі

2.3.3 Перевірка зовнішнього сигналу скидання reset_spec_dev_extern під час виконання алгоритму в третій версії моделі

Згідно з рис. 2.9, в момент 1300 наносекунд відбувається скидання автомата, під час якого обнуляються всі внутрішні регістри автомата та, зокрема, стан автомата.

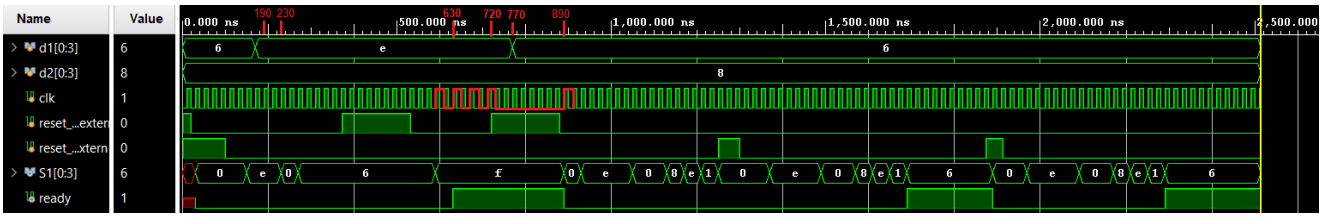


Рисунок 2.9 – Перевірка скидання автомата в процесі виконання алгоритму в HDL описі

Згідно з рис. 2.10, створено третю версію моделі, в якій блок головного автомата «mainAutomat» або «spec_dig_dev_model» має вхід зовнішнього сигналу скидання reset_spec_dev_extern.

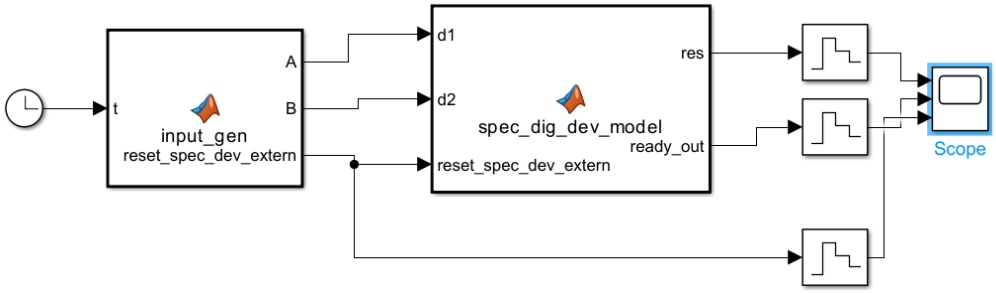
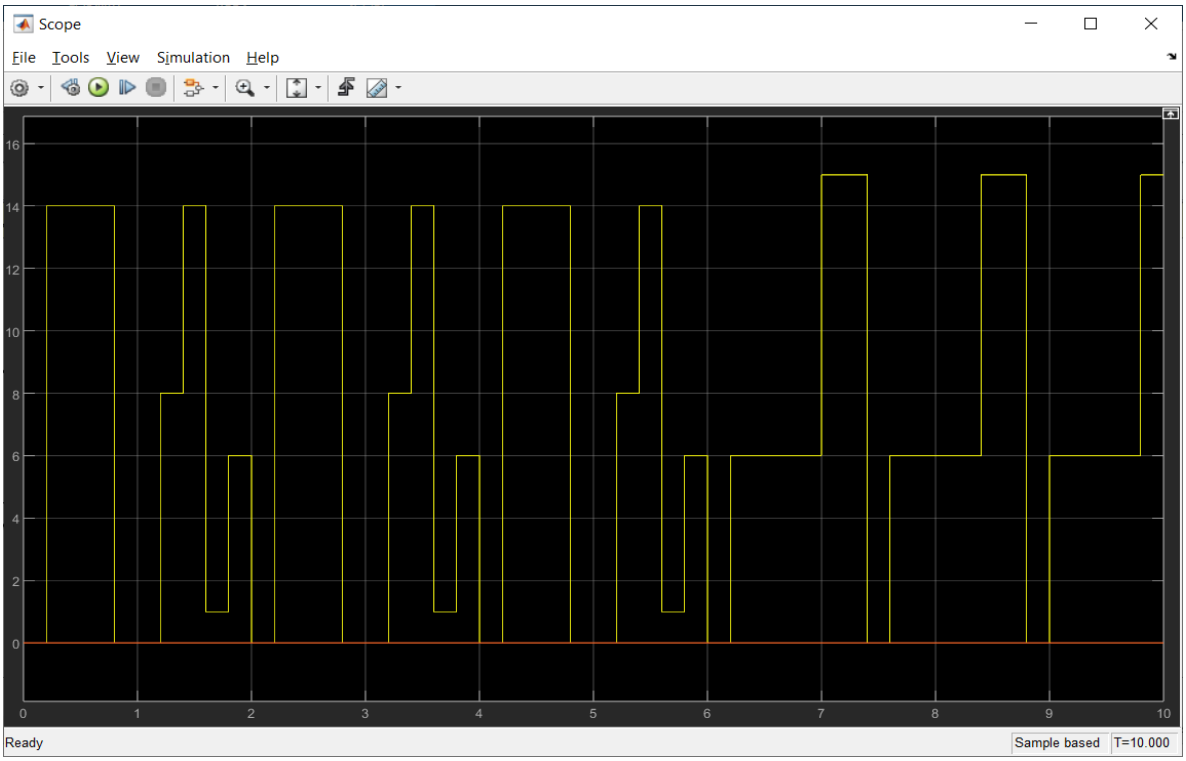
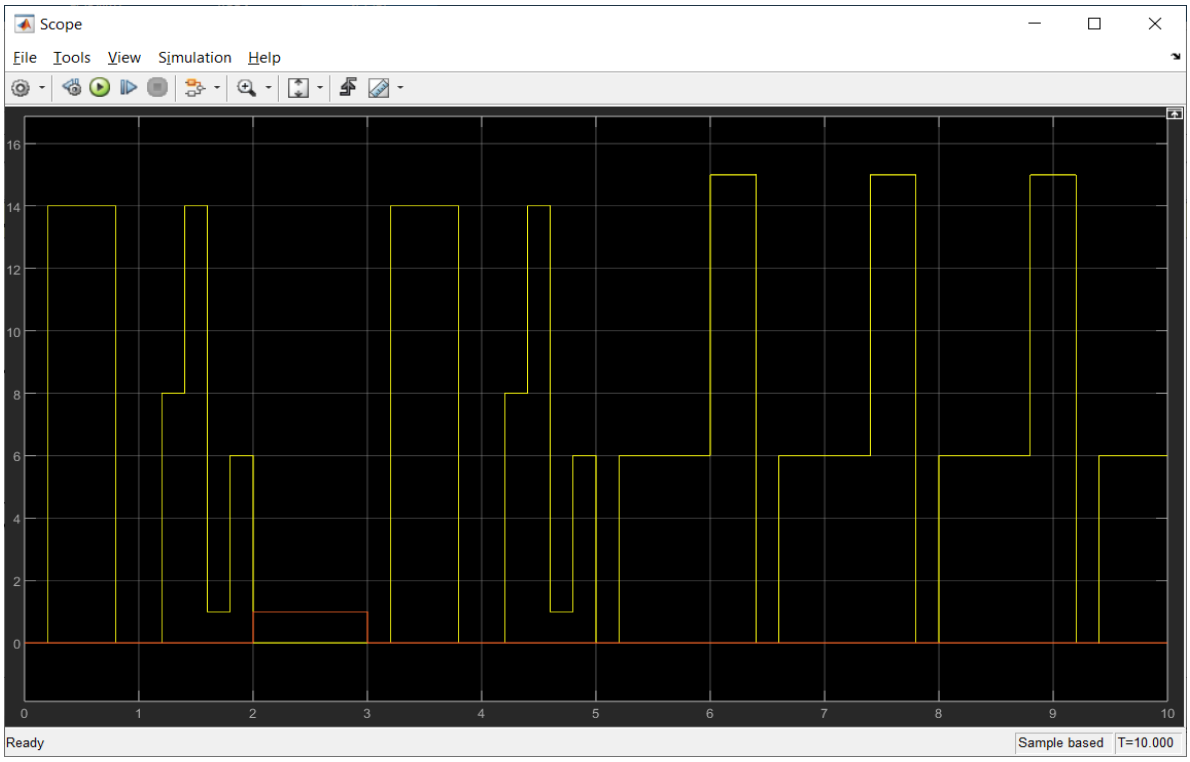


Рисунок 2.10 – Третя версія моделі HDL опису

В третій версії моделі в блоці головного автомата блок «switch» огорнутий в умовний блок з умовою перевірки сигналу скидання. Згідно з рис. 2.11, протестовано (лістинг в додатку Б.5) застосування скидання (помаранчевий графік) в головному автоматі, де можна спостерігати обнулення регістра S1 під час скидання автомата.



а)



б)

а – скидання відсутнє; б – скидання відбувається

Рисунок 2.11 – Результат виконання тесту 2

2.3.4 Перевірка зовнішнього сигналу скидання reset_spec_dev_extern після виконання алгоритму (при ready = 1) в четвертій версії моделі

Згідно з рис. 2.12, в момент 1870 наносекунд відбувається скидання автомата при ready = 1, під час якого обнуляються всі внутрішні регістри автомата та, зокрема, стан автомата.

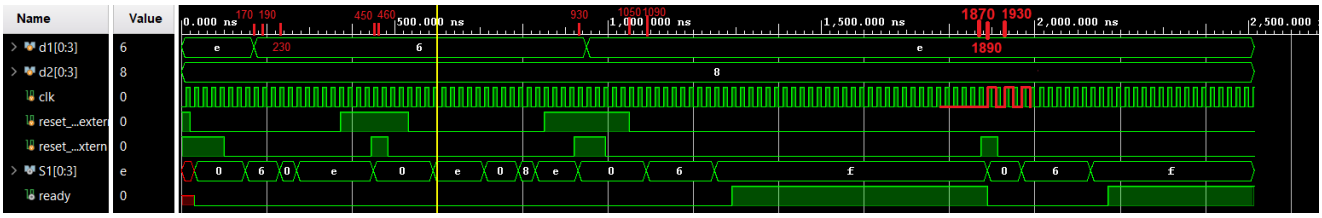


Рисунок 2.12 – Перевірка скидання автомата після виконання алгоритму в HDL описі

Згідно з рис. 2.13, створено четверту версію моделі, в якій реалізовано механізм призупинення виконання алгоритму після формування фінального результату виконання алгоритму. Блок головного автомата «mainAutomat» або «spec_dig_dev_model» тепер має вхід внутрішнього дозволу тактування reset_divider. Додано блоки «to_first_state_gen» (лістинг в додатку Б.2) та «inner_ena» (лістинг в додатку Б.3). Також, в блоці «input_gen» додано вихід зовнішнього сигналу дозволу тактування reset_divider_extern, який буде протестований пізніше як і блок «to_first_state_gen».

В четвертій версії моделі головний автомат огорнутий в умовний блок «reset_divier == 0». Це означає, що при встановлені внутрішнього сигналу дозволу тактування reset_divier автомат не змінює свого стану (призупиняє своє виконання). При виконанні умови «reset_divier == 1», автомат виконується. Значення сигналу reset_divier залежить від сигналу готовності результату ready. Тому, після завершення виконання алгоритму, встановлення сигналу готовності ready призводить до встановлення сигналу внутрішнього дозволу reset_divier.

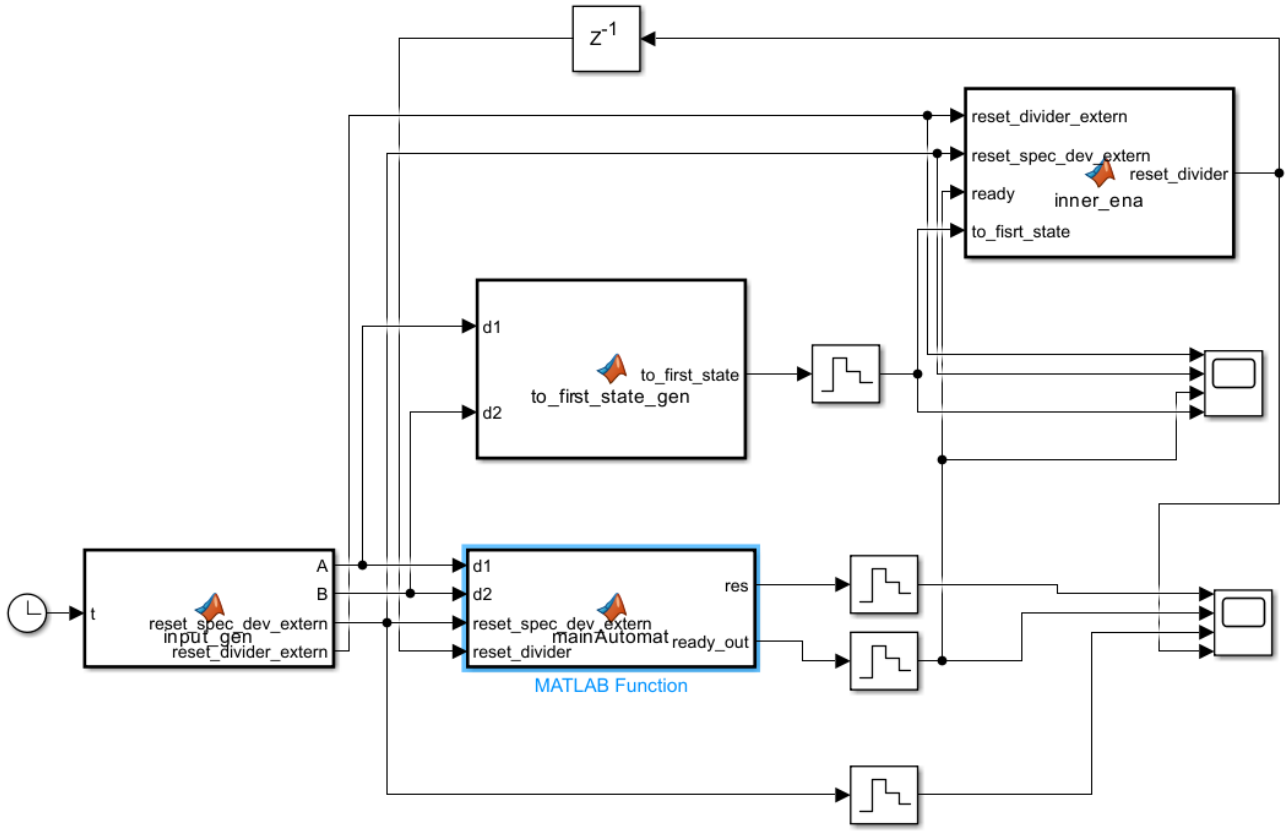


Рисунок 2.13 – Четверта версія моделі HDL опису

Зокрема, залежність сигналу reset_divier від ready описана в блоці, що реалізує схему формування внутрішнього сигналу дозволу «inner_ena». Загалом, в блоці «inner_ena» описано всі залежності сигналу внутрішнього дозволу тактування reset_divier від комбінації сигналів готовності результату виконання алгоритму ready, зовнішнього сигналу дозволу тактування reset_divider_extern та зовнішнього сигналу скидання reset_spec_dev_extern з врахуванням прапора переходу в початковий стан головного автомата to_first_state. Таким чином, в момент встановлення сигналу готовності ready та відсутності зміни операнда в попередній синхротакт або відсутності при цьому зовнішнього сигналу скидання reset_divider_extern, відбувається встановлення внутрішнього сигналу дозволу тактування reset_divider, що призупиняє виконання алгоритму до моменту зміни операнда або встановлення зовнішнього сигналу скидання reset_spec_dev_extern.

Згідно з рис. 2.14, протестовано (лістинг в додатку Б.6) застосування зовнішнього сигналу скидання reset_spec_dev_extern після завершення виконання алгоритму (при ready = 1).

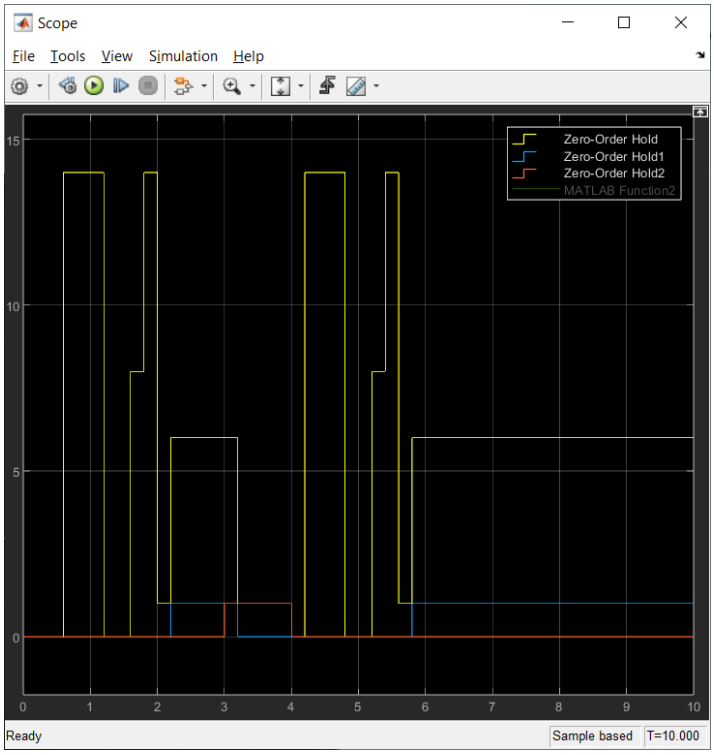


Рисунок 2.14 – Перевірка скидання при ready = 1 в моделі

Згідно з рис. 2.14, після завершення алгоритму, встановлюється сигнал готовності ready, що призупиняє виконання алгоритму. В автоматі обнулюються внутрішні регістри та регістр стану автомата при появі зовнішнього сигналу скидання reset_spec_dev_extern.

2.3.5 Перевірка перезапуску алгоритму при ready = 1 в разі зміни операнда в четвертій версії моделі

Згідно з рис. 2.15, в момент 550 наносекунд відбувається зміна операнда, в результаті чого перезапускається виконання алгоритму (автомат переходить в початковий стан) з новими значеннями операндів. Фактично, в момент зміни операнда алгоритм продовжує своє виконання. Проте, через те, що автомат перебуває в останньому стані, а наступним станом є початковий стан автомата, бо автомат є циклічним, це можна назвати перезапуском алгоритму.

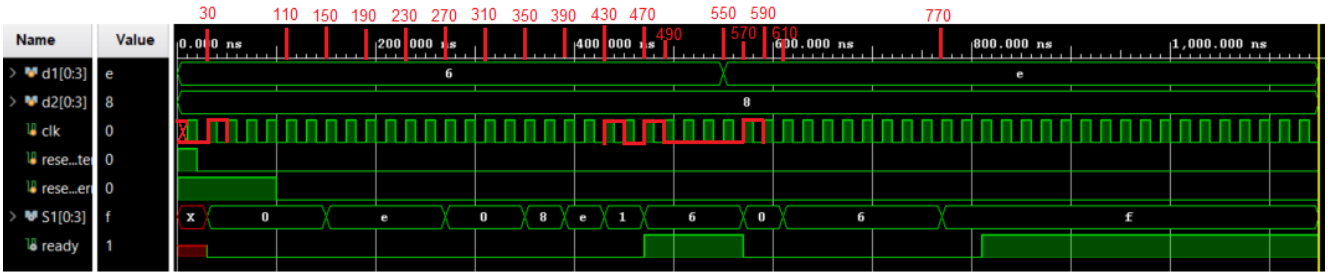
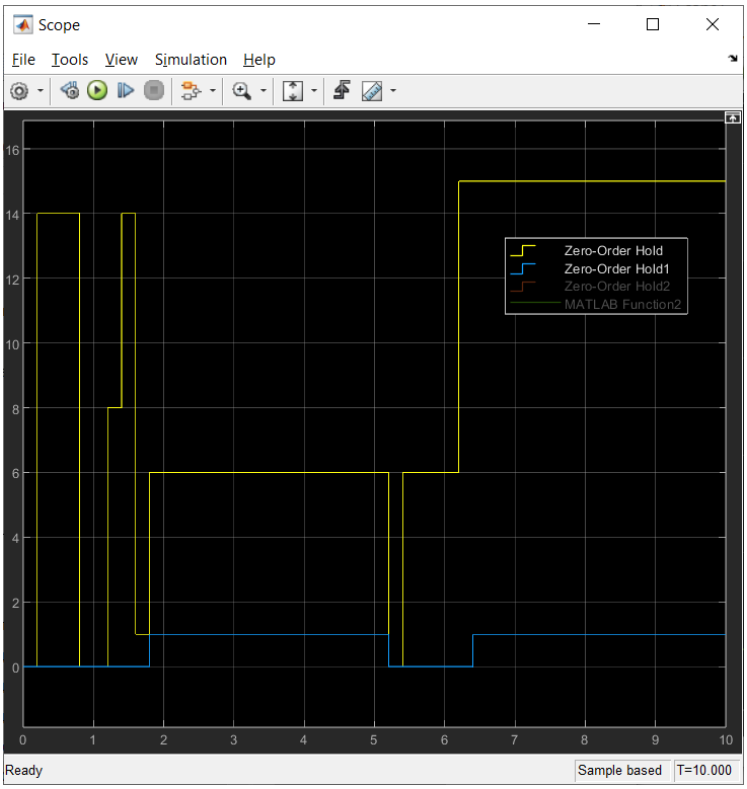
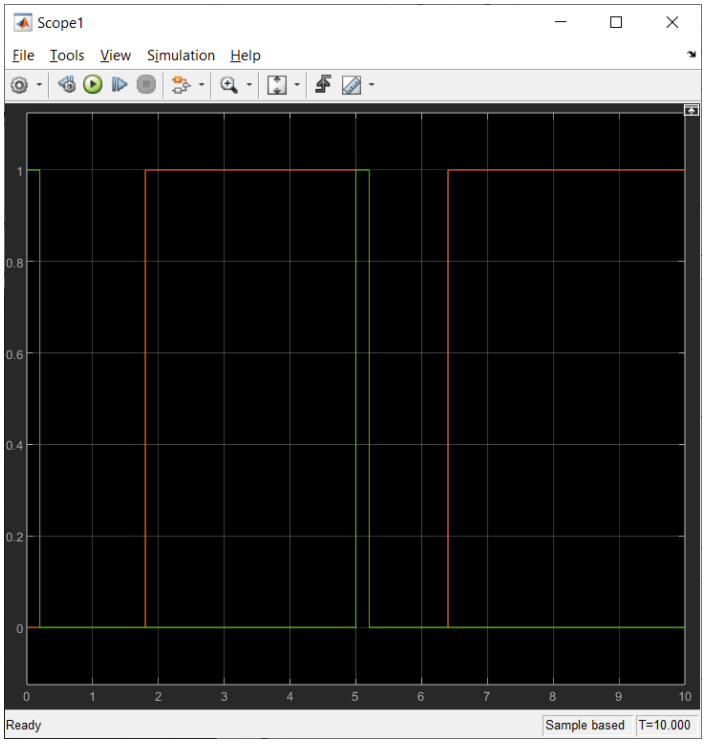


Рисунок 2.15 – Перевірка перезапуску алгоритму при ready = 1
в разі зміни операнда в HDL описі

Згідно з рис. 2.16, наведено результат виконання тесту (лістинг в додатку Б.7) з перевірки перезапуску алгоритму в моделі. На п’ятій секунді відбувається зміна операнда. В цей же момент відбувається встановлення прапора to_first_state. Прапор встановлюється в разі різниці значень одного з операндів в поточному та попередньому синхротактах (див. лістинг блоку «to_first_state_gen» в додатку Б.2). Прапор скидується, якщо операнди однакові в поточному та попередньому синхротактах, якщо в попередньому такті не було активного стану на зовнішньому сигналі дозволу тактування reset_divider_extern. Тобто, в даному випадку, прапор скидується в наступний синхротакт після встановлення. Отже, встановлення прапору відбувається протягом одного синхротакта, згідно з рис. 2.16 б), де зелений графік відображає сигнал прапора to_first_state, а помаранчевий графік – сигнал готовності ready. Проте, в момент встановлення прапора виконується умова перевірки встановлення прапора в блоці «inner_ena» (блок формування внутрішнього сигналу дозволу reset_divider) в умовному блоці «ready == 1», в результаті чого скидується сигнал внутрішнього дозволу тактування reset_divider, який був встановлений при встановлені сигналу готовності ready, і автомат продовжує своє виконання – переходить в наступний (початковий) стан, бо перебував в останньому стані, а виконання автомату є циклічним.



а)



б)

а – регістр S1; б – прапор to_first_state

Рисунок 2.16 – Результат роботи тесту 4

2.3.6 Перевірка перезапуску алгоритму під час виконання алгоритму (ready = 0) в разі зміни операнда в п’ятій версії моделі

Згідно з рис. 2.17, в п’ятій версії моделі додано вхід to_first_state в блоці головного автомата «mainAutomat» або «spec_dig_dev_model». В автоматі додано умову перевірки прапора to_first_state в кожному стані автомата, окрім першого стану, для можливості переривання виконання алгоритму та його перезапуску в момент зміни операндів. Отже, в момент зміни одного з операндів відбувається встановлення прапора to_first_state. В цей же момент відбувається виконання умови перевірки встановлення прапора to_first_state в певному стані автомата, в результаті чого автомат переходить в початковий стан.

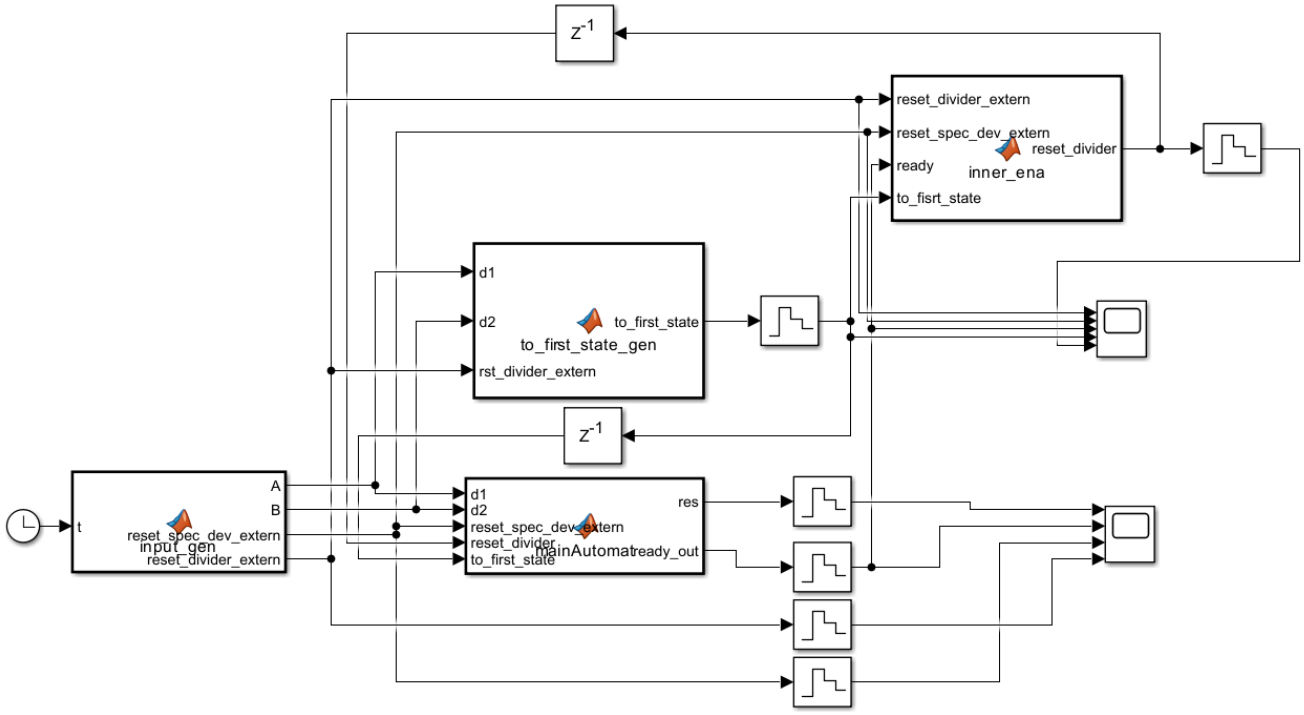


Рисунок 2.17 – П’ята версія HDL моделі

Згідно з рис. 2.18, в першу секунду відбувається зміна операнда, в результаті чого в наступному такті автомат переходить в початковий стан, де відбувається скидання внутрішніх регістрів автомата, а в наступному такті обирається інша гілка алгоритму, згідно з новим значенням операнда.

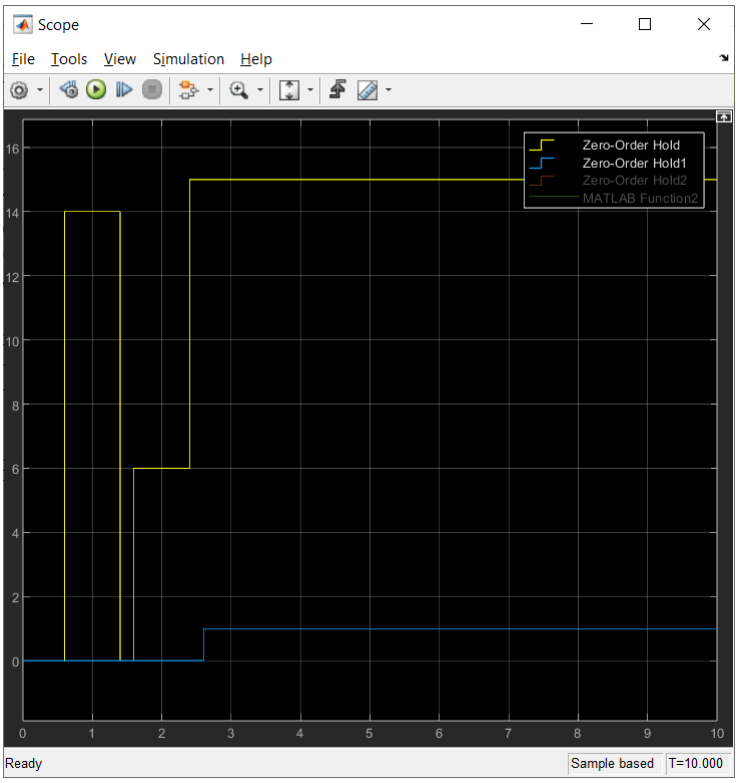
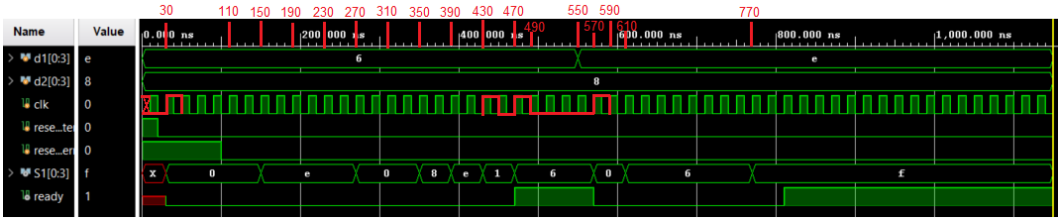


Рисунок 2.18 – Перевірка перезапуску алгоритму під час виконання алгоритму в разі зміни операнда

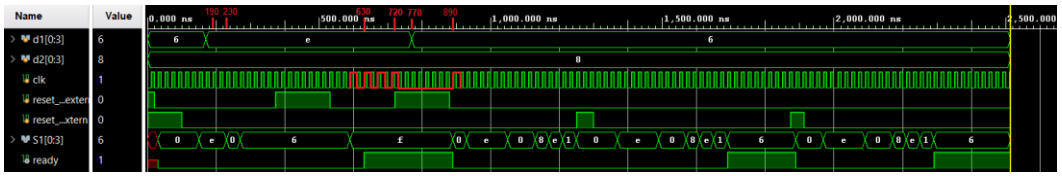
2.3.7 Перевірка зовнішнього сигналу дозволу тактування reset_divider_extern в п’ятій версії моделі

Згідно з рис. 2.19, перевіряється зовнішній сигнал дозволу тактування в HDL описі, де порівнюється тривалість встановлення проміжного результату виконання алгоритму в момент 500 наносекунд. При застосувані сигналу дозволу, тривалість збільшується. Отже, алгоритм призупиняє своє виконання під час reset_divider_extern = 1.

Згідно з рис. 2.20, порівнюється тривалість встановлення проміжного результату виконання алгоритму в момент 1.5 секунди (лістинг в додатку Б.9), де на рис. 2.20 б) зелений графік відображає зовнішній сигнал дозволу тактування reset_divider_extern.



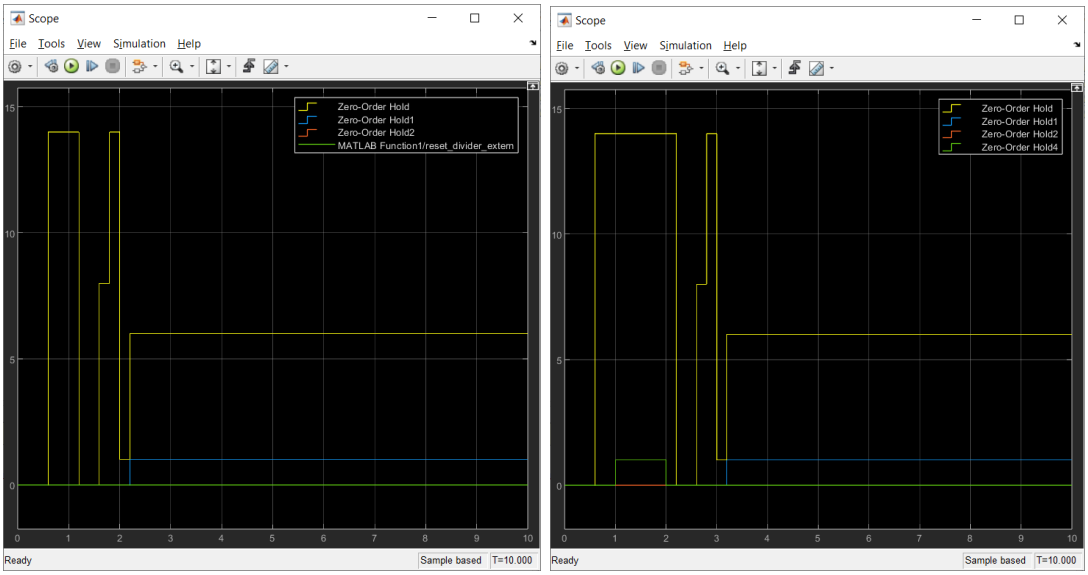
а)



б)

а – при reset_divider_extern = 0; б – при reset_divider_extern = 1

Рисунок 2.19 – Перевіряється зовнішній сигнал дозволу,
в HDL описі, в момент 500 наносекунд



а) б)

а – при reset_divider_extern = 0; б – при reset_divider_extern = 1

Рисунок 2.20 – Результат тесту 6

2.3.8 Перевірка сигналу скидання під час активного зовнішнього сигналу дозволу в п’ятій версії моделі

Згідно з рис. 2.21, в момент 450 наносекунд відбувається скидання автомата наступним чином. Щоб скидання відбулось необхідна наявність тактування та активного стану на сигналі скидання (reset_spec_dev_extern = 1). Тому, в блоці «inner_ena», при активному стані на сигналі скидання, відбувається відновлення тактування (reset_divider = 0), а наявність активного стану на сигналі скидання призводить до скидання автомата.

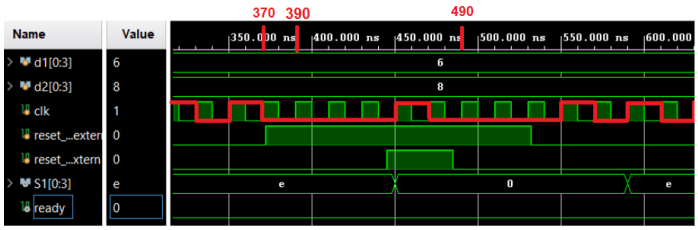


Рисунок 2.21 – Перевірка скидання при reset_divider_extern = 1 в HDL описі

Згідно з рис. 2.22, перевірено скидання автомата (лістинг в додатку Б.10) в моделі, де помаранчевий графік означає зовнішній сигнал скидання reset_spec_dev_extern.

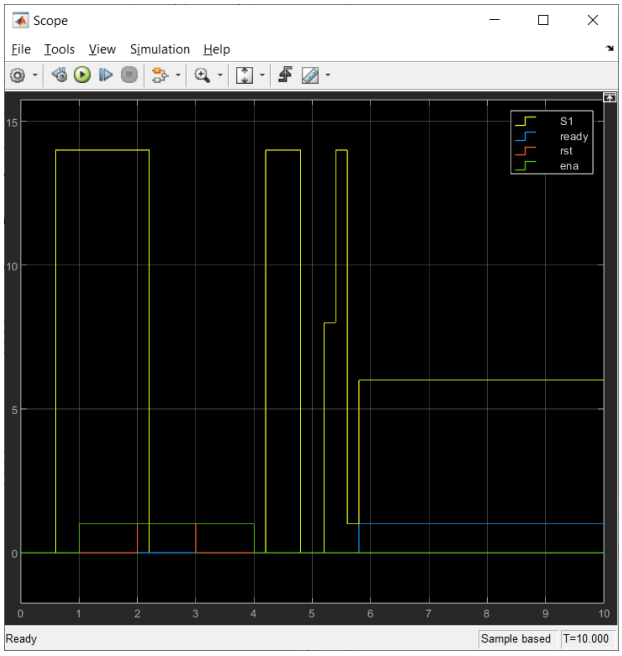


Рисунок 2.22 – Результат тесту 7

2.3.9 Перевірка сигналу скидання під час активного сигналу дозволу в момент зміни операнда в п’ятій версії моделі

Згідно з рис. 2.23, в момент 930 наносекунд відбувається скидання автомата, а в 950 наносекунд зміна операнда. Після завершення скидання в 995 наносекунд продовжується активний сигнал на вході дозволу тактування. Тому, автомат призупиняє своє виконання в початковому стані автомата та продовжує своє виконання після завершення активного стану в момент 1050 наносекунд.

Згідно з рис. 2.24, перевірено скидання при reset_divider_extern = 1 в момент зміни операнда в моделі (лістинг в додатку Б.11), де зелений графік означає зовнішній сигнал дозволу тактування reset_divider_extern.

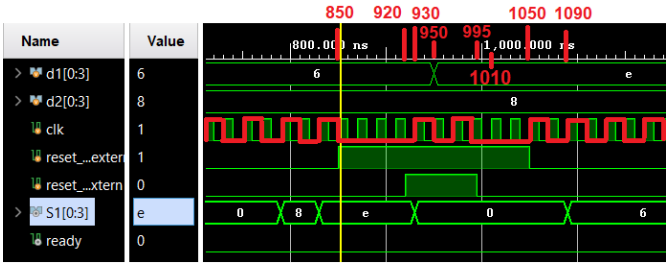


Рисунок 2.23 – Перевірка скидання при reset_divider_extern = 1 в момент зміни операнда в HDL описі

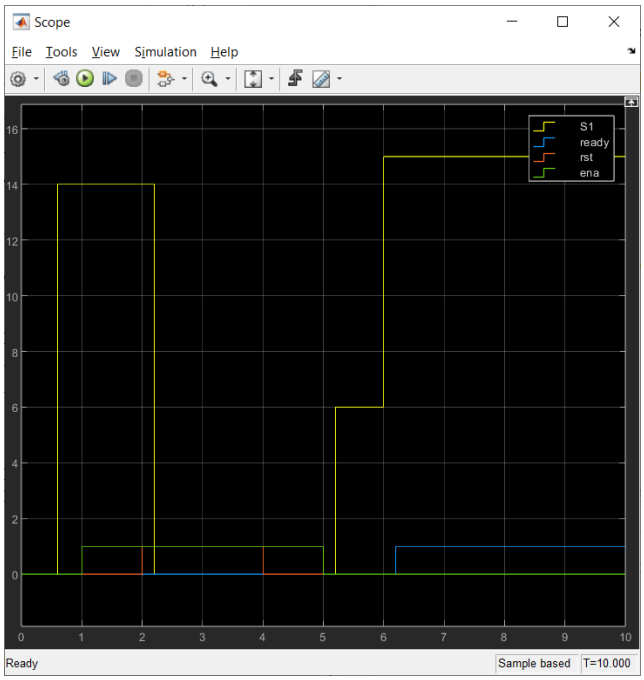
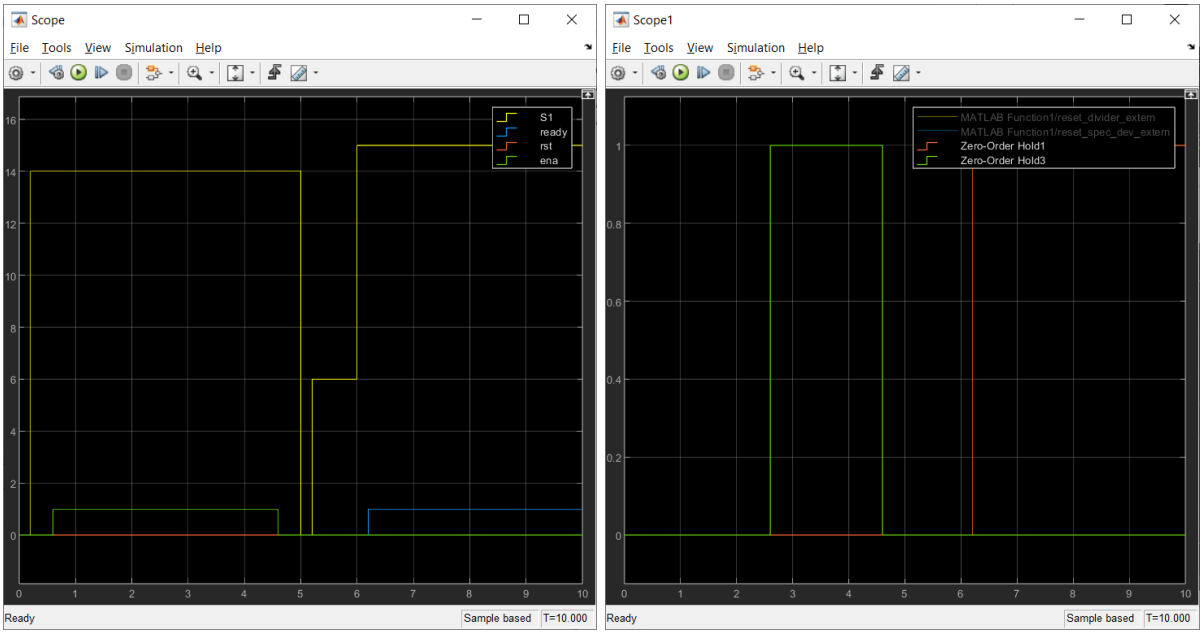


Рисунок 2.24 – Результат тесту 8

2.3.10 Перевірка перезапуску алгоритму при активному сигналі дозволу в разі зміни операнда в шостій версії моделі

Згідно з рис. 2.25, наведено результат виконання тесту 9, згідно з лістингом в додатку Б.12, де зелений графік на 2.25 рис. а) відображає зовнішній сигнал дозволу тактування reset_divider_extern. На 2.25 рис. б) зелений графік відображає значення прапора to_first_state, а помаранчевий – сигнал готовності результату ready. В момент 2.5 секунди відбувається зміна операнда під час активного сигналу на зовнішньому вході дозволу тактування (reset_divider_extern = 1). В момент зміни операнда встановлюється прапор to_first_state, що ідентифікує зміну хоча б одного операнда.



а) б)

а – регістр S1; б – прапор to_first_state

Рисунок 2.25 – Результат тесту 9

В шостій версії моделі додано умову «rst_divider_extern ~= 1», що дозволяє зберігти значення прапора після його встановлення до завершення активного стану на зовнішньому сигналі дозволу тактування reset_divider_extern. Це необхідно для

реакції автомата на зміну операнда під час його призупинення та його перезапуску після призупинення.

2.3.11 Перевірка перезапуску алгоритму при ready = 1 та активному сигналі дозволу в разі зміни операнда в сьомій версії моделі

Згідно з 2.26 рис., в момент 770 наносекунд відбувається зміна операнда. В цей же момент встановлюється прапор to_first_state. Значення прапору запам’ятовується поки reset_divider_extern = 1. Після скидання зовнішнього сигналу дозволу тактування reset_divider_extern, виконується умова «ready == 1 && reset_divider_extern == 0 && reset_spec_dev_extern == 0» в блоці «inner_ena», в результаті чого відбувається скидання внутрішнього сигналу дозволу тактування reset_divider і автомат продовжує своє виконання.

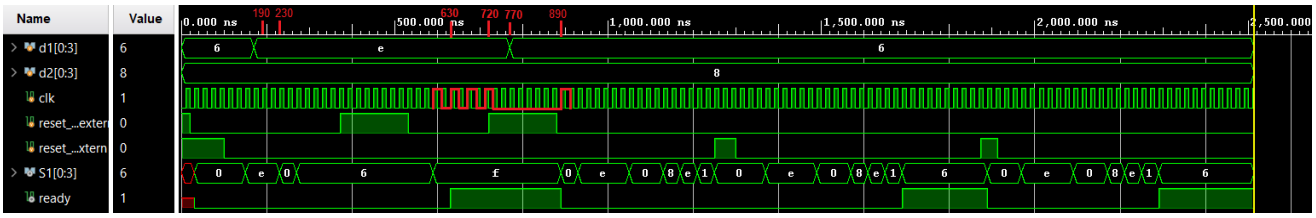
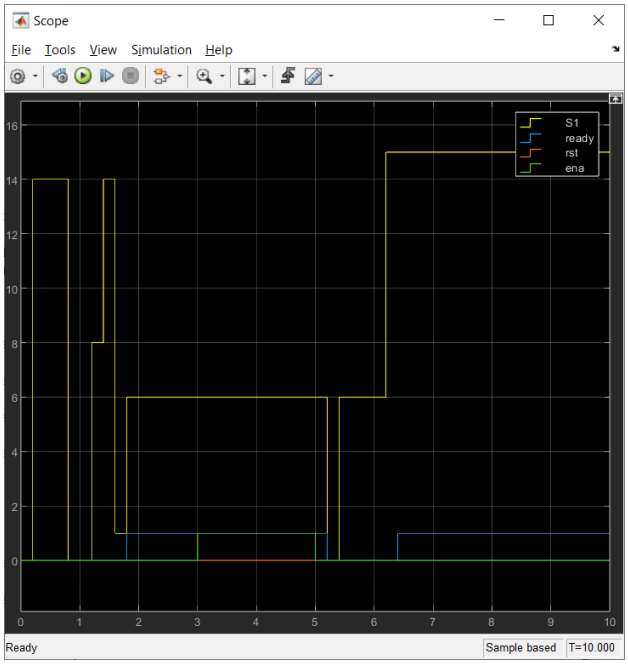


Рисунок 2.26 – Перевірка перезапуску алгоритму при ready = 1 та reset_divider_extern = 1 в HDL описі

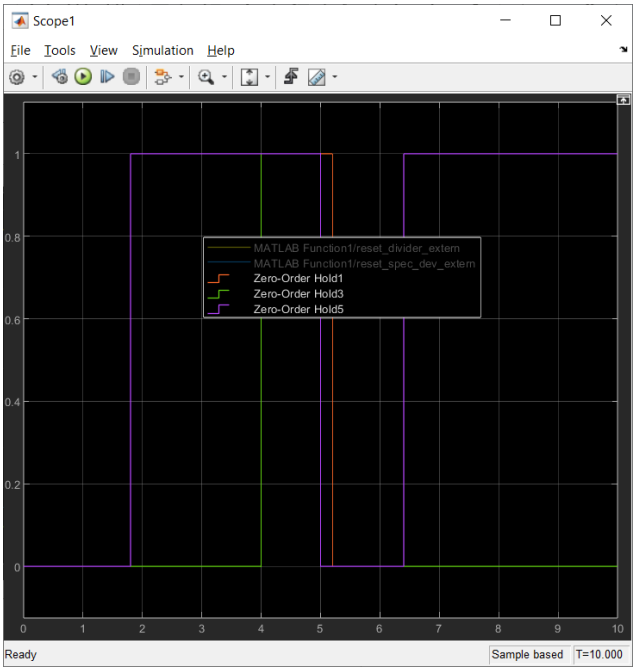
Скидання внутрішнього сигналу дозволу reset_divider відбувається, бо в попередній синхротакт був встановлений прапор to_first_state. Саме в сьомій версії моделі додано буфер, що зберігає значення прапора to_first_state в попередній синхротакт. Це необхідно, бо після скидання зовнішнього сигналу дозволу тактування reset_divider_extern відбувається скидання прапора to_first_state, а умова «ready == 1 && reset_divider_extern == 0 && reset_spec_dev_extern == 0» в блоці «inner_ena» виконується в синхротакт після скидання прапора to_first_state.

Згідно з рис. 2.27, перевіряється перезапуск алгоритму при ready = 1 та активному сигналі дозволу в разі зміни операнда в моделі (лістинг в додатку Б.13). На рис. 2.27 а) зелений графік відображає зовнішній сигнал дозволу тактування reset_divider_extern. На рис. 2.27 б) зелений графік відображає значення прапора

to_first_state, помаранчевий – сигнал готовності ready, а фіолетовий – сигнал внутрішнього сигналу дозволу тактування reset_divider.



а)



б)

а – регістр S1; б – сигнал reset_divider

Рисунок 2.27 – Результат тесту 10

2.3.12 Перевірка перезапуску алгоритму після зміни операнда, при ready = 1 та активному сигналі дозволу в разі скидання в сьомій версії моделі

Згідно з рис. 2.28, в момент 850 наносекунд відбувається зміна операнда. В той же момент встановлюється прапор to_first_state, який скидається в момент встановлення зовнішнього сигналу скидання reset_spec_dev_extern разом із скиданням автомата.

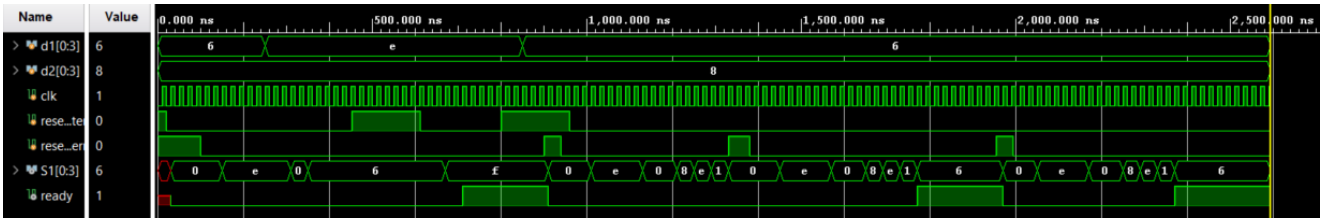
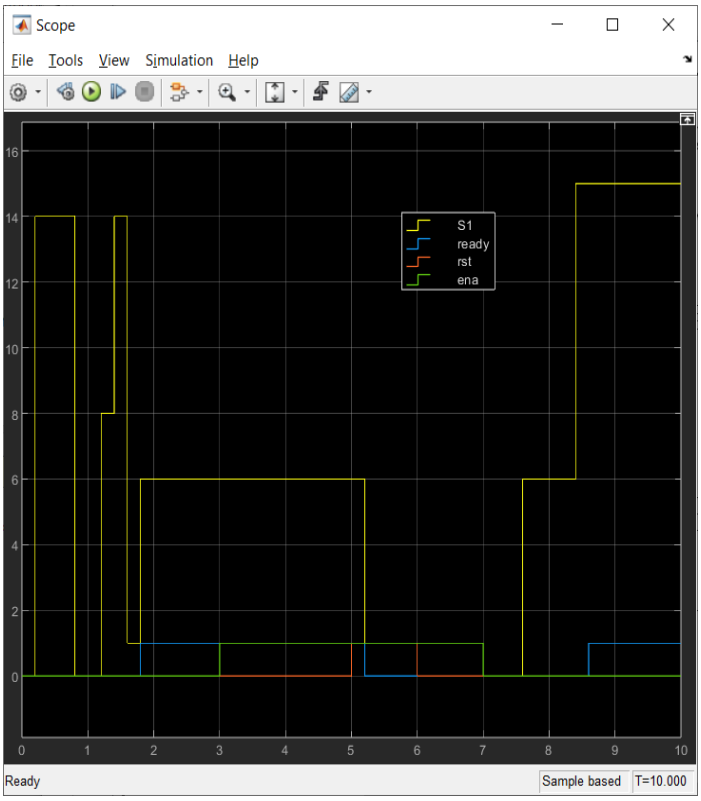


Рисунок 2.28 – Перевірка перезапуску після зміни операнда, при ready = 1 та reset_divider_extern = 1 в разі скидання

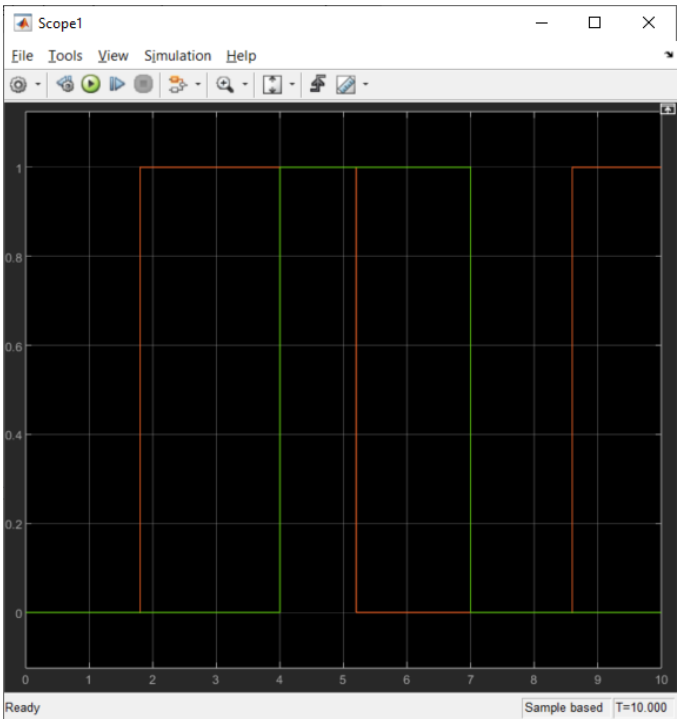
На рис. 2.29 наведено результат виконання тесту 11, згідно з лістингом в додатку Б.14, де на рис. 2.29 а) зелений графік означає зовнішній сигнал дозволу тактування reset_divider_extern, а на рис. 2.29 б) зелений графік означає прапор to_first_state, а помаранчевий – сигнал готовності результату ready.

2.4 Порівняння Simulink моделі та HDL опису

В моделі відсутній синхросигнал clk, бо всі блоки в моделі запускаються періодично кожний симуляційний крок. Тому, як в проєкті лише один синхросигнал, то поведінка, коли всі блоки моделі запускаються з однаковою частотою, є коректною. Синхросигнал меншої частоти clk_divided, що є виходом дільника частоти, відсутній в моделі як і дільник частоти, бо ці елементи схеми призначені для відлагодження HDL опису на реальній FPGA.



а)



б)

а – значення S1; б – значення to_first_state

Рисунок 2.29 – Результат тесту 11

В моделі відсутні так звані сигнали-посередники rst_to_first_state та rst_rst_spec_dev_flag. Вони використовується для скидання прапорів to_first_state та rst_spec_dev_flag (відсутній в моделі). Сигнали-посередники використовуються в HDL описах, що використовуються для Intel FPGA проєктах, через вимоги компілятора Quartus, який формує потік бітів для програмування FPGA на основі HDL опису. Вимога полягає в забороні виконувати операцію присвоювання (наприклад, встановлення або скидання) для одного і того ж сигналу (в даному випадку це прапори to_first_state та rst_spec_dev_flag) в декількох паралельних блоках (наприклад, в блоках always) [9].

Тому, як, наприклад, встановлення to_first_state повинне відбуватися при події зміни даних (в блоці always, що реагує на подію зміни даних), то, згідно з вимогами компілятора Quartus, і скидання to_first_state повинне відбуватися в цьому ж блоці always. Але, згідно з логікою роботи HDL опису, скидання to_first_state повинне відбуватися при події переходу головного автомата в стан 1. Тобто, подія скидання to_first_state відбувається в іншому паралельному блоці. Тому, використовується сигнал-посередник rst_to_first_state для відстеження події скидання та передачі інформації про неї в окремий блок, де відбувається разом встановлення та скидання прапору. Отже, сигнал-посередник встановлюється при переході головного автомата в стан 1. У відповідь на подію встановлення сигналу rst_to_first_state, прапор to_first_state скидується.

В моделі відсутній блок фільтрування дребезгу (англ. debounce filter), який застосовується для вхідних операндів d1 та d2. Ці блоки необхідні, якщо сигнал має дребезг, який може виникати в разі генерації значень операндів, наприклад, за допомогою кнопок або перемикачів. Саме перемикачі використовуватися для генерації значень операндів на відлагоджувальному стенді FPGA для тестування запрограмованого FPGA проєкта. Тому, як в модельних вхідних сигналах дребезг відсутній, то даний фільтр в моделі не потрібний.

3 РЕІНЖИНІРИНГ ДРУКОВАНОЇ ПЛАТИ

В процесі аналізу HDL опису (FPGA проєкта) також виникає необхідність в аналізі зовнішньої, по відношенню до FPGA, частини друкованої плати. Адже, після створення моделі HDL опису, необхідно її протестувати. Для цього, необхідно отримати результати роботи HDL опису та його моделі в однакових умовах. Щоб це зробити треба подати на входи FPGA тестові дані. Проте, на початку треба визначити формат тестових даних, протокол передачі, схему з'єднання фізичного інтерфейсу з входами/виходами FPGA. Отже, виникає потреба в аналізі всієї або частини друкованої плати, щоб протестувати роботу моделі HDL опису в процесі застосування різних фізичних інтерфейсів. Тому, реінжиніринг друкованої плати є підзадачею реінжинірингу FPGA проєкта, результатом чого є список з'єднань (англ. netlist), який буде використано на завершальному етапі аналізу HDL опису – тестуванні моделі HDL опису.

3.1 Огляд друкованої плати

Обрана друкована плата призначена для тестування комунікаційних вузлів RS-485, RS-232 та Ethernet. Також, плата має двохрядний роз'єм J4 (рис. 3.1) з 14-ма контактами. Шість контактів роз'єма призначені для прямого з'єднання ззовні з входами мікроконтролера (контакти роз'єма 3, 4, 7, 8, 11, 12). Три контакти роз'єма призначені для з'єднання ззовні з виходами мікроконтролера через гальванічну розв'язку (контакти роз'єма 1, 5, 9). Плата, також має двохрядний роз'єм J19 з 10-ма контактами для прямого з'єднання ззовні з входами/виходами мікроконтролера (контакти роз'єма 2, 4, 6, 7, 8, 9).

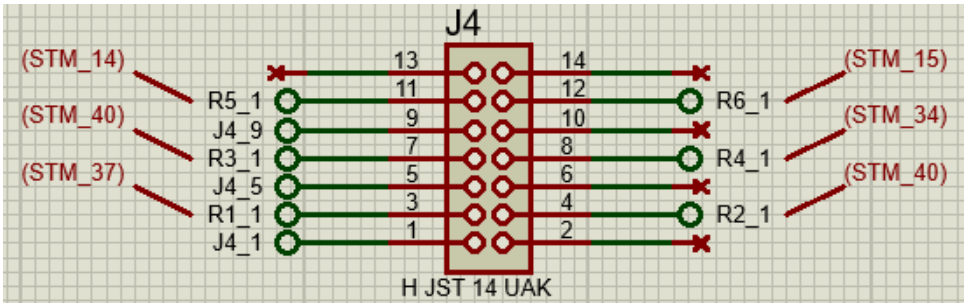


Рисунок 3.1 – Двохрядний роз’єм J4 з 14-ма контактами

Список з’єднань – це, в найпростішому випадку, список з’єднань між контактами електричних компонентів друкованої плати (інтегральні схеми, роз’єми, резистори та ін.). Такий список з’єднань може мати наступний вигляд:

– «U1:1 – U2:3»;

– «U1:2 – U2:6»;

– «U2:1 – U3:1».

Позначення U – це умовне графічне позначення (УГП) певного компоненту (рис. 3.2). Для прискорення складання списку з’єднань його можна згенерувати на основі електричної діаграми (рисунок в додатку В.1). Електрична діаграма – графічне зображення електричної схеми, що складається з УГП електричних компонентів (у вигляді прямокутників з назвою компоненту та його портами, його номером та назвою) та з’єднань між ними.

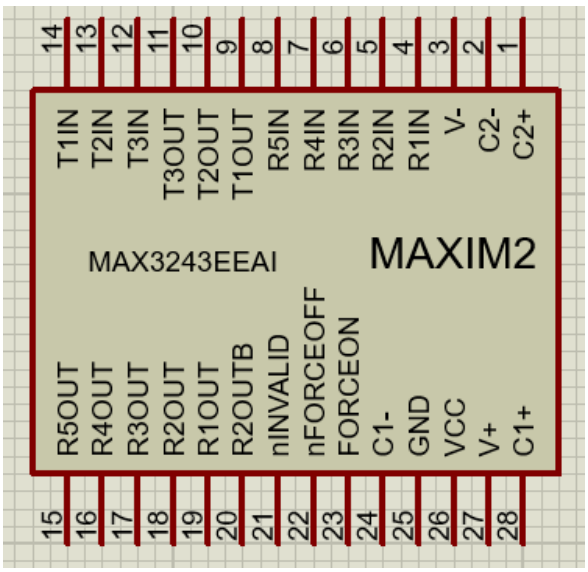


Рисунок 3.2 – УГП мікросхеми

3.2 Створення списку з’єднань

3.2.1 Визначення зв’язків між компонентами

Електричну діаграму побудовано в програмному забезпечені Proteus 8.13 SP0 (Build 31525). При побудові електричної діаграми, використано наступні інструменти: мультиметер, збільшувальне скло та ліхтарик. Мультиметер необхідний для підтвердження або визначення з’єднань між компонентами. Це відбувається в режимі виявлення обриву ланцюга. Якщо обрив між контактами компонентів відсутній, то звучить сигнал. Це означає, що з’єднання існує. Для визначення всіх з’єднань необхідно прослухати кожний контакт з кожним іншим контактом. Адже, один контакт може бути з’єднаний з декількома іншими. В той же час, ні всі доріжки друкованої плати є видимими, бо всі сучасні друковані плати мають більше ніж два шари доріжок та мають, по меншій мірі, один шар з’єднання всередині друкованої плати. Проте, в даному випадку, відтворено лише з’єднання з видимих шарів друкованої плати: шар з переднього (рис. 3.3) та зворотнього (рис. 3.4) боків. Часто, відтворення одного з’єднання відбувається таким чином. Спочатку, очима відстежуємо доріжку. Потім, перевіряємо мультиметром правильність спостереження.

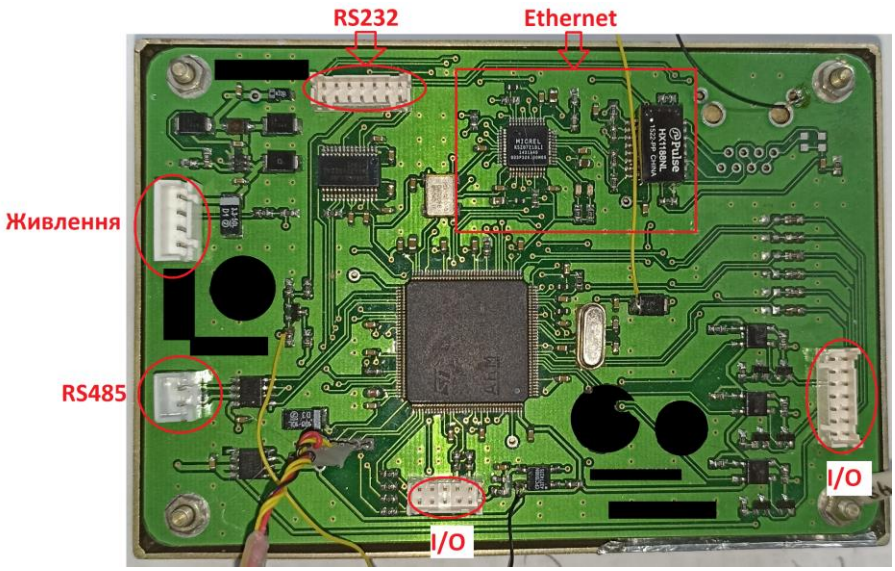


Рисунок 3.3 – Передній бік друкованої плати

Переднім вважається бік, на якому розташована більшість компонентів. В даному випадку, всі компоненти розташовані на передньому боці. Часто доріжка починається на одному боці та продовжується на іншому. Тоді, існує сквозний отвір (рис. 3.5) в місці переходу доріжки на інший бік.

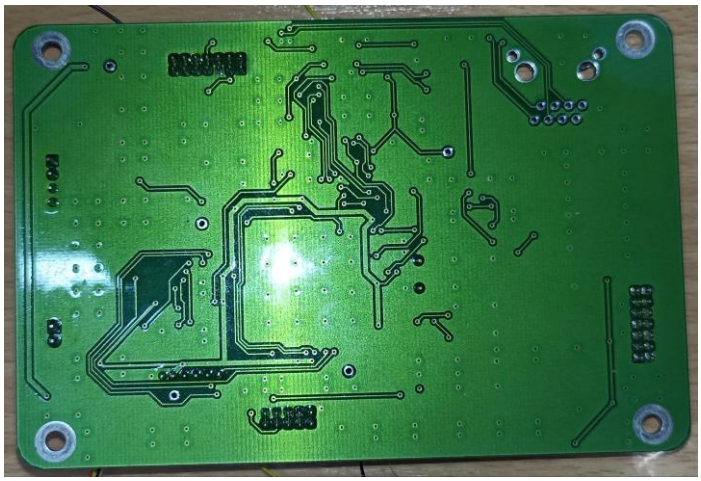


Рисунок 3.4 – Зворотній бік друкованої плати

Проте, без мультиметра складно співставити отвір на одному та іншому боках, бо таких отворів може бути багато і вони можуть близько розташовуватися один від одного. Тоді, використовується мультиметер. Робиться приблизне спостереження, ціллю якого є визначити які контакти або скрізні отвори з’єднані доріжкою на зворотньому боці. Потім, виконується перевірка гіпотези шляхом прослуховування потенційних кінців з’єднання на передньому боці.

Рисунок 3.5 – Друкована плата з позначеннями

3.2.2 Визначення номіналів резисторів

Також, мультиметер можна використовувати для приблизного визначення номіналів резисторів. Номінали є приблизними, бо резистор під'єднаний до електричного ланцюга. Більш надійним є спосіб визначення номіналів згідно з маркуванням присутнім на резисторах. Проте, необхідно точно визначити, яке маркування використовується. Іноді, поверхня з маркування може бути пошкоджена. А також, може бути не зрозуміло з якого боку дивитися на резистор. Адже, він може бути перевернутий. Отже, необхідно порівняти виміряне значення з табличним. Якщо значення приблизно (допустимо відхилення ± 10%) рівні, то маркування визначено вірно. Якщо значення сильно відрізняються, значить маркування визначено не вірно. Нехай, на мультиметрі маємо номінал 326 Ом, а згідно з маркуванням «3 Digit EIA» та значенням маркування «331» отримуємо 330 Ом. Отже, маркування визначено вірно.

3.2.3 Пошук даташитів

Ліхтарик та збільшувальне скло необхідне для визначення написів (маркувань або назв компонентів) на корпусах та визначення першого контакту мікросхеми. Маркування необхідне для визначення номіналу компонента або пошуку даташиту. При цьому, треба звертати увагу, що латинська літера «Q» може бути схожою на цифру нуль «0», якщо нижня риска літери затерлась. Помилка в одну літеру може унеможливити пошук даташиту навіть зі штучним інтелектом. Треба зважати на те, що на малих за розміром корпусах зазвичай вказується маркування замість назви.

Проте, не рідко трапляються випадки, коли точне маркування визначити не можна через наявні пошкодження на корпусі. В таких випадках, можна звертатися до сервісів штучного інтелекту (ШІ) або по допомогу на спеціалізованих форумах. ШІ жодного разу не надав очікувану допомогу. Він формує посилання на даташити, які не мають відношення до цільового компоненту. Отже, рекомендується звертатися по допомогу на спеціалізованих форумах. У всіх випадках була отримана очікувана відповідь. Рекомендується використовувати форум «edaboard». Всі відповіді були отримані за допомогою нього [10, 11, 12, 13, 14]. Також, можна

спробувати використати схожий форум «eevblog». Проте, не рекомендується використовувати форум «electronics stackexchange». Адміністрація даного форуму скаржиться на короткий опис запитань. Наприклад, якщо тіло питання складається з питання із заголовку, маркування та фотографії корпусу, то таке питання може бути видалене адміністрацією. Проте, часто цієї інформації достатньо для пошуку даташиту. Іноді може бути потрібною фотографія, яка включає прилежні компоненти на друкованій платі.

3.2.4 Визначення першого контакту на корпусі

Перший контакт на корпусі можна визначити декількома способами. Зазвичай на корпусі є позначення, що вказує на кут, в якому знаходиться перший контакт. На інтегральних схемах (далі – ІС або мікросхема) позначення має вигляд видавленої точки (рис. 3.6, а) або надрукованої білої точки (рис. 3.6, б) в одному з кутів корпусу.

Якщо контакти розташовані лише з двох боків корпусу, то проблем з визначенням першого контакту не має. Проте, якщо контакти розташовані з чотирьох боків (рис. 3.7), то цієї інформації не достатньо для визначення першого контакту, бо першим може бути контакт з одного чи іншого боку в тому ж куті.

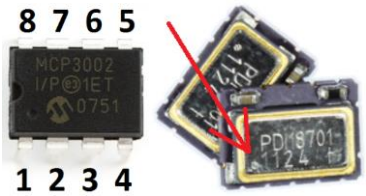


Рисунок 3.6 – Позначення першого контакту на корпусі

Для цього, необхідно знайти розпіновку мікросхеми в їх даташиті. Якщо корпус перевернути таким чином, щоб точка була з лівого верхнього кута корпусу, то перший контакт буде знаходитися з лівого боку мікросхеми.

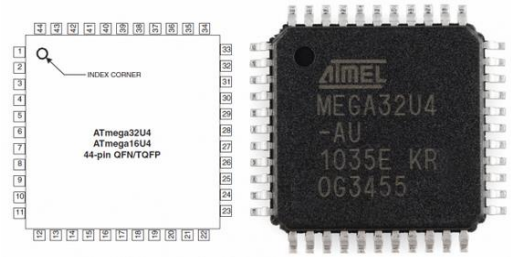


Рисунок 3.7 – Визначення першого контакту для корпусів

Інший спосіб, це звернути увагу на місце монтажу контакту мікросхеми. Якщо перший контакт під'єднаний до корпусу (англ. chassis), то навколо контакту може бути темно зелена квадратна рамка. Якщо перший контакт під'єднаний до доріжки, то від доріжки може відходити додаткова доріжка, що закінчується тупиком.

Поляризовані конденсатори в прямокутному корпусі або діоди мають надруковану білу лінію з боку катоду. Нумерація контактів мікросхеми, зазвичай, відбувається по колу проти годинникової стрілки від першого контакту. Нумерація роз'ємів, зазвичай, відбувається в шахматному порядку. В роз'ємах з двох рядів контактів, навпроти першого контакту – другий, над першим контактом – третій, навпроти третього – четвертий і т.д. Перший контакт на роз'ємах визначається за допомогою позначки на корпусі роз'єма. В даному випадку, позначка має вигляд видавленого трикутника.

3.2.5 Вивчення позначень на друкованій платі

Згідно з рис, показано сквозний отвір, який необхідний для з'єднання двох доріжок на передньому та зворотньому боках друкованої плати. Показано керамічний конденсатор, який одним боком під'єднаний до корпусу – один бік змонтований на світло-зелену область, що є корпусом. Один з контактів мікросхеми також під'єднана до корпусу, але обведена темно-зеленою квадратною рамкою, що є позначкою першого контакту. Другий контакт мікросхеми також під'єднана до корпусу, але не обведена також рамкою. Останній контакт мікросхеми не під'єднана (англ. not connected або NC) до жодного з компонентів та корпусу. Різниця монтажу першого та останнього контакту полягає у тому, що перший

контакт окреслений рамкою з виразними гострими кутами, а також є невелика світло-зелена область біля контакту.

Іноді, не під'єднаний контакт можна сплутати з під'єднаною, якщо доріжка до контакту проходить під корпусом мікросхеми як на рис. 3.8 під номером 1. Щоб перевірити це, необхідно прослухати відомий кінець доріжки та всі контакти мікросхеми. На рис. 3.8 під номером 2 наведено ще один приклад під'єднання контакту до корпусу.

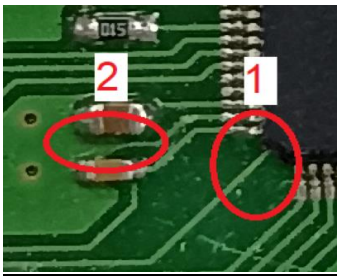


Рисунок 3.8 – Доріжка під'єднується до контакту мікросхеми під її корпусом та вигляд іншої доріжки, під'єднаної до корпусу друкованої плати

На рис. 3.9 наведено вигляд тестової точки, що використовується для тестування напруги на певних вузлах друкованої плати. Для тестування необхідно, за допомогою мультиметра в режимі вимірювання напруги, прикласти червоний його пробник до тестової точки, а чорний пробник до корпусу друкованої плати. У якості корпусу можна використовувати місця монтажу друкованої плати по її краях. Ці місця є оголеними від лаку, навідмін від інших ділянок плати, де необхідно цей лак проколювати пробником, щоб дістатися до корпусу. Це можна зробити, якщо в якості пробника використовувати голки.

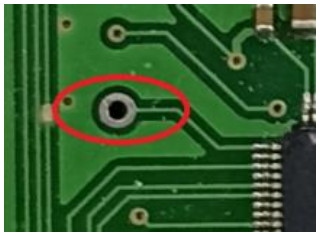


Рисунок 3.9 – Тестова точка для перевірки напруги

ВИСНОВКИ

Проаналізовано попередні наукові роботи [6, 7, 8], пов’язані з реінжинірингом FPGA проєкта. Проте, виявлено, що всі попередні дослідження сконцентровані на реінжинірингу потоку бітів, результатом чого є HDL опис. Натомість, в даній роботі досліджується реінжиніринг HDL опису, результатом чого є Simulink модель. Дана задача є актулальною у випадку необхідності вдосконалення наявного рішення. Проте, якщо при цьому відсутня документація та фаховий персонал для проведення вдосконалення.

Проаналізовано наявні методи реінжинірингу, а саме інструмент автогенерації Simulink-моделі на основі HDL опису (importhdl). Натомість, даний підхід не дозволяє створити наочну модель HDL опису, легку для сприйняття людиною. Тому, ручний підхід створення моделі є найбільш доцільним для створення наочної моделі. Також, інструмент автогенерації висуває вимоги до вихідного HDL опису, найкритичнішими з яких є наявність не більше одного синхросигналу в описі, заборонено використовувати перелік HDL шаблонів та заборонено використовувати стандартні модулі логічних вентилів.

Розроблено метод для ручного реінжинірингу FPGA проєкта:

– відтворення тракту обробки даних HDL опису;

– створення блок-діаграми HDL опису;

– створення текстового документа з описом блоків блок-діаграми;

– створення MATLAB-моделей на основі HDL модулів;

– створення Simulink-моделі на основі MATLAB-моделей.

Реалізовано Simulink-модель, згідно з запропонованим методом. Розроблено одинадцять тестів для тестування моделі. Через вимоги компілятора Quartus, опис HDL складніший та довший ніж MATLAB код Simulink моделі. Тому, модель наочно відображає весь функціонал HDL опису без надлишкової структурної складності.

Проведено реінжиніринг друкованої плати, в результаті чого отримано список з’єднань, згенерований в Proteus на основі електричної діаграми друкованої плати, відтвореної в Proteus, та сформовано рекомендації з процесу реінжинірингу. Успішна генерація списку з’єднань означає, що електричну діаграму створено вірно. Створена діаграма може бути використана на завершальному етапі реінжинірингу FPGA проєкта – тестуванні моделі. Тому, як для тестування моделі необхідні тестові дані, які можна отримати лише з’ясувавши яким чином оброблюються дані перед надходження до FPGA проєкта.

Таким чином, мета роботи досягнена, усі поставлені задачі виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Military, Aerospace & Government. URL: https://www.altera.com/fpga-solutions/ military-aerospace-government (date of access: 01.10.2025).

2. Zhang, T. et al. "A Comprehensive FPGA Reverse Engineering Tool-Chain: From Bitstream to RTL code". IEEE, 2019. URL: https://ieeexplore.ieee.org/document/8653869 (date of access: 02.10.2025).

3. Importhdl. URL: https://www.mathworks.com/help/hdlcoder/ref/importhdl.html (date of access: 04.10.2025).

4. Supported Verilog Constructs for HDL Import. URL: https://www.mathworks.com/help/hdlcoder/ug/supported-verilog-constructs-for-hdl-import.html (date of access: 06.10.2025).

5. Verilog Dataflow Modeling with HDL Import. URL: https://www.mathworks.com/help/hdlcoder/ug/verilog-dataflow-modeling-with-hdl-import.html (date of access: 06.10.2025).

6. Kashani, S. et al. "Bitfiltrator: A general approach for reverse-engineering Xilinx bitstream formats". IEEE, 2022. URL: https://ieeexplore.ieee.org/document/10035199/ (date of access: 11.10.2025).

7. Stowasser, H. "An Abstract Approach To FPGA LUT Bitstream Reverse Engineering". University of Cincinnati, 2022. URL: https://etd.ohiolink.edu/acprod/odb_etd/ws/send_file/send (date of access: 12.10.2025).

8. Mannhee, C. et al. "Automated Reverse Engineering Tools for FPGA Bitstream Extraction and Logic Estimation". ISOCC, 2022. URL: https://pure.korea.ac.kr/en/publications/automated-reverse-engineering-tools-for-fpga-bitstream-extraction/ (date of access: 14.10.2025).

9. Altera Corporation. "Altera Corporation: Cyclone. Cyclone Device Handbook, Volume 1". Altera Corporation, 2008. URL: https://cdrdv2.intel.com/v1/dl/getContent/654210 (date of access: 12.11.2025).

10. STM32F415xx STM32F417xx. URL: https://www.st.com/resource/en/datasheet/stm32f415rg.pdf (date of access: 13.11.2025).

11. MAXIM MAX3485. URL: https://www.analog.com/media/cn/technical-documentation/data-sheets/1079.pdf (date of access: 16.11.2025).

12. MAXIM MAX3243EEAI. URL: https://www.mouser.com/datasheet/2/609/MAX3221_MAX3243-3112066.pdf (date of access: 20.11.2025).

13. MICREL KS8721BL/SL. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/ks8721bl-sl.pdf (date of access: 26.11.2025).

14. 733Q Low-Dropout Voltage Regulators With Integrated Delayed Reset Function datasheet (Rev. F). URL: https://www.ti.com/lit/ds/symlink/tps73.pdf (date of access: 02.12.2025).

ДОДАТОК А

HDL опис для створення моделі

Лістинг А.1 – Verilog HDL опис, на основі якого створено Simulink модель

```
`timescale 1ns / 1ps
module spec_dig_dev(d1, d2, reset_divider_extern,
reset_spec_dev_extern, clk, S1, ready);
input [0:3] d1, d2;
input reset_divider_extern, reset_spec_dev_extern, clk;
output [0:3] S1;
output reg ready;

reg [0:3] A, B, S1, S2, S3;
reg [3:0] Y;
reg reset_divider;
reg to_first_state;
reg rst_to_first_state;
reg [0:1] automat_state;
reg [0:1] automat_state2;
reg rst_spec_dev_flag;
reg rst_rst_spec_dev_flag;
reg [0:3] tmp_d1_filt, tmp_d2_filt;
reg tmp_rst_to_first_state;
reg tmp_reset_spec_dev_extern;
reg tmp_rst_rst_spec_dev_flag;

wire clk_divided;
wire [0:3] d1_filt, d2_filt;

debounce_filter #(
    .CNTR_WIDTH(10)
) filter_d1 (
    .clock(clk),
```

```
    .reset(reset_spec_dev_extern),
    .button(d1),
    .pressed(d1_filt)
);

debounce_filter #(
    .CNTR_WIDTH(10)
) filter_d2 (
    .clock(clk),
    .reset(reset_spec_dev_extern),
    .button(d2),
    .pressed(d2_filt)
);

always @ (clk, reset_divider_extern, reset_spec_dev_extern, ready)
begin
        case({ready,
            reset_divider_extern,
            reset_spec_dev_extern})
        3'b000:  begin reset_divider <= 0; end
        3'b001:  begin reset_divider <= 0; end
        3'b010:  begin reset_divider <= 1; end
        3'b011:  begin reset_divider <= 0; end
        3'b100:  begin
            if(to_first_state) begin
                reset_divider <= 0;
            end
            else begin
                reset_divider <= 1;
            end
        end
        3'b101:  begin reset_divider <= 0; end
        3'b110:  begin reset_divider <= 1; end
        3'b111:  begin reset_divider <= 0; end
        endcase
```

```
end

always @ (posedge clk)
begin
        if(tmp_d1_filt != d1_filt | tmp_d2_filt != d2_filt |
tmp_rst_to_first_state != rst_to_first_state) begin
                    case(automat_state)
                2'b00:  begin
                                    to_first_state <= 0;
                                    automat_state <= 2'b11;
                                end
                    2'b10:  begin
                                    to_first_state <= 0;
                                    automat_state <= 2'b01;
                                end
                    2'b01:  begin
                                    to_first_state <= 0;
                                    automat_state <= 2'b11;
                                end
                    2'b11:  begin
                                    if(rst_spec_dev_flag == 1'b0)
                                         begin
                                              to_first_state <= 1;
                                              automat_state <= 2'b10;
                                         end
                                    else
                                         automat_state <= 2'b01;
                                end
                endcase
        end

          tmp_d1_filt <= d1_filt;
          tmp_d2_filt <= d2_filt;
          tmp_rst_to_first_state <= rst_to_first_state;
end
```

```
always @ (posedge clk)
begin
     if(tmp_reset_spec_dev_extern != reset_spec_dev_extern |
tmp_rst_rst_spec_dev_flag != rst_rst_spec_dev_flag)
     begin
          case(automat_state2)
               2'b00:  begin
                    rst_spec_dev_flag <= 0;
                    automat_state2 <= 2'b01;
               end
               2'b01:  begin
                    rst_spec_dev_flag <= 0;
                    automat_state2 <= 2'b11;
               end
               2'b11:  begin
                    rst_spec_dev_flag <= 1;
                    automat_state2 <= 2'b01;
               end
          endcase
     end

     tmp_reset_spec_dev_extern <= reset_spec_dev_extern;
     tmp_rst_rst_spec_dev_flag <= rst_rst_spec_dev_flag;
end

divider divider1(clk,reset_divider,clk_divided);

always @ (posedge clk_divided)begin
if (!reset_spec_dev_extern)
 case (Y)
0: begin
 A <= d1_filt;
 B <= d2_filt;
 S1 <= 0;
```

```
 S2 <= 0;
 S3 <= 0;
 ready <= 0;
 rst_to_first_state <= 1;
 Y <= 1;
 end
1: begin
     rst_to_first_state <= 0;
     rst_rst_spec_dev_flag <= 0;
     if (to_first_state) begin
        Y <= 0;
     end
     else begin
         S2 <= A << 2;
         S1 <= A^B;
         if(A<=9) Y <= 2;
         else Y <= 10;
     end
 end
2: begin
    if (to_first_state) begin
        Y <= 0;
     end
     else begin
    S3 <= B >> 3;
    Y <= 3;
    end
  end
3: begin
    if (to_first_state) begin
        Y <= 0;
     end
     else begin
         S3 <= S1 << 3;
         S2 <= S2 + S3;
```

```
            Y <= 4;
        end
    end
4: begin
    if (to_first_state) begin
        Y <= 0;
    end
    else begin
        S1 <= B << 2;
        S3 <= S3 + S1;
        Y <= 5;
    end
end
5: begin
    if (to_first_state) begin
        Y <= 0;
    end
    else begin
        S3 <= S3 + S2;
        Y <= 6;
    end
  end
6: begin
    if (to_first_state) begin
        Y <= 0;
    end
    else begin
        S1 <= S1 + B;
        Y <= 7;
    end
  end
7: begin
    if (to_first_state) begin
        Y <= 0;
    end
```

```
     else begin
        S1 <= A + S1;
        Y <= 8;
      end
    end
8: begin
        if (to_first_state) begin
        Y <= 0;
     end
     else begin
        S1 <= ~S1;
        Y <= 9;
     end
    end
9: begin
    if (to_first_state) begin
        Y <= 0;
     end
     else begin
        S1 <= S1^S3;
        ready <= 1;
        Y <= 0;
     end
    end
10: begin
    if (to_first_state) begin
        Y <= 0;
     end
     else begin
         B <= B << 1;
         A <= A + B;
         Y <= 11;
     end
 end
11: begin
```

```
    if (to_first_state) begin
        Y <= 0;
     end
     else begin
         A <= A >> 3;
         B <= B & S2;
         Y <= 12;
     end
 end
12: begin
    if (to_first_state) begin
        Y <= 0;
     end
     else begin
        B <= ~B;
        Y <= 13;
     end
    end
 13: begin
    if (to_first_state) begin
        Y <= 0;
     end
     else begin
        S1 <= B | S1;
        Y <= 14;
      end
    end
 14: begin
    if (to_first_state) begin
        Y <= 0;
     end
     else begin
        S1 <= A + S1;
        ready <= 1;
        Y <= 0;
```

```
     end
    end
endcase
else
begin
 A <=0; B <= 0; S1 <= 0;
 S2 <= 0; S3 <= 0;
 Y <= 0; ready <= 0;
 rst_rst_spec_dev_flag <= 1;
end
end

endmodule

`timescale 1ns / 1ps
module divider(
    input clk,
    input reset,
    output reg clk_divided
    );

    reg [31:0] counter;

    always @ (posedge clk)
    begin
        if(!reset) begin
            counter <= counter + 1;
            if(counter == 25000000) begin
                clk_divided <= ~clk_divided;
                counter <= 0;
            end
        end
        else begin
            counter <= 0;
            clk_divided <= 0;
```

```
        end
    end

/*
 * Debounce Filter Module
 */

module debounce_filter #(
    parameter CNTR_WIDTH = 10,
    parameter D_WIDTH = 4
)(
    input      clock,
    input      reset,
    input      [0:D_WIDTH-1] button,
    output reg [0:D_WIDTH-1] pressed
);

// 2-Flop synchroniser to transfer button signal to clock
// domain and avoid concurrency issues.
reg [D_WIDTH:0] buttonSync;
always @ (posedge clock or posedge reset) begin
    if (reset) begin
        buttonSync <= 2'b0;
    end else begin
        buttonSync <= {buttonSync[0], button};
    end
end

// Debounce Filter logic
// Requires state be stable for 2**CNTR_WIDTH cycles before
// output state is changed.
reg [0:D_WIDTH-1] stable;
reg [CNTR_WIDTH-1:0] filter;

always @ (posedge clock or posedge reset) begin
```

```
    if (reset) begin
        filter <= 1'b0;
    end else if (stable) begin
        // Reset filter timer while stable
        filter <= 1'b0;
    end else begin
        // Otherwise let it run.
        filter <= filter + 1'b1;
    end
end

always @ (posedge clock or posedge reset) begin
    if (reset) begin
        stable      <= 'h0;
        pressed     <= 'h0;
    end else if (stable) begin
        // Button is stable, check if it has changed
        if (buttonSync != pressed) begin
            // An edge is detected, no longer stable.
            stable  <= 'h0;
        end
    end else begin
        // Button might be changing, or it might be noise
        if (buttonSync == pressed) begin
            // Returned to its old state, so assume noise.
            stable  <= 'hF;
        end else if (filter == {(CNTR_WIDTH){1'b1}}) begin
            // Counter reached top, stable for long enough to
            // update the button state.
            pressed <= buttonSync;
            stable  <= 'hF;
        end
    end
end
endmodule
```

ДОДАТОК Б

Модель та тести

Лістинг Б.1 – Фінальна версія Simulink блоку «mainAutomat» або «spec_dig_dev_model»

```
function [res,ready_out] = mainAutomat(d1, d2,
reset_spec_dev_extern, reset_divider, to_first_state)
persistent bit4 wrap Y A B S1 S2 S3 ready buf_to_first_state
if isempty(bit4)
   bit4 = numerictype(0,4,0);  % unsigned, 4-bit
   wrap = fimath('OverflowAction','Wrap', 'RoundingMethod','Zero');
   Y = 1;
   A = fi(0, bit4, wrap);
   B = fi(0, bit4, wrap);
   S1 = fi(0, bit4, wrap);
   S2 = fi(0, bit4, wrap);
   S3 = fi(0, bit4, wrap);
   ready = 0;
   buf_to_first_state = zeros(1, 2);
end
buf_to_first_state(2) = buf_to_first_state(1);
buf_to_first_state(1) = to_first_state;
mask = fi(15, bit4, wrap);
if reset_divider == 0 %дозвіл тактування
   if reset_spec_dev_extern == 0
       switch Y
           case 1
               A = fi(d1, bit4, wrap);
               B = fi(d2, bit4, wrap);
               S1 = fi(0, bit4, wrap);
               S2 = fi(0, bit4, wrap);
               S3 = fi(0, bit4, wrap);
               ready = 0;
```

```
    Y = 2;
case 2
         if buf_to_first_state(2) ~= 1
             S2 = fi(bitsll(A,2), bit4, wrap);
             S1 = fi(bitxor(A,B), bit4, wrap);
             if A<=9
                 Y = 3;
             else
                 Y = 11;
             end
         else
            Y = 1;
         end
case 3
         if buf_to_first_state(2) ~= 1
            S3 = fi(bitsrl(B,3), bit4, wrap);
            Y = 4;
         else
            Y = 1;
         end
case 4
         if buf_to_first_state(2) ~= 1
             S2 = fi(S2 + S3, bit4, wrap);
             S3 = fi(bitsll(S1,3), bit4, wrap);
             Y = 5;
         else
            Y = 1;
         end
case 5
         if buf_to_first_state(2) ~= 1
             S3 = fi(S3 + S1, bit4, wrap);
             S1 = fi(bitsll(B,2), bit4, wrap);
             Y = 6;
         else
            Y = 1;
```

```
         end
case 6
        if buf_to_first_state(2) ~= 1
            S3 = fi(S3 + S2, bit4, wrap);
            Y = 7;
        else
            Y = 1;
        end
case 7
        if buf_to_first_state(2) ~= 1
            S1 = fi(S1 + B, bit4, wrap);
            Y = 8;
        else
            Y = 1;
        end
case 8
        if buf_to_first_state(2) ~= 1
            S1 = fi(S1 + A, bit4, wrap);
            Y = 9;
        else
            Y = 1;
        end
case 9
        if buf_to_first_state(2) ~= 1
            S1 = fi(bitxor(mask, S1), bit4, wrap);
            Y = 10;
        else
            Y = 1;
        end
case 10
        if buf_to_first_state(2) ~= 1
            S1 = fi(bitxor(S1,S3), bit4, wrap);
            ready = 1;
            Y = 1;
        else
```

```
            Y = 1;
        end
case 11
         if buf_to_first_state(2) ~= 1
             A = fi(A + B, bit4, wrap);
             B = fi(bitsll(B,1), bit4, wrap);
             Y = 12;
         else
            Y = 1;
         end
case 12
         if buf_to_first_state(2) ~= 1
             A = fi(bitsrl(A,3), bit4, wrap);
             B = fi(bitand(B,S2), bit4, wrap);
             Y = 13;
         else
            Y = 1;
         end
case 13
        if buf_to_first_state(2) ~= 1
            B = fi(bitxor(mask, B), bit4, wrap);
            Y = 14;
        else
            Y = 1;
        end
case 14
        if buf_to_first_state(2) ~= 1
            S1 = fi(bitor(S1, B), bit4, wrap);
            Y = 15;
        else
            Y = 1;
        end
case 15
        if buf_to_first_state(2) ~= 1
            S1 = fi(S1 + A, bit4, wrap);
```

```
                    ready = 1;
                    Y = 1;
                else
                    Y = 1;
                end
        end
    else
            A = fi(0, bit4, wrap);
            B = fi(0, bit4, wrap);
            S1 = fi(0, bit4, wrap);
            S2 =  fi(0, bit4, wrap);
            S3 = fi(0, bit4, wrap);
            Y = 1;
            ready = 0;
    end
end
res = S1;
ready_out = ready;
```

Лістинг Б.2 – Simulink блок «to_first_state_gen»

```
function to_first_state = to_first_state_gen(d1, d2,
rst_divider_extern)
persistent flag buf_d1 buf_d2 firstClk
if isempty(flag)
    flag = 0;
    buf_d1 = zeros(1, 2);
    buf_d2 = zeros(1, 2);
    firstClk = 0;
end
firstClk = firstClk + 1;
buf_d1(2) = buf_d1(1);
buf_d1(1) = d1;
buf_d2(2) = buf_d2(1);
buf_d2(1) = d2;
if(firstClk ~= 1)
```

```
    if buf_d1(1) ~= buf_d1(2) || buf_d2(1) ~= buf_d2(2)
        flag = 1;
    else
        if rst_divider_extern ~= 1
            flag = 0;
        end
    end
end
to_first_state = flag;
```

Лістинг Б.3 – Simulink блок «inner_ena»

```
function reset_divider = inner_ena(reset_divider_extern, ...
    reset_spec_dev_extern, ready, to_fisrt_state)
persistent rst buf_to_first_state
if isempty(rst)
    rst = 0;
    buf_to_first_state = zeros(1, 2);
end
buf_to_first_state(2) = buf_to_first_state(1);
buf_to_first_state(1) = to_fisrt_state;
if ready == 0 && reset_divider_extern == 0 && reset_spec_dev_extern
== 0
    rst = 0;
elseif ready == 0 && reset_divider_extern == 0 &&
reset_spec_dev_extern == 1
    rst = 0;
elseif ready == 0 && reset_divider_extern == 1 &&
reset_spec_dev_extern == 0
    rst = 1;
elseif ready == 0 && reset_divider_extern == 1 &&
reset_spec_dev_extern == 1
    rst = 0;
elseif ready == 1 && reset_divider_extern == 0 &&
reset_spec_dev_extern == 0
    if buf_to_first_state(2) == 1
```

```
        rst = 0;
    else
        rst = 1;
    end
elseif ready == 1 && reset_divider_extern == 0 &&
reset_spec_dev_extern == 1
    rst = 0;
elseif ready == 1 && reset_divider_extern == 1 &&
reset_spec_dev_extern == 0
    rst = 1;
elseif ready == 1 && reset_divider_extern == 1 &&
reset_spec_dev_extern == 1
    rst = 0;
end
reset_divider = rst;
```

Лістинг Б.4 – Simulink блок «input_gen», тест 1, перевірка правої та лівої гілок алгоритму (без сигналу готовності ready в моделі)

```
function [A, B] = input_gen(t)
    persistent A_value
    persistent B_value
    persistent last_time

    if isempty(B_value)
        B_value = 8;
    end
    update_interval = 2;

    if isempty(A_value)
        A_value = 6;
        last_time = t;
    else
        if t - last_time >= update_interval
            A_value = 14;
        end
```

```
    end
    A = A_value;
    B = B_value;
end
```

Лістинг Б.5 – Simulink блок «input_gen», тест 2, перевірка сигналу скидання reset_spec_dev_extern під час виконання алгоритму

```
function [A, B, reset_spec_dev_extern] = input_gen(t)
    persistent A_value
    persistent B_value
    persistent rst_value
    persistent last_time

    rst_set_interval  = 1;
    rst_reset_interval  = 1;
    A_update_interval = 5;

    if isempty(B_value)
        B_value = 8;
    end
    if isempty(rst_value)
        rst_value = 0;
        last_time = t;
    else
        if t - last_time >= rst_set_interval
            rst_value = 1;
        end
        if t - last_time >= rst_reset_interval
            rst_value = 0;
        end
    end

    if isempty(A_value)
        A_value = 6;
        last_time = t;
```

```
    else
        if t - last_time >= A_update_interval
            A_value = 14;
        end
    end
    A = A_value;
    B = B_value;
    reset_spec_dev_extern  = rst_value;
end
```

Лістинг Б.6 – Simulink блок «input_gen», тест 3, перевірка сигналу скидання reset_spec_dev_extern після виконання алгоритму (при ready = 1)

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
    persistent A_value
    persistent B_value
    persistent rst_value
    persistent ena_value
    persistent last_time

    rst_set_interval  = 3;
    rst_reset_interval  = 4;
    if isempty(ena_value)
        ena_value = 0;
    end

    if isempty(B_value)
        B_value = 8;
    end
    if isempty(rst_value)
        rst_value = 0;
        last_time = t;
    else
        if t - last_time >= rst_set_interval
            rst_value = 1;
```

```
        end
        if t - last_time >= rst_reset_interval
            rst_value = 0;
        end
    end

    if isempty(A_value)
        A_value = 6;
    end
    A = A_value;
    B = B_value;
    reset_spec_dev_extern  = rst_value;
    reset_divider_extern = ena_value;
end
```

Лістинг Б.7 – Simulink блок «input_gen», тест 4, перевірка перезапуску алгоритму при ready = 1 в разі зміни операнда

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
    persistent A_value
    persistent B_value
    persistent rst_value
    persistent ena_value
    persistent last_time

    rst_set_interval  = 1;
    rst_reset_interval  = 1;
    A_update_interval = 5;
    if isempty(ena_value)
        ena_value = 0;
    end

    if isempty(B_value)
        B_value = 8;
    end
```

```
    if isempty(rst_value)
        rst_value = 0;
        last_time = t;
    else
        if t - last_time >= rst_set_interval
            rst_value = 1;
        end
        if t - last_time >= rst_reset_interval
            rst_value = 0;
        end
    end

    if isempty(A_value)
        A_value = 6;
        last_time = t;
    else
        if t - last_time >= A_update_interval
            A_value = 14;
        end
    end
    A = A_value;
    B = B_value;
    reset_spec_dev_extern  = rst_value;
    reset_divider_extern = ena_value;
end
```

Лістинг Б.8 – Simulink блок «input_gen», тест 5, перевірка перезапуску алгоритму під час виконання алгоритму в разі зміни операнда

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
    persistent A_value
    persistent B_value
    persistent rst_value
    persistent ena_value
    persistent last_time
```

```
    rst_set_interval  = 1;
    rst_reset_interval  = 1;
    A_update_interval = 1;
    if isempty(ena_value)
        ena_value = 0;
    end

    if isempty(B_value)
        B_value = 8;
    end
    if isempty(rst_value)
        rst_value = 0;
        last_time = t;
    else
        if t - last_time >= rst_set_interval
            rst_value = 1;
        end
        if t - last_time >= rst_reset_interval
            rst_value = 0;
        end
    end
    if isempty(A_value)
        A_value = 6;
        last_time = t;
    else
        if t - last_time >= A_update_interval
            A_value = 14;
        end
    end
    A = A_value;
    B = B_value;
    reset_spec_dev_extern  = rst_value;
    reset_divider_extern = ena_value;
end
```

Лістинг Б.9 – Simulink блок «input_gen», тест 6, перевірка сигналу дозволу reset_divider_extern

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
   persistent A_value
   persistent B_value
   persistent rst_value
   persistent ena_value
   persistent last_time

   rst_set_interval  = 1;
   rst_reset_interval  = 1;
   ena_set_interval  = 1;
   ena_reset_interval  = 2;
   if isempty(ena_value)
       ena_value = 0;
       last_time = t;
   else
       if t - last_time >= ena_set_interval
           ena_value = 1;
       end
       if t - last_time >= ena_reset_interval
           ena_value = 0;
       end
   end

   if isempty(B_value)
       B_value = 8;
   end
   if isempty(rst_value)
       rst_value = 0;
       last_time = t;
   else
       if t - last_time >= rst_set_interval
           rst_value = 1;
```

```
        end
        if t - last_time >= rst_reset_interval
            rst_value = 0;
        end
    end
    if isempty(A_value)
        A_value = 6;
    end
    A = A_value;
    B = B_value;
    reset_spec_dev_extern  = rst_value;
    reset_divider_extern = ena_value;
end
```

Лістинг Б.10 – Simulink блок «input_gen», тест 7, перевірка сигналу скидання під час активного зовнішнього сигналу дозволу

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
    persistent A_value
    persistent B_value
    persistent rst_value
    persistent ena_value
    persistent last_time

    rst_set_interval  = 2;
    rst_reset_interval  = 3;
    ena_set_interval  = 1;
    ena_reset_interval  = 4;
    if isempty(ena_value)
        ena_value = 0;
        last_time = t;
    else
        if t - last_time >= ena_set_interval
            ena_value = 1;
        end
```

```
        if t - last_time >= ena_reset_interval
            ena_value = 0;
        end
    end

    if isempty(B_value)
        B_value = 8;
    end
    if isempty(rst_value)
        rst_value = 0;
        last_time = t;
    else
        if t - last_time >= rst_set_interval
            rst_value = 1;
        end
        if t - last_time >= rst_reset_interval
            rst_value = 0;
        end
    end

    if isempty(A_value)
        A_value = 6;
    end
    A = A_value;
    B = B_value;
    reset_spec_dev_extern  = rst_value;
    reset_divider_extern = ena_value;
end
```

Лістинг Б.11 – Simulink блок «input_gen», тест 8, перевірка сигналу скидання під час активного сигналу дозволу в момент зміни операнда

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
    persistent A_value
    persistent B_value
```

```
persistent rst_value
persistent ena_value
persistent last_time

rst_set_interval  = 2;
rst_reset_interval  = 4;
ena_set_interval  = 1;
ena_reset_interval  = 5;
A_update_interval = 3;
if isempty(ena_value)
    ena_value = 0;
    last_time = t;
else
    if t - last_time >= ena_set_interval
        ena_value = 1;
    end
    if t - last_time >= ena_reset_interval
        ena_value = 0;
    end
end

if isempty(B_value)
    B_value = 8;
end
if isempty(rst_value)
    rst_value = 0;
    last_time = t;
else
    if t - last_time >= rst_set_interval
        rst_value = 1;
    end
    if t - last_time >= rst_reset_interval
        rst_value = 0;
    end
end
```

```
  if isempty(A_value)
      A_value = 6;
      last_time = t;
  else
      if t - last_time >= A_update_interval
          A_value = 14;
      end
  end
   A = A_value;
   B = B_value;
   reset_spec_dev_extern  = rst_value;
   reset_divider_extern = ena_value;
end
```

Лістинг Б.12 – Simulink блок «input_gen», тест 9, перевірка перезапуску алгоритму при активному сигналі дозволу в разі зміни операнда

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
  persistent A_value
  persistent B_value
  persistent rst_value
  persistent ena_value
  persistent last_time

  rst_set_interval  = 1;
  rst_reset_interval  = 1;
  ena_set_interval  = 0.5;
  ena_reset_interval  = 4.5;
  A_update_interval = 2.5;

  if isempty(ena_value)
      ena_value = 0;
      last_time = t;
  else
```

```
    if t - last_time >= ena_set_interval
        ena_value = 1;
    end
    if t - last_time >= ena_reset_interval
        ena_value = 0;
    end
end

if isempty(B_value)
    B_value = 8;
end

if isempty(rst_value)
    rst_value = 0;
    last_time = t;
else
    if t - last_time >= rst_set_interval
        rst_value = 1;
    end
    if t - last_time >= rst_reset_interval
        rst_value = 0;
    end
end

if isempty(A_value)
    A_value = 6;
    last_time = t;
else
    if t - last_time >= A_update_interval
        A_value = 14;
    end
end
A = A_value;
B = B_value;
reset_spec_dev_extern  = rst_value;
```

```
  reset_divider_extern = ena_value;
end
```

Лістинг Б.13 – Simulink блок «input_gen», тест 10, перевірка перезапуску алгоритму при ready = 1 та активному сигналі дозволу в разі зміни операнда

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
  persistent A_value
  persistent B_value
  persistent rst_value
  persistent ena_value
  persistent last_time

  rst_set_interval  = 1;
  rst_reset_interval  = 1;
  ena_set_interval  = 3;
  ena_reset_interval  = 5;
  A_update_interval = 4;

  if isempty(ena_value)
      ena_value = 0;
      last_time = t;
  else
      if t - last_time >= ena_set_interval
          ena_value = 1;
      end
      if t - last_time >= ena_reset_interval
          ena_value = 0;
      end
  end

  if isempty(B_value)
      B_value = 8;
  end
  if isempty(rst_value)
```

```
        rst_value = 0;
        last_time = t;
    else
        if t - last_time >= rst_set_interval
            rst_value = 1;
        end
        if t - last_time >= rst_reset_interval
            rst_value = 0;
        end
    end

 if isempty(A_value)
     A_value = 6;
     last_time = t;
 else
     if t - last_time >= A_update_interval
         A_value = 14;
     end
 end
  A = A_value;
  B = B_value;
  reset_spec_dev_extern  = rst_value;
  reset_divider_extern = ena_value;
end
```

Лістинг Б.14 – Simulink блок «input_gen», тест 11, перевірка перезапуску алгоритму після зміни операнда, при ready = 1 та активному сигналі дозволу в разі скидання

```
function [A, B, reset_spec_dev_extern, reset_divider_extern] =
input_gen(t)
  persistent A_value
  persistent B_value
  persistent rst_value
  persistent ena_value
```

```
persistent last_time

rst_set_interval  = 5;
rst_reset_interval  = 6;
ena_set_interval  = 3;
ena_reset_interval  = 7;
A_update_interval = 4;

if isempty(ena_value)
    ena_value = 0;
    last_time = t;
else
    if t - last_time >= ena_set_interval
        ena_value = 1;
    end
    if t - last_time >= ena_reset_interval
        ena_value = 0;
    end
end

if isempty(B_value)
    B_value = 8;
end
if isempty(rst_value)
    rst_value = 0;
    last_time = t;
else
    if t - last_time >= rst_set_interval
        rst_value = 1;
    end
    if t - last_time >= rst_reset_interval
        rst_value = 0;
    end
end
```

```
 if isempty(A_value)
     A_value = 6;
     last_time = t;
 else
     if t - last_time >= A_update_interval
         A_value = 14;
     end
 end
  A = A_value;
  B = B_value;
  reset_spec_dev_extern  = rst_value;
  reset_divider_extern = ena_value;
end
```

ДОДАТОК В

Слайди презентації

Рисунок В.1 – Слайд презентації 1

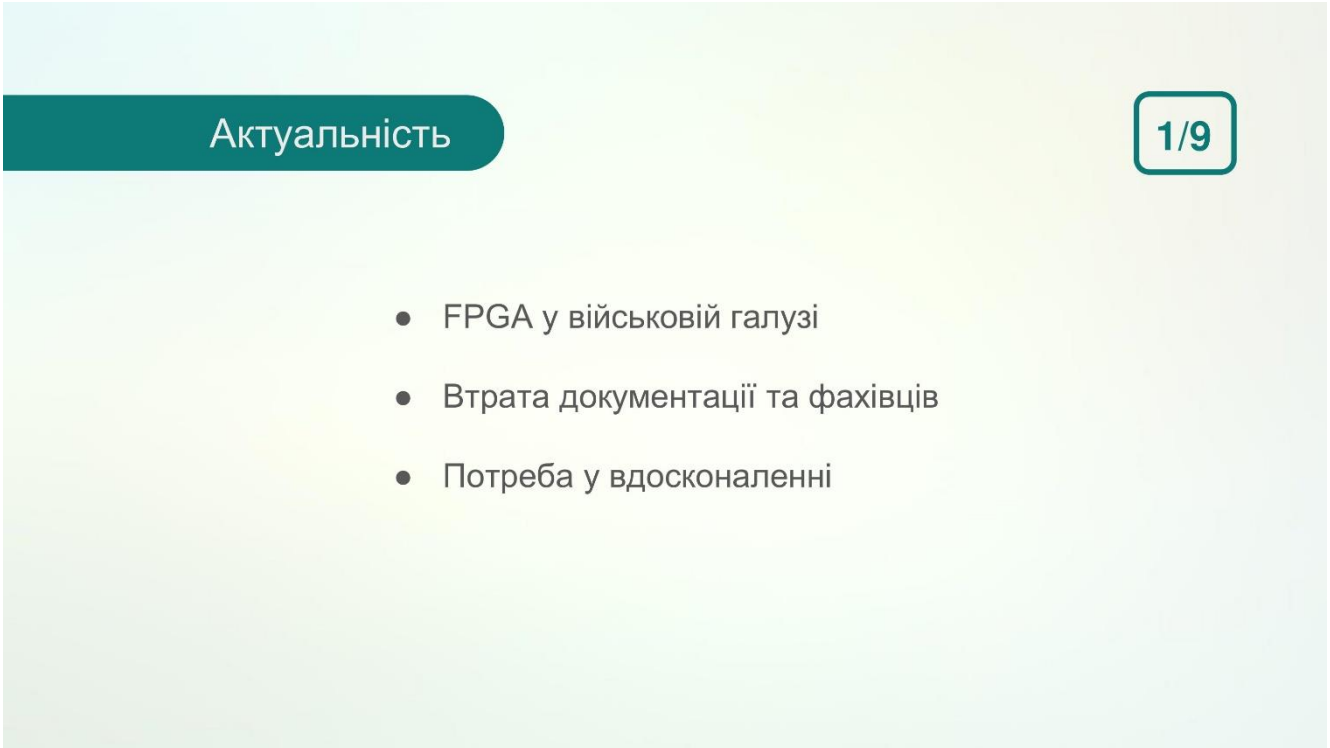


Рисунок В.2 – Слайд презентації 2

Рисунок В.3 – Слайд презентації 3

Рисунок В.4 – Слайд презентації 4

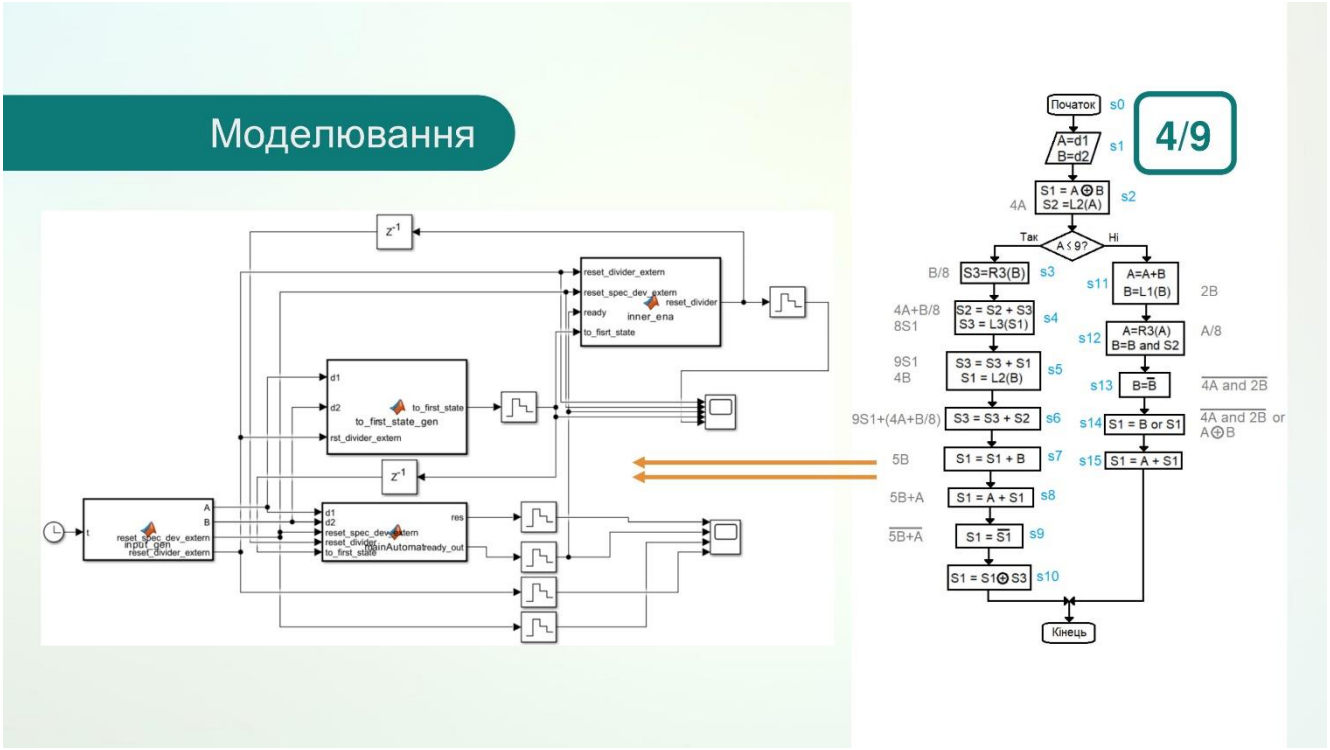


Рисунок В.5 – Слайд презентації 5

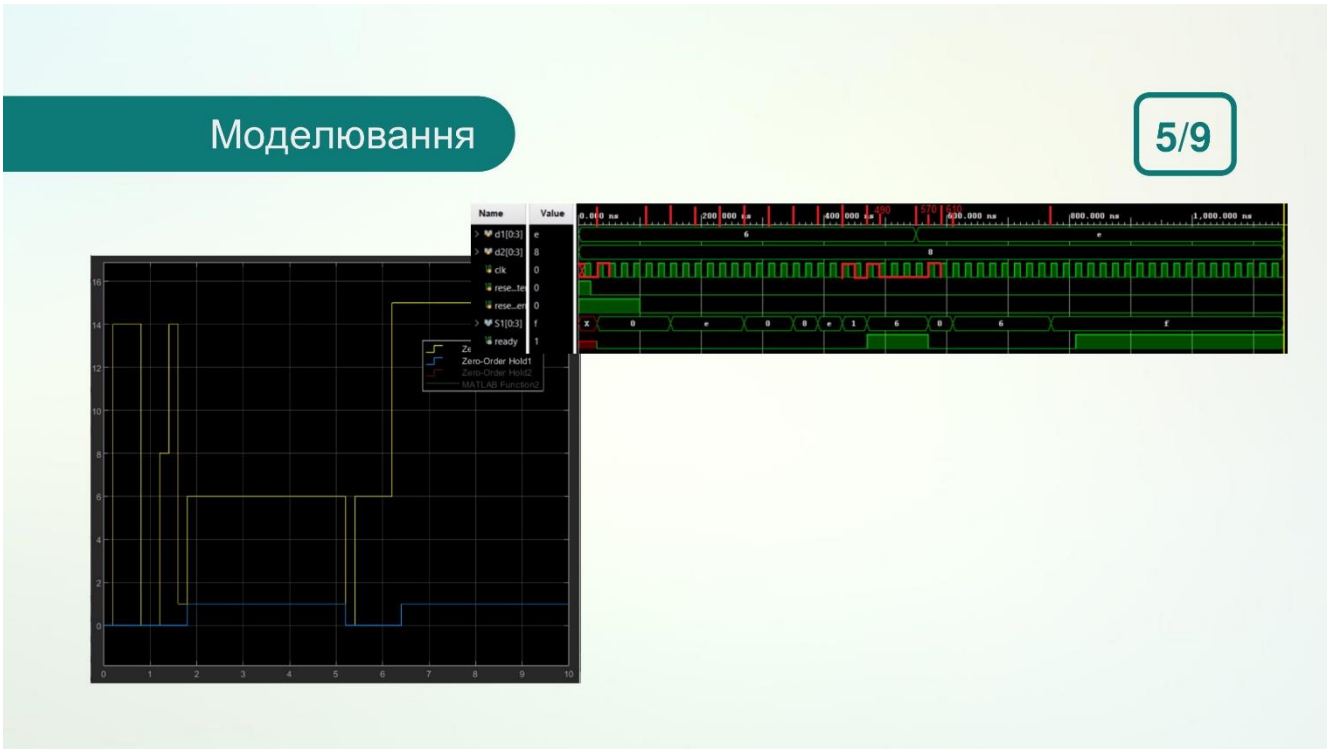


Рисунок В.6 – Слайд презентації 6

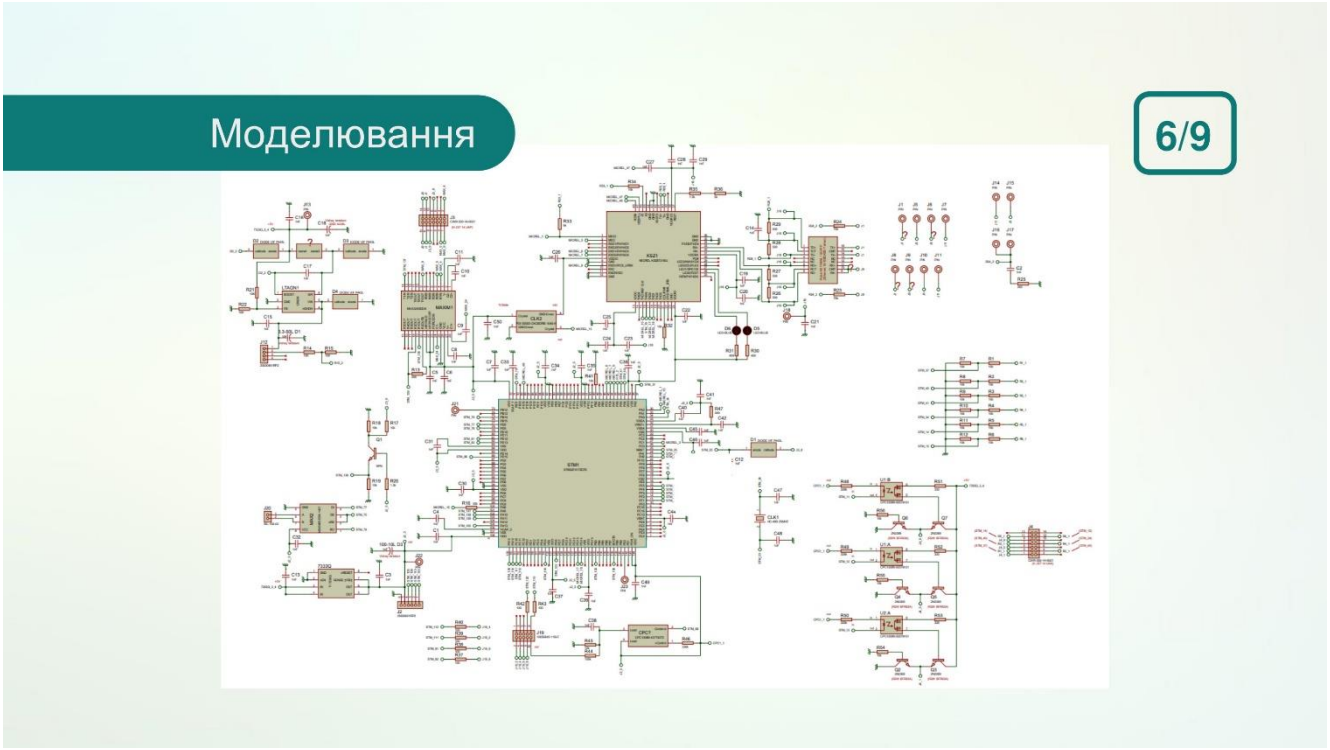


Рисунок В.7 – Слайд презентації 7

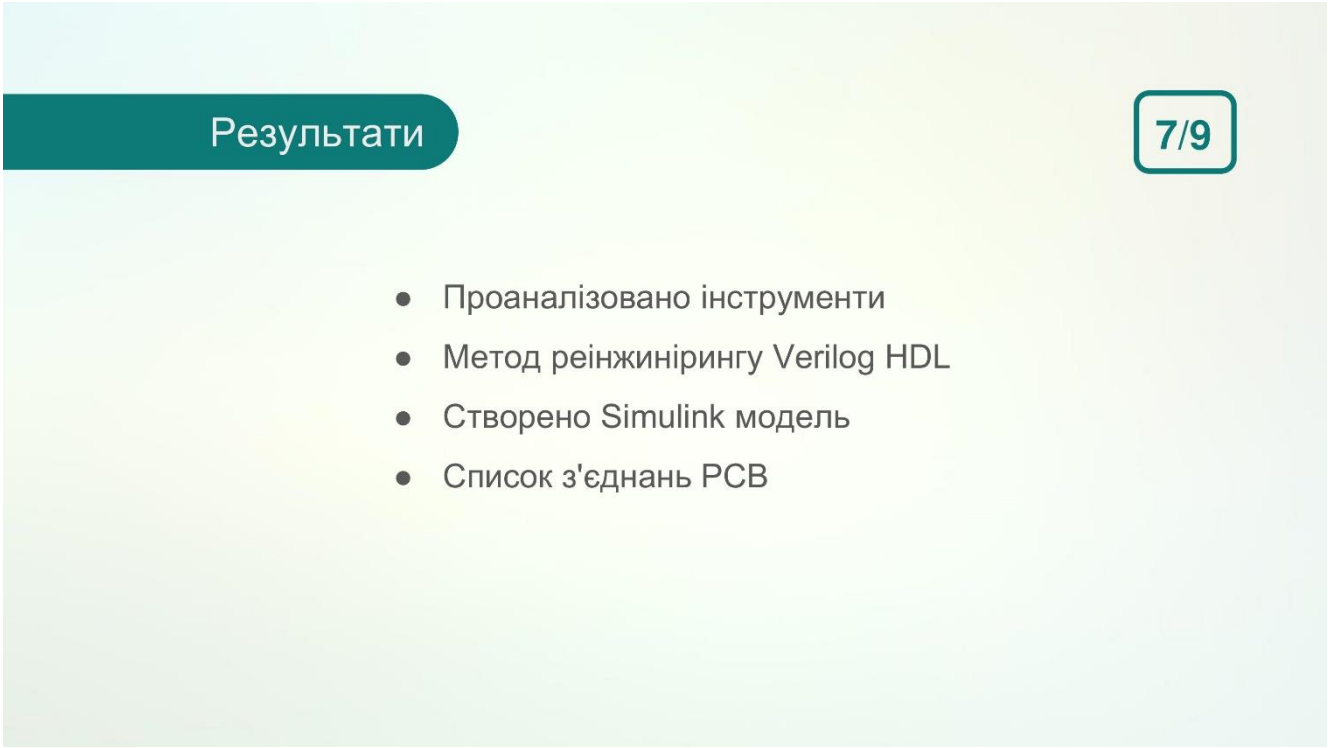


Рисунок В.8 – Слайд презентації 8

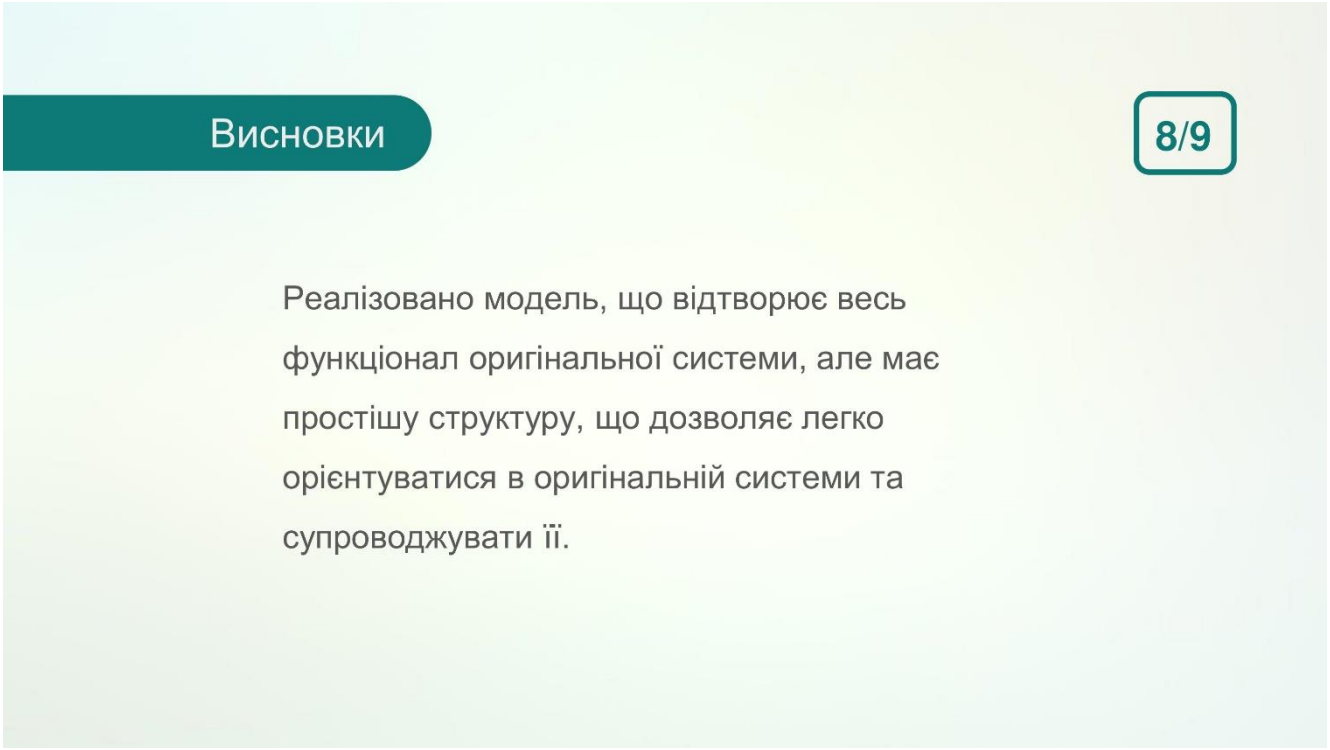


Рисунок В.9 – Слайд презентації 9

Рисунок В.10 – Слайд презентації 10