Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

ДИПЛОМНА РОБОТА МАГІСТРА

на тему:

«Синтез електронних секвенційних схем мовою опису обладнання Verilog в середовищі проектування Intel Quartus Prime»

студента 2 року навчання
спеціальності 123 «Комп'ютерна інженерія»
Рябова Дмитра Олександровича

Науковий керівник:
к. ф.-м. н., доцент кафедри
радіофізики, електроніки та комп'ютерних систем,
ЗАГОРОДНЮК Сергій Петрович

До захисту допускаю
Протокол засідання кафедри від
"__"_____2023 р. №_____

Зав. кафедри, к.ф.-м.н., доцент
Юрій БОЙКО
_____

Київ 2023

РЕФЕРАТ

Обсяг роботи за об’ємом складає 76 сторінки, містить 24 рисунки, 5 таблиць, та 9 додатків; використано 20 інформаційних джерел.

СИНТЕЗ ЕЛЕКТРОНИХ СХЕМ, МОВА ОПИСУ АПАРАТУРИ VERILOG, INTEL QUARTUS PRIME, FPGA, MODELSIM.

Кваліфікаційна робота є актуальною з огляду на масове поширення і використання сучасних спеціалізованих приладів - контролерів ПЛІС (програмована логічна інтегральна схема). Функціональність таких контролерів може бути визначена інженером-розробником за допомогою мови опису обладнання VHDL, Verilog або System Verilog та може бути змінена багатократно в залежності від змісту і суті поточної технічної задачі. Тому широке застосування таких приладів створює для молодих інженерів-дослідників необхідність детально опанувати і закріпити уміння швидко та ефективно віддзеркалювати функціональність нової електронної схеми у структуру та опис цієї схеми вказаними мовами опису обладнання і у такий спосіб перетворити ПЛІС-контролер на стендовий експериментальний зразок нового приладу, що розробляється.

Мета дослідження :

- Розробити схему довільного реального керуючого пристрою, яка має функціональність і логіку роботи скінченого автомату Мілі. Побудувати граф переходів. Розробити дві нові схеми автомату, які мають якісну відмінність та підвищену складність по відношенню до першої схеми.
- Синтезувати електронну схему кожного скінченого автомату мовою опису обладнання Verilog у середовищі проектування Inter Quartus Prime та виконати перевірку логіки роботи схеми за допомогою

вбудованого симулятора ModelSIM, включенного до середовища Quartus.

- Для кожної електронної схеми застосувати програмний механізм перевірки TestBench, який дозволяє перевірити швидкодію роботи схеми, а також стабільність і коректність перемикання станів скінченого автомату.
- Реалізувати скінчений автомат найвищого рівня складності у вигляді одного головного об'єднуючого Verilog-файлу та двох допоміжних Verilog-файлів, які обмінюються між собою сигналами.

Інструменти розробки:

- Intel Quartus Prime v.17.1 – це програмний комплекс для проектування і аналізу цифрових схем, який включає в себе інструменти для створення, тестування та реалізації складних цифрових схем, підтримує різні цифрові пристрої, включаючи FPGA, і містить багато вбудованих бібліотек та компонентів, що допомагають прискорити процес проектування.
- ModelSim – це програмне забезпечення для симуляції апаратури, що використовується для верифікації і аналізу цифрових і аналогових електронних схем. Дозволяє створювати імітацію поведінки схеми, за для визначення її працездатності та виявленні можливих помилок або проблем в роботі до того, як схема буде реалізована на апаратурі.
- Мова програмування: Verilog

Результат роботи: Синтезовано електронні цифрові схеми для трьох концептуально різних секвенційних пристроїв на основі скінчених автоматів Мура та Мілі поширеною стандартизованою мовою опису обладнання Verilog.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

FPGA – Field-Programmable Gate Array

ПЛІС – програмовані логічні інтегральні схеми

ІС – інтегральні схеми

EDA – Electronic Design Automation

PLD – Programmable logic device

CPLD – Complex programmable logic device

SoC – System on chip

HDL – Hardware Description Language

ПК – персональний комп’ютер

ВСТУП

На цей час технічні вимоги до ефективності та якості обробки великих обсягів інформації та складних високочастотних сигналів вимагають розробки нових спеціалізованих електронних пристроїв, складу яких можуть належати вольтметри, осцилографи, фазометри, аналізатори спектрів,, різного роду датчики, лічильники, фільтри, модулятори/демодулятори та інші інтегралльні та диференціальні схеми. Один з ключових етапів при створенні таких пристроїв - це синтез електронних схем. Мова опису обладнання Verilog та середовище проектування Intel Quartus Prime дозволяють моделювати та синтезувати електронні схеми більш ефективно.

В сучасному світі програмовані логічні інтегральні схеми (ПЛІС) знаходять все більше застосування у різних галузях промисловості. Один з ключових елементів ПЛІС – це скінченні автомати, які виконують необхідні обчислення та керують функціонуванням цілого пристрою.

Розробка ефективних методів та алгоритмів структурного синтезу скінченних автоматів є актуальною проблемою в галузі розробки програмованого обладнання. Швидкість та ефективність розробки складних автоматів є важливими параметрами у конкурентному світі технологій. Дослідження в галузі синтезу скінченних автоматів мовою опису Verilog та їх виконання на контролерах програмованих логічних інтегральних схем має великий практичний потенціал і може бути використана для розробки нових технологічних рішень у різних галузях промисловості, таких як автомобільна промисловість, електроніка, медицина, телекомунікації та інші. Окрім розробки методів та алгоритмів структурного синтезу скінченних автоматів мовою опису обладнання Verilog, в рамках дослідження буде проведено моделювання роботи автоматів в середовищі ModelSim. Моделювання є важливим етапом при розробці промислових пристроїв, оскільки дозволяє

перевірити правильність роботи скінченого автомата та виявити можливі помилки в роботі пристрою до його програмування. Крім того, ModelSim надає можливість використовувати різні інструменти для аналізу та відлагодження коду, що сприяє підвищенню ефективності розробки та скороченню часу налагодження.

РОЗДІЛ 1. ПЕРЕВАГИ ТА ОСОБЛИВОСТІ ПРОГРАМОВАНИХ ЛОГІЧНИХ ІНТЕГРАЛЬНИХ СХЕМ (FPGA), АНАЛІЗ ЇХ ВИРОБНИКІВ, ПРОГРАМНИХ СЕРЕДОВИЩ ТА МОВ ОПИСУ АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Програмовані логічні інтегральні схеми (ПЛІС), особливості та переваги FPGA.

Програмовані логічні інтегральні схеми (ПЛІС) - це електронні компоненти, які містяться на одному чіпі і виконують функції логічної обробки сигналів.[1] Зазвичай, ПЛІС використовуються для виконання складних операцій з обробки даних, таких як цифрова обробка сигналів, криптографія, керування системами та інші.

Одним з видів ПЛІС є програмовані матриці логічних вентилів (FPGA). FPGA містять матрицю з логічних вентилів, яку можна програмувати для виконання різноманітних завдань.[2] Оскільки FPGA можна програмувати, та перепрограмовувати, вони можуть виконувати багато завдань одночасно, що робить їх дуже потужними інструментами для виконання складних обчислень.

Однією з головних переваг FPGA є їх гнучкість. FPGA можуть бути програмовані для виконання будь-яких функцій, що робить їх ідеальними для використання в складних системах, де вимоги до обчислювальних потужностей змінюються з часом. Крім того, FPGA мають дуже велику швидкість обчислень, що робить їх ідеальними для використання в системах реального часу.

Іншою важливою перевагою FPGA є їх енергоефективність. FPGA використовують менше енергії, ніж традиційні мікроконтролери, що дозволяє їм працювати на довгі періоди часу без підзарядки.[3] Крім того,

FPGA можуть виконувати обчислення при нижчих напругах, що дозволяє їм знизити споживання енергії.

Однак, FPGA мають свої недоліки. Перш за все, вони вимагають більше часу і зусиль для програмування порівняно з іншими типами електронних компонентів, таких як мікроконтролери.[16] Також, FPGA можуть бути більш витратними, ніж інші типи ПЛІС, тому використання їх у проектах може вимагати додаткових коштів.

Загалом, FPGA є потужними та гнучкими інструментами для виконання складних обчислень, що дозволяє їх використовувати в багатьох різних сферах. Однак, перед використанням FPGA у проекті, необхідно ретельно проаналізувати вимоги до проекту та визначити, чи є FPGA оптимальним вибором для даного завдання.

Отже, в даному розділі дипломної роботи було розглянуто особливості та переваги FPGA як типу програмованих логічних інтегральних схем. Виявлено, що FPGA мають багато переваг, таких як гнучкість, швидкість обчислень та енергоефективність, що робить їх ідеальними для використання в складних системах. Однак, перед використанням FPGA у проекті, необхідно ретельно проаналізувати вимоги до проекту та визначити, чи є FPGA оптимальним вибором для даного завдання.

1.2 Intel Altera та Xilinx – передові виробники програмованих логічних інтегральних схем (ПЛІС)

Intel Altera та Xilinx – два з найбільших виробників програмованих логічних інтегральних схем, які займають значну частку на ринку ПЛІС. Обидва виробники надають широкий асортимент для різних застосувань, від промислової автоматизації до споживчих електронних пристроїв.[4]

Intel Altera – це підрозділ компанії Intel, що спеціалізується на розробці та виробництві програмованих логічних інтегральних схем. Один з основних продуктів Intel Altera – це серія ПЛІС(FPGA). FPGA відрізняються великою гнучкістю та швидкодією, що дозволяє їх використовувати в багатьох застосуваннях. FPGA компанії Intel Altera мають від 2,5 тисяч до 5 мільйонів системних вентилів та можуть працювати на частоті до 1 ГГц. Крім того, FPGA Intel Altera мають високий рівень інтеграції та підтримують різні стандарти, такі як PCI Express, Gigabit Ethernet, HDMI та інші. Їх продукція включає широкий спектр ПЛІС-пристроїв, від бюджетних Cyclone до потужних Stratix. Одним з ключових конкурентних переваг Intel Altera є їх технологія емуляції FPGA, відома як FPGA в розробці (FPGA in the Loop - FPGAIL). Вона дозволяє розробникам тестувати програми для FPGA в середовищі емуляції, що дозволяє значно прискорити процес розробки та відладки. Також Intel Altera має велику кількість програмованих ресурсів та широку підтримку програмного забезпечення, такого як Quartus Prime.

Щодо Xilinx, це ще один великий виробник ПЛІС, який займає важливе місце на ринку програмованих логічних інтегральних схем.[6] Один з головних продуктів Xilinx – це також серія FPGA, які мають широке застосування в багатьох галузях. Для поліпшення продуктивності і зменшення споживання енергії, Xilinx розробила архітектуру UltraScale, яка використовує 20-нм технології. Останнім часом Xilinx активно просувається в галузі інтеграції програмних інструментів, аналогових пристроїв і функцій безпеки в ПЛІС, виробляє програмовані логічні інтегральні схеми з високою пропускною здатністю для використання в мережах зі швидкістю передачі даних до 600 Гбіт/с. Це забезпечується за рахунок використання передових технологій виробництва. Крім того, має підтримку програмного засобу,

такого як Vivado, яка значно спрощує процес розробки та налагодження ПЛІС-систем

Intel Altera та Xilinx - дві провідні компанії в галузі програмованих логічних інтегральних схем.[5] Вони надають широкі можливості для проектування та розробки продуктів у різних галузях, таких як телекомунікації, автомобільна промисловість, медична техніка, аерокосмічна промисловість і багато іншого.

1.2.1 Порівняння середовищ розробки Quartus та Vivado

Сфера проектування цифрової електроніки стрімко зросла в останні роки з розробкою все більш складних інтегральних схем (IC). Для проектування цих IC широко використовуються засоби автоматизації електронного проектування (EDA). Двома найпопулярнішими інструментами EDA для проектування цифрової електроніки є Quartus і Vivado.

Розглянемо спочатку Quartus — це програмний пакет, розроблений Altera Corporation для проектування програмованих логічних пристроїв (PLD), включаючи програмовані вентильні матриці (FPGA) і комплексні програмовані логічні пристрої (CPLD).[7] Quartus надає повний набір інструментів для проектування, моделювання та перевірки цифрових схем. Quartus має кілька переваг перед іншими інструментами EDA. По-перше, він забезпечує зручний інтерфейс, який дозволяє розробникам швидко та легко проектувати цифрові схеми. По-друге, він підтримує широкий спектр PLD, у тому числі від Altera та інших виробників, що робить його придатним для різноманітних застосувань. По-третє, надає повний набір інструментів моделювання та перевірки (ModelSim), які допомагають розробникам переконатися, що їхні схеми функціонують правильно.

Відносно Vivado — це також інструмент EDA, проте, він в свою чергу розроблений компанією Xilinx для проектування ПЛІС і систем на кристалі (SoC).[9] Він надає повний набір інструментів для проектування, моделювання та перевірки цифрових схем. Vivado підтримує широкий спектр FPGA і SoC Xilinx і надає такі функції, як проектування на системному рівні. Vivado має ряд своїх переваг перед іншими інструментами EDA. По-перше, він забезпечує високооптимізовану схему проектування, що дозволяє розробникам швидко й ефективно проектувати цифрові схеми. По-друге, він підтримує широкий спектр Xilinx FPGA і SoC, що робить його придатним для різноманітних застосувань. Нарешті, Vivado надає повний набір інструментів моделювання та перевірки (Vivado Simulator), які допомагають розробникам переконатися, що їхні схеми функціонують правильно.

	Quartus	Vivado
Мови програмування	VHDL, Verilog, SystemVerilog	VHDL, Verilog, SystemVerilog, C, C++
Графічний інтерфейс	Має більш традеційний інтерфейс користувача який базується на вікнах	Має більш сучасний та інтуїтивно зрозумілий інтерфейс
Підтримка платформ	Підтримує FPGA на платформі Intel та ще деяких менш відомих компаній	Підтримує FPGA виключно на платформы Xilinx

Сумістність	Windows та Linux	Windows, Linux та MacOS

Таблиця 1 – Порівняння середовищ розробки Quartus та Vivado

Підсумовуючи, і Quartus, і Vivado є потужними інструментами EDA для проектування цифрової електроніки. У той час як Quartus розроблено в основному для ПЛІС Altera та деяких інших, Vivado розроблено виключно для ПЛІС та систем на процесорі Xilinx. Обидва інструменти мають свої переваги та недоліки, і вибір між ними залежить від конкретних вимог дизайн-проекту. Незалежно від того, який інструмент обрано, важливо розуміти, що кожне з вище описаних EDA надає влаштовані інструменти для моделювання (ModelSim та Vivado Simulator), що є надто важливо при проектуванні цифрової електроніки.

1.2.2 Порівняння середовищ симуляції ModelSim та Vivado Simulator

У сучасному світі проектування та розробки електронних пристроїв та систем, середовища симуляції є невід'ємною частиною процесу верифікації. Для симуляції цифрових пристроїв та систем, широко використовуються два основних інструменти: ModelSim та Vivado Simulator. Обидва інструменти є дуже потужними і здатними допомогти при верифікації складних цифрових систем. Однак, їхні можливості, переваги та недоліки можуть відрізнятися, тому важливо провести детальне порівняння, щоб вибрати оптимальне середовище для конкретного проекту.

ModelSim є однією з найбільш популярних систем симуляції апаратних засобів, що використовуються для верифікації і тестування апаратного забезпечення.[8] Він розроблений компанією Mentor Graphics, яка займається розробкою програмного забезпечення для електронної промисловості.

ModelSim підтримує мови опису апаратури VHDL і Verilog, а також більшість стандартів, пов'язаних з цими мовами.

Однією з особливостей ModelSim є його широкий набір функцій і можливостей, які дозволяють користувачам легко та швидко виконувати різноманітні завдання. Наприклад, система має вбудовані засоби аналізу коду та пошуку помилок, можливість генерувати звіти про тестування та забезпечення підтримки різних форматів файлів вхідного та вихідного коду. Окрім того, ModelSim дозволяє користувачам виконувати симуляції на різних рівнях абстракції, включаючи графічну симуляцію, так що користувач може спостерігати роботу своєї схеми в реальному часі. Ця функція дозволяє виявляти помилки в апаратурі та виправляти їх на ранніх етапах розробки.[10]

Однією з переваг ModelSim є його швидкість. Система дозволяє виконувати симуляції на великих і складних схемах, використовуючи оптимізовані алгоритми, що забезпечують високу швидкість обробки.

Недоліками ModelSim є те, що він не підтримує низку платформ, що може обмежувати його використання для певних проектів. Наприклад, його не можна використовувати на операційних системах Mac OS та Linux. Ще одним недоліком ModelSim є те, що він не є дуже ефективним для роботи зі складними та великими проектами. Збільшення розміру проекту може призвести до значного збільшення часу симуляції, що може стати проблемою для швидкого вирішення проблем.

Загалом, ModelSim є потужним та високоякісним середовищем для симуляції інтегральних схем, яке дозволяє розробникам перевірити та відлагодити свої проекти до їх реалізації на платі. Незважаючи на деякі

недоліки, це середовище залишається популярним вибором для професійних інженерів та студентів, які працюють з інтегральними схемами.

Vivado Simulator - це симулятор HDL-коду, розроблений компанією Xilinx для використання з їхніми програмними продуктами. Vivado Simulator є частиною інтегрованого середовища розробки Xilinx Vivado, яке включає в себе інструменти для синтезу, розміщення та маршрутизації FPGA-схем.[11]

Унікальністю Vivado Simulator є можливість розробки і симуляції FPGA-схем у єдиному інтегрованому середовищі, що зменшує час, необхідний для розробки та тестування FPGA-схем.

Перевагою є підтримка багатьох мов програмування, таких як Verilog, VHDL та SystemVerilog. Крім того, він має широкі можливості для візуалізації результатів симуляції, зокрема, можливість відстежування значень сигналів у реальному часі.

Недоліком Vivado Simulator є те, що його інтерфейс користувача може бути складним для розуміння та використання, особливо для початківців. Крім того, час, необхідний для настройки та запуску симуляції, може бути тривалим у порівнянні з іншими симуляторами. Також важливою недоліком є те, що Vivado Simulator підтримує тільки пристрої Xilinx, що обмежує його застосування до проектів, що не використовують ці пристрої.

Однак, Vivado Simulator має кілька переваг перед ModelSim, зокрема, він є більш сучасним і продуктивним інструментом, що підтримує такі функції, як многопоточність, оптимізацію використання пам'яті, та інші технології, що роблять його більш ефективним в роботі зі складними проектами. Також, Vivado Simulator має інтегрований інтерфейс з іншими інструментами Xilinx, такими як Vivado Design Suite, що дозволяє

користувачам виконувати всі етапи проектування, від симуляції до виконання на реальних пристроях, в одній інтегрованій середовищі розробки.

Отже, можна зазначити, що обидва інструменти, ModelSim та Vivado Simulator, є потужними засобами для симуляції HDL-коду. Однак при виборі між цими інструментами необхідно враховувати особливості свого проекту, включаючи підтримувані пристрої, мови програмування, та здатність працювати з конкретними вимогами проекту. Крім того, важливо враховувати особливості інтерфейсу користувача та час, необхідний для запуску симуляції. Вибір між ModelSim та Vivado Simulator повинен бути здійснений на основі потреб вашого проекту та особливостей роботи з обраним інструментом.

1.3 Порівняння мов опису апаратного забезпечення, їх переваги та недоліки

Одним із ключових елементів проектування апаратного забезпечення є мова опису апаратури (HDL). Існує декілька різних мов опису апаратного забезпечення, включаючи основні Verilog, VHDL та SystemVerilog, кожна з яких має свої переваги та недоліки. Порівняння цих мов може допомогти розробникам вибрати мову, яка найкраще відповідає їхнім потребам та вимогам проекту. Однак, при виборі мови опису апаратного забезпечення необхідно враховувати багато чинників, таких як складність вивчення мови, наявність інструментальних засобів, швидкість синтезу та ефективність розробки. В цьому пункті дипломної роботи будуть проаналізовані основні мови опису апаратного забезпечення їх переваги та недоліки, що допоможе зробити обґрунтований вибір при проектуванні апаратного забезпечення.

	VHDL	Verilog	SystemVerilog

Синтаксис	Заснований на Pascal	С-подібний	Заснований на Verilog та С
Типи даних	Строго типізована мова	Не строго типізована мова	Строго типізована мова з можливістю використання нестрого типізованих змінних
Підтримка об’єктів	Підтримує моделювання поведінки, структурного та фізичного рівнів	Підтримує моделювання на рівні структури	Підтримує всі рівні моделювання, включаючи поведінковий
Можливості синтезу	Високі	Високі	Високі
Підтримка динамічної розробки	Так	Так	Так
Цільова платформа	FPGA та ASIC	FPGA та ASIC	FPGA та ASIC

Таблиця 2 – Порівняння основних мов опису апаратного забезпечення

В таблиці наведені основні характеристики порівняння мов опису апаратного забезпечення, проте необхідно додати ще декілька слів. У порівнянні з VHDL, Verilog та SystemVerilog є менш формальними мовами

опису апаратного забезпечення, що дає можливість розробникам швидше створювати прототипи та приступати до верифікації розроблених моделей. Окрім того, Verilog має простіший синтаксис, що робить його легшим у вивченні та використанні.[12] Ще одною перевагою Verilog є можливість використання атрибутів, які дозволяють змінювати функціональність моделей залежно від вхідних даних, моделювання обмежується структурним рівнем, але він має здатність до паралельних обчислень, що робить його ідеальним для використання в проектуванні апаратного забезпечення.

В свою чергу SystemVerilog був розроблений як розширення Verilog з метою вирішення деяких його недоліків та додавання нових функцій.[13] SystemVerilog є більш потужним інструментом для моделювання апаратного забезпечення порівняно з Verilog і VHDL.[14] Він підтримує багато нових функцій, включаючи об'єктно-орієнтований підхід до програмування. Також має здатність до асинхронного та синхронного процесів, що дає можливість використовувати його для моделювання різних типів апаратних пристроїв. Однак важливо зауважити, що дана мова є набагато скалднішою у порівнянні з VHDL та Verilog, не є стандартизованою, та має більший обсяг коду для виконання тих самих завдань, що може призвести до складнощів у підтримці та управлінні кодом.[15]

Узагальнюючи, варто сказати, що кожна мова опису апаратного забезпечення має свої переваги та недоліки, які можуть бути важливими для розробки апаратних засобів. Вибір мови опису апаратного забезпечення повинен залежати від вимог проекту та володіння розробником конкретною мовою. Перш за все, необхідно враховувати мету проекту, тип проекту та складність апаратного забезпечення. Якщо проект вимагає високого рівня абстракції, варто розглянути SystemVerilog, оскільки вона надає можливість використовувати об'єктно-орієнтований підхід. Якщо проект повинен бути

ефективним, варто розглянути Verilog, оскільки вона надає можливість більш точного моделювання апаратного забезпечення та гнучкості у використанні з іншими мовами програмування.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ КОНЦЕПТУАЛЬНО РІЗНИХ ДОРОЖНІХ СВІТЛОФОРІВ У СЕРЕДОВИЩІ РОЗРОБКИ INTEL QUARTUS PRIME

2.1 Встановлення програмного забезпечення для проектування та модуляції електронних схем

Інсталяція Quartus Prime та ModelSim є досить не складним етапом, проте вимагає деяких уточнень. Нижче наведені основні кроки для встановлення програмного забезпечення Quartus Prime та ModelSim на персональний комп’ютер.

1. Потрібно завантажити інсталяційний файл Quartus Prime з офіційного сайту компанії Intel, та переконатися, що завантажена версія програмного забезпечення відповідає операційній системі встановленій на ПК.

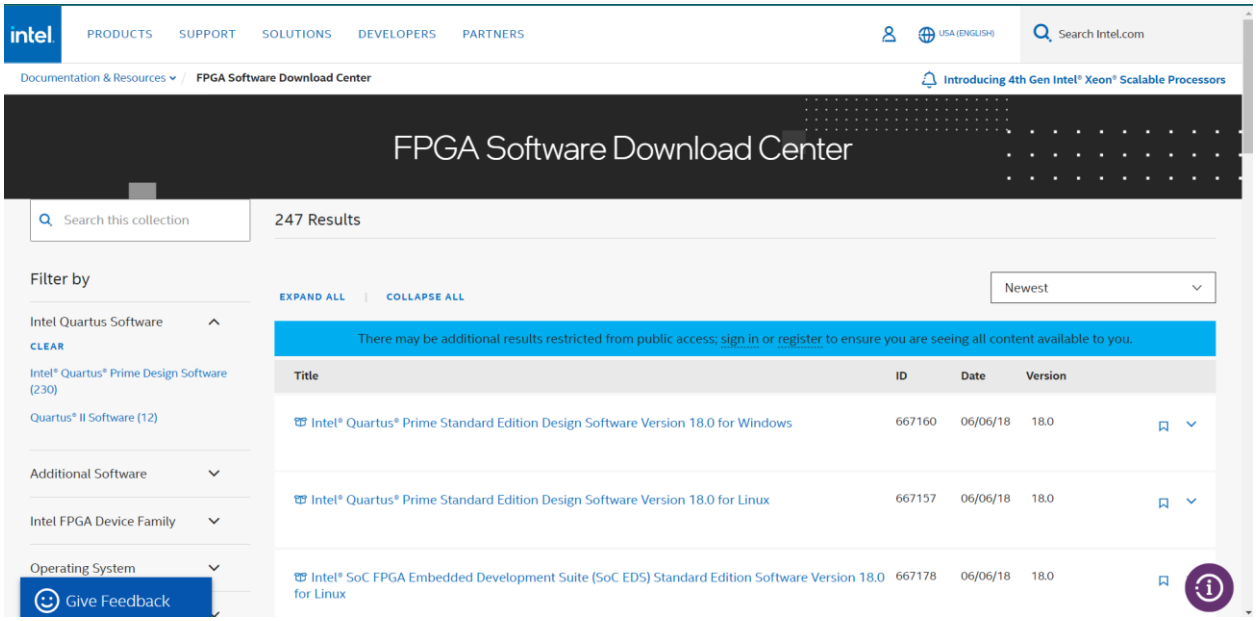

Рисунок 1 - офіційний сайт компанії Intel

В завантаженому архіві буде присутній файл readme.txt, перед встановленням варто з ним ознайомитись, та переконатись що версія програми підтримує вашу операційну систему, в вище згаданому файлі описуються усі версії операційних систем сумісні з даною версією програми.

2. Інсталювати програму Quartus Prime слідуючи інструкціям інсталяції.

3. Якщо під час інсталяції Quartus Prime не було запропоновано встановити додатково ModelSim – встановлюємо його окремо, в архіві буде додано окремий інсталятор.

4. Запускаємо, та перевіряємо роботу програми.

2.2 Створення проекту у Intel Quartus Prime та налаштування середовища для модуляції роботи у ModelSim

Створення проекту у середовищі розробки Intel Quartus Prime включає в себе наступні етапи:

1. Запуск програмного забезпечення Quartus Prime, у лівій верхній частині потрібно обрати File -> New Project Wizard (рисунок №2).

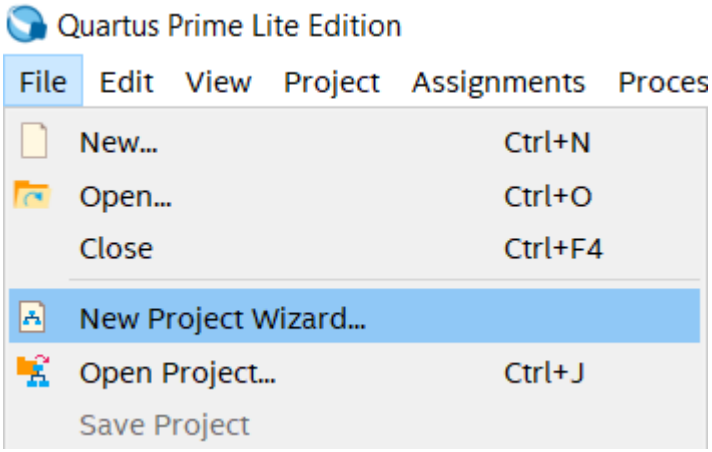

Рисунок 2 - створення нового проекту

2. У наступному вікні (рисунок №3) потрібно обрати шлях де буде зберігатися проект, написати назву проекту, та назву головно файлу проекту, а саме top-level design entity. Головний файл проекту або головний модуль в описі апаратної частини включає в себе всі під модулі необхідні для функціонування системи. Top-level design entity містить всі входи та виходи, та забезпечує правильну взаємодію між всіма

компонентами системи. Це важливо для забезпечення коректної роботи та відлагодження проблем, які можуть виникнути в процесі розробки.

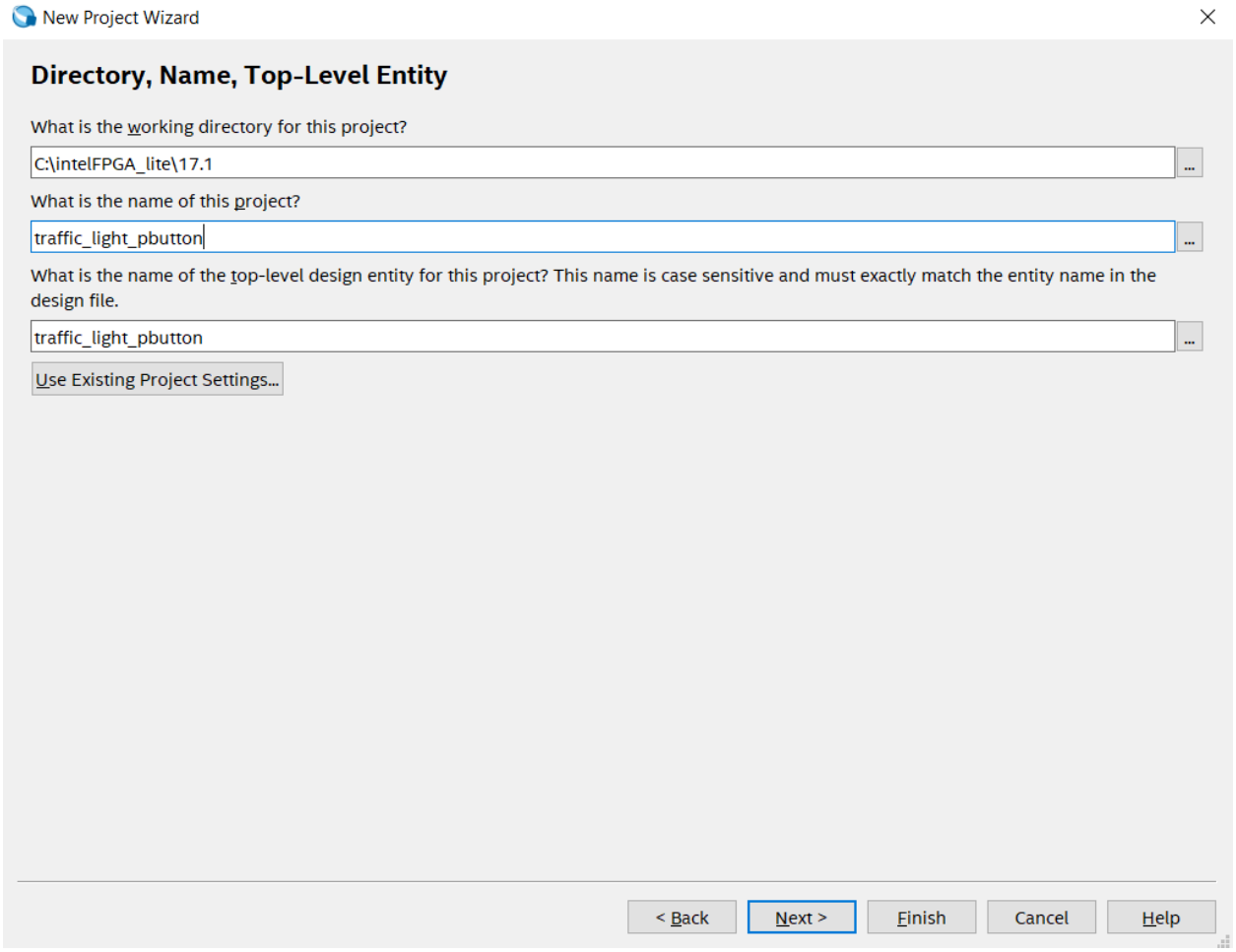

Рисунок 3 - вікно обрання директорії проекту, назви проекту та назви головного файлу проекту

3. Обираємо тип проекту (рисунок №4). Якщо це перший проект, то можна вибрати “Empty project”. Якщо ж це проект з готовими файлами, то можна вибрати “Project template”, в моєму випадку я обираю пустий проект.

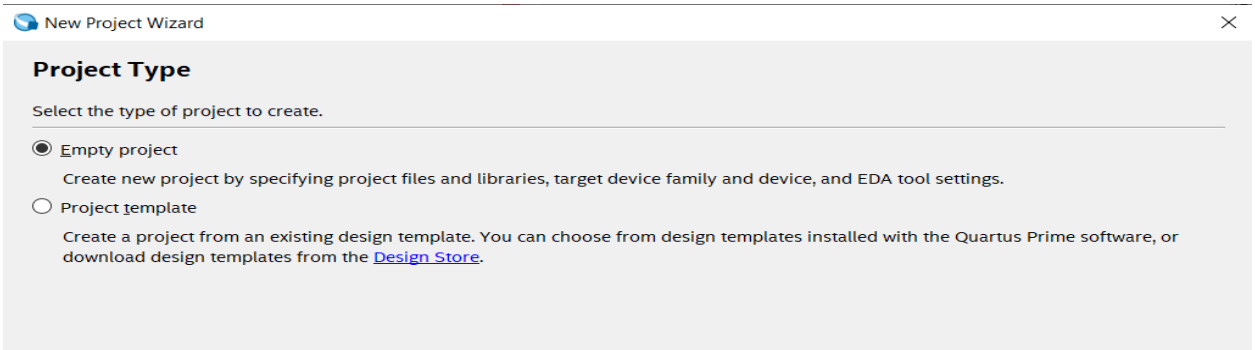

Рисунок 4 - обрання типу проекту

4. Визначаємось з типом пристрою для якого буде розроблюватись проект (рисунок №5). Якщо у переліченому списку немає пристрою для якого повинен був розроблятись проект, потрібно перейти на офіційний сайт використавши посилання зазначене на рисунку “Device Support List” переконатися що встановлена версія Quartus підтримує потрібний пристрій, якщо так – скачати та встановити необхідні файли, в іншому випадку скористатися іншою версією програми.

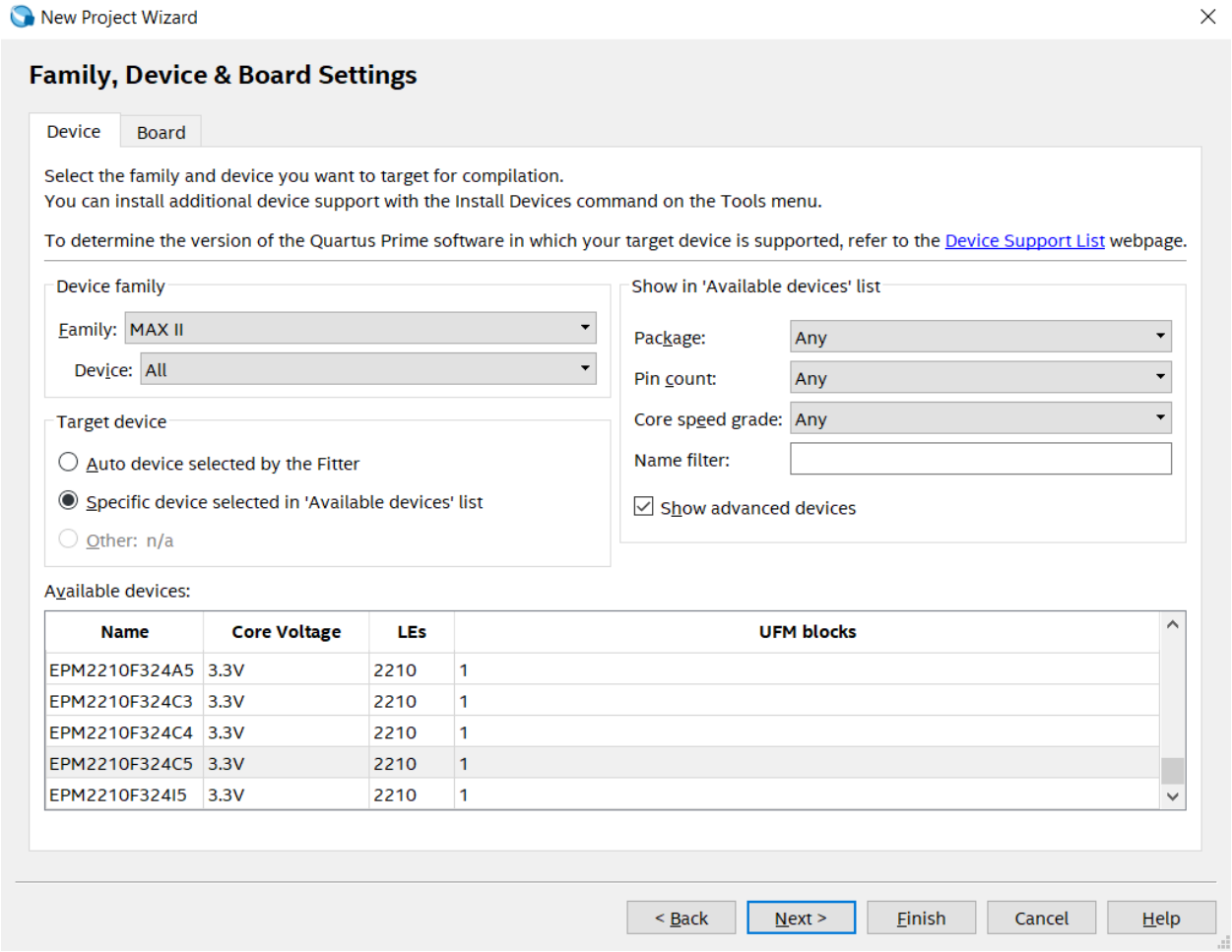

Рисунок 5 - обрання типу пристрою для якого буде розроблюватись проект

5. В останньому пункті доступні додаткові інструменти для перевірки та від лагодження роботи проекту. В моєму випадку симуляція роботи проекту буде відбуватися в ModelSim, а сам проект написаний мовою опису Verilog.

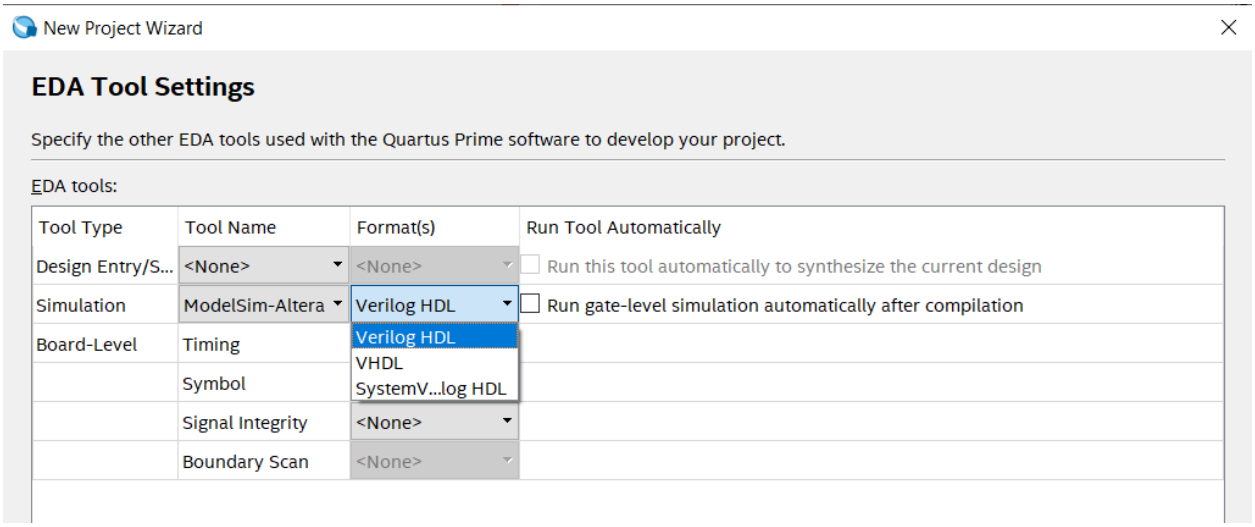

Рисунок 6 - доступні додаткові інструменти для розробки проекту

6. Важливо! До чи після створення проекту, налаштувати Quartus для роботи з ModelSim, для цього потрібно перейти в Tools -> Options -> EDA Tools Options. У розділі для ModelSim обрати каталог “win32aloem” який в свою чергу знаходиться в каталозі “modelsim_ase”, ця операція є обов’язковою для того щоб Quartus міг взаємодіяти з ModelSim.[19]

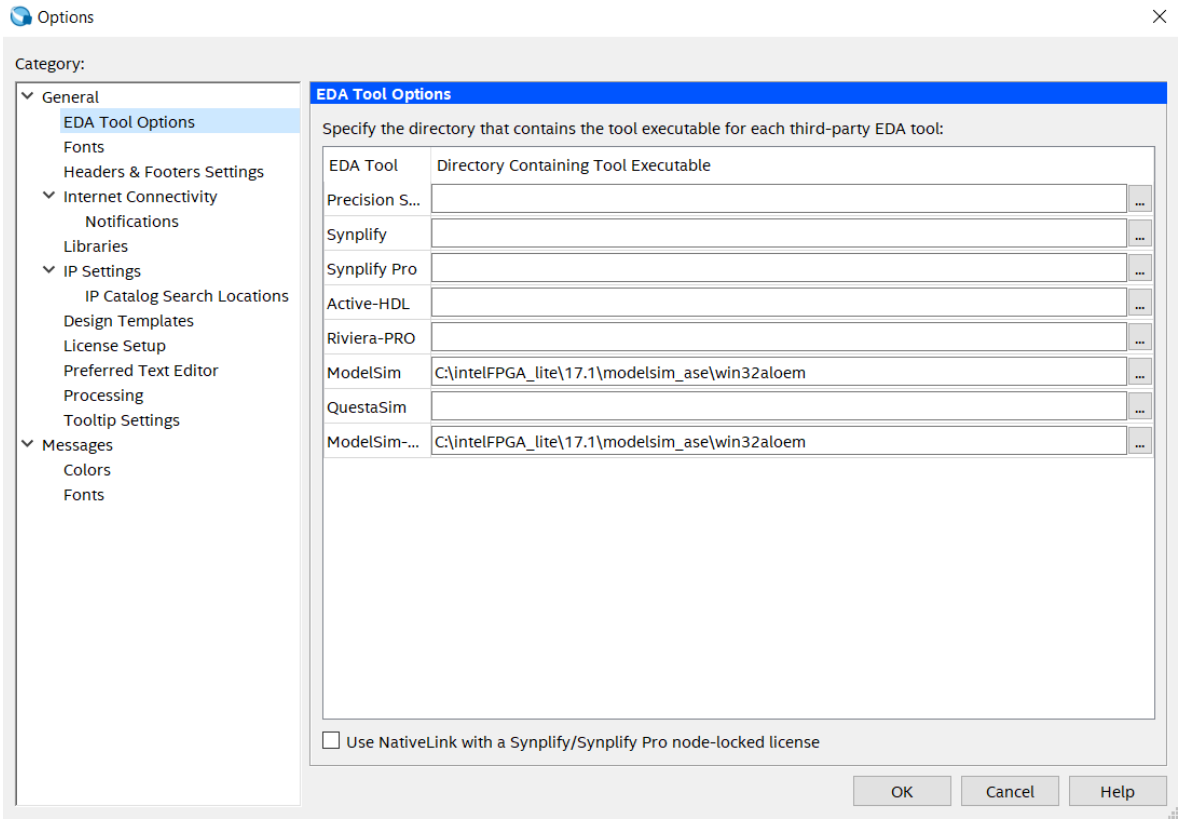

Рисунок 7 - налаштування взаємодії Quartus та ModelSim

2.3 Моделювання роботи дорожнього світлофору, що включає в себе світлофор для авто, пішоходів та кнопку для пішоходів

2.3.1 Опис роботи світлофору

Світлофор - це один з найбільш важливих елементів організації дорожнього руху. Він забезпечує безпечний та ефективний рух транспортних засобів на вулицях. Розробка та удосконалення світлофорної системи є актуальною проблемою у сфері дорожнього руху та містобудування.

Завдання розділу полягає в розробці Verilog моделі системи управління дорожнім світлофором, на основі автомата, що може бути синтезований в програмовану логічну схему.

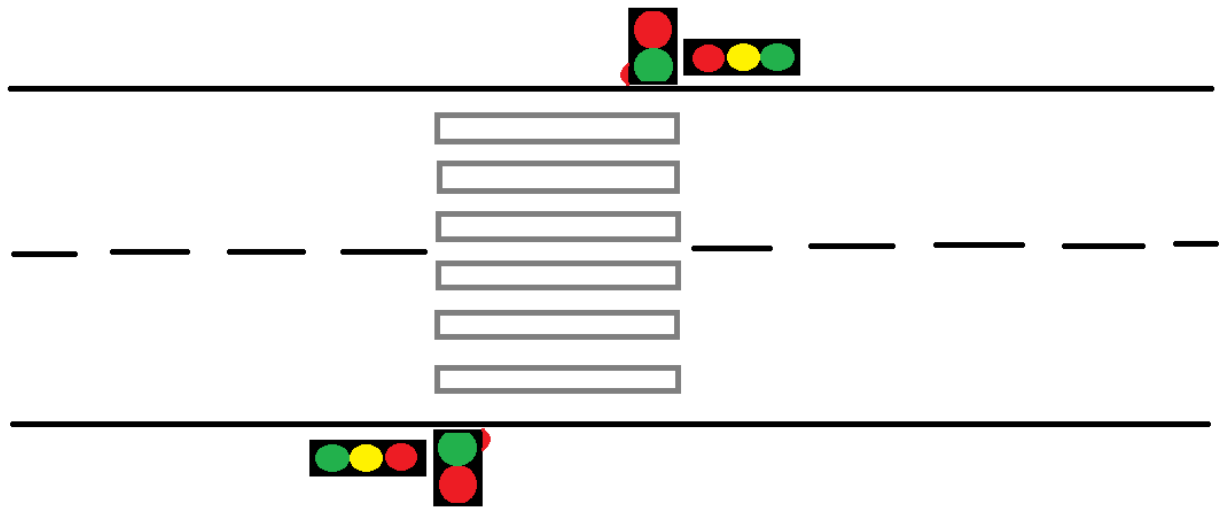

Рисунок 8 - спрощена схема дорожнього світлофору

Розглянемо алгоритм роботи автомата (рисунок №9) реалізованого для вище наведеного дорожнього світлофору (рисунок №8). Полягає він у тому, що для автомобілів завжди горить зелене світло, до того моменту поки не буде натиснута кнопка, яка повідомляє що пішоходи хочуть перейти дорогу. Модель автомату побудована таким чином, що кожен окремий світлофор має можливість мати різні часові затримки. В наведеному випадку модель реалізована так, що при натисканні кнопки, спрацьовує запуск двох таймерів.

Перший таймер - «count» відповідає за затримку перед перемикання автомату до іншого стану, у випадку переходу стану автомату з S0 до S1 це час від натиснення кнопки. Значення умов для перемикання стану автомату відрізняються, а значення таймеру анулюється після кожного переходу стану автомату.

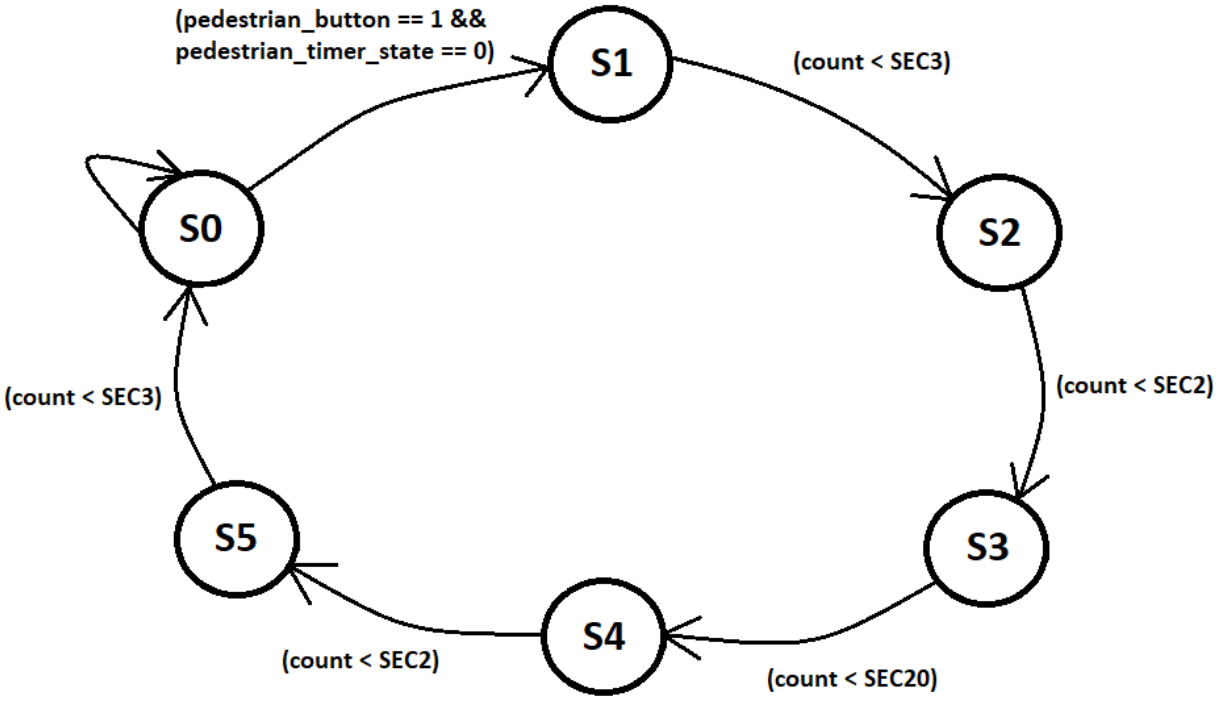

Рисунок 9 - граф логічних переходів автомату

Другий таймер – “pedestrian_timer_state” відповідає за блокування подання сигналу з кнопки після її натиснення на відведений час (60 секунд), реалізовано це для того щоб не порушувати роботу автомату, а саме щоб унеможливити постійне зелене світло для пішоходів та червоне для авто.

Стани автомату	Стан дорожнього світлофору (Light)	Стан пішохідного світлофору (Pedestrian_light)
S0	001	10

S1	010	10
S2	100	10
S3	100	01
S4	100	10
S5	110	10

Таблиця 3 - відповідність станів автомату та значень станів світлофорів

Для змінення станів автомату описаних в таблиці №3, де S0 – S5 це значення станів автомату, та двох вихідних регістрів “Light” та “Pedestrian_light”, де регістр “Light” визначено як 3-бітний вектор, де біти нумеруються від 2 (найбільш значущого) до 0 (найменш значущого). Це означає, що “Light” може приймати будь-яке значення від 0 (000) до 7 (111) в двійковій формі. Регістр “Pedestrian_light” визначено як 2-бітний вектор, де біти нумеруються від 1 (найбільш значущого) до 0 (найменш значущого). Це означає, що “Pedestrian_light” може приймати будь-яке значення від 0 (00) до 3 (11) в двійковій формі. Кожен з описаних станів відповідає за колір світлофору. Для автомобільного світлофору є чотири різних значення, це 001 – зелений колір, 010 – жовтий колір, 100 – червоний колір та 110 – червоно-жовтий колір відповідно. Для пішохідного світлофору є всього два стани, а саме 01 – зелений колір та 10 – червоний. Обидва регістри “Light” та “Pedestrian_light” оголошені як “output reg”, що означає, що вони є вихідними портами і можуть бути оновлені модулем, до якого вони належать. Ключове слово “reg” вказує, що регістри реалізовані як тригери, що означає, що їх значення буде зберігатися та оновлюватися на фронті сигналу годинника.

```
always @(posedge clk or posedge rst)
        if (rst == 1) begin
                state <= S0;
                count <= 0;
```

```
            pedestrian_button_state <= 0;
            pedestrian_timer_state <= 0;
end else begin
if (pedestrian_button == 1 && pedestrian_timer_state == 0) begin
            pedestrian_timer_state <= SEC60;
            pedestrian_button_state = 1;
end
case(state)
            S0: if(pedestrian_button_state == 1) begin
                        if(count < SEC8) begin
                                    state <= S0;
                                    count <= count + 1;
                                    pedestrian_timer_state <= pedestrian_timer_state -1;
                        end else begin
                                    state <= S1;
                                    count <= 0;
                        end
               end else begin
                        state <= S0;
                        count <= 0;
               end

            S1: if(count < SEC3) begin
                        state <= S1;
                        count <= count + 1;
                        pedestrian_timer_state <= pedestrian_timer_state - 1;
            end else begin
                        state <= S2;
                        count <= 0;
            end
```

В наведеному фрагменті реалізації автомату мовою опису Verilog команда always @(posedge clk or posedge rst) вказує, що конструкція виконується постійно за будь-яких змін вхідних сигналів clk або rst, де clk – це тактовий сигнал, або сигнал синхронізації цифрової схеми, а rst – сигнал скидання, що використовується для встановлення початкового стану схеми та анулює таймери.

При першому натисканні кнопки пішоходів “pedestrian_button” присвоюється змінній стану кнопки пішоходів “pedestrian_button_state” значення одиниці, та значення таймеру “pedestrian_timer_state” рівне 60 секундам. При кожному наступному натисканні кнопки “pedestrian_button” таймер “pedestrian_timer_state” поки він не прийме значення 0.

В звичайному стані по передньому фронту значення clk, автомат перевіряє чи “pedestrian_button_state” не має значення одиниці, анулює значення “count” та зберігає стан автомату S0. В іншому випадку при

натиснутій кнопці для пішоходів автомат зберігає свій стан протягом відповідної кількості секунд зазначеній в умові для переходу, збільшуючи значення таймеру “count” та зменшуючи значення таймеру “pedestrian_timer_state” по передньому фронту значення clk. Повна реалізація алгоритму роботи продемонстрована у додатку А.

2.3.2 Створення Test Bench для моделювання роботи світлофору у ModelSim

Test Bench - це спеціальний модуль у програмі для симуляції апаратних засобів, який дозволяє тестувати модулі апаратної складової до їх реалізації на апаратному рівні, а також надає можливість створення симуляційного середовища, що дозволяє перевіряти правильність роботи вихідних даних моделі апаратного забезпечення відповідно до заданих вхідних даних.[20] Дозволяє провести широкий спектр тестів для перевірки модулів апаратних засобів перед їх розгортанням на живому обладнанні. Він допомагає виявляти помилки та відлагоджувати моделі до того, як вони будуть вироблені.

Для створення Test Bench необхідно виконати наступний алгоритм дій:

1. Створити новий файл того ж розширення, що й головний файл у проекті Quartus.
2. У файлі додати необхідні залежності та бібліотеки, оголосити вхідні та вихідні порти.
3. Описати логіку роботи, опираючись на технічне завдання проекту.
4. Скомпілювати реалізований файл.

Для реалізації Test Bench`а роботи дорожнього світлофору описаного в попередньому пункті було створено файл - traffic_light_pbutton_tb, його код

продемонстровано у додатку Б. Модуль traffic_light_pbutton_tb містить у собі три вхідні сигнали:

```
reg clk;
reg rst;
reg pedestrian_button;
```

Кожен з них це сигнал регістра, який використовується як вхідний сигнал системи.

“clk” - використовується для синхронізації та керування таймінгом операцій системи. Цей сигнал підключений до джерела годинника, такого як кристалічний генератор або генератор годинника.

“rst” - використовується як сигнал скидання системи. Коли цей сигнал активується, система буде ініціалізована та переведена в відомий стан.

“pedestrian_button” - використовується як вхідний сигнал для системи. Цей сигнал може бути підключений до фізичної кнопки або перемикача та буде використовуватися для позначення того, коли пішохід бажає перетнути вулицю.

Два сигнали wire:

```
wire [2:0] light;

wire [1:0] pedestrian_light;
```

“Wire” - це один з типів даних у мові опису апаратури Verilog. Він використовується для оголошення сигналів, які в свою чергу використовуються для передачі даних між компонентами цифрової системи.

Wire є беззнаковим, тобто не має знака, і може приймати два стани – “0” (нуль) або “1” (одиниця). [17] Він може бути підключений до інших діодів, регістрів, комбінаційної логіки або будь-яких інших елементів цифрової системи, які приймають вхідні дані.

“Wire” можна порівняти з електричним проводом, який передає сигнали від одного пристрою до іншого в електричній схемі. Таким чином, “wire” може використовуватись для передачі даних між різними компонентами цифрової системи, що дозволяє їм спілкуватися та працювати разом.

Змінні "Light" та "Pedestrian_light" є векторами з трьох та двох бітів відповідно, та використовуються для керування набором світлофорів для авто та пішоходів. Кожен біт векторів відповідає за свій сигнал світлофору.

```
traffic_light_pbutton TRAFFIC_LIGHT_PBUTTON(
    .clk(clk),
    .rst(rst),
    .pedestrian_button(pedestrian_button),
    .light(light),
    .pedestrian_light(pedestrian_light)
);
```

Вище описано головний модуль програми, який містить порти, що підключаються до його цифрової схеми.

Основною частиною Test Bench`у є наступний блок коду, в якому отримують значення вхідні сигнали, тим самим після симуляції перевіряють роботу автомату.

```
initial clk = 1;
always #(`clk_period/2) clk = ~clk;

initial begin
            rst = 0;
   pedestrian_button = 0;
   #`clk_period;

   rst = 1;
   #`clk_period;

   rst = 0;
   #`clk_period;

   pedestrian_button = 1;

   #`clk_period;

   pedestrian_button = 0;


   begin : anonymous_block
      integer i;
```

```
        for (i = 0; i < 20; i = i + 1) begin
            #`clk_period;
        end
    end

    pedestrian_button = 1;

            #`clk_period;

    pedestrian_button = 0;

            begin : anonymous_block2
        integer j;
        for (j = 0; j < 60; j = j + 1) begin
            #`clk_period;
        end
    end

    pedestrian_button = 1;

    #`clk_period;

    pedestrian_button = 0;

    $stop;
end
```

Перші два рядки коду ініціалізують змінну “clk” як 1, та встановлюють безкінечний цикл, який змінює стан “clk” з періодом, вказаним у параметрі “#(`clk_period/2)”, тобто половину від періоду “clk”.

Далі, за допомогою блоку “initial begin”, задаються початкові значення для змінних “rst” та “pedestrian_button”. Затім, за допомогою команди “#`clk_period”, встановлюється затримка, що дорівнює періоду “clk”.

Після цього, спочатку змінна “rst” встановлюється у значення 1 на протязі одного періоду “clk”, після чого знову встановлюється в 0 з такою ж затримкою, після чого ця ж операція відбувається зі змінною “pedestrian_button”.

Згодом виконується цикл, в якому описана затримка двадцяти періодів “clk”, для того щоб знову про симулювати натискання кнопки “pedestrian_button”. Далі виконуються аналогічні дії тільки затримка збільшена на шістдесят періодів. Останній рядок “$stop” зупиняє симуляцію після закінчення виконання Test Bench`у. Таким чином, даний код

використовується для тестування модуля “traffic_light_pbutton”, який був описаний раніше.

2.3.3 Результати роботи дорожнього світлофору в середовищі ModelSim

Для демонстрації роботи дорожнього світлофору потрібно скомпілювати проект і завдяки попереднім налаштуванням, та реалізації Test Bench`y програма ModelSim автоматично завантажується. Виконується перенесення усіх вхідних та вихідних сигналів на екран візуального представлення сигналів (Wave), які моделюються в апаратній моделі, що в свою чергу дозволяє переглядати значення сигналів в різні моменти часу та визначати їх залежність один від одного.

Wave відображає сигнали у вигляді графіків, де горизонтальна вісь відповідає часу, а вертикальна – значенню сигналу. Графіки можуть містити декілька сигналів, які можна відобразити на одному для порівняння.[18]

Для відображення роботи розробленої схеми на екрані візуального представлення сигналів, потрібно виконати команду “run -all” та “stop” у вікні для введення команд, в результаті чого будуть промодельовані графіки усіх сигналів (рисунок 10).

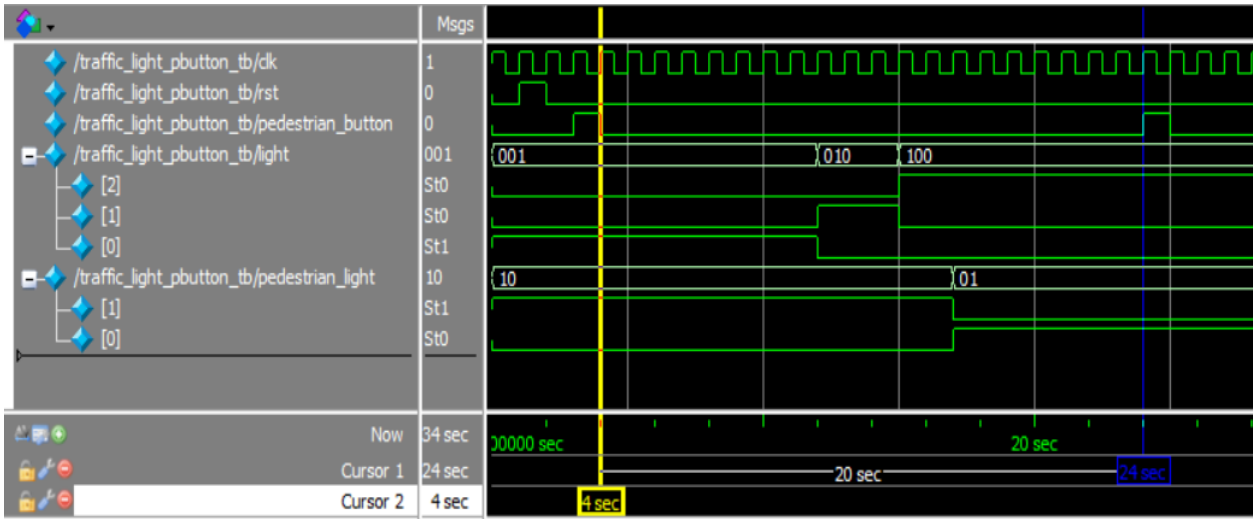

Рисунок 10 - графіки переходів станів автомату для вище описаного Test Bench`y

На вище наведеному рисунку продемонстрована перша частина роботи Test Bench. Проаналізувавши симуляцію, можна побачити один виклик сигналу скидання системи, двічі натискання кнопки, яка відповідає за бажання пішохода перетнути вулицю, та різницю між ними у часі в двадцять секунд. Особливу увагу потрібно звернути на те, що при другому натисканні кнопки автомат не змінює свій стан, а просто ігнорує натискання цієї кнопки. В результаті можна зробити висновок, на основі рисунку №9 та таблиці №3, що стани автомату перемикаються відповідно до описаних умов.

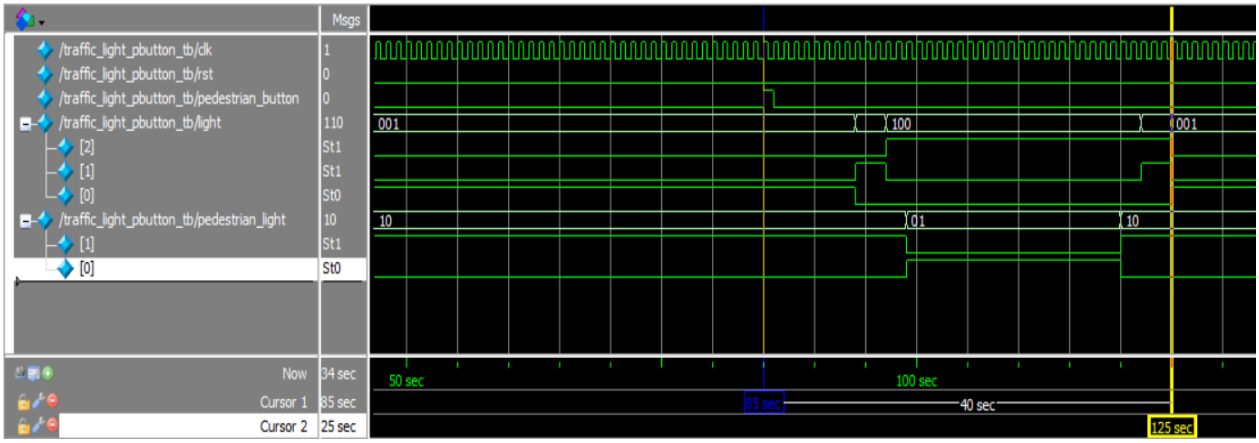

Рисунок 11 - графіки переходів станів автомату для вище описаного Test Bench`у (Продовження)

На рисунку №11 продемонстровано продовження графіків переходів станів автомату. На вісімдесят п’ятій секунді роботи була знову натиснута кнопка, яка відповідає за бажання пішохода перетнути вулицю. Важливо звернути увагу, що до натискання кнопки, автомат зберігає свій стан.

Виконавши зміни у файлі Test Bench, а саме змінивши наступні рядки коду:

```
initial begin
  rst = 0;
  pedestrian_button = 0;
  #`clk_period;
  rst = 1;
  #`clk_period;

  rst = 0;
  #`clk_period;
```

```
begin : anonymous_block

        integer k;
        for (k = 0; k < 80; k = k + 1) begin
                pedestrian_button = 1;

                #`clk_period;

                pedestrian_button = 0;

                #`clk_period;
        end
end

$stop;

end
```

Виконано повторну компіляцію проекту, та отримано нові графіки (рисунок №12), для впевненості коректності роботи реалізованого дорожнього світлофору.

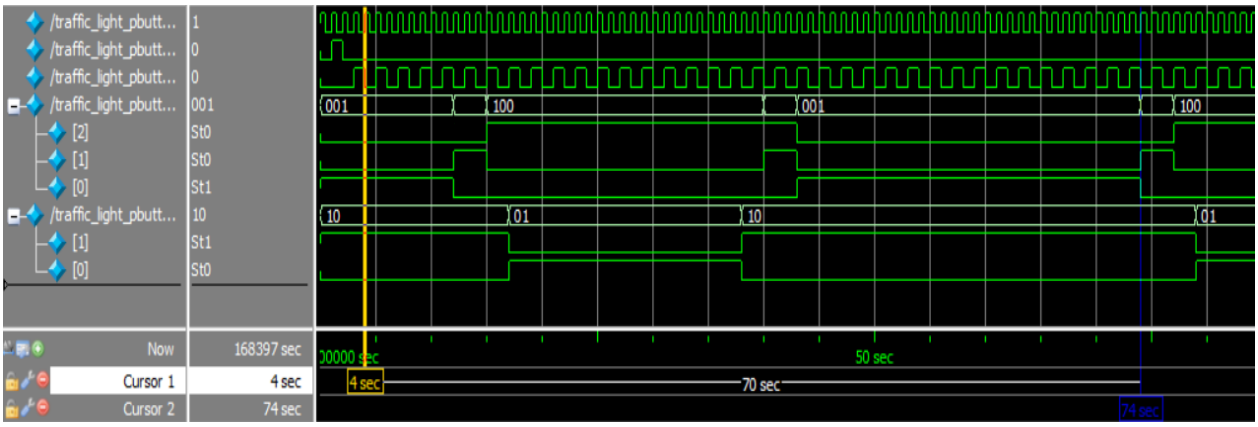

Рисунок 12 - графіки переходів станів автомату для зміненого Test Bench`y

Розглянувши продемонстровану симуляцію на рисунку №12, можна побачити постійне натиснення кнопки яка відповідає за бажання пішохода перетнути вулицю і при цьому робота схеми залишається відповідною до описаної логіки роботи.

Отже, можна зробити висновок, що робота розробленого модуля що включає в себе світлофор для авто, пішоходів та кнопку для пішоходів, працює коректно.

2.4 Моделювання роботи дорожнього перехрестя

2.4.1 Опис роботи дорожнього перехрестя

Дорожнє перехрестя - це місце, де декілька доріг перетинаються або з'єднуються. У більшості випадків дорожні перехрестя мають світлофори, які керують рухом транспорту та пішоходів.

Завдання розділу полягає в розробці Verilog моделі системи управління дорожнім перехрестям, на основі розробленого автомата, що може бути синтезований в програмовану логічну схему.

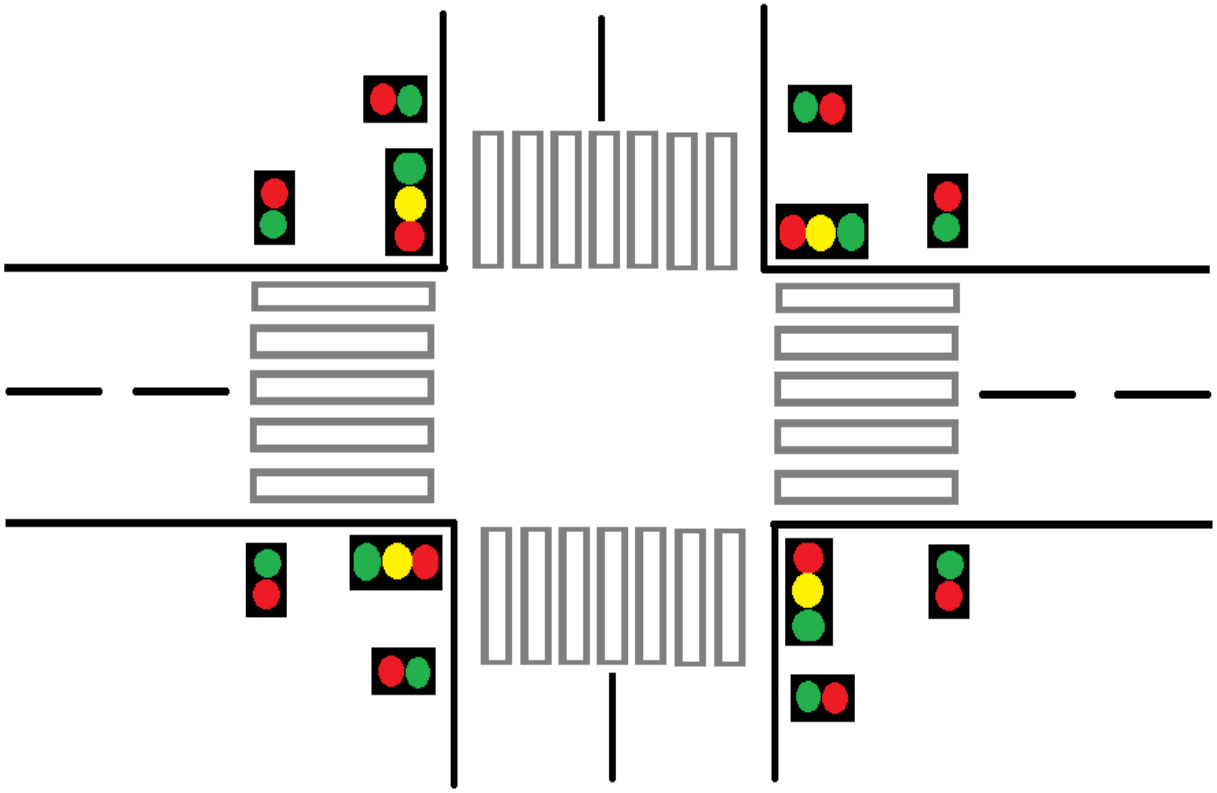

Рисунок 13 – спрощена схема дорожнього перехрестя

Схема дорожнього перехрестя яка наведена на рисунку №13 має чотири світлофори для авто, а також вісім світлофорів для пішоходів, проте автомат буде відповідати за стани світлофорів в одному напрямку руху, дорожні світлофори в напрямку північ – південь управляються одним

регістром “Light_A”, в напрямку захід – схід регістром “Light_B”. Аналогічно для пішохідних світлофорів.

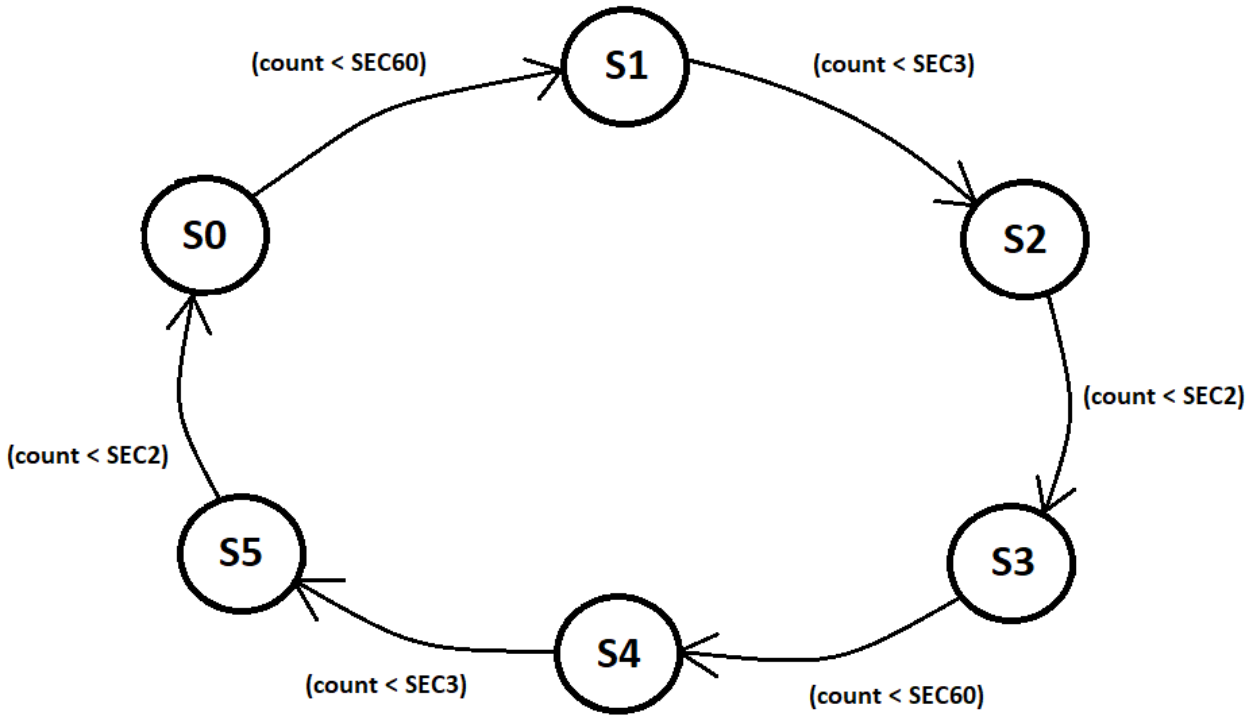

Рисунок 14 - граф логічних переходів автомату для дорожнього перехрестя

Для реалізації роботи схеми розроблено та описано граф переходів стану автомату (рисунок 14), де S0 – S5 це стани автомату, а count < SEC2 , count < SEC3 , count < SEC60 – умови переходу.

Стан автомату	Light_A	Light_B	Pedestrian_Light_A	Pedestrian_Light_B
S0	001	100	10	01
S1	010	100	10	10
S2	100	110	10	10
S3	100	001	01	10
S4	100	010	10	10
S5	110	100	10	10

Таблиця 4 - відповідність станів автомату та значень станів світлофорів

Опис роботи дорожнього перехрестя продемонстрованого вище включає в себе регістри “Light_A” та “Light_B” які визначені як 3-бітні вектори, де біти нумеруються від 2 (найбільш значущого) до 0 (найменш значущого).

Регістри “Pedestrian_light_A” та “Pedestrian_light_B” визначені як 2-бітні вектори, де біти нумеруються від 1 (найбільш значущого) до 0 (найменш значущого). Обидва регістри оголошені як “output reg”, що означає, що вони є вихідними портами і можуть бути оновлені модулем, до якого вони належать. Регістр “Pedestrian_light_A” відповідає за роботу пішохідних світлофорів в напрямку захід – схід, відповідно “Pedestrian_light_B” – за роботу в напрямку північ – південь.

Кожен з описаних станів відповідає за колір світлофору. Для автомобільного світлофору є чотири різних значення, це 001 – зелений колір, 010 – жовтий колір, 100 – червоний колір та 110 – червоно-жовтий колір відповідно. Для пішохідного світлофору є всього два стани, а саме 01 – зелений колір та 10 – червоний.

Робота реалізованого дорожнього перехрестя відбувається по передньому фронту зміни значення вхідного сигналу clk. Присутній також вхідний сигнал скидання автомату у початковий стан - rst. Таймер count, який відповідає за перехід станів автомату, при кожному новому стані оновлюється і приймає значення рівне нулю.

2.4.2 Створення Test Bench для моделювання роботи перехрестя у ModelSim

Для реалізації Test Bench`а роботи дорожнього перехрестя описаного в попередньому пункті було створено файл - traffic_light_with_pedestrian_tb, його код продемонстровано у додатку Г. Модуль

traffic_light_with_pedestrian_tb містить у собі два вхідні сигнали clk та rst, та чотири вихідні сигнали:

```
reg clk;
reg rst;

wire [2:0] light_A;
wire [2:0] light_B;
wire [1:0] pedestrian_light_A;
wire [1:0] pedestrian_light_B;
```

Змінні “Light_A” та “Light_B” є векторами з трьох бітів, та використовуються для керування набором світлофорів для авто. Відповідно “Pedestrian_Light_A” та “Pedestrian_Light_B” – вектори з двох бітів, та використовуються для керування світлофорів для пішоходів. Кожен біт векторів відповідає за свій сигнал світлофору.

На початку файлу оголошено - `define clk_period 1000000000.

Це значення періоду вхідного сигналу clk яке рівне 1000000000 наносекунд, або одній секунді. Основною частиною Test Bench`у є блок коду наведений нижче, в якому описується логіка роботи автомату.

```
initial clk = 1;
always #(`clk_period/2) clk = ~clk;

initial begin
rst = 0;
#`clk_period;

rst = 1;
#`clk_period;

rst = 0;
#`clk_period;

$stop;
end
```

В наведеному блоці коду для модуляції роботи дорожнього перехрестя, протягом трьох періодів роботи автомату, він скидає своє значення в встановлені початкові, та відпрацьовує відповідно до описаного алгоритму роботи.

2.4.3 Результати роботи перехрестя в середовищі ModelSim

За для демонстрації роботи розробленого автомату насамперед компілюємо проект в середовищі Quartus, в результаті автоматично завантажується середовище ModelSim та відображаються усі сигнали на екран візуального представлення сигналів (Wave). Результат роботи дорожнього перехрестя для Test Bench`у наведеного в додатку Г, має наступний вигляд (рисунок №15).

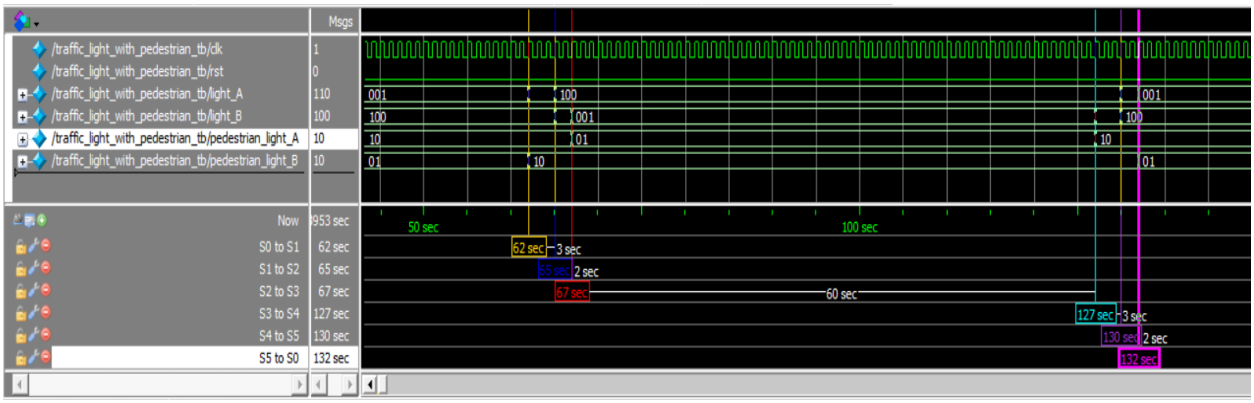

Рисунок 15 - графіки переходів станів автомату дорожнього перехрестя

Проаналізувавши симуляцію на рисунку №15 можна побачити один з циклів роботи автомату дорожнього перехрестя.

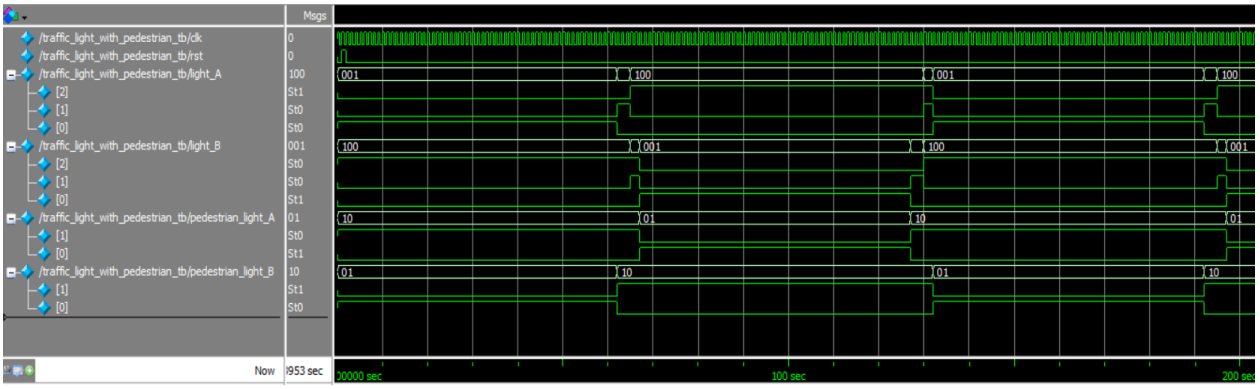

Рисунок 16 - декілька циклів переходів станів автомату

На основі графу логічних переходів автомату (рисунок №14) та відповідності станів автомату та значень станів світлофорів (таблиця №4)

зроблено висновок, що розроблений модуль дорожнього перехрестя працює абсолютно правильно.

2.5 Проектування та моделювання роботи ускладненого дорожнього перехрестя

2.5.1 Проектування та опис роботи ускладненого дорожнього перехрестя

Ускладнене дорожнє перехрестя - це перехрестя на дорозі, яке має складну геометрію, та вимагає від водіїв високої концентрації та уваги через наявність, різних напрямків руху, розгалужень, великої кількості знаків та сигнальних систем. Такі перехрестя можуть призводити до заторів, небезпеки дорожнього руху та аварій, якщо водії не дотримуються правил дорожнього руху або не уважно слідкують за дорожньою обстановкою.

Завдання розділу полягає в розробці Verilog моделей системи управління ускладненим дорожнім перехрестям, на основі спроектованого автомата, що може бути синтезований в програмовану логічну схему.

Розроблена схема ускладненого дорожнього перехрестя (рисунок №17) включає в себе шість світлофорів для авто, з яких два мають додаткові секції дозволу повороту праворуч, а також вісім світлофорів для пішоходів. Для опису роботи введені умовні позначення для автомобільних світлофорів TL1 (Traffic Light №1) – TL6 (Traffic Light №6), та PD1 (Pedestrian №1), PD2, PD5, PD6 для пішохідних відповідно. Перехрестя має наступний алгоритм роботи:

1. У першому стані автомату S0 автомобільні світлофори TL1, TL3 та допоміжний для повороту праворуч TL2R світиться зеленим кольором, TL2, TL4, TL5 та TL6 в цей час світиться червоним, TL5R вимкнуто. Пішохідні світлофори PD2 та PD6 світяться зеленим кольором, PD1 та PD5 – червоним. Час роботи – шістдесят секунд.

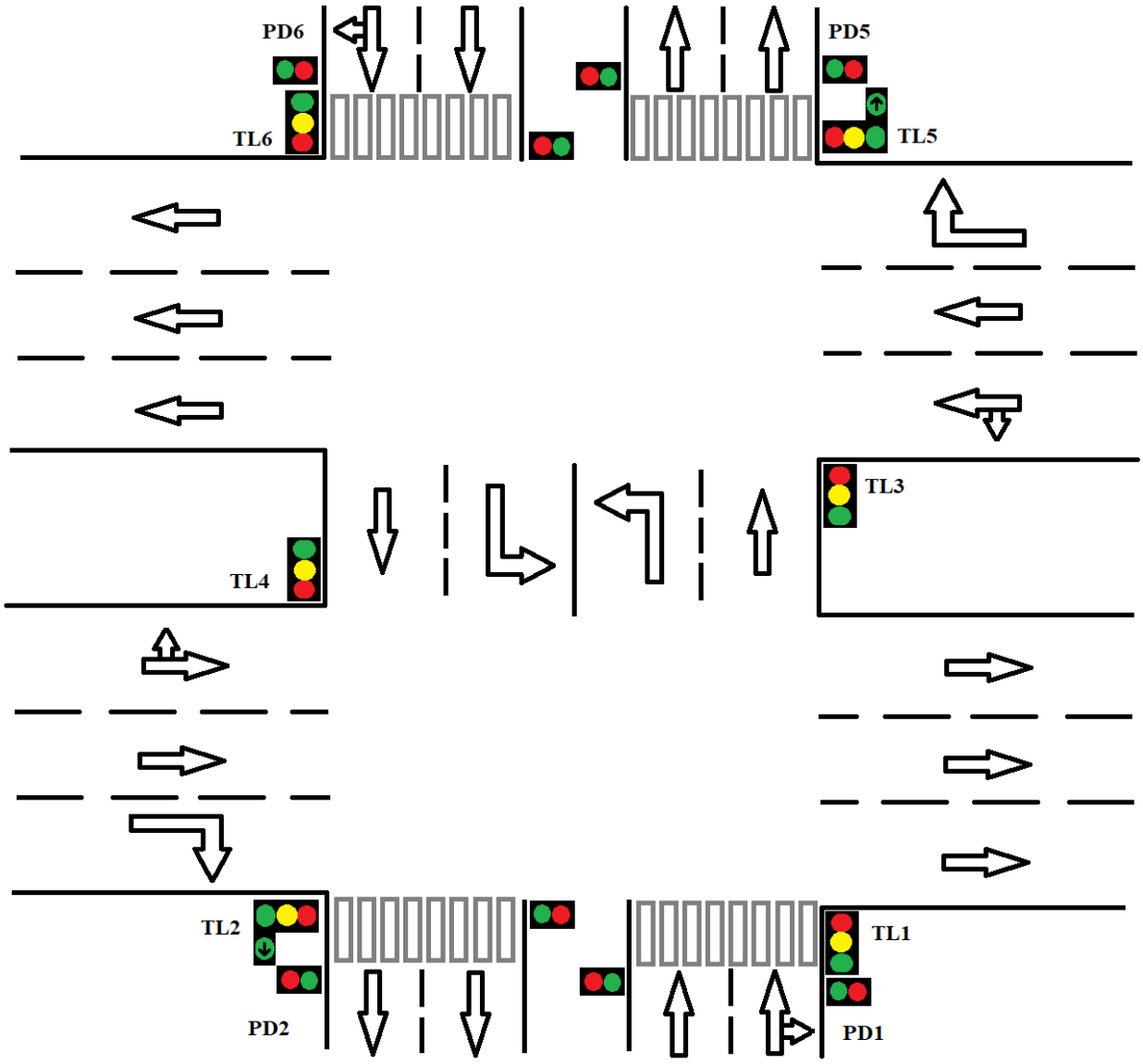

Рисунок 17 - ускладнена схема дорожнього перехрестя

2. Другій стан автомату S1 - автомобільні світлофори TL2, TL4, TL5 та TL6 світяться червоним світлом, TL3 та TL2R – зеленим, TL5R – вимкнутий, TL1 – має жовтий колір. Пішохідні світлофори PD2 та PD6 світяться зеленим кольором, PD1 та PD5 – червоним. Час роботи – дві секунди.
3. Третій стан автомату S2 - автомобільні світлофори TL1, TL2, TL4, TL5 та TL6 світяться червоним світлом, TL3 – жовтим, TL2R та TL5R – вимкнуті. Усі пішохідні світлофори світяться червоним кольором. Час роботи – дві секунди.

4. Четвертий стан автомату S3 - автомобільні світлофори TL1, TL2, TL3 та TL5 світяться червоним світлом. TL4 та TL6 – червоно-жовтим, TL2R – вимкнутий, TL5R – має зелений колір. Усі пішохідні світлофори горять червоним кольором. Час роботи – дві секунди.
5. П’ятий стан автомату S4 - автомобільні світлофори TL1, TL2, TL3 та TL5 світяться червоним світлом. TL4, TL6 та TL5R – зеленим, TL2R – вимкнутий. Пішохідні світлофори PD1 та PD5 світяться зеленим кольором, PD2 та PD6 – червоним. Час роботи – шістдесят секунд.
6. Шостий стан автомату S5 - автомобільні світлофори TL1, TL2, TL3 та TL5 світяться червоним світлом, TL4 та TL5R – зеленим, TL6 – жовтим, TL2R – вимкнутий. Пішохідні світлофори PD1 та PD5 світяться зеленим кольором, PD2 та PD6 – червоним. Час роботи – дві секунди.
7. Сьомий стан автомату S6 - автомобільні світлофори TL1, TL2, TL3, TL5 та TL6 світяться червоним світлом, TL4 – жовтим, TL2R та TL5R – вимкнуті. Усі пішохідні світлофори світяться червоним кольором. Час роботи – дві секунди.
8. Восьмий стан автомату S7 - автомобільні світлофори TL1, TL3, TL4 та TL6 світяться червоним світлом, TL2 та TL5 – червоно-жовтим, TL2R та TL5R – вимкнуті. Усі пішохідні світлофори світяться червоним кольором. Час роботи – дві секунди.
9. Дев’ятий стан автомату S8 - автомобільні світлофори TL1, TL3, TL4 та TL6 світяться червоним світлом, TL2, TL5, TL2R та TL5R – зеленим. Усі пішохідні світлофори світяться також зеленим кольором. Час роботи – шістдесят секунд у не пікові години роботи, та сто двадцять у пікові.
10. Десятий стан автомату S9 - автомобільні світлофори TL1, TL3, TL4 та TL6 світяться червоним світлом, TL2 та TL5 - жовтим, TL2R та

TL5R – вимкнуті. Усі пішохідні світлофори світяться червоним кольором. Час роботи – дві секунди.

11. Одинадцятий стан автомату S10 – усі світлофори світяться червоним світлом, TL2R та TL5R – вимкнуті. Час роботи – дві секунди.
12. Останній дванадцятий стан автомату S11 - автомобільні світлофори TL2, TL4, TL5 та TL6 світяться червоним світлом, TL1 та TL3 – червоно-жовтим, TL2R та TL5R – вимкнуті. Усі пішохідні світлофори світяться червоним кольором. Час роботи – дві секунди.

Автомобільні світлофори визначені як 3-бітні вектори де значення бітів відповідають кольору світлофору. 001 – зелений колір, 010 – жовтий колір, 100 – червоний колір та 110 – червоно-жовтий колір відповідно. Пішохідний світлофор, та допоміжні автомобільні визначені як 2-бітні вектори, де для пішохідного світлофору 01 – зелений колір та 10 – червоний, а для допоміжного автомобільного 01 – зелений, та zz – виключений.

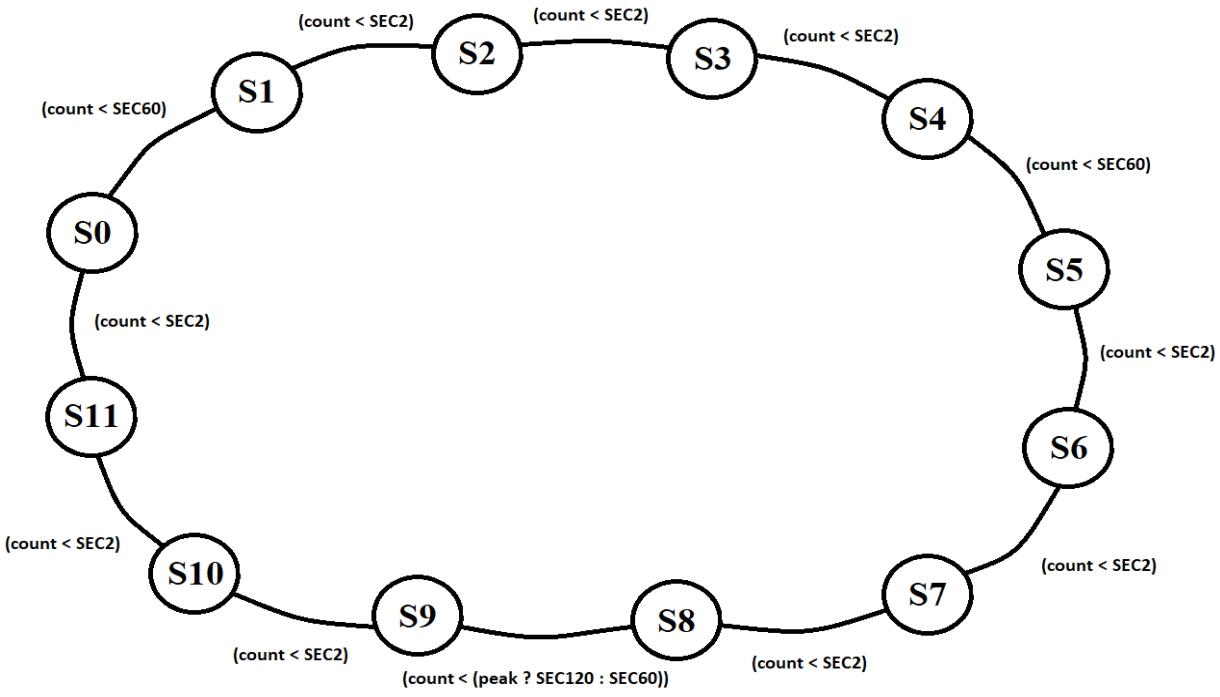

Рисунок 18 - граф логічних переходів автомату для ускладненого дорожнього перехрестя

Для реалізації роботи схеми розроблено та описано граф переходів стану автомату (рисунок 18), де S0 – S11, це стани автомату, а count < SEC2 , count < SEC60 та count < (peak ? SEC120 : SEC60) – умови переходу.

В описаному алгоритмі роботи присутнє твердження пікові та не пікові години роботи. Для реалізації зміни умов автомату, було реалізовано модуль двадцяти чотирьох форматного годинника, який працює змінюючи своє значення по вхідному сигналу clk. Так як значення clk рівне одній секунді, кожного разу збільшується значення секунди на одиницю поки не буде рівним шістдесяти, при цьому значення секунд анулюється, а значення хвилин збільшується на одиницю, аналогічно відбувається з хвилинами. Коли значення годин досягає значення двадцять чотири, годинник повністю анулюється.

```
always @(posedge(clk) or posedge(reset))
   begin
      if(reset == 1'b1) begin
         seconds = 0;
         minutes = 0;
         hours = 0;
                        end
      else if(clk == 1'b1) begin
         seconds = seconds + 1;
         if(seconds == 60) begin
            seconds = 0;
            minutes = minutes + 1;
            if(minutes == 60) begin
               minutes = 0;
               hours = hours + 1;
              if(hours ==  24) begin
                  hours = 0;
               end
            end
         end
      end
   end
```

А також модуль визначення пікових годин руху. Піковими годинами руху визначено період з 07:00 до 09:00, 12:00 – 14:00, 17:00 – 19:00.

```
always @(posedge(clk))
          begin
      if(((hours >= 7) && (hours <= 9)) || ((hours >= 12) && (hours <= 14)) || ((hours >= 17) && (hours <= 19)))
begin
                                        peak <= 1;
                        end else begin
```

```
                peak <= 0;
            end
        end
```

У наведені години зміна стану автомату з S8 в S9 збільшено з шістдесяти до сто двадцяти секунд.

Структура проекту для роботи вище згаданого автомату має наступний вигляд (рисунок №19):

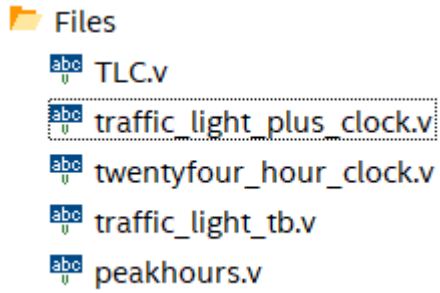

Рисунок 19 - структура проекту

Головний файл TLC.v відповідає за усю роботу програми, в ньому оголошуються усі вхідні та вихідні сигнали проекту.

```
input clk,
input reset,
output [2:0] TL1,
output [2:0] TL2,
output [1:0] TL2R,
output [2:0] TL3,
output [2:0] TL4,
output [2:0] TL5,
output [1:0] TL5R,
output [2:0] TL6,
output [1:0] PdL1,
output [1:0] PdL2,
output [1:0] PdL5,
output [1:0] PdL6,
output peak,
output [4:0] hours,
output [5:0] minutes,
output [5:0] seconds
```

А також усі описані модулі проекту, що в свою чергу дозволяє файлам (окремим модулям) взаємодіяти між собою.

```
twentyfour_hour_clock CLOCK(
        .clk(clk),
        .reset(reset),
        .hours(hours),
        .minutes(minutes),
```

```
        .seconds(seconds)
);

traffic_light_plus_clock TRAFFIC_LIGHT(
        .clk(clk),
        .reset(reset),
        .TL1(TL1),
        .TL2(TL2),
        .TL2R(TL2R),
        .TL3(TL3),
        .TL4(TL4),
        .TL5(TL5),
        .TL5R(TL5R),
        .TL6(TL6),
        .PdL1(PdL1),
        .PdL2(PdL2),
        .PdL5(PdL5),
        .PdL6(PdL6),
        .peak(peak)
);

peakhours PEAK(
        .clk(clk),
        .hours(hours),
        .peak(peak)
);
```

2.5.2 Створення та опис Test Bench для моделювання роботи перехрестя у ModelSim

Для реалізації Test Bench`а роботи ускладненого дорожнього перехрестя описаного в попередньому пункті було створено файл - traffic_light_ tb, його код продемонстровано у додатку З.

Модуль розробленого Test Bench`у описує усі вхідні та вихідні сигнали проекту, сигнали світлофорів для авто та пішоходів, пікові години роботи, а також години, хвилини та секунди для реалізованого годинника.

```
reg clk;
reg reset;

wire [2:0] TL1;
wire [2:0] TL2;
wire [1:0] TL2R;
wire [2:0] TL3;
wire [2:0] TL4;
wire [2:0] TL5;
wire [1:0] TL5R;
wire [2:0] TL6;
wire [1:0] PdL1;
wire [1:0] PdL2;
wire [1:0] PdL5;
wire [1:0] PdL6;
wire peak;
wire [4:0] hours;
```

```
wire [5:0] minutes;
wire [5:0] seconds;
```

Взаємодія Test Bench`у відбувається з головним модулем програми, а саме з TLC.

```
TLC Tlc (
    .clk(clk),
    .reset(reset),
    .TL1(TL1),
    .TL2(TL2),
    .TL2R(TL2R),
    .TL3(TL3),
    .TL4(TL4),
    .TL5(TL5),
    .TL5R(TL5R),
    .TL6(TL6),
    .PdL1(PdL1),
    .PdL2(PdL2),
    .PdL5(PdL5),
    .PdL6(PdL6),
    .peak(peak),
    .hours(hours),
    .minutes(minutes),
    .seconds(seconds)
);
```

Можлива також реалізація Test Bench`ів для кожного окермого модуля, якщо потрібно перевірити якусь маленьку частину проекту, в такому випадку потрібно викликати потрібний модуль, та передавати необхідні сигнали.

На початку файлу оголошено - `define clk_period 1000000000.

Це значення періоду вхідного сигналу clk, яке рівне 1000000000 наносекунд, або одній секунді. Основною частиною Test Bench`у є блок коду наведений нижче, в якому описується логіка роботи автомату.

```
 initial clk = 1;
always #(`clk_period/2) clk = ~clk;

initial begin
rst = 0;
#`clk_period;

rst = 1;
#`clk_period;

rst = 0;
#`clk_period;
$stop;
end
```

В наведеному блоці коду для модуляції роботи дорожнього перехрестя, протягом трьох періодів роботи автомату, він скидає своє значення в встановлені початкові, та відпрацьовує відповідно до описаного алгоритму роботи.

2.5.3 Результати роботи ускладненого дорожнього перехрестя

Після компіляції проекту в середовищі Quartus, та завантаження ModelSim, відображаються усі сигнали на екран візуального представлення сигналів (Wave). Результат роботи дорожнього перехрестя для Test Bench`у наведеного в додатку З, має наступний вигляд (рисунок №20).

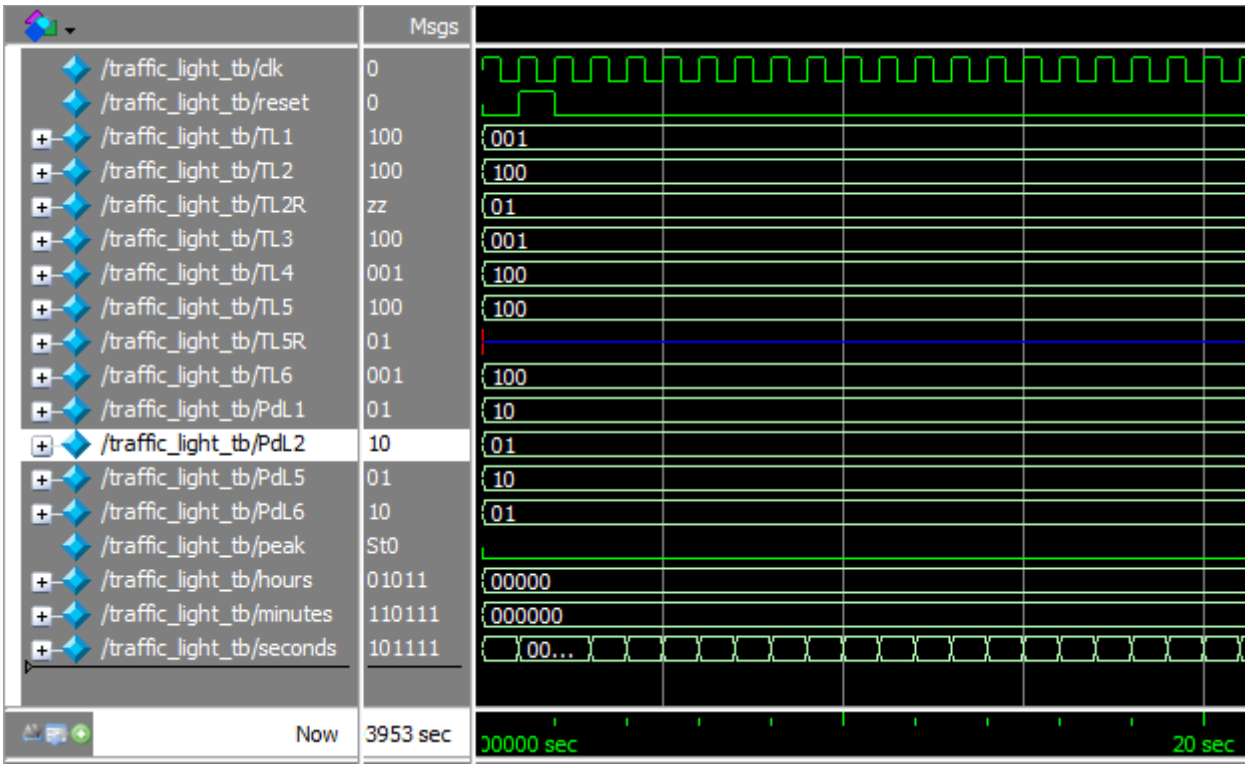

Рисунок 20 - Графіки початкового стану автомату

Після спрацювання сигналу reset, протягом шістдесяти секунд автомат зберігає свій стан S0. В описаному алгоритмі роботи автомату для стану S0 автомобільні світлофори TL1, TL3 та допоміжний для повороту праворуч TL2R світиться зеленим кольором (стан 001), TL2, TL4, TL5 та TL6 в цей час світиться червоним (100), TL5R вимкнуто (zz). Пішохідні світлофори PD2 та

PD6 світяться зеленим кольором (01), PD1 та PD5 – червоним (10). Час роботи – шістдесят секунд.

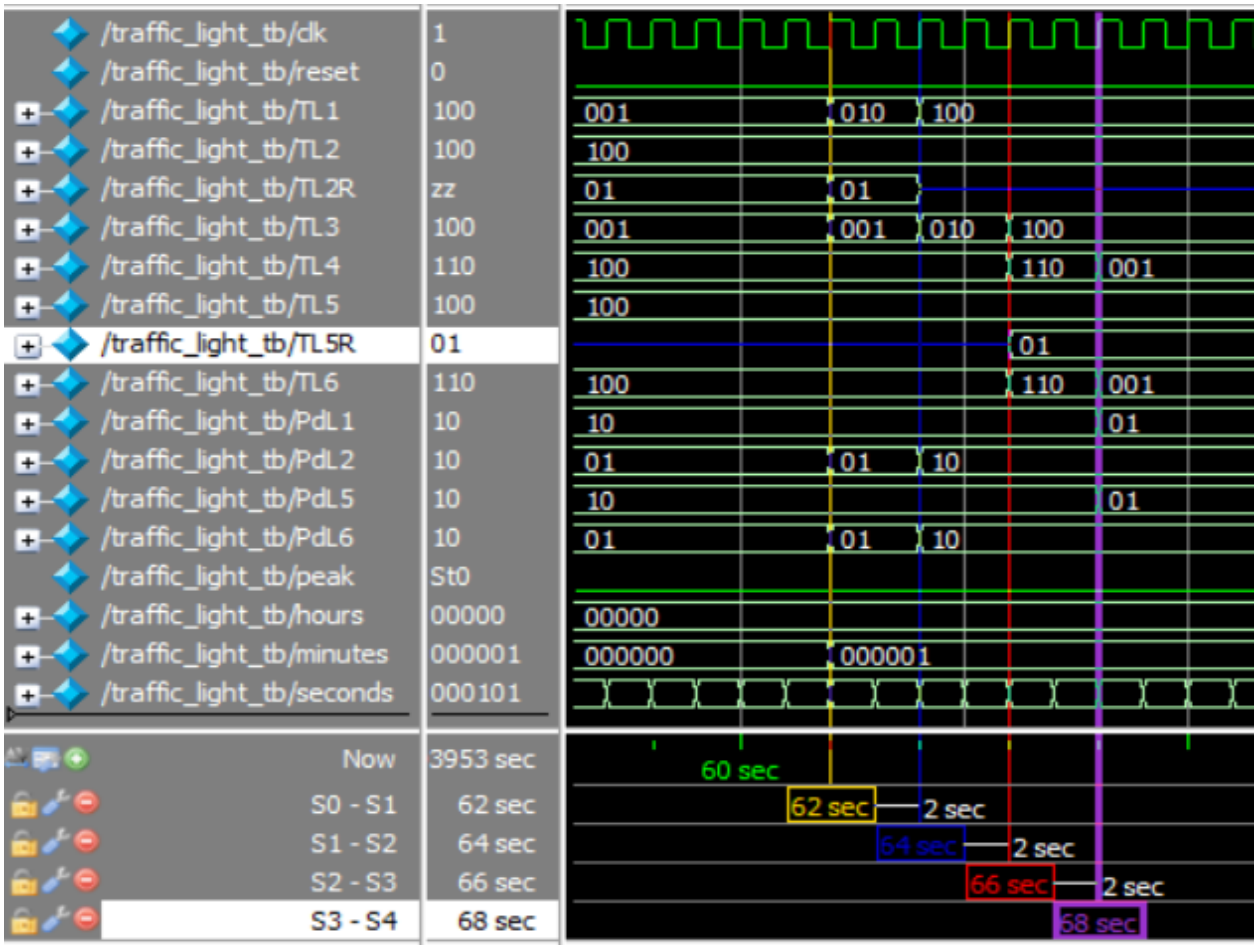

Рисунок 21 - графіки переходів станів автомату S0 - S4

Наступні графіки переходів станів автомату S0 – S4 (рисунок №21), та S4 – S8 (рисунок №22), S8 – S0 (рисунок №23) повністю дотримуються алгоритму роботи (таблиця 5).

Стан автомату	TL1	TL2	TL2R	TL3	TL4	TL5	TL5R	TL6	PD1	PD2	PD5	PD6
S0	001	110	01	001	100	100	zz	100	10	01	10	01
S1	010	100	01	001	100	100	zz	100	10	01	10	01
S2	100	100	zz	010	100	100	zz	100	10	10	10	10

S3	100	100	zz	100	110	100	01	110	10	10	10	10
S4	100	100	zz	100	001	100	01	001	01	10	01	10
S5	100	100	zz	100	001	100	01	010	01	10	01	10
S6	100	100	zz	100	010	100	zz	100	10	10	10	10
S7	100	110	zz	100	100	110	zz	100	10	10	10	10
S8	100	001	01	100	100	001	01	100	01	01	01	01
S9	100	010	zz	100	100	010	zz	100	10	10	10	10
S10	100	100	zz	100	100	100	zz	100	10	10	10	10
S11	110	100	zz	110	100	100	zz	100	10	10	10	10

Таблиця 5 - відповідність станів автомату та значень станів світлофорів

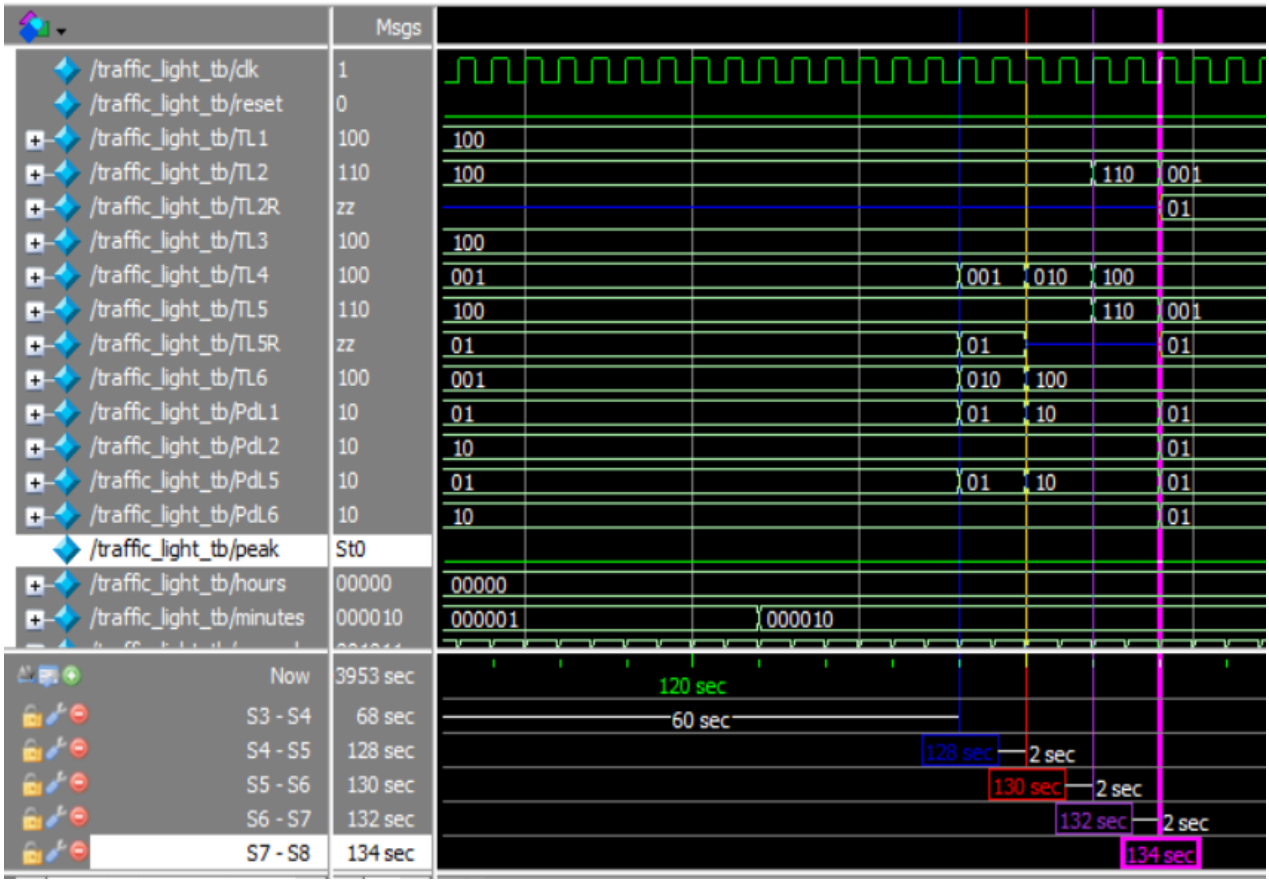

Рисунок 22 - графіки переходів станів автомату S4 - S8

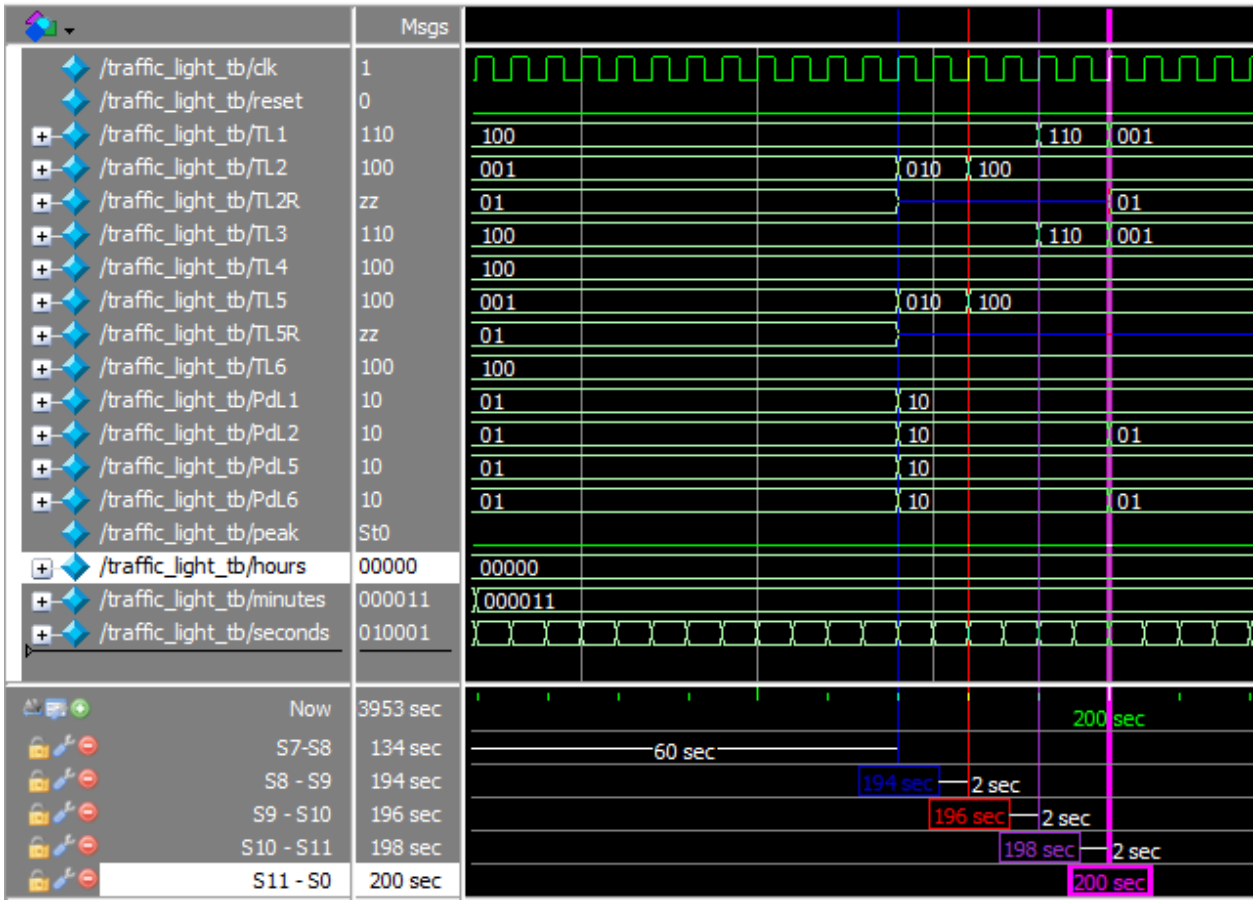

Рисунок 23 - графіки переходів станів автомату S8 - S0

Симуляція роботи годинника для відстеження пікових годин визначається за допомогою п’яти бітного регістру для годин та шести бітних регістрів для хвилин та секунд. На початку симуляції роботи годинник починає свою роботу з значення 00:00:00, а значення peak приймає значення рівне 0. На 200 секунді роботи програми значення годинника рівне 197 секунді (значення отримане після переведення з двійкової системи в десяткову), тому що на початку програми подається на вхід сигнал reset, який перезавантажує роботу програми. При першій піковій годині роботи 07:00:00 на годиннику, а це значення 00111 для п’яти бітного регістру годин, значення peak стає рівним 1, при цьому умова затримки для перемикання автомату з стану S8 в S9 змінюється з 60 на 120 секунд (рисунок №24).

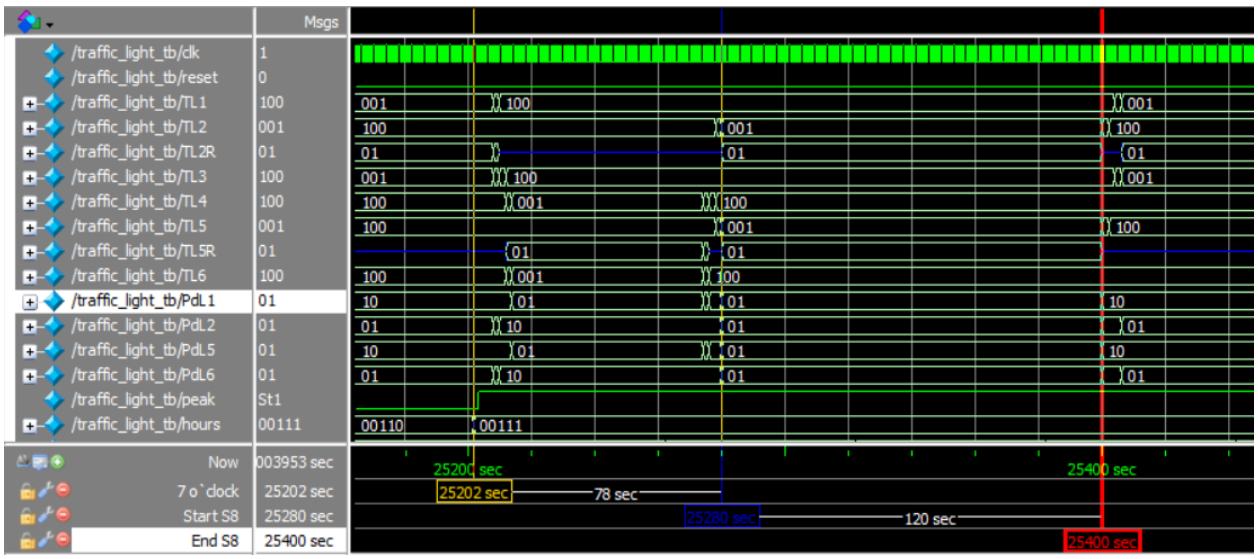

Рисунок 24 - зміна роботи автомату у пікові години

Проаналізувавши усі наведені результати симуляції ускладненого дорожнього світлофора, зроблено висновок, що модуляція в середовищі ModelSim є дуже ефективною для аналізу помилок в роботі автомату, а сам автомат працює згідно описаним для нього умовам роботи.

ВИСНОВКИ

В дипломній синтезовано набір концептуально різних електричних схем, функціональність та логіка роботи яких основані на скінчених автоматах Мура та Мілі, що дозволило сформулювати наступні висновки:

1. Порівняння симуляторів від виробників ПЛІС показало, що середовище симуляції ModelSim, що входить до середовища розробки Intel Quartus Prime, є одним з найбільш популярних систем симуляції, має широкий набір функцій, вбудовані засоби аналізу коду та пошуку помилок. Недоліками ModelSim є те, що він не підтримує інші операційні системи окрім Windows, зокрема Mac OS та Linux. Другим недоліком ModelSim є те, що він є неефективним для роботи зі складними та великими проектами. Vivado Simulator - це симулятор виробника Xilinx, має широкі можливості для візуалізації результатів симуляції, має кілька переваг перед симулятором ModelSim, зокрема, підтримує функції багатопоточності, оптимізації використання пам'яті та інші технології для ефективної роботі зі складними проектами. Недоліком Vivado Simulator є те, що інтерфейс користувача є складним для розуміння, особливо для початківців. Тому для синтезу електронних схем в роботі обрано і використано як інструмент симуляції саме програму ModelSim.

2. Порівняння мов опису обладнання показало, що Verilog та System Verilog є менш формальними мовами ніж мова VHDL, що дає можливість розробникам швидше створювати прототипи та приступати до верифікації розроблених моделей. Мова Verilog має простіший синтаксис, що робить його легшим у вивченні та використанні, її перевагою є наявність атрибутів, які дозволяють змінювати функціональність моделей залежно від вхідних даних. В свою чергу, мова SystemVerilog розроблена як розширення мови Verilog, але є набагато складнішою у порівнянні з VHDL та Verilog, не

є стандартизованою, та має більший обсяг коду для виконання однакових завдань, тому в роботі усі схеми синтезовані мовою Verilog.

3. В роботі продемонстровано, що використання мови Verilog та середовища проектування Inter Quartus Prime з симулятором ModelSim дозволяю інженерам-розробникам синтезувати комбінаційні та секвенційні електронні схеми практично довільної складності, з перспективою запрограмувати електронну схему в контролер ПЛІС і у такий отримати повнофункціональний апаратний прилад, готовий до використання у промислових масштабах. Як приклад, в роботі синтезовано три різні електронні схеми для управління дорожними світлофорами, які можуть бути використані організації дорожнього руху на складних перехрестях. Перша схема працює в залежності від зовнішнього вхідного сигналу - завки від пішохода, який бажає перейти дорогу, друга схема вхідних сигналів не має, третя схема керує світлофорами для складного перехрестя, що включає у себе три автомобільні дороги та шість пішохідних переходів.

4. Для кожної електронної схеми застосовано програмний механізм перевірки TestBench, який дозволяє перевірити швидкодію роботи схеми, а також стабільність і коректність перемикання станів електронної схеми як скінченого автомату. Електронна схема для організації руху ускладненого дорожнього перехрестя синтезована у вигляді одного головного об'єднуючого Verilog-файлу та допоміжних Verilog-файлів, які використовують спеціальний програмний механізм для обміну сигналами між собою. Таким чином, основним результатом роботи є синтезовані електронні цифрові схеми стандартизованою мовою Verilog для трьох різних секвенційних пристроїв з логікою роботи скінчених автоматів Мура та Мілі.

ПЕРЕЛІК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Технології проектування комп'ютерних систем [Електронний ресурс] / [В. А. Лахно, Б. С. Гусєв, В. В. Смолій та ін.]. – 2019. – Режим доступу до ресурсу: https://nubip.edu.ua/sites/default/files/u34/posibnik_-_tehnologiyi_proektuvannya_kompyuternih_sistem.pdf.
2. ASIC or FPGA? Each solution has Advantaches and Disadvantages [Електронний ресурс] – Режим доступу до ресурсу: https://www.swindonsilicon.com/asic-fpga-advantages-and-disadvantages/
3. Ian Kuon, Russell Tessier and Jonathan Rose (2008), "FPGA Architecture: Survey and Challenges"
4. Šćekić O. FPGA Comparative Analysis [Електронний ресурс] / Ognjen Šćekić. – 2005. – Режим доступу до ресурсу: http://home.etf.rs/~vm/os/vlsi/razno/Altera%20vs%20Xilinx%20%28Scekic%29%20.pdf.
5. Xilinx FPGA vs Altera FPGA: What Is the Difference? [Електронний ресурс] – Режим доступу до ресурсу: https://www.nextpcb.com/blog/xilinx-fpga-vs-altera-fpga.
6. Jacques L Athow, and Asim J Al-Khalili, "Implementation of large-integer hardware multiplier in Xilinx FPGA", Proceeding of the 15th IEEE International Conference on Electronics, Circuits and Systems (ICECS'08), St.Julien's, Sep 2008, pp.1300-1303.
7. Introduction to the Quartus® II Software [Електронний ресурс]. – 2010. – Режим доступу до ресурсу: https://courses.cs.washington.edu/courses/cse467/15wi/docs/Quartus_II_Intro.pdf.
8. Sutherland S. SystemVerilog, ModelSim, and You [Електронний ресурс] / Stuart Sutherland. – 2004. – Режим доступу до ресурсу: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=38b5fade8a473e434ecf0f6c10ed659798a242af.

9. Vivado Design Suite User Guide [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2022_2/ug893-vivado-ide.pdf.
10. Getting Started with Quartus II Simulation Using the ModelSim-Altera Software [Електронний ресурс]. – 2011. – Режим доступу до ресурсу: http://www.univasf.edu.br/~romulo.camara/novo/wp-content/uploads/2013/07/tutorial-quartus_modelsim.pdf.
11. Vivado Design Suite Tutorial [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.studocu.com/row/document/%E7%94%B5%E5%AD%90%E7%A7%91%E6%8A%80%E5%A4%A7%E5%AD%A6/digital-image-processing/ug937-vivado-design-suite-simulation-tutorial/22766511.
12. Zwagerman, Michael D., "High Level Synthesis, a Use Case Comparison with Hardware Description Language" (2015).
13. What's the Difference Between VHDL, Verilog, and SystemVerilog? [Електронний ресурс] // 2014 – Режим доступу до ресурсу: https://www.electronicdesign.com/resources/whats-the-difference-between/article/21800239/whats-the-difference-between-vhdl-verilog-and-systemverilog.
14. Mukherjee R. COMPREHENSIVE SYSTEMVERILOG-SYSTEMC-VHDL MIXED-LANGUAGE DESIGN METHODOLOGY [Електронний ресурс] / R. Mukherjee, G. Verma, S. Kakkar – Режим доступу до ресурсу: https://dvcon-proceedings.org/wp-content/uploads/comprehensive-systemverilog-systemc-vhdl-mixed-language-design-methodology.pdf.
15. Rudra Mukherjee and Sachin Kakkar, "System Verilog – VHDL Mixed Design Reuse Methodology", DVCon 2006.
16. Haskell R. Digital Design Using Digilent FPGA Boards - VHDL / Active-HDL Edition / Richard E. Haskell, Darrin M. Hanna. – LBE Books Rochester Hills, MI, 2009

17. Palnitkar S. Verilog HDL: A Guide to Digital Design and Synthesis / Samir Palnitkar., 2003. – 415 с. – (Second Edition).
18. ModelSim® GUI Reference Manual [Електронний ресурс]. – 1991. – Режим доступу до ресурсу: https://www.microsemi.com/document-portal/doc_view/136365-modelsim-me-10-4c-gui-reference-manual-for-libero-soc-v11-7.
19. Jesse op den Brown. Using ModelSim with Quartus II Block Design Files [Електронний ресурс] / Jesse op den Brown – Режим доступу до ресурсу: https://ds.opdenbrouw.nl/quartus/convertBDFtoVHDL.pdf.
20. Using the ModelSim-Intel FPGA Simulator with VHDL Testbenches [Електронний ресурс] – Режим доступу до ресурсу: http://mems.ece.dal.ca/eced4260/Testbench.pdf.

ДОДАТКИ

Додаток А

Реалізація модулю дорожнього світлофору, що включає в себе світлофор для авто, пішоходів та кнопку для пішоходів

```
module traffic_light_pbutton(
   input clk,
   input rst,
         input pedestrian_button,
   output reg [2:0] light,
         output reg [1:0] pedestrian_light);

         reg[5:0] state;
         reg[5:0] count;
         reg[5:0] pedestrian_timer_state;
         reg[2:0] pedestrian_button_state;
         localparam S0 = 6'b000001,
                                    S1 = 6'b000010,
                                          S2 = 6'b000100,
                                          S3 = 6'b001000,
                                          S4 = 6'b010000,
                                          S5 = 6'b100000;
         localparam SEC60 = 8'd60,
                                          SEC20 = 5'd20,
                                          SEC10 = 4'd10,
                                          SEC8 = 4'd8,
                                          SEC3 = 4'd3,
                                          SEC2 = 4'd2;

         always @(posedge clk or posedge rst)
                        if (rst == 1) begin
                                          state <= S0;
                                          count <= 0;
                                          pedestrian_button_state <= 0;
                                          pedestrian_timer_state <= 0;
                        end else begin
                                          if (pedestrian_button == 1 && pedestrian_timer_state == 0) begin
                                                   pedestrian_timer_state <= SEC60;
                                                   pedestrian_button_state = 1;
                                          end
                                          if (pedestrian_timer_state != 0) begin
                                                   pedestrian_timer_state <= pedestrian_timer_state - 1;
                                          end
                                          case(state)
                                                   S0: if(pedestrian_button_state == 1) begin
                                                                                    if(count < SEC8) begin
                                                                                             state <= S0;
                                                                                             count <= count + 1;
                                                                                    end else begin
                                                                                             state <= S1;
                                                                                             count <= 0;
                                                                                    end
                                                                           end else begin
                                                                                    state <= S0;
                                                                                    count <= 0;
                                                                           end

                                                   S1: if(count < SEC3) begin
                                                                                    state <= S1;
                                                                                    count <= count + 1;
```

```
                                        end else begin
                                                state <= S2;
                                                count <= 0;
                                        end

                                S2: if(count < SEC2) begin
                                                state <= S2;
                                                count <= count + 1;
                                        end else begin
                                                state <= S3;
                                                count <= 0;
                                        end

                                S3: if(count < SEC20) begin
                                                state <= S3;
                                                count <= count + 1;
                                        end else begin
                                                state <= S4;
                                                count <= 0;
                                        end

                                S4: if(count < SEC2) begin
                                                state <= S4;
                                                count <= count + 1;
                                        end else begin
                                                state <= S5;
                                                count <= 0;
                                        end

                                S5: if(count < SEC3) begin
                                                state <= S5;
                                                count <= count + 1;
                                        end else begin
                                                state <= S0;
                                                count <= 0;
                                                pedestrian_button_state <= 0;
                                        end

                                        default state <= S0;
                                endcase
                        end
        always @(*)
        begin
                case(state)
                        S0: begin light <= 3'b001; pedestrian_light <= 2'b10; end
                        S1: begin light <= 6'b010; pedestrian_light <= 2'b10; end
                        S2: begin light <= 6'b100; pedestrian_light <= 2'b10; end
                        S3: begin light <= 6'b100; pedestrian_light <= 2'b01; end
                        S4: begin light <= 6'b100; pedestrian_light <= 2'b10; end
                        S5: begin light <= 6'b110; pedestrian_light <= 2'b10; end
                default begin light <= 6'b100; pedestrian_light <= 2'b10; end
                endcase
        end
endmodule
```

Додаток Б

Test Bench для дорожнього світлофору, що включає в себе світлофор для авто, пішоходів та кнопку для пішоходів

```
`timescale 1ns / 1ps
`define clk_period 1000000000
```

```
module traffic_light_pbutton_tb();

reg clk;
reg rst;
reg pedestrian_button;
wire [2:0] light;
wire [1:0] pedestrian_light;

traffic_light_pbutton TRAFFIC_LIGHT_PBUTTON(
    .clk(clk),
    .rst(rst),
    .pedestrian_button(pedestrian_button),
    .light(light),
    .pedestrian_light(pedestrian_light)
);
initial clk = 1;
always #(`clk_period/2) clk = ~clk;

initial begin
            rst = 0;
  pedestrian_button = 0;
  #`clk_period;
  rst = 1;
  #`clk_period;
  rst = 0;
  #`clk_period;
  pedestrian_button = 1;
  #`clk_period;
  pedestrian_button = 0;
  begin : anonymous_block
      integer i;
      for (i = 0; i < 20; i = i + 1) begin
         #`clk_period;
      end
  end
  pedestrian_button = 1;
          #`clk_period;
  pedestrian_button = 0;
          begin : anonymous_block2
      integer j;
      for (j = 0; j < 60; j = j + 1) begin
         #`clk_period;
      end
  end
  pedestrian_button = 1;

  #`clk_period;

  pedestrian_button = 0;

  $stop;
end

endmodule
```

Додаток В

Реалізація модулю дорожнього перехрестя

```
module traffic_light_with_pedestrian(
            input clk,
            input rst,
            output reg [2:0] light_A,
            output reg [2:0] light_B,
            output reg [1:0] pedestrian_light_A,
```

```
output reg [1:0] pedestrian_light_B);

 reg[5:0] state;
 reg[5:0] count;

 localparam S0 = 6'b000001,
                                 S1 = 6'b000010,
                                         S2 = 6'b000100,
                                         S3 = 6'b001000,
                                         S4 = 6'b010000,
                                         S5 = 6'b100000;


 localparam SEC60 = 8'd60,
                                         SEC3 = 4'd3,
                                         SEC2 = 4'd2;


 always @(posedge clk or posedge rst)
                         if (rst == 1) begin
                                         state <= S0;
                                         count <= 1;
                         end else
                                         case(state)
                                                 S0: if(count < SEC60) begin
                                                                 state <= S0;
                                                                         count <= count + 1;
                                                         end else begin
                                                                         state <= S1;
                                                                         count <= 1;
                                                         end

                                                 S1: if(count < SEC3) begin
                                                                 state <= S1;
                                                                         count <= count + 1;
                                                         end else begin
                                                                         state <= S2;
                                                                         count <= 1;
                                                         end

                                                 S2: if(count < SEC1) begin
                                                                 state <= S2;
                                                                         count <= count + 1;
                                                         end else begin
                                                                         state <= S3;
                                                                         count <= 1;
                                                         end

                                                 S3: if(count < SEC60) begin
                                                                 state <= S3;
                                                                         count <= count + 1;
                                                         end else begin
                                                                         state <= S4;
                                                                         count <= 1;
                                                         end

                                                 S4: if(count < SEC3) begin
                                                                 state <= S4;
                                                                         count <= count + 1;
                                                         end else begin
                                                                         state <= S5;
                                                                         count <= 1;
                                                         end

                                                 S5: if(count < SEC1) begin
```

```
                                        state <= S5;
                                                count <= count + 1;
                                        end else begin
                                                state <= S0;
                                                count <= 1;
                                        end

                                default state <= S0;
                        endcase

        always @(*)
        begin
                case(state)
                                        S0: begin light_A <= 3'b001; light_B <= 3'b100; pedestrian_light_A <=
2'b10; pedestrian_light_B <= 2'b01;end
                                        S1: begin light_A <= 6'b010; light_B <= 3'b100; pedestrian_light_A <=
2'b10; pedestrian_light_B <= 2'b10;end
                                        S2: begin light_A <= 6'b100; light_B <= 3'b110; pedestrian_light_A <=
2'b10; pedestrian_light_B <= 2'b10;end
                                        S3: begin light_A <= 6'b100; light_B <= 3'b001; pedestrian_light_A <=
2'b01; pedestrian_light_B <= 2'b10;end
                                        S4: begin light_A <= 6'b100; light_B <= 3'b010; pedestrian_light_A <=
2'b10; pedestrian_light_B <= 2'b10;end
                                        S5: begin light_A <= 6'b110; light_B <= 3'b100; pedestrian_light_A <=
2'b10; pedestrian_light_B <= 2'b10;end
                        default begin light_A <= 6'b100; light_B <= 3'b100; pedestrian_light_A <= 2'b10;
pedestrian_light_B <= 2'b10;end
                        endcase
        end

endmodule
```

Додаток Г

Test Bench для дорожнього перехрестя

```
`timescale 1ns / 1ps

`define clk_period 1000000000

module traffic_light_with_pedestrian_tb();

reg clk;
reg rst;
wire [2:0] light_A;
wire [2:0] light_B;
wire [1:0] pedestrian_light_A;
wire [1:0] pedestrian_light_B;

traffic_light_with_pedestrian TRAFFIC_LIGHT_WITH_PEDESTRIAN(
		.clk(clk),
		.rst(rst),
		.light_A(light_A),
		.light_B(light_B),
		.pedestrian_light_A(pedestrian_light_A),
		.pedestrian_light_B(pedestrian_light_B)
		 );

		 initial clk = 1;
		 always #(`clk_period/2) clk = ~clk;

		 initial begin
		 rst = 0;
		 #`clk_period;

		 rst = 1;
		 #`clk_period;

		 rst = 0;
		 #(`clk_period * 200);

		 $stop;
		 end

endmodule
```

Додаток Д

Головний файл ускладненого дорожнього світлофору

```
module TLC(
	input clk,
	input reset,
	output [2:0] TL1,
	output [2:0] TL2,
	output [1:0] TL2R,
	output [2:0] TL3,
	output [2:0] TL4,
	output [2:0] TL5,
	output [1:0] TL5R,
	output [2:0] TL6,
	output [1:0] PdL1,
	output [1:0] PdL2,
	output [1:0] PdL5,
	output [1:0] PdL6,
	output peak,
	output [4:0] hours,
	output [5:0] minutes,
	output [5:0] seconds
);

twentyfour_hour_clock CLOCK(
	.clk(clk),
	.reset(reset),
	.hours(hours),
	.minutes(minutes),
	.seconds(seconds)
);

traffic_light_plus_clock TRAFFIC_LIGHT(
	.clk(clk),
	.reset(reset),
	.TL1(TL1),
	.TL2(TL2),
	.TL2R(TL2R),
	.TL3(TL3),
	.TL4(TL4),
	.TL5(TL5),
	.TL5R(TL5R),
	.TL6(TL6),
	.PdL1(PdL1),
	.PdL2(PdL2),
	.PdL5(PdL5),
	.PdL6(PdL6),
	.peak(peak)
);

peakhours PEAK(
	 .clk(clk),
	 .hours(hours),
	 .peak(peak)
);

endmodule
```

Додаток Е

Реалізація модулю ускладненого дорожнього світлофору

```
module traffic_light_plus_clock(
   input clk,
   input reset,
            input peak,
   output reg [2:0] TL1,
   output reg [2:0] TL2,
            output reg [1:0] TL2R,
            output reg [2:0] TL3,
            output reg [2:0] TL4,
            output reg [2:0] TL5,
            output reg [1:0] TL5R,
            output reg [2:0] TL6,
            output reg [1:0] PdL1,
            output reg [1:0] PdL2,
            output reg [1:0] PdL5,
            output reg [1:0] PdL6);

            reg[5:0] state;
            reg[7:0] count;
            localparam S0 = 4'b0000,
                                    S1 = 4'b0001,
                                            S2 = 4'b0010,
                                            S3 = 4'b0011,
                                            S4 = 4'b0100,
                                            S5 = 4'b0101,
                                            S6 = 4'b0110,
                                            S7 = 4'b0111,
                                            S8 = 4'b1000,
                                            S9 = 4'b1001,
                                            S10 = 4'b1010,
                                            S11 = 4'b1011;
            localparam SEC120 = 8'd120,
                                            SEC60 = 8'd60,
                                            SEC2 = 4'd2;
            always @(posedge clk or posedge reset)
                                    if (reset == 1) begin
                                            state <= S0;
                                            count <= 1;
                                    end else
                                            case(state)
                                                    S0: if(count < SEC60) begin
                                                                    state <= S0;
                                                                            count <= count + 1;
                                                            end else begin
                                                                    state <= S1;
                                                                    count <= 1;
                                                            end

                                                    S1: if(count < SEC2) begin
                                                                    state <= S1;
                                                                            count <= count + 1;
                                                            end else begin
                                                                    state <= S2;
                                                                    count <= 1;
                                                            end

                                                    S2: if(count < SEC2) begin
                                                                    state <= S2;
                                                                            count <= count + 1;
                                                            end else begin
```

```
                        state <= S3;
                        count <= 1;
            end

S3: if(count < SEC2) begin
                    state <= S3;
                        count <= count + 1;
            end else begin
                        state <= S4;
                        count <= 1;
            end

S4: if(count < SEC60) begin
                    state <= S4;
                        count <= count + 1;
            end else begin
                        state <= S5;
                        count <= 1;
            end

S5: if(count < SEC2) begin
                    state <= S5;
                        count <= count + 1;
            end else begin
                        state <= S6;
                        count <= 1;
            end

S6: if(count < SEC2) begin
                    state <= S6;
                        count <= count + 1;
            end else begin
                        state <= S7;
                        count <= 1;
            end

S7: if(count < SEC2) begin
                    state <= S7;
                        count <= count + 1;
            end else begin
                        state <= S8;
                        count <= 1;
            end

S8: if(count < (peak ? SEC120 : SEC60)) begin
                    state <= S8;
                        count <= count + 1;
            end else begin
                        state <= S9;
                        count <= 1;
            end

S9: if(count < SEC2) begin
                    state <= S9;
                        count <= count + 1;
            end else begin
                        state <= S10;
                        count <= 1;
            end

S10: if(count < SEC2) begin
                    state <= S10;
                        count <= count + 1;
            end else begin
                        state <= S11;
```

```
                                        count <= 1;
                                end

                        S11: if(count < SEC2) begin
                                        state <= S11;
                                        count <= count + 1;
                                end else begin
                                        state <= S0;
                                        count <= 1;
                                end

                        default state <= S0;
                endcase

always @(*)
begin
        case(state)
                S0: begin TL1 <= 3'b001;
                                TL2 <= 3'b100;
                                TL2R <= 2'b01;
                                TL3 <= 3'b001;
                                TL4 <= 3'b100;
                                TL5 <= 3'b100;
                                TL5R <= 2'bzz;
                                TL6 <= 3'b100;
                                PdL1 <= 2'b10;
                                PdL2 <= 2'b01;
                                PdL5 <= 2'b10;
                                PdL6 <= 2'b01;
                end
                S1: begin TL1 <= 3'b010;
                                TL2 <= 3'b100;
                                TL2R <= 2'b01;
                                TL3 <= 3'b001;
                                TL4 <= 3'b100;
                                TL5 <= 3'b100;
                                TL5R <= 2'bzz;
                                TL6 <= 3'b100;
                                PdL1 <= 2'b10;
                                PdL2 <= 2'b01;
                                PdL5 <= 2'b10;
                                PdL6 <= 2'b01;
                end
                S2: begin TL1 <= 3'b100;
                                TL2 <= 3'b100;
                                TL2R <= 2'bzz;
                                TL3 <= 3'b010;
                                TL4 <= 3'b100;
                                TL5 <= 3'b100;
                                TL5R <= 2'bzz;
                                TL6 <= 3'b100;
                                PdL1 <= 2'b10;
                                PdL2 <= 2'b10;
                                PdL5 <= 2'b10;
                                PdL6 <= 2'b10;
                end
                S3: begin TL1 <= 3'b100;
                                TL2 <= 3'b100;
                                TL2R <= 2'bzz;
                                TL3 <= 3'b100;
                                TL4 <= 3'b110;
                                TL5 <= 3'b100;
                                TL5R <= 2'b01;
                                TL6 <= 3'b110;
                                PdL1 <= 2'b10;
```

```
                                PdL2 <= 2'b10;
                                PdL5 <= 2'b10;
                                PdL6 <= 2'b10;
end
S4: begin TL1 <= 3'b100;
                                TL2 <= 3'b100;
                                TL2R <= 2'bzz;
                                TL3 <= 3'b100;
                                TL4 <= 3'b001;
                                TL5 <= 3'b100;
                                TL5R <= 2'b01;
                                TL6 <= 3'b001;
                                PdL1 <= 2'b01;
                                PdL2 <= 2'b10;
                                PdL5 <= 2'b01;
                                PdL6 <= 2'b10;
end
S5: begin TL1 <= 3'b100;
                                TL2 <= 3'b100;
                                TL2R <= 2'bzz;
                                TL3 <= 3'b100;
                                TL4 <= 3'b001;
                                TL5 <= 3'b100;
                                TL5R <= 2'b01;
                                TL6 <= 3'b010;
                                PdL1 <= 2'b01;
                                PdL2 <= 2'b10;
                                PdL5 <= 2'b01;
                                PdL6 <= 2'b10;
end
S6: begin TL1 <= 3'b100;
                                TL2 <= 3'b100;
                                TL2R <= 2'bzz;
                                TL3 <= 3'b100;
                                TL4 <= 3'b010;
                                TL5 <= 3'b100;
                                TL5R <= 2'bzz;
                                TL6 <= 3'b100;
                                PdL1 <= 2'b10;
                                PdL2 <= 2'b10;
                                PdL5 <= 2'b10;
                                PdL6 <= 2'b10;
end
S7: begin TL1 <= 3'b100;
                                TL2 <= 3'b110;
                                TL2R <= 2'bzz;
                                TL3 <= 3'b100;
                                TL4 <= 3'b100;
                                TL5 <= 3'b110;
                                TL5R <= 2'bzz;
                                TL6 <= 3'b100;
                                PdL1 <= 2'b10;
                                PdL2 <= 2'b10;
                                PdL5 <= 2'b10;
                                PdL6 <= 2'b10;
end
S8: begin TL1 <= 3'b100;
                                TL2 <= 3'b001;
                                TL2R <= 2'b01;
                                TL3 <= 3'b100;
                                TL4 <= 3'b100;
                                TL5 <= 3'b001;
                                TL5R <= 2'b01;
                                TL6 <= 3'b100;
                                PdL1 <= 2'b01;
```

```
                                        PdL2 <= 2'b01;
                                        PdL5 <= 2'b01;
                                        PdL6 <= 2'b01;
                                end
                                S9: begin TL1 <= 3'b100;
                                        TL2 <= 3'b010;
                                        TL2R <= 2'bzz;
                                        TL3 <= 3'b100;
                                        TL4 <= 3'b100;
                                        TL5 <= 3'b010;
                                        TL5R <= 2'bzz;
                                        TL6 <= 3'b100;
                                        PdL1 <= 2'b10;
                                        PdL2 <= 2'b10;
                                        PdL5 <= 2'b10;
                                        PdL6 <= 2'b10;
                                end
                                S10: begin TL1 <= 3'b100;
                                        TL2 <= 3'b100;
                                        TL2R <= 2'bzz;
                                        TL3 <= 3'b100;
                                        TL4 <= 3'b100;
                                        TL5 <= 3'b100;
                                        TL5R <= 2'bzz;
                                        TL6 <= 3'b100;
                                        PdL1 <= 2'b10;
                                        PdL2 <= 2'b10;
                                        PdL5 <= 2'b10;
                                        PdL6 <= 2'b10;
                                end
                                S11: begin TL1 <= 3'b110;
                                        TL2 <= 3'b100;
                                        TL2R <= 2'bzz;
                                        TL3 <= 3'b110;
                                        TL4 <= 3'b100;
                                        TL5 <= 3'b100;
                                        TL5R <= 2'bzz;
                                        TL6 <= 3'b100;
                                        PdL1 <= 2'b10;
                                        PdL2 <= 2'b10;
                                        PdL5 <= 2'b10;
                                        PdL6 <= 2'b10;
                                end
                        default begin  TL1 <= 3'b001;
                                        TL2 <= 3'b100;
                                        TL2R <= 2'b01;
                                        TL3 <= 3'b001;
                                        TL4 <= 3'b100;
                                        TL5 <= 3'b100;
                                        TL5R <= 2'bzz;
                                        TL6 <= 3'b100;
                                        PdL1 <= 2'b10;
                                        PdL2 <= 2'b01;
                                        PdL5 <= 2'b10;
                                        PdL6 <= 2'b01;
                                end
                        endcase
                end
endmodule
```

Додаток Є

Реалізація модулю двадцяти чотирьох форматного годинника

```
module twentyfour_hour_clock(
```

```
            input clk,
            input reset,
            output reg [5:0] seconds,
            output reg [5:0] minutes,
            output reg [4:0] hours);


    always @(posedge(clk) or posedge(reset))
    begin
        if(reset == 1'b1) begin
            seconds = 0;
            minutes = 0;
            hours = 0;
                        end
        else if(clk == 1'b1) begin
            seconds = seconds + 1;
            if(seconds == 60) begin
                seconds = 0;
                minutes = minutes + 1;
                if(minutes == 60) begin
                    minutes = 0;
                    hours = hours + 1;
                   if(hours ==  24) begin
                       hours = 0;
                    end
                end
            end
        end
    end

endmodule
```

Додаток Ж

Реалізація модулю визначення пікових годин роботи світлофору

```
module peakhours(
        input clk,
        input [4:0] hours,
        output reg peak);

        always @(posedge(clk))
        begin
    if(((hours >= 7) && (hours <= 9)) || ((hours >= 12) && (hours <= 14)) || ((hours >= 17) && (hours <= 19))) begin
                                peak <= 1;
                end else begin
                                peak <= 0;
                end
        end

endmodule
```

Додаток З

Test Bench для ускладненого дорожнього перехрестя

```
`timescale 1ns / 1ps

`define clk_period 1000000000

module traffic_light_tb;
    reg clk; reg reset;
    wire [2:0] TL1;
    wire [2:0] TL2;
    wire [1:0] TL2R;
    wire [2:0] TL3;
    wire [2:0] TL4;
    wire [2:0] TL5;
    wire [1:0] TL5R;
    wire [2:0] TL6;
    wire [1:0] PdL1;
    wire [1:0] PdL2;
    wire [1:0] PdL5;
    wire [1:0] PdL6;
    wire peak;
    wire [4:0] hours;
    wire [5:0] minutes;
    wire [5:0] seconds;

TLC Tlc (
    .clk(clk),
    .reset(reset),
    .TL1(TL1),
    .TL2(TL2),
    .TL2R(TL2R),
    .TL3(TL3),
    .TL4(TL4),
    .TL5(TL5),
    .TL5R(TL5R),
    .TL6(TL6),
    .PdL1(PdL1),
    .PdL2(PdL2),
    .PdL5(PdL5),
    .PdL6(PdL6),
    .peak(peak),
    .hours(hours),
    .minutes(minutes),
    .seconds(seconds)
);

            initial clk = 1;
            always #(`clk_period/2) clk = ~clk;

            initial begin
                        reset = 0;
                        #`clk_period;
                        reset = 1;
                        #`clk_period;
                        reset = 0;
                        #(`clk_period * 200);
                        $stop;
            end

endmodule
```