Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

Система автоматизованого тестування великої кількості VHDL-проектів.
(тема)

Виконав: студент 2 курсу, групи СКСм-19-1
Лопатіна А.О.
(прізвище, ініціали)
Спеціальність 123 Комп'ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Спеціалізовані комп'ютерні системи
(повна назва освітньої програми)
Керівник проф. Хаханова І.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Автоматизації проектування обчислювальної техніки
Рівень вищої освіти другий (магістерський)
Спеціальність 123 – Комп'ютерна інженерія
Тип програми Освітньо-професійна
Освітня програма Спеціалізовані комп'ютерні системи

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____» _____ 2020 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ (ПРОЕКТ)

Студентові Лопатіній Анні Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Система автоматизованого тестування великої кількості VHDL-проектів

затверджена наказом по університету від "30" жовтня 2020 р. № 1489Ст

2. Термін подання студентом роботи до екзаменаційної комісії

3. Вхідні дані до роботи Розробити систему для автоматизації перевірки проектів, написаних на мові VHDL, забезпечити перевірку проекту на відповідність варіанту та коректність виконання завдання

4. Перелік питань, що потрібно опрацювати в роботі Огляд предметної області, аналіз ролі автоматизації в освітній сфері та основних завдань процесу автоматизованого навчання та його переваги, аналіз структури автоматизованої системи, аналіз та вибір засобів розробки, розробка алгоритму функціонування системи

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 11 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання 01.09.2020

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	01.09.2020–06.09.2020	
2	Огляд предметної області	07.09.2020–20.09.2020	
3	Аналіз та вибір засобів розробки	21.09.2020–04.10.2020	
4	Розробка структури системи	05.10.2020–18.10.2020	
5	Розробка алгоритму функціонування	19.10.2020–01.11.2020	
6	Розробка програмної частини	02.11.2020–22.11.2020	
7	Оформлення пояснювальної записки	23.11.2020–11.12.2020	

Студент _____
(підпис)

Керівник роботи (проекту) _____ проф. Хаханова І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 82 с., 35 рис., 5 табл., 3 дод., 18 джерел.

АВТОМАТИЗАЦІЯ, ВЕРІФІКАЦІЯ, ТЕСТУВАННЯ, VHDL, ШАБЛОН, КОНТРОЛЬ ЗНАНЬ, АВТОМАТИЗОВАНА СИСТЕМА

Метою атестаційної роботи є розробка системи автоматизованого тестування великої кількості VHDL-проектів. У ході виконання атестаційної роботи проводилося вивчення механізмів автоматизації, аналіз використання автоматизації в освітній сфері, розробка структури та алгоритму для автоматизованого тестування проектів, створених за допомогою мови VHDL.

Основним завданням є перевірка великої кількості проектів на відповідність варіанту та коректність виконання завдання, що реалізується за допомогою написання скрипта для тестування бази проектів.

Перевірку великої кількості проектів буде забезпечено за допомогою скрипта, в результаті роботи якого кожний проект буде перевірено на відповідність варіанту за допомогою шаблону перевірки та запущений в консольній версії Active HDL для тестування. Всі результати перевірок будуть занесені до таблиці, яка буде зберігатися у файлі для подальшого використання викладачем.

ABSTRACT

Master’s thesis: 82 pages, 35 figures, 5 tables, 3 appendices, 18 sources.

AUTOMATION, VERIFICATION, TESTING, VHDL, TEMPLATE, KNOWLEDGE CONTROL, AUTOMATED SYSTEM.

The purpose of this thesis is to develop a system of automated testing of a large number of VHDL-projects. During the attestation work the study of automation mechanisms, analysis of the use of automation in education, development of structure and algorithm for automated testing of projects created using VHDL language was carried out.

The main task is to check a large number of projects for compliance with the option and the correctness of the task, which is implemented by writing a script to test the database of projects.

Verification of a large number of projects will be provided by a script, as a result of which each project will be checked for compliance with the option using a validation template and run in the console version of Active HDL for testing. All test results will be recorded in a table, which will be stored in a file for later use by the teacher.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

VHDL ((Very high speed integrated circuits) Hardware Description Language) - мова опису апаратури інтегральних схем

САПР – система автоматизованого проектування

UVC (Universal Verification Component) – універсальна методологія верифікації

CDC (Clock Domain Crossing) – перетин тактової частоти

FSM (Finite-state Machine) машина кінцевих станів

RTL (Register Transfer Level) – рівень регістрових передач

TCL (Tool Command Language) – командна мова

JSON (JavaScript Object Notation) – стандартний текстовий формат для подання структурованих даних

ВСТУП

Сьогодні будь-яка сфера людської діяльності знаходиться в процесі розвитку. Не виключенням стала і освітня сфера. Технології відіграють значну роль в освітньому сегменті, сприяючи вдосконаленню навчального процесу. Зокрема, автоматизація стає важливою та значущою частиною не тільки нашого повсякденного життя, а і активно поширюється на освітні системи.

Автоматизація визначається як процес виконання повторюваних і буденних завдань за допомогою технологій, що призводить до швидших і точніших результатів. Робота проводиться з використанням мінімального людського втручання, а це означає, що це спільні зусилля між машинами та людьми. Розвиток автоматизації, навіть в сфері освіти, є неминучою реальністю. Автоматизація процесів в освіті допомагає викладачам та студентам мінімізувати ручні зусилля, створюючи більшу ефективність та дозволяє вчителям приділяти більше часу діяльності, що сприяє навчанню учнів.

Один з відносно нових методів контролю знань – машинний контроль. Цей метод дозволяє за допомогою автоматизованої системи перевірки результатів виконувати тестування робіт. В результаті чого підвищується кількісна та якісна складові процесу перевірки робіт [1].

Наприклад, автоматизована перевірка завдання може допомогти викладачам більше часу витрачати на допомогу учням та стати способом оптимізації свого часу. Викладач може приділити більшу увагу студентам, роботи яких не пройшли автоматизовану перевірку, особисто перевіривши завдання та провести зі студентом роботу над помилками. Тобто викладач не витрачає час на перевірку правильно виконаних завдань та може відвести більше часу на аналіз неправильно виконаних робіт та на зворотній зв'язок з

автором роботи.

Метою роботи є аналіз ролі автоматизації в освітній сфері, розробка структури та алгоритму для автоматизованого тестування проектів на мові VHDL та реалізація програмного забезпечення.

Сучасний етап розвитку освітніх закладів визначається значним збільшенням числа студентів, в тому числі, які навчаються за заочною та дистанційною формами навчання.

Рівень навчального навантаження впливає на якість навчальної роботи викладачів, при зростанні навантаження на педагога якість його роботи знижується. Одним із шляхів зниження навчального навантаження є автоматизування окремих функцій педагога. Контроль знань повністю покладається на викладача, незважаючи на можливість частково автоматизувати цей процес. Впровадження інструментальних засобів, що забезпечують автоматизування процесу перевірки знань і навичок, допомогло б вирішити частину проблем системи вищої освіти.

Якщо порівнювати з іншими методами контролю, автоматизований контроль знань в навчальному процесі має ряд явних переваг, серед яких: об'єктивність оцінки результатів, зручна кількісна форма вираження результатів, підвищена стійкість до фальсифікацій, швидка обробка результатів, високий ступінь стандартизації, єдність вимог.

В даній роботі пропонується автоматизувати частину дій викладача, а саме перевірку проектів студентів, створених мовою VHDL. Процес перевірки VHDL-проекту включатиме такі етапи, як верифікація проекту та компіляція.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Роль автоматизації в освітній сфері

Контроль знань під час навчального процесу на сьогоднішній день належить до однієї з найбільш актуальних проблем у сфері освіти. З одного боку, потрібні великі часові витрати, з іншого боку, необхідність аналізу отриманих даних і скорочення етапу реагування на отримані результати. Прийняття рішень в даній області традиційно практично цілком покладається на викладача.

Безсумнівно, якість навчальної роботи викладачів залежить від навчального навантаження, при зростанні навантаження на викладача якість його роботи знижується. Зниження навчального навантаження неможливо без автоматизації окремих функцій викладача, що допоможе їм робити свою роботу краще та ефективніше.

Даний етап розвитку освіти ставить перед викладачем спектр завдань, пов'язаних з переорієнтацією процесу навчання на забезпечення високого рівня інтерактивності і зворотного зв'язку зі студентами. Сучасні освітні технології націлені на створення умов для підвищення ступеня результативності та об'єктивності контролю якості засвоєння матеріалу учнями. Реалізації цих освітніх потреб сприяють різні системи автоматизації викладацької діяльності, які в майбутньому стануть обов'язковим атрибутом ефективного процесу навчання.

На сьогоднішній день проблема автоматизації в системі освіти вважається однією з найбільш актуальних. Ведуться активні дослідження в області автоматизації управління діяльністю підрозділів і кадрового складу вузів [2, 3], управління науково-дослідницької [4, 5] і організаційно-управлінської [6] діяльністю системи освіти. Є і більш приватні, вузькі з

проблематики роботи, що стосуються, наприклад, автоматизації рейтингової оцінки діяльності викладачів [7], автоматизації діяльності вчителя з розробки системи уроків [8].

Як показує аналіз наявних робіт, переважна більшість автоматизованих систем направлена на бізнес-процеси конкретних навчальних закладів. При цьому самі системи великою мірою збігаються функціональністю, найчастіше дублюючи одне одного.

Актуальною є проблема створення повноцінних і легких у використанні систем автоматизації діяльності викладача і студента, пов'язаних безпосередньо з викладанням конкретної навчальної дисципліни. Завданнями автоматизації процесу навчання можна назвати:

- удосконалення процесу навчання;
- віддалене керування процесом навчання;
- підвищення рівня знань студентів;
- підвищення ефективності процесу навчання;
- підвищення ефективності перевірки завдань;
- прискорення отримання результатів контролю знань.

Використання сучасних технологій в навчальному процесі, як засіб контролю якості знань, набуває все більшого поширення. Як показує практика, при належній підготовці, автоматизація контролю дозволяє помітно підвищити, перш за все, індивідуальність самого контролю, варіювати його в залежності від здібностей і освітніх цінностей учнів.

Автоматизований контроль підвищує об'єктивність самого контролю, дозволяє оцінювати якість знань не тільки «в загальному і цілому», але забезпечує кількісну оцінку якості засвоєння того чи іншого розділу навчального курсу.

Ще одним важливим аспектом використання автоматизованого контролю, як показала практика, стає стимулювання і мотивування студентів до самоосвітньої діяльності. За умови дотримання належних дидактичних

правил, методи автоматизованого контролю надають інформацію про якість знань не тільки викладачеві, а й самим студентам. Ці дидактичні можливості засобів комп'ютерного автоматичного контролю якості знань актуалізували численні дослідження даної проблематики. В результаті, в останні роки отримали належні освітлення в науковій літературі різні аспекти автоматизації контролю якості з допомогою комп'ютера: дидактичні, методичні, технологічні, програмні.

Проте швидке вдосконалення засобів обчислювальної техніки, розширення їх можливостей, як технічних (швидкість обчислень, швидкодія, обсяг пам'яті), так і програмних призводять до розширення їх дидактичних і методичних можливостей. Практично це означає, що, незважаючи на численні публікації з проблеми організації автоматизованого контролю, наукове дослідження проблеми як і раніше не є вичерпним.

Технічні можливості постійно розширюються, з’являються нові програмні засоби, що дозволяє постійно вдосконалювати форми і методи організації автоматичного контролю знань учнів.

В даний час все більш широке поширення набувають контролюючі програми зі зворотним зв'язком, що спираються на застосування звуку, зображень (динамічних і статичних) та презентацій.

Такий розвиток обчислювальної техніки розширює не тільки дидактичні можливості автоматизації, але й саме коло його застосування від навчальних курсів природних і точних дисциплін до навчальних курсів гуманітарного та соціального змісту. При цьому системне, науково обґрунтоване використання автоматизованого контролю якості знань підвищує освітню ефективність контролю більш ніж в два рази. Серед причин, що впливають на ефективність контролю, особливу роль відіграє фактор часу. Психологами встановлено, що оцінка корисності результату діяльності убуває відповідно до квадратом часу його досягнення – чим більший час досягнення результату, тим менш привабливим є сам результат.

Тому відстрочка заохочення або покарання за успішний або поганий результат навчальної діяльності знижує ефективність цих заходів впливу. Ця умова має виняткове значення в області навчання і виховання, зокрема, в практиці застосування педагогічної оцінки. Іншими словами, результат контролю якості знань повинен бути доступним негайно, прямо після виконання оцінюваної роботи.

1.2 Структура автоматизованої системи

Моніторинг результатів навчання з використанням сучасних засобів інформаційних технологій в процесі навчання порівняно з іншими методами контролю має ряд очевидних переваг, серед яких: об'єктивність оцінки результатів, зручна кількісна форма вираження результатів, підвищена стійкість до фальсифікацій, швидка обробка результатів, єдині вимоги для всіх учнів, високий ступінь стандартизації, виключення суб'єктивізму при оцінці результатів.

При цьому зручність в наявності кількісних показників виражається в можливості порівняння знань і умінь одних учнів з іншими, або відстеження динаміки засвоєння знань одним учням в процесі навчання.

Існуючі на сьогоднішній день технічні засоби контролю знань за тематикою контрольованих питань можна розділити на дві категорії: засоби контролю теоретичних знань і засоби контролю практичних навичок.

Засоби контролю теоретичних знань служать для перевірки засвоєння учнем отриманих теоретичних відомостей; на сьогоднішній день дані засоби контролю займають домінуючі позиції серед засобів контролю знань. Скоріш за все, це пов'язано з простотою реалізації контролю за допомогою стандартних засобів обчислювальної техніки, а також можливістю застосування створених програмних оболонок для створення тестів з різних предметів.

У той же час процес підготовки фахівця технічного профілю не може протікати без отримання практичних навичок роботи в ході виконання лабораторних робіт. Однак автоматизація процесу контролю за такою навчальною роботою є набагато складнішою, тому на сьогоднішній день практично відсутні технічні засоби контролю практичних навичок.

Коректність виконання проекту – відповідність функціональності проекту вимогам, яка з'ясовується за допомогою верифікації.

Верифікацією називається перевірка відповідності проекту (або деякого проміжного результату проектування: прототипу чи моделі) висунутим до неї вимогам (проектної документації). Якщо проект відповідає вимогам, він називається коректним (правильним); в іншому випадку проект називається некоректним (помилковим), а сам факт невідповідності вимогам – помилкою. Таким чином, верифікація – це аналіз коду на предмет наявності або відсутності в ньому помилок. Говорячи перебільшено, результатом верифікації є вердикт: негативний, якщо в проекті знайдена хоча б одна помилка, або позитивний, якщо доведено, що помилок немає.

Така перевірка може приймати різні форми: інспекція коду (перегляд та рецензування «вихідного коду»); статичний аналіз (пошук типових помилок, наприклад читання неініціалізованих змінних); тестування (запуск проекту на прикладах і перевірка коректності результатів); імітаційне моделювання (створення виконавчої моделі системи і її дослідження в спеціальному оточенні); формальна верифікація (побудова логічної моделі системи і її аналіз засобами математичної логіки). Підходів багато, проте завдання верифікації настільки важке, що жоден з них не гарантує коректність дійсно складних проектів; кращі результати, як показує практика, досягаються при спільному використанні різних методів.

Метою роботи є дослідження та розробка методів і засобів, що дозволяють автоматизувати частину дій викладача, таких як перевірка робіт студентів, їх верифікація, аналіз результатів та прийняття рішення щодо

коректності виконання. Під роботою розуміється проект, створений за допомогою мови VHDL [9].

Схема пропонованої системи показана на рисунку 2.1.

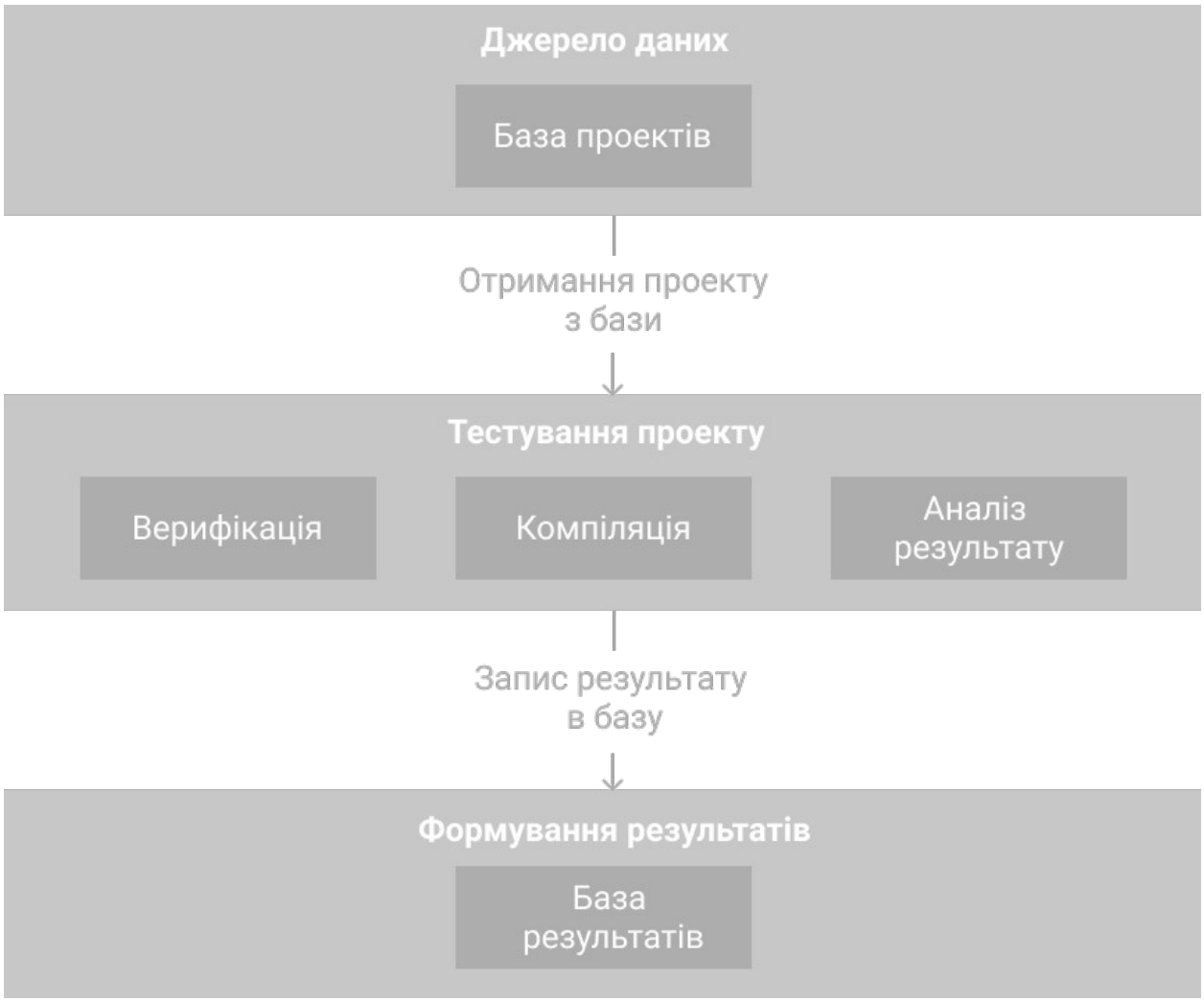

Рисунок 2.1 – Структура системи

Джерелом даних служить база проектів, що представляє собою таблицю Excel з переліком проектів для перевірки, кожен проект має номер варіанту, прізвище студента, та локальний шлях до папки з файлами проекту. Усі проекти з бази будуть проходити автоматизований етап тестування.

Тестування проекту складається з верифікації вихідного коду, компіляції проекту та прийняття рішення про коректність виконання завдання студентом.

Етап верифікації служить для перевірки проекту на відповідність вхідним даними (вимогам, зазначеним у варіанті завдання). Якщо верифікація пройдена не вдало, етап тестування завершиться і у базу результатів буде доданий негативний результат, інакше проект буде запущений в консольній версії Active HDL для компіляції і після чого у базу результатів буде доданий відповідний результат.

База результатів зберігається у вигляді таблиці з переліком усіх перевірених проектів та результатів їх перевірки. Якщо проект не пройшов етап верифікації, результатом перевірки буде повідомлення «Проект не відповідає вимогам варіанту». У разі вдалої верифікації проект буде скомпільований. Якщо етап компіляції пройдено успішно, результатом перевірки буде повідомлення «Проект пройшов усі перевірки», інакше результатом буде – «Проект не пройшов компіляцію».

1.3 Постановка завдання

Мета атестаційної роботи – розробка системи автоматизованого тестування проектів, створених мовою VHDL. Процес перевірки VHDL-проекту включає в себе кілька етапів. Головними етапами є верифікація проекту та компіляція, після проходження яких можна стверджувати чи правильно студентом виконане поставлене завдання. Також процес автоматизації тестування повинен включати взаємодію з базою проектів та процес формування результатів.

Основні завдання проектування наступні:

- аналіз методів верифікації проектів;
- аналіз засобів розробки автоматизованої системи;

– розгляд структури автоматизованої системи, виділення основних етапів автоматизації та розробка структурної схеми;

– розробка алгоритму автоматизованого тестування;

– розробка програмного забезпечення для автоматизованої системи.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Вибір засобів розробки

Розробка запропонованої автоматизованої системи передбачає реалізацію скрипта для описання послідовності дій, необхідних для автоматизації етапів перевірки проектів.

Скриптові мови дозволяють автоматизувати деяку задачу, яку без скрипту користувач робив би вручну. У скриптових мовах немає потреби в компіляції. Інтерпретатор може перетворювати текст в проміжний код, але цього не потрібно робити в окремій стадії. Такий код, якщо це передбачено конструкцією мови, буде згенеровано при першому запуску скрипта. Можливості мов скриптів зазвичай представлені у вигляді більш високого рівня абстрагування від системи, ніж в компільованих мовах програмування. Команди в скриптах – це в більшості своїй команди, доступні в командному рядку операційної системи. Скрипти можуть також містити свої власні команди.

Існує чимало мов програмування, за допомогою яких робляться скрипти. Розглянемо популярні мови загального призначення, такі як Python, JavaScript і Ruby.

JavaScript – це легка, об'єктно-орієнтована мова з функціями першого класу, найвідоміша скриптова мова для веб-сторінок, але також використовується у багатьох не браузерних середовищах, наприклад, в додатках для автоматизації дій, написання макросів, організації доступу з боку веб-служб. JavaScript класифікують як об'єктно-орієнтовану скриптову мову програмування з динамічною типізацією. Також JavaScript частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема:

динамічну та слабку типізацію, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

JavaScript має багато можливостей для написання скриптів автоматизації, але за рахунок обмежених можливостей взаємодії з файловою системою, дана мова не зможе задовольнити вимогам розроблюваної системи.

Ruby – інтерпретована скриптова мова високого рівня для швидкого і зручного об'єктно-орієнтованого програмування. Ruby має велику кількість засобів для обробки текстів, для вирішення системних завдань. Ruby є повністю вільною мовою програмування з можливістю копіювання, модифікації і поширення. Ruby перенесений на безліч платформ.

До переваг мови можна віднести просунуті методи маніпуляції рядками і текстом, простий і чистий синтаксис, розширення можливостей за допомогою бібліотек, написаних на C або Ruby. До недоліків відносяться недолік інформаційних ресурсів, присвячених Ruby і меншу продуктивність у порівнянні з іншими мовами, також мова від початку була розроблена під Unix-подібні системи і тому поступається в роботі у Windows.

Python – це мова програмування загального призначення, широко застосовується як інтерпретована мова для скриптів різного призначення. Як і Ruby, Python має на меті наблизити синтаксис реальної програми, написаної на ньому, до опису задачі на псевдокоді, що дозволяє програмісту зменшити обсяг програми. Основною перевагою мови є безліч корисних бібліотек і розширень мови, які можна легко використовувати в своїх проектах завдяки уніфікованого механізму імпорту та програмним інтерфейсам [10].

Стандартна бібліотека Python дуже обширна і пропонує широкий спектр можливостей. Вона містить вбудовані модулі, які забезпечують доступ до функціональних можливостей системи, таких як робота з файловою системою, які інакше були б недоступні програмістам, а також модулі, що забезпечують стандартизовані рішення для багатьох проблем, що

виникають у програмуванні. Інсталятори Python для платформи Windows зазвичай включають всю стандартну бібліотеку і часто також містять багато додаткових компонентів. Для Unix-подібних операційних систем Python зазвичай надається за замовчуванням, як набір пакетів. Також перевагою є широкий вибір середовищ розробки, доступних на різних платформах.

Отже, можливості мови Python зможуть задовільнити вимоги до написання скрипта для автоматизації в силу легкості розробки, наявності великої кількості бібліотек та підтримці на різних платформах.

У якості середовища розробки для мови Python доступно безліч інструментів, серед яких можна виділити Visual Studio Code – безкоштовний повнофункціональний редактор коду від Microsoft для Windows, Linux і MacOS. Його можливості – відладка, підсвічування синтаксису, інтелектуальне завершення коду, рефакторинг і інтеграція з Git. Для початку роботи з Python може знадобитися кілька додаткових пакетів, але встановити їх досить просто, так як Visual Studio Code надає свій власний набір розширень.

2.2 Структура VHDL-проекту

2.2.1 Особливості проектування на VHDL

Мова VHDL призначена для опису проектів різного ступеня складності – від найпростішого вентиля до цілої системи, що складається з апаратних і програмних частин. Вона дозволяє будувати моделі на різних рівнях абстракції, виконувати імітаційне моделювання і генерувати тимчасові діаграми, вести документування проекту, здійснювати синтез структури по поведінковому опису, верифікувати проект формальними методами [11].

В основі мови лежать наступні принципи побудови:

– підтримка функціональної декомпозиції (функціональна ієрархія і рекурсія);

– підтримка структурної декомпозиції (структурна ієрархія);

– представлення системи у вигляді паралельно функціонуючих взаємодіючих процесів;

– використання абстрактних типів даних;

– використання подійного моделювання;

– підтримка різних рівнів абстракції та деталізації представлення проекту.

Основні засоби мови VHDL включають:

– бібліотеки;

– модулі: інтерфейси об'єкта проекту, архітектурні тіла, конфігурації, пакети, тіла пакетів;

– підпрограми: процедури, функції;

– скалярні типи даних;

– складові типи даних;

– об'єкти: константи, змінні, сигнали, порти, параметри налаштування;

– операції: логічні, порівняння, арифметичні;

– вираження;

– послідовні оператори: очікування, затвердження, призначення сигналу, присвоювання змінної, виклику процедури, умовний, селективний, циклу, повернення та інші;

– паралельні оператори: блоку, процесу, паралельного виклику процедури, паралельного затвердження, паралельного призначення сигналу, конкретизації компонента, генерації.

Розробка автоматизованої системи перевірки VHDL-проектів передбачає етап тестування, під час якого відбувається аналіз коду для верифікації проекту. Тому розглянемо детальніше структуру VHDL-проекту.

Проект в системі проектування на основі VHDL представляється сукупністю ієрархічно пов'язаних текстових фрагментів, званих проектними модулями [12]. Розрізняють первинні і вторинні проектні модулі, при цьому:

<Первинний модуль> :: =

<Декларація суті>

| <Декларація пакета>

| <Декларація конфігурації>

<Вторинний модуль> :: =

<Тіло архітектури>

| <Тіло пакета>

Декларація сутності (entity) визначає ім'я проекту і його інтерфейс, тобто порти і параметри налаштування. Тіло архітектури сутності описує той чи інший спосіб функціонування пристрою і (або) його структуру.

Кожній сутності зіставляється одне або кілька архітектурних тіл, кілька вторинних модулів, що відповідають одному первинному, складають набір можливих альтернативних реалізацій об'єкта, представленого первинним модулем. Наприклад, одній сутності може відповідати кілька архітектурних тіл, що відрізняються ступенем деталізації опису і навіть алгоритмом перетворення даних.

Первинні і відповідні їм вторинні модулі можуть зберігатися в різних файлах або записуватися в одному файлі. Важливо лише, щоб вони були скомпільовані в загальну проектну бібліотеку, причому первинний модуль компілюється раніше підлеглого йому вторинного. При записі первинного і вторинного модулів в одному файлі первинний модуль записується раніше відповідного йому вторинного.

Типовий текст програми на VHDL має наступну структуру:

<VHDL-програма> :: =

{

{<Оголошення бібліотеки>}

{<Оголошення використання>}

<Первинний модуль>

}

{<Вторинний модуль>}

<Оголошення бібліотеки> :: = library <ім'я бібліотеки>;

<Оголошення використання> :: =

use <ім'я бібліотеки>. < ідентифікатор >;

Ідентифікатори – це назви бібліотек, які використовуються в об'єкті проекту. Вони вказують транслятору місце розташування цих бібліотек. Опис «use» вказує, які пакети і які елементи цих пакетів можуть бути використані в об'єкті.

Тобто, текст являє собою довільний набір первинних і вторинних модулів, причому кожному первинному модулю може передувати вказівка на бібліотеки і пакети, інформація з яких буде потрібна для побудови цього модуля. Після цього необхідно вказати, що вони будуть використані за допомогою відповідного оголошення. Як ім'я модуля часто вживають «all» – в цьому випадку будуть використані всі наявні модулі. І навіть якщо кілька первинних модулів посилаються на одні і ті ж бібліотеки і модулі, декларація використання повинна передувати кожному первинному модулю окремо [13].

2.2.2 VHDL-модель

Синтаксис декларації сутності має вигляд:

<Декларація суті> :: =

entity <ім'я проекту> is

[<Оголошення параметрів налаштування>]

[<Оголошення портів>]

[<Розділ декларацій>]

[Begin <розділ оператора>]

end [entity] <ім'я проекту>;

<Оголошення параметрів настройки> :: =

generic (<ім'я>: <тип> [: = <вираз>]

{; <Ім'я>: <тип> [: = <вираз>]});

<Оголошення портів> :: =

port (<ім'я>: <режим> <тип> [: = <вираз>]

{; <Ім'я>: <режим> <тип> [: = <вираз>]};

<Режим> :: = in | out | inout

Оголошення параметрів налаштування включається до декларації сутності для створення проектів, які передбачається використовувати як фрагменти в різних інших проектах, причому можлива модифікація деяких властивостей даного компонента, точніше вибір параметра з безлічі значень, певного типом.

Визначення портів задає імена вхідних (in), вихідних (out) та двонапрямлених (inout) ліній передачі інформації і тип даних, що передаються через порти. Оголошення портів, як випливає з представлених правил синтаксису, не обов'язково. Така конструкція дозволяє ефективно поєднувати опис власне проектованого пристрою і алгоритм його тестування.

Для параметрів і вхідних портів можна задавати значення за замовчуванням. Вони приймаються, якщо відповідним одиницям інформації не присвоєно інші значення в модулях вищого рівня ієрархії.

Розділ декларацій суті має такий же зміст, що і розділ декларацій тіла архітектури – оголошуються локальні типи даних, підпрограми, сигнали і т.п. При цьому оголошені об'єкти доступні у всіх архітектурних тілах, підлеглих цієї сутності.

Розділ операторів суті може містити тільки дуже обмежений набір службових операторів, які на практиці застосовуються нечасто. Тому зазвичай цей розділ залишається порожнім [14].

Розглянемо структуру коду VHDL-моделі на прикладі цифрової схеми, яку показано на рисунку 2.1.

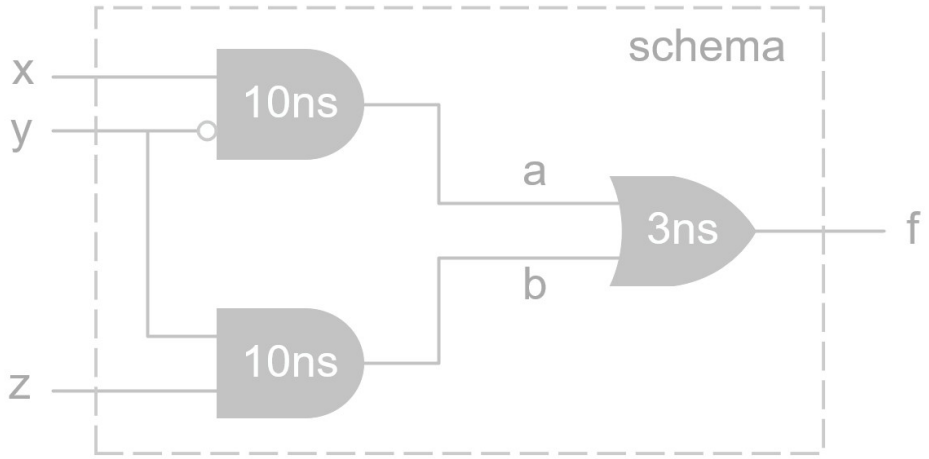

Рисунок 2.1 – Приклад цифрової схеми

На рисунку 2.1 показано, що є три вхідних порти (x, y, z) і один вихідний порт (f). Вхідні і вихідні порти мають ширину один біт.

Функціональність схеми – це виконання логічної операції AND двох входів і виведення результату на вихідний порт. Для опису вхідних і вихідних портів цієї схеми використовуються наступні рядки коду, показані на рисунку 2.2.

```
1 entity schema is
2 port (x: in std_logic;
3       y: in std_logic;
4       z: in std_logic;
5       f: out std_logic);
6 end schema;
```

Рисунок 2.2 – Опис вхідних і вихідних портів схеми

Розглянемо, що визначає кожен рядок:

– перший рядок вказує довільну назву для описуваної схеми. Слово «schema», яке знаходиться між ключовими словами «entity» і «is», визначає ім'я цього модуля;

– рядки з 2 по 5 визначають вхідні і вихідні порти схеми. Порівнюючи ці рядки зі схемою, зображеної на рисунку 2.1, бачимо, що порти схеми разом зі своїми функціями перераховані після ключового слова «port». Наприклад, в рядку 3 зазначається, що у нас є порт під назвою «y». Цей порт є входом, як зазначено ключовим словом «in» після двокрапки. Тип даних std_logic використовується для опису одно-бітового цифрового сигналу. Оскільки всі порти входів/виходів на рисунку 2.1 передають одиницю або нуль, ми можемо використовувати для цих портів тип даних «std_logic»;

– рядок 6 визначає кінець оператора «entity».

Отже, фрагмент коду «entity» визначає:

1) назву описуваної схеми;

2) порти схеми разом зі своїми характеристиками, а саме: вхід/вихід і тип даних, які повинні передаватися цими портами.

Фактично фрагмент коду «entity» описує інтерфейс модуля з навколишнім середовищем. На додаток до інтерфейсу схеми з навколишнім середовищем необхідно описати роботу схеми.

Тіло архітектури представляє змістовний опис проекту. Архітектурне тіло в певному сенсі підпорядковане відповідній сутності.

Розрізняють структурні архітектурні тіла (описують проект у вигляді сукупності компонентів і їх з'єднань), поведінкові архітектурні тіла (описують проект як сукупність виконуваних дій) і змішані тіла. Формальних ознак, за якими тіло можна було б віднести до того чи іншого типу, не існує – мова йде скоріше не про чітку класифікацію, а про стилі опису, для кожного з яких характерними є певні оператори і які краще відображають різні аспекти одного і того ж об'єкта (структурний і поведінковий відповідно).

Визначено наступні правила запису архітектурних тіл:

<Архітектурне тіло> :: =

architecture <ім'я архітектури> of <ім'я сутності> is

<Розділ декларацій>

begin

<Розділ операторів>

end [architecture] <ім'я архітектури>;

Тут ім'я архітектури – це будь-який індивідуальне ім'я проектного модуля. Ім'я сутності задає первинний модуль, якому підпорядковується архітектурне тіло. В розділі декларацій оголошуються локальні для цього модуля інформаційні одиниці – типи даних, сигнали, підпрограми тощо Розділ операторів описує правила функціонування і (або) конструювання пристрою в термінах мови.

Оголошення в тілі архітектури таке ж, як в блоці і їм може бути: оголошення і тіло процедури або функції, оголошення типу і підтипу, оголошення файлу, псевдоніма, константи, глобальної змінної, оголошення і специфікація атрибута, оголошення групи, опис «use», а також оголошення компонентів. Оголошені в тілі архітектури типи, сигнали, підпрограми видимі тільки в межах цієї архітектури. Виконавчу частину архітектури складають паралельні оператори, такі як процес, блок, паралельне присвоювання сигналу і ін. Ці оператори виконуються паралельно.

Так як всі оператори у виконавчій частині тіла архітектури – паралельні, їх взаємний порядок – байдужий, хорошим стилем вважається, коли паралельні оператори ставляться в послідовності, відповідної ланцюжках вершин граф-схеми алгоритму, реалізованого в архітектурі.

На рисунку 2.1 функціональність схеми – це виконання логічних операцій і отримання результату на вихідному порту. Опис архітектури показано на рисунку 2.3.

Розглянемо, що визначає кожен рядок:

– рядок 7 дає назву «arch» для архітектури, яка буде описана в наступних рядках. Це ім'я зустрічається між ключовими словами «architecture» і «of». Цей рядок також пов'язує цю архітектуру з «schema». Іншими словами, ця архітектура буде описувати роботу «schema»;

– рядок 8 вказує внутрішні лінії схеми як сигнали a та b;

– рядок 9 вказує початок опису архітектури;

– рядки з 10 по 12 використовують синтаксис VHDL для опису роботи схеми. Результат присвоюється сигналу або вихідному порту за допомогою оператора присвоювання «<=», якщо логічний елемент схеми має затримку, вона вказується після ключового слова «after»;

– рядок 13 вказує кінець опису архітектури. Як згадувалося вище, ці рядки коду описують внутрішню роботу схеми.

Об'єднавши рядки, отримаємо опис «schema» в VHDL, показаний на рисунку 2.4.

```
7  architecture arch of schema is
8  signal a,b: std_logic;
9  begin
10   a <= x and not y after 10 ns;
11   b <= y and z after 10 ns;
12   f <= a or b after 3 ns;
13 end arch;
```

Рисунок 2.3 – Опис архітектури

```
1  entity schema is
2  port (x: in std_logic;
3        y: in std_logic;
4        z: in std_logic;
5        f: out std_logic);
6  end schema;
7  architecture arch of schema is
8  signal a,b: std_logic;
9  begin
10   a <= x and not y after 10 ns;
11   b <= y and z after 10 ns;
12   f <= a or b after 3 ns;
13 end arch;
```

Рисунок 2.4 – Опис схеми

Однак ще потрібно додати ще декілька рядків коду. Оскільки в наведеному вище прикладі використовується тип даних «std_logic», тому потрібно додати в код пакет «std_logic_1164» з бібліотеки «ieee». Логічні оператори для типу даних «std_logic» також визначені в пакеті «std_logic_1164». Остаточний код показаний на рисунку 2.5.

Тут додаємо два нових рядки перед «entity» та «architecture». Перший рядок додає бібліотеку «ieee», а другий рядок вказує, що потрібно пакет «std_logic_1164» з даної бібліотеки. Оскільки «std_logic» є широко використовуваним типом даних, в проект майже завжди додається бібліотека «ieee» і пакет «std_logic_1164». Тепер дану модель, що включає оголошення об'єкта і тіло архітектури можна транслювати і моделювати.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity schema is
5  port (x: in std_logic;
6        y: in std_logic;
7        z: in std_logic;
8        f: out std_logic);
9  end schema;
10
11 architecture arch of schema is
12 signal a,b: std_logic;
13 begin
14   a <= x and not y after 10 ns;
15   b <= y and z after 10 ns;
16   f <= a or b after 3 ns;
17 end arch;
```

Рисунок 2.5 – Остаточний опис схеми

2.2.3 VHDL-Testbench

TestBench – це код на мові опису апаратури, який моделює середу функціонування розроблюваного пристрою, формує вхідні тестові послідовності і може аналізувати отримані результати тестування. Він створений для того, щоб описати, які дії будуть здійснені з перевіряється модулем, і побачити його реакцію.

Як правило, такий код не містить вхідних або вихідних сигналів, тобто має порожній «entity», містить оператор реалізації компонента і оператори, що формують вхідні набори.

Приклад Testbench для тестування схеми на рисунку 2.1 показано на рисунку 2.6.

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

entity schema_tb is
end schema_tb;

architecture arch_tb of schema_tb is
  component schema is
  port (x, y, z: in std_logic;
  f: out std_logic);
end component;

signal test : std_logic_vector(2 downto 0);
signal Res : std_logic;

begin
UUT: schema
port map (x => test(2),y => test(1), z => test(0), f => Res);
test <= "000",
     "001" after 15 ns,
     "010" after 30 ns,
     "011" after 45 ns,
     "100" after 60 ns,
     "101" after 75 ns,
     "110" after 90 ns,
     "111" after 105 ns;
end arch_tb;
```

Рисунок 2.6 – Приклад Testbench для тестування схеми

Структура Testbench:

– «entity», не має портів (порожній заголовок сутності);

– «architecture», містить декларацію тестованого компонента;

– «configuration»: містить підключення тестованого модулю і формування тестової послідовності.

У рядку 8 використовується конструкція декларації компонента «component» для оголошення тестованої моделі; ім'я компонента і порти повинні збігатися з ім'ям і портами тестованої моделі.

Оператор реалізації компонента у рядку 15 «UUT: schema port map (x => test (2), y => test (1), z => test (0), f => Res)» підключає тестований блок, де UUT – мітка копії компонента, EX1 – ім'я компонента (таке ж як в декларації компонента), «port map» – описує зв'язок портів тестованого пристрою і сигналів TestBench в форматі «ім'я порту => ім'я сигналу».

2.4 Методи верифікації проектів

Одним з головних завдань при розробці автоматизованої системи перевірки проектів є верифікація програмного коду. Метою верифікації є виявлення помилок, вразливостей, некоректно реалізованих властивостей і вимог [15]. Розглянемо основні засоби верифікації, які можуть бути використані в розроблюваній системі.

Шляхом функціональної верифікації виконуються такі групи тестів:

перевірка поведінки в штатних ситуаціях, що регламентуються специфікацією на пристрій. Тобто перевіряємо ситуації, коли все йде добре;

перевірка відхилень від штатних ситуацій, але в рамках специфікації. Тобто коли щось йде не так, але ми знали, що таке може бути і знаємо як в цій ситуації працювати;

нестандартні ситуації, будь-які випадкові ситуації (наприклад, випадкових змін станів логічних елементі). Тобто це коли може відбутися все, що завгодно і треба переконатися, що пристрій вийде після цього в робочий стан.

Перші дві стадії піддаються автоматизації за допомогою UVC (Universal Verification Component) і досить швидко можна наростити обсяг різних тестів, в тому числі – які генеруються автоматично. Третя стадія вимагає неординарного підходу і досвіду, дуже складно автоматизується, тому що більшість ситуацій – це окремий алгоритм, можливо скрипт для САПР або інструкції для «ручних» перевірок.

Метрики функціональної верифікації – це показники охоплення проекту тестами. Вони потрібні для того, щоб зрозуміти які ще тести необхідно розробити для перевірки можливих ситуацій і скільки приблизно часу може зайняти верифікація. На жаль, тільки один тип метрик оцінюється на підставі вихідного коду проекту, визначення критеріїв для інших типів – це результат інтелектуальної праці. До того ж, необхідно пам'ятати, що досягнення бажаних показників одним типом метрик ніяк не говорить про працездатність в цілому, завжди необхідно оцінювати комплекс.

Типи метрик:

а) функціональне покриття. Показує на скільки повно перевірені функції. Критерії даного покриття можуть визначатися планом тестування і впровадженням спеціальних конструкцій в тестове оточення, що відстежують виконувалася чи ні та чи інша функція, змінювалися чи певним чином дані тощо. Інформація від конструкцій, впроваджених в вихідний код, може бути автоматично зібрана засобами САПР.

б) покриття коду – зміна стану конструкцій вихідного коду в ході тестів. Збирається автоматично засобами САПР, не вимагає внесення будь-яких конструкцій в вихідний код. Наприклад:

1) перемикання регістрів (Toggle Coverage);

2) активність кожного рядка коду (Line Coverage);

3) активність виразів (Statement Coverage);

4) активність ділянки коду всередині умовного оператора або процедури (Block Coverage);

5) активність всіх гілок умовних операторів таких як if, case, while, repeat, forever, for, loop (Branch Coverage);

6) зміна всіх станів (true, false) складових логічних виразів (Expression Coverage);

7) стану кінцевого автомата (Finite-State Machine Coverage);

в) покриття тверджень. Твердження – це спеціальні мовні конструкції, які відстежують різні події і послідовність, і за заданими критеріями визначають правомірність їх виникнення.

Методи функціональної верифікації:

– прямі, осмислені тести. Якщо приймається цей метод в проекті, то план верифікації складається з тестів спрямованих на перевірку поведінки в конкретних цікавлять точках (станах). Перевірити всі можливі ситуації, особливо в складних проектах, майже не можливо. Також проблеми, які можуть виникнути в ситуаціях не покритих тестами не виявляються до того, як пристрій починає використовуватися в реальних умовах. Зазвичай в цих тестах використовуються метрики функціонального покриття;

– концепція створення тестів, спрямована на досягнення певного «тестового покриття». Спираються на метрики, щоб зрозуміти які тести необхідно додати в план верифікації, щоб досягти цільових показників готовності проекту. Необхідно використовувати інструменти аналізу покриття, щоб подивитися, що ще додати в план верифікації;

– верифікація подачею випадкових впливів. Це дійсно автоматичні тести з генерацією випадкових впливів. Метод дуже витратний спочатку, тому що потрібен тривалий час на підготовку інструментів. Після того як перший етап підготовки пройдено, то тестування може запускатися автоматично, багаторазово з різними вихідними даними. При виявленні невідповідності «assertion», команда розробників і тестувальників приступає до аналізу виявленої помилки. У реальному проекті не можна використати тільки цей метод, тому що за його допомогою можна зібрати покриття коду і

покриття тверджень, а вони можуть нічого не говорити про правильність роботи, тобто відповідність технічним умовам. Його необхідно доповнювати функціональними тестами. Для реалізації даної методології потрібно впровадити "затвердження" (assertion) у всіх важливих точках вихідного коду і тестового оточення; розробити генератори випадкових впливів і сценарії їх роботи, тобто впливи випадкові, але мають обмеження діапазону, порядок подачі.

Статичний аналіз коду – це дослідження, яке виконується без фактичного виконання проекту, в той час як динамічний аналіз дає аналізувати всі шляхи виконання коду. На сьогодні існують дві найпопулярніші групи методів статичної верифікації: це методи дедуктивного дослідження та методи перевірки моделі [16].

Методи статичного аналізу не залежать від використання компілятора і середовища, що дає виявити приховані помилки. В розроблюваній системі пропонується проводити статичний аналіз коду для перевірки виконання вимог варіанту завдання. Визначимо можливі помилки, які свідчать про невідповідність виконання проекту згідно з завданням:

неправильна структура проекту (відсутність одного чи декількох структурних блоків);

невідповідність назв вхідних або вихідних портів;

неправильний тип вхідних або вихідних портів;

невідповідність сигналів;

неправильний тип сигналів.

Такий аналіз добре автоматизується і може бути практично повністю покладений на програмний інструментарій, хоча іноді необхідно вручну визначити, наприклад, прийняті в проекті стандарти кодування. Однак він застосовується лише до коду або до певних форматів уявлення проектних артефактів, і здатний виявляти тільки обмежену кількість типів помилок. Проблемою цих методів є або велика надмірність, при використанні методів

строго аналізу, або вірогідність пропустити помилки, при використані методів що ще генерують повідомлення про помилку. Інструменти автоматичної перевірки на основі статичного аналізу застосовуються досить широко, оскільки не вимагають спеціальної підготовки і досить зручні у використанні.

З існуючих інструментів верифікації цифрових схем можна виділити ALINT-PRO – це рішення для перевірки проекту RTL(Register transfer level), написаного на VHDL, Verilog і SystemVerilog, яке орієнтоване на перевірку стилю кодування, угод про імена, невідповідностей моделювання, правильних описів. В центрі уваги:

дотримання стандартів щодо оформлення коду і угод прро імена;

невідповідності між результатами симуляції до і після синтезу моделі;

оптимальність синтезу;

коректність опису кінцевих автоматів;

усунення потенційних проблем на наступних етапах проектування;

коректність проектування дерев тактових сигналів і сигналів скидання;

перетин тактових доменів, доменів асинхронного скидання;

проектування з орієнтацією на тестопридатність;

переносимість і можливість повторного використання коду.

Статичний аналіз RTL-опису та проектних обмежень дозволяє виявляти критичні проблеми на початковій стадії проектування пристрою, уникаючи значних тимчасових витрат. Запуск ALINT-PRO перед симуляцією

RTL-коду і логічного синтезу запобігає виникненню помилок на наступних стадіях проектування і знижує кількість ітерацій, необхідних для завершення проекту.

ALINT-PRO має добре розроблену, інтуїтивно зрозумілу структуру, яка пропонує функції для ефективного аналізу проекту, включаючи перегляд схеми RTL, переглядач FSM, перегляд тактових імпульсів і скидів, перегляд схеми управління, перегляд складності, перегляд порушень, і спеціальні інструменти, такі як переглядач CDC, засоби перегляду RDC і схеми CDC для аналізу перетинів в доменах «clock» і ініціалізації.

Налаштування існуючого проекту HDL для аналізу в ALINT-PRO підтримується шляхом читання зовнішніх форматів файлів проекту (Aldec Active-HDL, Aldec Riviera-PRO, Xilinx Vivado, Xilinx ISE, Intel Quartus), інтерпретуючи типові сценарії моделювання (команди сумісності, такі як vcom, vlog, vsim, vlib), а також традиційні списки файлів для інструментів пакетного режиму і дуже прості інструменти графічного інтерфейсу для прямого імпорту окремих файлів і цілих каталогів. Приклад роботи з графічним інтерфейсом ALINT-PRO показано на рисунку 2.7. Поле «Violation Viewer» відображає порушення, знайдені у коді.

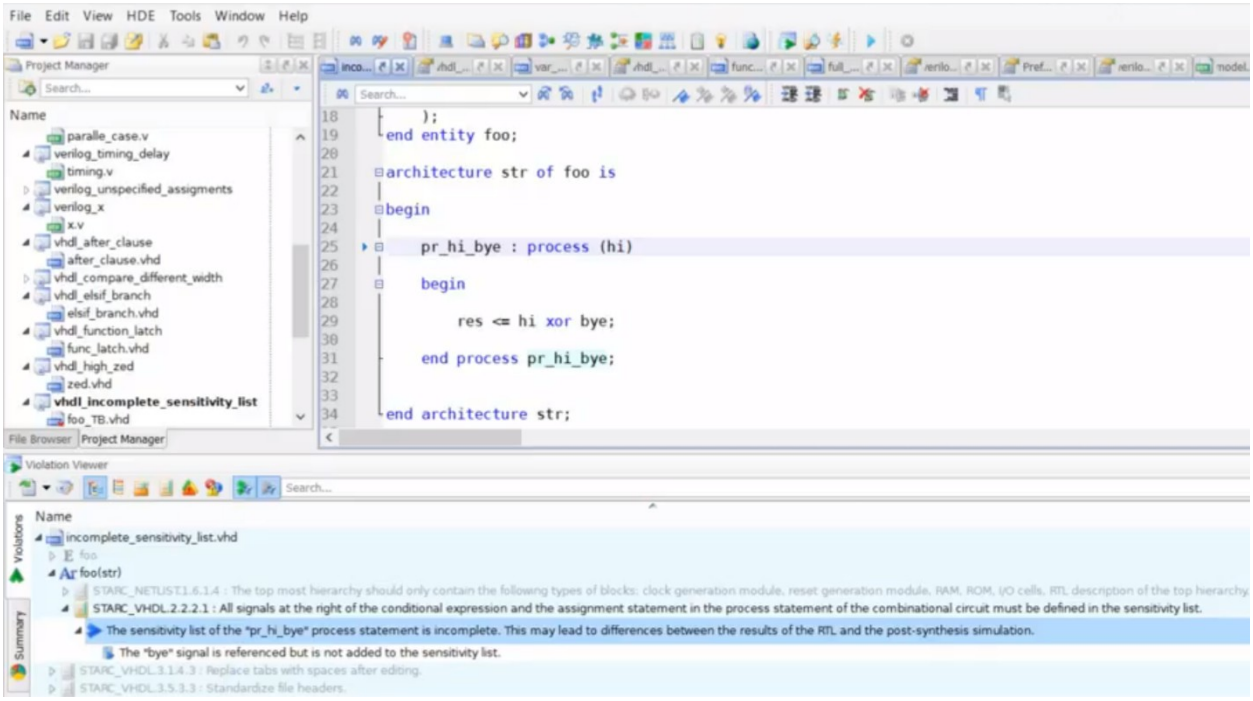

Рисунок 2.7 – Приклад роботи ALINT-PRO

ALINT-PRO підтримує 2 різні методики статичного аналізу: повну перевірку і блокову перевірку. Обидва методи доповнюють один одного і зазвичай застосовуються на різних етапах циклу проектування.

Повна перевірка на рівні чіпа виконує аналіз ієрархії проекту в цілому, з конкретними значеннями загальних параметрів, які розповсюджуються від екземплярів верхнього рівня до кінцевих підмодулей, а також із застосуванням призначених для користувача тимчасових обмежень. Ця методологія традиційна для типових інструментів статичного аналізу HDL і частіше використовується під час періодів приймання проекту, або для типів перевірки, які вимагають одночасного вивчення всього проекту в якості вже інтегрованої системи (перевірки синхронізації і скидання, CDC, RDC, DFT). Ця методологія передбачає, що весь або, принаймні, велика частина тестованого проекту вже реалізована.

Фрагментація блоків – це відносно новий підхід, особливий вид розробки, який розглядає блоки проектування HDL незалежно один від одного і використовує стандартні значення загальних параметрів. Цей стиль обробки дозволяє пізніше додавати відсутні елементи в проектах, такі як ще не реалізовані частини ієрархії. Це робить модульний статичний аналіз найбільш цінним як при застосуванні до окремих вихідних файлів, незалежно від інших файлів, так і відразу після того, як файли були змінені дизайнером. Тому дуже важливі перевірки правил виконуються дуже рано, майже відразу після написання самого коду, що скорочує цикл зворотного зв'язку для дизайнера від днів або тижнів до декількох секунд. Додатком для блокової перевірки є інтеграція з HDL-редакторами, причому блокова перевірка виконується у фоновому режимі, що передбачено в останніх версіях Aldec Active-HDL і Riviera-PRO, а також в деяких популярних сторонніх редакторах HDL.

Зрозуміло, об'єднання блоків не усуває необхідність використання повної традиційної перевірки, оскільки поєднання незалежних блоків може створити додаткові проблеми після інтеграції. Проте, систематичне застосування одиничного статичного аналізу може значно скоротити кількість перевірок під час періодів приймання проекту.

ALINT-PRO включає в себе бібліотеки правил, засновані на рекомендаціях з проектування STARC (Центр наукових досліджень напівпровідникових технологій) і RMM (Керівництво по методиці повторного використання), в яких використовуються кращі методи розробки дизайну, використовувані компаніями в усьому світі. Для критично важливих проектів Aldec надає бібліотеки правил, засновані на рекомендаціях DO-254, орієнтованих на аналіз критичних проблем, що впливають на стабільність проекту.

Стратегія верифікації в ALINT-PRO складається з трьох ключових елементів: статична структурна верифікація, настройка проектних обмежень і динамічна функціональна верифікація. Перші два кроки виконуються в ALINT-PRO, в той час як динамічні перевірки здійснюються за допомогою інтеграції з симуляторами (підтримуються Riviera-PRO, Active-HDL і ModelSim) на основі автоматично створеного випробувального тесту. Цей підхід виявляє потенційні проблеми з метастабільністю при моделюванні RTL, які в іншому випадку потребували б виявлення за допомогою лабораторних тестів. Налагодження проблем CDC і RDC досягається за допомогою потужних схем і механізмів перехресного зондування HDE, а також всебічних звітів і API на основі TCL, який дозволяє переглядати результати синтезу, синхронізувати і скидати структури, виявляти перетин доменів «clock» і «reset», а також ідентифікувати синхронізатори [17].

3 РОЗРОБКА АЛГОРИТМУ

На рисунку 3.1 показана блок-схема алгоритму для розроблюваної системи. Кожен проект перевіряється за наступним алгоритмом:

Першим кроком є взаємодія з джерелом даних – базою проектів. даних База проектів представляє собою таблицю Excel з переліком проектів для перевірки, кожен проект має номер варіанту, прізвище студента, та локальний шлях до папки з файлами проекту, як показано на рисунку 3.2. На цьому етапі відбувається зчитування з файлу інформації про проекти. Якщо в базі є неперевірений проект, він обирається для тестування.

Далі відбувається верифікація – перевірка проекту на відповідність вимогам варіанту. Якщо перевірка проходить не вдало – завдання вважається невиконаним і відповідний результат заноситься до бази результатів.

У разі позитивного результату верифікації виконується перевірка на коректність виконання. Проект компілюється за допомогою консольної версії Active HDL.

На основі результатів компіляції приймається рішення чи виконано завдання правильно і далі відповідний результат заноситься до бази [18].

База результатів представляє собою таблицю Excel, що містить прізвище студента, варіант завдання та результат перевірки у вигляді повідомлення, як показано на рисунку 3.3. У разі успішного проходження всіх перевірок, повідомлення буде виглядати так: «Проект пройшов усі перевірки», у разі не проходження перевірки на етапі верифікації чи тестування, повідомлення буде виглядати відповідно: «Варіант не відповідає» або «Компіляція з помилками».

Рисунок 3.1 – Блок-схема алгоритму

	A	B	C
1	Surname	Variant	Path
2	Karnitskaya	1	D:\Testing\Task1\Karnitskaya _var1
3	Hvorostovsky	3	D:\Testing\Task1\Hvorostovsky_var3
4	Hasymov	6	D:\Testing\Task1\Hasymov_var6
5	Tatishev	11	D:\Testing\Task1\Tatishev_var11
6	Dzhumayev	16	D:\Testing\Task1\Dzhumayev_var16
7	Tsvetaeva	4	D:\Testing\Task1\Tsvetaeva_var4
8	Chistovich	7	D:\Testing\Task1\Chistovich_var7
9	Vasilyev	2	D:\Testing\Task1\Vasilyev _var2
10	Georgiyev	8	D:\Testing\Task1\Georgiyev_var8
11	Dudinsky	10	D:\Testing\Task1\Dudinsky_var10
12	Yeremeyeva	9	D:\Testing\Task1\Yeremeyeva_var9
13	Vorobyov	5	D:\Testing\Task1\Vorobyov_var5
14	Zaytsev	13	D:\Testing\Task1\Zaytsev_var13
15	Lapshin	12	D:\Testing\Task1\Lapshin_var12

Рисунок 3.2 – Приклад бази проектів

	A	B	C
1	Surname	Variant	Result
2	Karnitskaya	1	All checks passed
3	Hvorostovsky	3	All checks passed
4	Hasymov	6	Variant does not match
5	Tatishev	11	All checks passed
6	Dzhumayev	16	All checks passed
7	Tsvetaeva	4	Compiled with errors
8	Chistovich	7	All checks passed
9	Vasilyev	2	Variant does not match
10	Georgiyev	8	Variant does not match
11	Dudinsky	10	Compiled with errors
12	Yeremeyeva	9	All checks passed
13	Vorobyov	5	All checks passed
14	Zaytsev	13	Compiled with errors
15	Lapshin	12	All checks passed

Рисунок 3.3 – Приклад бази результатів

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розроблювана автоматизована система буде активно взаємодіяти з файловою системою, тому розглянемо основні методи роботи з файлами в мові програмування Python.

Щоб отримати можливість працювати з файлом в Python, для початку його необхідно створити. Зробити це можна стандартними засобами операційної системи, перейшовши в потрібний каталог і створивши новий документ. Однак аналогічна дія виконується і за допомогою методу «open» в, з яким треба передати в якості параметрів назву файлу і режим його обробки.

Наступний код демонструє отримання змінної «file» посилання на новий документ. Якщо запустити цю програму, вона створить текстовий файл «test.txt» в папці, де зберігається вихідний код:

```
file = open ( "test.txt", "w");
file.close ();
```

Якщо ж файл з вказаним ім'ям «test.txt» вже існує в каталозі з кодом, програма просто продовжить роботу з ним, не створюючи новий документ. Як можна помітити, ім'я файлу є першим параметром методу «open». Відразу за ним йде спеціальна літера, яка позначає метод обробки даних. В даному випадку «w» означає «write», тобто запис.

Після виконання будь-яких маніпуляцій над файлом, його обов'язково слід закрити за допомогою функції «close», щоб гарантовано уникнути втрати інформації.

У попередньому прикладі для доступу до файлу був використаний відносний шлях, який не містить в собі вичерпних відомостей про місцезнаходження об'єкта на жорсткому диску. Для того, щоб задати шлях до файлу на жорсткому диску, його необхідно прописати в якості першого аргументу функції «open».

Перед строковим літералом можна використовували символ «r», для відключення екранування. Інакше компілятор вважатиме послідовність «\t» як символ табуляції і поверне помилку.

Режим відкриття задається за допомогою спеціальних символів, які використовуються в мові Python при відкритті файлу. Вони вказують програмі, як саме потрібно відкривати. Всі вони представлені в таблиці 4.1, яка містить їх сигнатуру і короткий опис призначення.

Таблиця 4.1 – Режими відкриття файлу

Символ	Значення
r	Відкриття для читання
w	Відкриття для запису, а якщо файду не існує по заданому шляху, то створюється новий
x	Відкриття для запису, але тільки якщо файлу ще не існує, інакше буде видано виняток
a	Відкриття на додаткову запис, щоб інформація додавалася в кінець документа
b	Відкриття в бінарному режимі
t	Відкриття в текстовому режимі
+	Відкриття одночасно на читання і запис

Користуючись другим аргументом методу «open», можна комбінувати різні режими роботи з файлами, вказуючи, наприклад, «rb» для читання записаних даних в двійковому режимі.

Ще один приклад: відміна "r +" і "w +" полягає в тому, що в другому випадку створиться новий файл, якщо такого немає. У першому ж випадку виникне виключення. При використанні "r +" і "w +" файл буде відкритий і на читання і на запис.

Об'єкт, який повертає функція «open», містить посилання на існуючий файл. Також в ньому є інформація про створений документі, яка представлена у вигляді чотирьох основних полів. Всі вони описані в таблиці 4.2, яка містить їх імена і значення.

Таблиця 4.2 – Інформація про файл

Властивість	Значення
name	Повертає ім'я файлу
mode	Повертає режим, в якому був відкритий
closed	Повертає true, якщо файл закритий
softspace	Повертає true, якщо при виведенні даних з файлу не слід окремо додавати символ пробілу

Щоб отримати властивості файлу, досить скористатися оператором доступу, тобто точкою.

Запис в файл в Python здійснюється за допомогою методу «write». Метод викликаємо у об'єкта, який посилається на існуючий файл. Важливо пам'ятати, що для цього слід попередньо відкрити документ за допомогою функції «open» і вказати режим запису символом «w». Метод «write» приймає в якості аргументу дані, які потрібно помістити в текстовий файл

Якщо необхідно додати нову інформацію до записаним раніше даними, слід заново викликати функцію «open», вказавши їй як режим роботи символ «a». В іншому випадку всі відомості з файлу «test.txt» будуть повністю вилучені.

Для читання інформації з файлу, слід викликати метод «read» через об'єкт, який посилається на існуючий документ. Також необхідно не забувати вказувати «r» в якості другого параметра функції «open» при відкритті текстового файлу.

При відкритті може виникнути помилка. Наприклад, якщо зазначений файл не знайдений. Тому треба оброблювати винятки. В Python можна скористатися конструкцією «with», в такому разі не треба буде обробляти виключення і навіть закривати файл.

Також в розроблюваній системі будуть широко використовуватись функції для роботи з рядками. В Python вони є самостійним типом даних, а

значить за допомогою вбудованих функцій мови над ними можна робити різні операції і форматувати їх для подальшого використання.

Отримати новий рядок можна кількома способами: за допомогою відповідного літерала або ж викликавши готову функцію. Розглянемо перший метод, який продемонстрований нижче. Тут змінна string отримує значення some text, завдяки оператору присвоювання:

string = 'some text';

Щоб задати власне форматування тексту, досить застосовувати спеціальні керуючі символи зі зворотним слешем, як це показано в наступному прикладі. Тут використовується символ табуляції \t, а також знак переходу на новий рядок \n:

string = "some \ ttext \ nnew line here";

Службові символи для форматування рядків виконують свої функції автоматично, але іноді це заважає, наприклад, коли потрібно зберегти шлях до файлу на диску. Щоб їх відключити, необхідно застосувати спеціальний префікс r перед першою лапками литерала. Таким чином, зворотні слеши будуть ігноруватися програмному забезпеченні під час її запуску.

string = r"D:\dir\new";

Таблиця 4.3 демонструє перелік використовуваних в мові Python службових символів для форматування рядків. Як правило, більшість з них дозволяють змінювати положення каретки для виконання перекладу рядка, табуляції або повернення каретки.

Таблиця 4.3 – Символи для форматування рядків

Символ	Призначення
\n	Переведення каретки на новий рядок
\b	Повернення каретки на один символ назад
\f	Переведення каретки на нову сторінку
\r	Повернення каретки на початок рядка
\t	Горизонтальна табуляція
\v	Вертикальна табуляція

Символ	Призначення
\0	Символ Null

Виконати форматування окремих частин рядка, задавши в якості її компонентів якісь об'єкти програми дозволяє символ %, вказаний після літерала.

Більш зручне форматування виконується за допомогою функції format. Їй необхідно передати в якості аргументів об'єкти, які повинні бути включені в рядок, а також вказати місця їх розташування за допомогою числових індексів, починаючи з нульового:

```
string = "text";
number = 10;
newString = "this is {0} and digit {1}". format (string, number);
```

Дуже часто використовується для приведення типів до строковому увазі функція str. З її допомогою можна створити новий рядок з літерала, який приймає в якості аргументу. Даний приклад демонструє ініціалізацію

змінної string новим значенням some text:

```
string = str("some text");
```

Аргументом цієї функції можуть бути змінні різних типів, наприклад числа або списки. Ця функція дозволяє в Python перетворити в рядок різні типи даних. Для отримання довжини рядка в символах використовується функція len. Метод find дозволяє здійснювати пошук в рядку. За допомогою нього в Python можна знайти одиночний символ або підрядок в будь-якій іншій послідовності символів. Як результат свого виконання він повертає індекс першої літери шуканого об'єкта, при цьому нумерація здійснюється з нуля.

Метод replace служить для заміни певних символів або підрядків на введену програмістом послідовність символів.

Для того щоб розділити рядок на кілька підрядків за допомогою зазначеного роздільника, слід викликати метод split. За замовчуванням його роздільником є пробіл.

Виконати зворотне перетворення, перетворивши список рядків в один можна за допомогою методу join. Метод strip використовується для автоматичного видалення пробілів з обох сторін рядка.

Ознайомитися з функціями і методами, використовуваними в Python 3 для роботи з рядками можна з таблиці 4.4. У ній також наведено методи, що дозволяють взаємодіяти з регістром символів.

Таблиця 4.4 – Методи для роботи з рядками

Метод	Призначення
str(obj)	Перетворює об'єкт до рядкового вигляду
len(s)	Повертає довжину рядка
find(s, start, end), rfind(s, start, end)	Повертає індекс першого і останнього входження підрядка в s в межах від start до end

Продовження таблиці 4.4

Метод	Призначення
replace(s, ns)	Змінює обрану послідовність символів в s на нову підрядок ns
split(c)	Розбиває на підрядки за допомогою обраного роздільник c
join(c)	Об'єднує список рядків в одну за допомогою обраного роздільника c
strip(s), lstrip(s), rstrip(s)	Прибирає пробіли з обох сторін s, тільки зліва або тільки справа
center(num, c), ljust(num, c), rjust(num, c)	Повертає відцентрований рядок, вирівняний по лівому і по правому краю з довжиною num і символом c по краях
lower(), upper()	Переклад всіх символів в нижній і верхній регістр

startwith(ns), endwith(ns)	Перевіряє, чи починається або закінчується рядок підрядком ns
islower(), isupper()	Перевіряє, чи складається рядок тільки з символів в нижньому і верхньому регістрі
isalpha()	Перевіряє, чи складається рядок тільки з букв
isdigit()	Перевіряє, чи складається рядок тільки з цифр
isnumeric()	Перевіряє, чи є рядок числом

Для перевірки вихідного коду також будуть використовуватися регулярні вирази. В Python для роботи з регулярними виразами є модуль «re». Для використання його потрібно імпортувати:

import re;

Найчастіше регулярні вирази використовуються для:

пошуку в рядку;

розбиття рядка на підрядка;

заміни частини рядка.

Бібліотека «re» надає наступні методи для цих завдань:

метод re.match(pattern, string), шукає по заданому шаблону на початку рядка;

методи start() і end(), використовуються для того, щоб дізнатися початкову і кінцеву позицію знайденого рядка;

метод re.search(pattern, string), схожий на match(), але він шукає не тільки на початку рядка. На відміну від попереднього, search() поверне об'єкт;

метод re.findall(pattern, string), повертає список всіх запропонованих варіантів. У методу findall() немає обмежень на пошук на початку або наприкінці рядка. Для пошуку рекомендується використовувати саме findall(), так як він може працювати і як re.search(), і як re.match();

метод re.split(pattern, string, [maxsplit = 0]), розділяє рядок за заданим шаблоном;

метод re.sub(pattern, repl, string), шукає шаблон в рядку і заміняє його на зазначений підрядок. Якщо шаблон не знайдений, рядок залишається незмінним.

Приведені вище методи вирішують проблему пошуку певної послідовності символів. Але що, якщо у нас немає певного шаблону, і нам треба повернути набір символів з рядка, який відповідає певним правилам? Таке завдання часто стоїть при добуванні інформації з рядків. Це можна зробити, написавши вираз з використанням спеціальних символів. Найбільш часто використовувані з них наведено у таблиці 4.5.

Таблиця 4.5 – Методи для роботи з рядками

Оператор	Опис
.	Один будь-який символ, крім нового рядка \n.
?	0 або 1 входження шаблону зліва
+	1 і більше входжень шаблону зліва
*	0 і більше входжень шаблону зліва
\w	Будь-яка цифра або буква (\W – все, крім букви або цифри)
\d	Будь-яка цифра [0-9] (\D - все, крім цифри)
\s	Будь-який символ пробілу (\ S – символ не пробілу)
\b	Межа слова
[..]	Один із символів в дужках ([^ ..] – будь-який символ, крім тих, що в дужках)
\	Екранування спеціальних символів (\. Означає точку або \+ – знак «плюс»)
^ и $	Початок і кінець рядка відповідно
{n,m}	Від n до m входжень ({, m} – від 0 до m)
a|b	Відповідає a або b
()	Групує вираз і повертає знайдений текст
\t, \n, \r	Символ табуляції, нового рядка і повернення каретки

	відповідно

4.1 Шаблони перевірки

Для кожного варіанту розроблено шаблон, відповідно до якого проект буде перевірено на відповідність варіанту. Він містить набір змінних, які вказані у завданні. Приклад шаблону для завдання приведено на рисунку 4.1.

```
{
  "model": {
    "keywords": [
      "library",
      "use",
      "entity",
      "port",
      "architecture",
      "signal",
      "begin",
      "process"
    ],
    "in_type": "std_logic",
    "in": [ "a", "b", "c" ],
    "out_type": "std_logic",
    "out": [ "f" ],
    "signals_type": "std_logic",
    "signals": [ "a0", "b0", "c0" ]
  },
  "testbench": {
    "keywords": [
      "library",
      "use",
      "entity",
      "port",
      "architecture",
      "component",
      "signal",
      "begin",
      "process",
      "map"
    ],
    "in_type": "std_logic",
    "in": [ "a", "b", "c" ],
    "out_type": "std_logic",
    "out": [ "f" ],
    "signals_type": "std_logic",
    "signals": [ "a0", "b0", "c0" ]
  }
}
```


Рисунок 4.1 – Приклад шаблону для варіанту завдання

Шаблони для кожного варіанту будуть зберігатися окремо, у json-файлах, наприклад, «var1.json». Вони можуть бути легко змінені у ході навчального процесу, якщо вимоги до завдання зміняться.

JSON – текстовий формат даних і багато середовищ програмування мають можливість читати (аналізувати) і генерувати його. Об'єкт JSON це

формат даних – ключ-значення, який зазвичай знаходиться в фігурних дужках. Пари ключ-значення розділені двокрапкою, наприклад "key": "value".

Кожна пара значень розділена двокрапкою, таким чином середина JSON виглядає так: {"key": "value", "key": "value", "key": "value"}.

Ключі в JSON знаходяться з лівого боку від двокрапки. Їх потрібно обертати в дужки, як з "key" і це може бути будь-який рядок. У кожному об'єкті, ключі повинні бути унікальними. JSON значення знаходяться з правого боку від двокрапки. Вони можуть бути одним з шести типів даних: рядком, числом, об'єктом, масивом, булевим значенням або null.

На ширшому рівні, значення можуть також складатися їх складних типів даних, таких як JSON об'єкт або масив.

Загальні властивості JSON:

JSON являється форматом даних, він містить тільки властивості, без методів;

JSON вимагає подвійних лапок, які будуть використовуватися навколо рядків і імен властивостей. Одиничні лапки недійсні;

навіть одна недоречна кома або двокрапка можуть привести до збою JSON-файлу і не працювати. Рекомендується перевіряти JSON за допомогою програм, наприклад, JSONLint;

в JSON в якості властивостей можуть використовуватися тільки рядки укладені в подвійні лапки.

В шаблоні перевірки містяться наступні об’єкти:

масив ключових слів, які визначають структурні блоки;

масив назв вхідних та вихідних портів;

типи вхідних та вихідних портів;

масив назв сигналів;

тип сигналів.

4.2 Взаємодія з базою проектів

Програмне забезпечення представляє собою управляючий скрипт та скрипт тестування. Управляючий скрипт керує налаштуваннями тестування – встановлює шляхи до бази проектів та каталогу з варіантами, оброблює можливі помилки та запускає модуль тестування для кожного проекту з бази. Повний код даного модулю приведений в ДОДАТКУ А. Розглянемо детальніше методи управляючого скрипта. Спочатку оголошуємо імпорт пакетів, як показано на рисунку 4.2.

```
import os
import sys
import openpyxl
from TestModule import TestModule
```

Рисунок 4.2 – Імпорт пакетів

Для роботи скрипта знадобляться такі пакети, як «os» для роботи з операційною системою, «sys» – надає системі особливі параметри і функції, «openpyxl» – надає можливості роботи з документами електронної таблиці Excel, «TestModule» – розроблюваний модуль тестування.

Визначаємо шляхи до бази проектів, каталогу з варіантами завдань та до бази результатів, як показано на рисунку 4.3.

```
if len(sys.argv) == 2 and sys.argv[1] == "help":
    print(f"{sys.argv[0]} \"path to input excel file\" \"path to folder with variants files\"")
    sys.exit()
elif len(sys.argv) == 3 and isinstance(os.sys.argv[1], str) and isinstance(os.sys.argv[2], str):
    script = sys.argv[0]
    xl_path = sys.argv[1]
    vars_path = sys.argv[2]
else:
    print("Invalid input parameters. Enter \"help\" for more information.")
    sys.exit()

xl_path_res = xl_path.replace(".xls", "_result.xls")
```

Рисунок 4.3 – Оголошення шляхів

Метод «sys.argv» отримує список аргументів командного рядка, що передаються сценарієм Python.

Елемент «sys.argv[0]» містить шлях до скрипта. В останні два параметри з командного рядка будуть передаватися шлях до бази проектів та до каталогу з варіантами. В змінну «xl_path_res» присвоюється шлях до бази результатів.

Шляхи, передані в командний рядок, можуть бути не знайдені в файловій системі, тому перевіряємо, що база проектів та каталог з варіантами існують, що показано на рисунку 4.4.

```
if not os.path.exists(xl_path) :
    print("Table path does not exist!")
    sys.exit()

if not os.path.exists(vars_path) :
    print("Vars path does not exist!")
    sys.exit()
```

Рисунок 4.4 – Обробка помилок

За допомогою методу «os.path.exists» визначаємо, чи шляхи існують. Якщо один з шляхів не існує, за допомогою метода «print» в консоль виводиться відповідне повідомлення та, за допомогою метода «sys.exit», скрипт завершує свою роботу.

Після успішного визначення необхідних шляхів створюємо об’єкт класу «TestModule», конструктор якого приймає параметром шлях до каталогу з варіантами, що видно на рисунку 4.5.

За допомогою метода «openpyxl.load_workbook» отримуємо по вказаному шляху файл з базою проектів.

Метод «active» повертає активний лист Excel документу.

Далі файл з результатами очищується за допомогою метода «delete_cols» та за допомогою метода «cell» отримуємо доступ до комірок

таблиці і встановлюємо властивості «value» у першому рядку таблиці. Створюємо цикл, як показано на рисунку 4.6. Він буде виконуватися поки у таблиці з проектами не закінчяться рядки, на що вказує умова «i <= sheet.max_row».

```
testModule = TestModule(vars_path)

sheet = openpyxl.load_workbook(filename = xl_path).active

if os.path.exists(xl_path_res):
    xl = openpyxl.load_workbook(filename = xl_path_res)
    xls = xl.active
    xls.delete_cols(1, 3)
    xl.save(xl_path_res)
else:
    res_sheet = openpyxl.Workbook()
    res_sheet.save(xl_path_res)

res_book = openpyxl.load_workbook(filename = xl_path_res)
res_sheet = res_book.active

res_sheet.cell(row=1, column=1).value="Surname"
res_sheet.cell(row=1, column=2).value="Variant"
res_sheet.cell(row=1, column=3).value="Result"
```

Рисунок 4.5 – Створення об’єкта класу «TestModule» та бази результатів

```
i = 2
while i <= sheet.max_row:
    surname = sheet.cell(row=i, column=1).value
    variant = sheet.cell(row=i, column=2).value

    res_sheet.cell(row=i, column=1).value=surname
    res_sheet.cell(row=i, column=2).value=variant

    folder_path = sheet.cell(row=i, column=3).value

    if not (isinstance(folder_path, str) and isinstance(variant, int)):
        i += 1
        continue

    if not os.path.exists(folder_path):
        res_sheet.cell(row=i, column=3).value = "Path is wrong"
        res_err = "Path is wrong"
        i += 1
    else:
        res_err = testModule.CheckProject(folder_path, variant)
        res_sheet.cell(row=i, column=3).value = res_err
        i += 1
    print(f"{surname}\t\tvar_{variant}\t\t{res_err}")

print("All variants are checked!")
res_book.save(xl_path_res)
```

Рисунок 4.6 – Перевірка проектів у циклі

4.3 Модуль тестування

Клас «TestModule» містить методи для тестування проекту. Повний код даного модулю показаний в ДОДАТКУ Б.

Для початку роботи треба оголосити імпорт пакетів, як показано на рисунку 4.7 та визначити повідомлення про помилки, як показано на рисунку 4.8.

```
import os
import re
import sys
import json
import subprocess
```

Рисунок 4.7 – Імпорт пакетів

```
var_not_exist = "Vars path does not exist!"
proj_not_exist = "Some of project files does not exist!"
```

Рисунок 4.8 – Повідомлення про помилки

Для роботи даного скрипта знадобляться такі пакети, як «os» для роботи з операційною системою, «sys» – надає системі особливі параметри і функції, «json» – надає можливості роботи з json-файлами, «re» – надає можливість використання регулярних виразів.

Модуль «subprocess» відповідає за виконання наступних дій: породження нових процесів, з'єднання с потоками стандартного введення, стандартного виводу, стандартного виводу повідомлень про помилки і отримання кодів повернення від цих процесів. Рекомендованим підходом до роботи з підпроцесами є використання допоміжних функцій. Для більш

складних випадків може бути використаний безпосередньо інтерфейс Popen, що виконує дочірню програму в новому процесі.

Конструктор класу приймає параметром шлях до каталогу з варіантами, як показано на рисунку 4.9.

```
def __init__(self, vars_path):
    if not os.path.exists(vars_path) :
        print(var_not_exist)
        sys.exit()
    self.var_path = vars_path
```

Рисунок 4.9 – Ініціалізація класу «TestModule»

В конструкторі класу здійснюється перевірка на існування каталогу з варіантами. За допомогою методу «os.path.exists» визначаємо, чи шлях до каталогу існує. Якщо метод повертає «false», шляху не існує і за допомогою метода «print» в консоль виводиться відповідне повідомлення, а за допомогою метода «sys.exit», скрипт завершує свою роботу. Інакше змінній «var_path» присвоюється рядок зі шляхом до варіанту.

Клас «TestModule» має наступні методи:

CheckProject;

ParseForVariant;

CheckInOut;

CheckSignals;

CompileProject.

Розглянемо кожен метод детальніше.

Метод «CheckProject» показано на рисунку 4.10, він керує перевірками проекту.

Спочатку в методі оголошуються такі змінні:

«var_path», містить шлях до варіанту;

«des_path», містить шлях до VHDL-моделі;

«test_path», містить шлях до VHDL-тестбенчу.

Метод «CheckProject» містить перевірки та коректність шляхів до VHDL-моделі та VHDL-тестбенчу за допомогою методу «os.path.exists».

```
def CheckProject(self, proj_path, var):
  var_path = f"{self.var_path}\\var{var}.json"
  des_path = f"{proj_path}\\design.vhd"
  test_path = f"{proj_path}\\testbench.vhd"

  if not os.path.exists(var_path) :
                print(var_not_exist)
                return var_not_exist

  if not (os.path.exists(des_path) and os.path.exists(test_path)):
                print(proj_not_exist)
                return proj_not_exist

  if not self.__ParseForVariant(proj_path, var_path):
                return "Variant does not match"

  if not self.__CompileProject(proj_path):
                return "Compiled with errors"

  return "All checks passed"
```

Рисунок 4.10 – Метод «CheckProject»

Рядки зі строковими літералами «f» на початку і фігурними дужками містять змінні, які в подальшому будуть замінені своїми значеннями. Якщо метод повертає «false», метод припиняє своє виконання і повертає в управляючий модуль відповідне повідомлення.

Якщо всі перевірки пройдені успішно, переходимо до виконання перевірки на відповідність варіанту.

На рисунку 4.11 показано метод «ParseForVariant», який перевіряє проект на відповідність варіанту.

Клас «json» надає методи для роботи з json-форматом, дозволяє кодувати і декодувати дані в зручному форматі. Його метод «json.load»

дозволяє отримати об’єкт з json-файлу, у метод «open» передається шлях до певного шаблону.

Для доступу до значень json-об’єкту в квадратних дужках треба задати ключ: «json_content["model"]». В змінну «model_content» записуємо код VHDL-моделі. Спочатку перевіряємо наявність всіх структурних блоків, якщо ця перевірка завершується вдало, починається перевірка на відповідність вхідних та вихідних портів за допомогою методу «CheckInOut» та перевірка на відповідність сигналів за допомогою методу «CheckSignals».

```
def __ParseForVariant(self, proj_path, var_path):
  json_content = json.load(open(var_path))

  model = json_content["model"]

  model_content = open(f"{proj_path}/design.vhd").read()

  for word in model["keywords"]:
      if word not in model_content:
         return False

  if not (self.__CheckInOut(model_content, model["in"], model["in_type"], "[ :]in ")
   and self.__CheckInOut(model_content, model["out"], model["out_type"], "[ :]out ")
   and self.__CheckSignals(model_content, model["signals"], model["signals_type"])):
      return False

  testbench = json_content["testbench"]

  testbench_content = open(f"{proj_path}/testbench.vhd").read()

  for word in testbench["keywords"]:
      if word not in testbench_content:
         return False

  if not (self.__CheckInOut(testbench_content, testbench["in"], testbench["in_type"], "[ :]in ")
   and self.__CheckInOut(testbench_content, testbench["out"], testbench["out_type"], "[ :]out ")
   and self.__CheckSignals(testbench_content, testbench["signals"], testbench["signals_type"])):
     return False

  return True
```

Рисунок 4.11 – Метод «ParseForVariant»

Далі, якщо попередня перевірка пройшла успішно, в змінну «testbench» записуємо код VHDL-тестбенчу та знову перевіряємо наявність всіх структурних блоків та відповідність вхідних та вихідних портів за

допомогою методу «CheckInOut». Коли хоча б один метод повертає «false» виконання перевірки завершується та формується відповідний результат.

Розглянемо детальніше метод «CheckInOut», що показаний на рисунку 4.12. Він приймає в параметри зміст vhd-файлу(код VHDL-моделі або VHDL-тестбенчу), список вхідних та вихідних портів, їх типи та паттерн для пошуку відповідності. В даному методі використовуються функції для роботи з регулярними виразами з модулю «re».

```
def __CheckInOut(self, file_content, in_list, types_list, pattern):
  i = len(re.findall(pattern, file_content))
  in_cpy = in_list.copy()
  find_index = 0

  while i:
      reg_exp_res = re.search(pattern, file_content[find_index:])
      type_str = file_content[reg_exp_res.end():file_content.find(";", reg_exp_res.end())]
      in_str = file_content[file_content.rfind("\n", 0, reg_exp_res.start()): reg_exp_res.start()]

      cnt_list = len(types_list)
      for t in types_list:
          if not type_str.upper().find(t.upper()) == -1:
              types_list.remove(t)
              break

      if len(types_list) == cnt_list:
          return False

      for v in in_list:
          if not re.search(("[ (,]" + v.upper() + "[ :,]"), in_str.upper()) == None:
              in_cpy.remove(v)

      find_index = reg_exp_res.end
      i -= 1

  if len(in_cpy) == 0:
      return True
  else:
      return False
```

Рисунок 4.12 – Метод «CheckInOut»

Для перевірки сигналів служить метод «CheckSignals», який показано на рисунку 4.13. Він приймає в параметри зміст vhd-файлу(код VHDL-моделі або VHDL-тестбенчу), список сигналів та їх типи. Для роботи з регулярними виразами використовуються функції з модулю «re».

Якщо перевірки на відповідність варіанту пройшли успішно, скрипт переходить до виконання методу «CompileProject» для компіляції проекту.

Компіляція VHDL-проекту виконується за допомогою Active-HDL Batch Mode. Щоб запустити Active-HDL в режимі оболонки, треба запустити файл «vsimsa.bat», який знаходиться в директорії встановлення Active-HDL.

Файл «vsimsa.bat» – це сценарій обгортки для симулятора пакетного режиму bin\vsimsa.exe. Коли запускається оболонка «vsimsa», у командному рядку можна виконувати команди Active-HDL, наприклад:

creating library

$ alib work

ALIB: Library `work' attached.

work = c:\my_designs\fpc_pll\work\work.lib

setting work library as the default target for all commands

$ set worklib work

VSIMSA: 'work' is now working library.

```
def __CheckSignals(self, file_content, sig_list, types_list):
  pattern = " signal "
  i = len(re.findall(pattern, file_content))
  sig_cpy = sig_list.copy()
  find_index = 0

  while i:
      find_index = file_content.find(pattern, find_index)
      type_str = file_content[file_content.find(":", find_index):file_content.find(";", find_index)]
      sig_str = file_content[find_index + len(pattern) - 1 : file_content.find(":", find_index) + 1]

      cnt_list = len(types_list)

      for t in types_list:
          if not type_str.upper().find(t.upper()) == -1:
              types_list.remove(t)
              break

      if len(types_list) == cnt_list:
          return False

      for v in sig_list:
          if not re.search(("[ ,]" + v.upper() + "[ :,]"), sig_str.upper()) == None:
              sig_cpy.remove(v)

      find_index += len(pattern)
      i -= 1
  if len(sig_cpy) == 0:
      return True
  else:
      return False
```

Рисунок 4.13 – Метод «CheckSignals»

Крім того, команди Active-HDL можливо помістити у файл макросу з розширенням .do або .tcl і виконати їх у пакетному режимі:

```
$ do runsim.do
```

Сценарій макросу runsim.do може містити такі команди:

```
# creating library
alib work
# setting work library as the default target for all commands
set worklib work
# compiling source files
acom fpc.vhd fpcTB.vhd
# starting simulation with tb_top as the top level module
asim fpc_tb
# running the simulation
run 1000us
# closing the simulation
endsim
quit
```

Також можна передати файл макросу як аргумент при запуску «vsimsa» і створити файл журналу з файлом «.bat», що містить такі рядки коду:

```
vsimsa -do runsim.do > mylog.log
exit
```

Командний рядок «vsimsa» повернеться, коли виконання «vsimsa» з макросом буде завершено без помилок. Щоб закрити оболонку «vsimsa» треба закінчити сценарій макросу командою «quit». Якщо помилки при виконанні макросу з’являються помилки, «vsimsa» потрапляє до оболонки інструменту, що дозволяє прочитати всі повідомлення та визначити джерело

проблеми. Для того, щоб «vsimsa» продовжувала виконувати макрос після повідомлення про помилку, додаємо наступний запис у файл макросу:

$ onerror {resume}

Найважливіші команди Active-HDL доступні як окремі виконувані файли та можуть бути викликані безпосередньо з оболонки операційної системи. Усі зовнішні команди знаходяться в підкаталозі «/bin» каталогу встановлення Active-HDL. Зовнішні команди можна використовувати для управління бібліотекою, компіляції, моделювання тощо.

Розглянемо список найбільш часто використовуваних команд Active-HDL.

Команди управління бібліотекою:

vlib (створює нову бібліотеку);

vlist (перелічує бібліотеки, видимі з поточного каталогу);

vdir (перелік вмісту бібліотеки);

vdel (видаляє скомпільовані одиниці з бібліотеки);

vmap (додає або видаляє зіставлення бібліотек).

Команди компіляції:

vcom (компілює файли VHDL);

vlog (компілює файли Verilog);

ccomp (компілює файли SystemC);

addcs (імпортує модулі SystemC із спільної бібліотеки об’єктів або бібліотеки динамічних посилань).

Повний приклад сценарію оболонки, який викликає зовнішні команди Active-HDL для компіляції та моделювання дизайну:

```
# create library v_bjack
vlib v_bjack
# compile source files
# working library is specified with –work switch
vlog -work v_bjack c:/my_designs/samples_83/v_bjack/src/*.v
```

```
# initialize simulation
vsim  -c -lib v_bjack V_BJACK_tb
run -all
endsim
```

Для запуску моделювання в пакетному(консольному) режимі потрібно використовувати «-c» параметр із зовнішньою командою «vsim». Якщо пропустити параметр «-c», буде запущено моделювання в режимі графічного інтерфейсу.

Active-HDL зберігає результати моделювання у «wave.asdb»,

що є значенням за замовчуванням для $waveformoutput, і може бути змінено за допомогою команди «set» з оболонки «vsimsa».

Запис сигналів за допомогою команди «trace». Ця команда відстежує історію вказаних сигналів у файлі бази даних симуляції (.asdb). За замовчуванням ім'я бази даних моделювання – wave.asdb. За замовчуванням цей файл створюється у поточному каталозі. Шлях та ім'я файлу форми сигналу можна замінити аргументом -asdb <ім'я файлу>, переданим команді asim. Важливо закрити "wave.asdb" в Active-HDL перед запуском будь-яких команд vsimsa або макросів.

Трасування всіх портів рекурсивно оголошених в області проектування UUT:

```
# create library v_bjack
log
log hands_played
vlib v_bjack
# compile source files
# working library is specified with –work switch
vlog -work v_bjack c:/my_designs/samples_83/v_bjack/src/*.v
# initialize simulation
vsim  +access +r -c -lib v_bjack V_BJACK_tb -do
```

```
trace -rec -ports UUT/*
run -all
log
endsim
```

Для відображення записаних сигналів за допомогою Waveform Viewer потрібно запустити Active-HDL та відкрити відповідний файл сигналу «wave.asdb».

Метод «CompileProject» показано на рисунку 4.14. Active-HDL-макромова надає можливість працювати в Active-HDL середовищі, не використовуючи її графічний інтерфейс. Дії можуть бути введені та виконані безпосередньо у консолі. Для ініціалізації одиничних команд такий спосіб роботи не дуже зручний. Але макрокоманди незамінні, коли слід виконати велику послідовність операцій. В такому випадку створюється текстовий файл, який містить команди – макрофайл або командний файл, який запускається на виконання. Для цього слід набрати команду в консолі:

do <filename> [<parameter_value> ...]

Параметр <filename> відповідає імені виконуваного макро-файла. Можна вказати повний шлях розташування файлу. Якщо ніякий шлях не вказано, за замовчуванням мається на увазі поточна робоча директорія Active-HDL.

Можна також ввести необов'язкові аргументи <parameter_value>, які будуть передані в макро-файл через параметри $1 ... $99. Якщо в командному рядку описано менше параметрів, ніж в макро-файлі, неописані параметри будуть замінені символом нового рядка.

Для виклику макро-файла будемо використовувати наступну команду:

do macro.do

Слід звернути увагу, що оператор «do» є макрокомандою і може викликати командний файл з аналогічного файлу. У середовищі Active-HDL

макро-файли є файлами ресурсу, тому можуть бути включені в проект як і будь-які інші ресурси.

Метод «CompileProject» приймає шлях до проекту як параметр. У каталозі з проектом створюємо макро-файл «macro.do». На випадок якщо файл існує, видаляємо його за допомогою метода «os.remove». За допомогою «open» відкриваємо файл для запису, на що вказує параметр «x» – відкриття на запис, якщо файлу не існує, інакше виняток.

```
def __CompileProject(self, proj_path):

  design_path = proj_path + r"\design.vhd"
  testbench_path = proj_path + r"\testbench.vhd"

  f = open(testbench_path, "r")
  testbench_cont = f.read()
  f.close()

  entity_name_beg = testbench_cont.find("entity ") + len("entity")
  entity_name_end = testbench_cont.find(" is", entity_name_beg)
  entity_name = testbench_cont[entity_name_beg:entity_name_end]

  sim_duration = 500

  macro_file = f"{proj_path}/macro.do"

  if os.path.exists(macro_file):
      os.remove(macro_file)

  f = open(macro_file, "x")
  f.write(f"alib  work\r\nset worklib work\r\nacom {design_path} {testbench_pat}\r\nasim
  {entity_name}\r\nrun {sim_duration}ns\r\nendsim")
  f.close()

  cmd = f"vsimsa.bat -do {macro_file}"
  PIPE = subprocess.PIPE
  p = subprocess.Popen(cmd, shell=True, stdin=PIPE, stdout=PIPE, stderr=subprocess.STDOUT)

  out = str(p.stdout.read(), "cp866")

  if out.find(" 0 Errors") < 0:
      return False

  if out.find("KERNEL: Simulation has finished.") < 0:
      return False
  else:
      return True
```

Рисунок 4.14 – Метод «CompileProject»

Далі формуємо шляхи до файлів проекту та записуємо у макро-файл за допомогою методу «write» наступні команди:

alib work

set worklib work

acom \шлях до файлу\design.vhd \шлях до файлу\testbench.vhd

asim testbench

run 500ns

endsim

Команди «alib work» та «set worklib work» відповідають за створення бібліотеки та її встановлення за замовчуванням для всіх команд. Компіляція відбувається за допомогою команди «acom», яка приймає параметром шляхи до необхідних файлів.

Команда «asim» ініціалізує симуляцію, «run» запускає її, для завершення симуляції використовуємо команду «endsim».

В консолі з робочої директорії Active-HDL викликаємо «vsimsa.bat» з параметром «-do {macro_file}», де {macro_file} містить повний шлях до створеного макро-файлу. У разі успішної компіляції у консоль буде виведено повідомлення, як показано на рисунку 4.15.

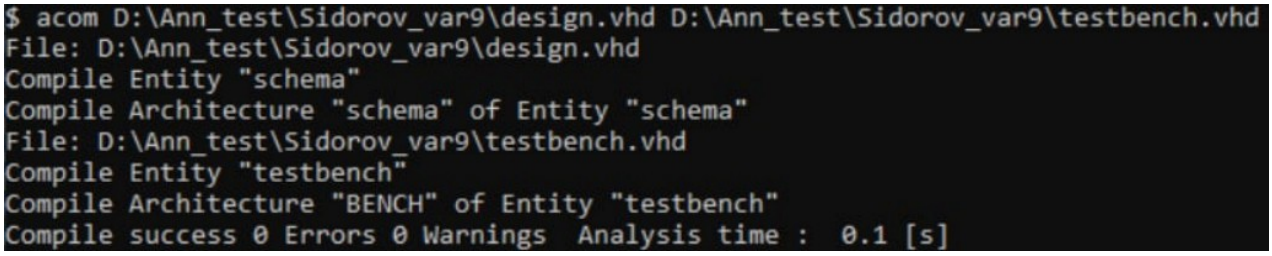

Рисунок 4.15 – Компіляція проекту

За допомогою методу «re.search» шукаємо у повідомленні підрядок «0 Errors», наявність якого свідчить про успішне виконання. Якщо його не знайдено, метод «CompileProject» повертає «False» і проект вважається не виконаним, після чого до бази результатів буде додано повідомлення

«Компіляція з помилками». В разі успішної компіляції буде здійснена симуляція, після якої формуються результати перевірки.

Якщо симуляція пройшла вдало, метод «CompileProject» повертає «True» і до бази результатів додається повідомлення «Проект пройшов усі перевірки».

Процес симуляції в командному рядку за допомогою команд Active-HDL показано на рисунку 4.16.

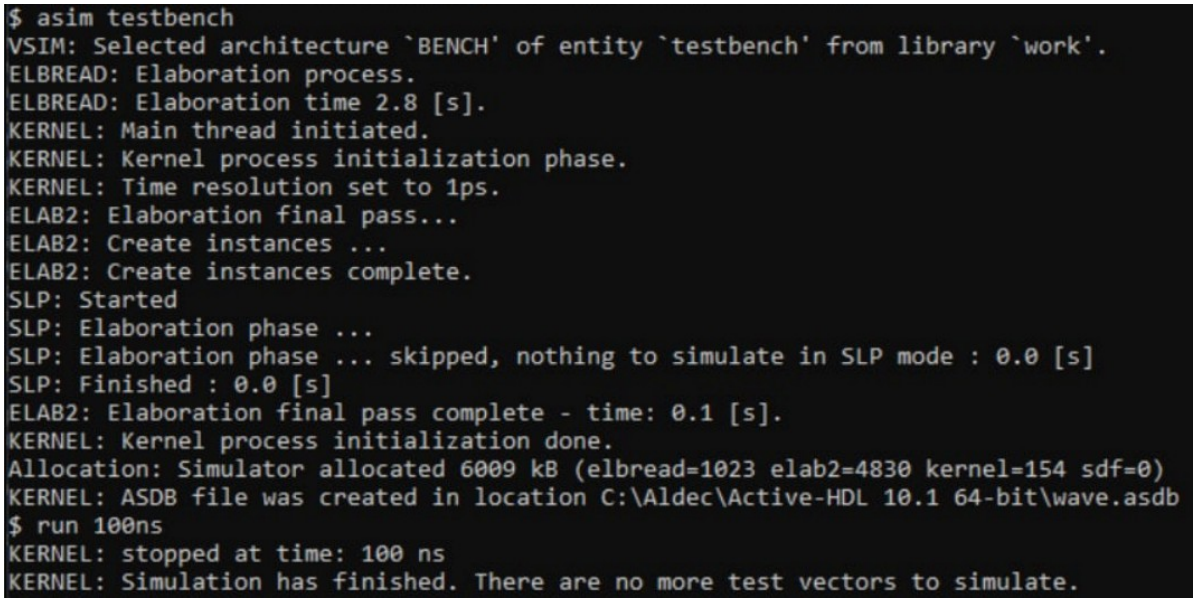

Рисунок 4.16 – Симуляція проекту

4.4 Тестування програмного забезпечення

Для тестування програмного забезпечення створимо базу проектів, як показано на рисунку 4.17.

	A	B	C
1	Surname	Variant	Path
2	Karnitskaya	1	D:\Testing\Task1\Karnitskaya _var1
3	Hvorostovsky	3	D:\Testing\Task1\Hvorostovsky_var3
4	Hasymov	6	D:\Testing\Task1\Hasymov_var6
5	Tatishev	11	D:\Testing\Task1\Tatishev_var11
6	Dzhumayev	16	D:\Testing\Task1\Dzhumayev_var16
7	Tsvetaeva	4	D:\Testing\Task1\Tsvetaeva_var4
8	Chistovich	7	D:\Testing\Task1\Chistovich_var7

Рисунок 4.17 – База проектів для тестування

Базу проектів розміщено за шляхом «D:\Testing\task1.xlsx», як показано на рисунку 4.18. В каталозі «D:\Testing\Task1» містяться проекти, як показано на рисунку 4.19. Каталог «D:\Testing\Task1_Vars» має json-файли з шаблонами для необхідних варіантів, що показано на рисунку 4.20.

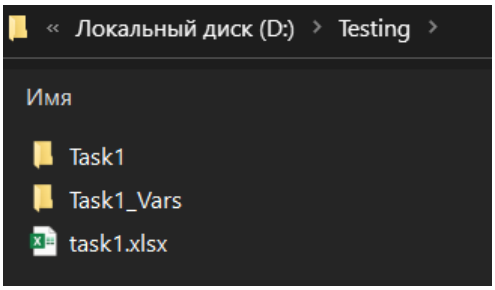

Рисунок 4.18 – Каталог з базою проектів

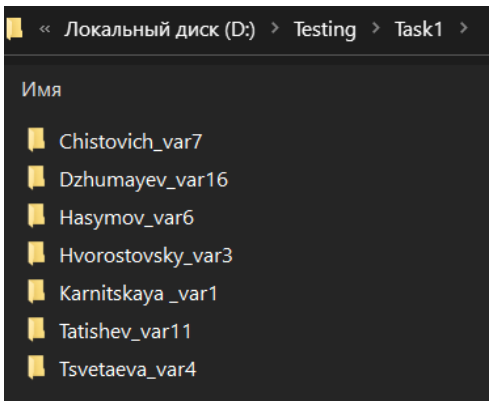

Рисунок 4.19 – Каталог з проектами

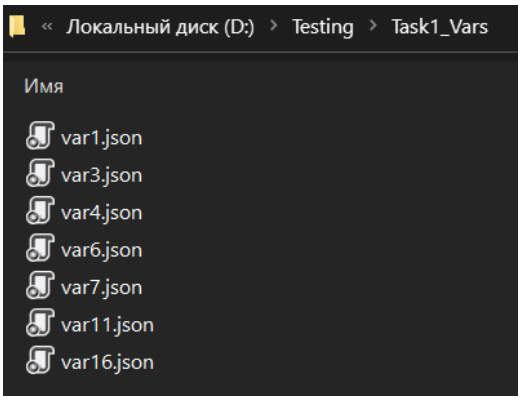

Рисунок 4.20 – Каталог з шаблонами завдань

Процес виконання скрипта показано на рисунку 4.21. В командному рядку вказується назва скрипту, другим параметром вказується шлях до бази проектів, третій параметр – шлях до каталогу з шаблонами перевірки.

```
C:\Users\Admin\Project>main.py D:\Testing\task1.xlsx D:\Testing\Task1_Vars
Karnitskaya        var_1        All checks passed
Hvorostovsky        var_3        All checks passed
Hasymov        var_6        Variant does not match
Tatishev        var_11        Variant does not match
Dzhumayev        var_16        All checks passed
Tsvetaeva        var_4        Compiled with errors
Chistovich        var_7        All checks passed
All variants are checked!
```

Рисунок 4.21 – Виконання скрипта

В даному випадку перший параметр – назва скрипта «main.py», другий – шлях до бази проектів – «D:\Testing\task1.xlsx», третім є шлях до каталогу з шаблонами перевірки – «D:\Testing\Task1_Vars». В результаті отримали рядок для запуска скрипта:

> main.py D:\Testing\task1.xlsx D:\Testing\Task1_Vars

Під час перевірки у консоль виводяться відповідні повідомлення про стан перевірених проектів, кожне з яких містить прізвище студента, варіант завдання та результат перевірки. Після завершення перевірки всіх проектів з бази, в консоль виводиться відповідне повідомлення і робота скрипта завершується.

Всі проекти було успішно перевірено. Сформовану базу результатів після тестового запуску показано на рисунку 4.22.

Розглянемо обробку помилок при запуску скрипта. У випадку вказання у командному рядку невірного шляху в параметрах, буде виведено відповідне повідомлення.

Спробуємо задати параметром неіснуючий шлях до каталогу з варіантами, як показано на рисунку 4.23.

Так як каталог з шаблонами завдань «D:\Testing\NotExist_Vars» не існує в консоль було виведено повідомлення «Vars path does not exist!».

	A	B	C
1	Surname	Variant	Result
2	Karnitskaya	1	All checks passed
3	Hvorostovsky	3	All checks passed
4	Hasymov	6	Variant does not match
5	Tatishev	11	Variant does not match
6	Dzhumayev	16	All checks passed
7	Tsvetaeva	4	Compiled with errors
8	Chistovich	7	All checks passed

Рисунок 4.22 – База результатів після тестування

```
C:\Users\Admin\Project>main.py D:\Testing\task1.xlsx D:\Testing\NotExist_Vars
Vars path does not exist!
C:\Users\Admin\Project>
```

Рисунок 4.23 – Перевірка на помилки

Також може бути не знайдена база проектів. Спробуємо передати в консоль невірний шлях до бази «D:\Testing\ NotExist.xsls», як показано на рисунку 4.24. В результаті було отримано повідомлення «Table path does not exist!».

```
C:\Users\Admin\Project>main.py D:\Testing\NotExist.xsls D:\Testing\Task1_Vars
Table path does not exist!
C:\Users\Admin\Project>
```

Рисунок 4.24 – Перевірка на помилки

У випадку, якщо в базі проектів шлях до проекту буде вказано невірно, в консоль буде виведено повідомлення «Path is wrong». Також в проекті можуть не бути необхідні файли моделі або тестбенчу, тоді в консоль буде виведено повідомлення «Some of project files does not exist!».

Якщо при перевірці скриптом не буде знайдено файл з шаблоном перевірки, в консоль буде виведено повідомлення «Vars path does not exist!».

ВИСНОВКИ

В результаті виконання атестаційної роботи проекту у першому розділі було проаналізовано предметну область питання автоматизованого контролю знань, визначено основні завдання автоматизації процесу навчання та її переваги, наведена загальна структурна схема роботи пропонованої системи та визначені основні завдання проектування.

Другий розділ містить аналіз засобів розробки програмного забезпечення, вибір мови програмування та середовища розробки. Розглянуто особливості проектування на VHDL та проаналізовано структуру VHDL-моделі та VHDL-Testbench. Також досліджено методи верифікації проектів та розглянуто інструмент для верифікації ALINT-PRO, його особливості та методи аналізу проектів.

У третьому розділі було розроблено алгоритм функціонування системи та його блок-схему, а також приведені приклади бази проектів для перевірки та бази з результатами.

В четвертому розділі було розроблено програмне забезпечення, описано основні засоби реалізації та етапи розробки системи, які включають розробку шаблонів перевірки, реалізацію взаємодії з базою проектів та модуль тестування.

На основі поставлених вимог і проведених досліджень була розроблена система, що дозволяє автоматизувати процес перевірки проектів, написаних на мові VHDL. Вона є масштабованою, що дозволяє розширювати систему залежно від вимог викладача. Також дана система має ряд істотних переваг:

– викладач звільняється від трудомістких рутинних операцій і може приділити більше часу індивідуальній роботі зі студентами;

– підвищується точність реєстрації результатів і виключаються помилки обробки вихідних даних;

– оперативність обробки даних дозволяє проводити в стислі терміни масовий контроль усіх студентів.

В основі роботи системи лежать такі етапи, як верифікація проекту та компіляція. Створена система дозволяє автоматично виявляти невідповідність виконання студентом проекту до заданого варіанту завдання та перевіряти коректність роботи проекту. Вона задовольняє всім вимогам технічного завдання і готова до застосування на практиці.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Лавров А.А., Лопатина А.А., Хаханова И.В., Разработка системы автоматизации тестирования большого объема проектов [Текст] // XIII Всеукраїнська науково-практична WEB конференція аспірантів, студентів та молодих вчених «Комп'ютерні інтелектуальні системи та мережі», 24 – 26 березня 2020 р. Харків: 2020. – С. 32-35

2. Архипова Е.Н., Кононова О.В., Крюков В.В., Шахгельдян К.И. Автоматизация рейтинговой оценки деятельности преподавателей // Университетское управление: практика и анализ. – 2010. – № 5. – С. 51-62.

3. Белова Е.В. Автоматизация организационно-управленческой деятельности как один из компонентов создания информационно-образовательной среды учебного заведения // Вестник МГПУ. Серия: Информатика и информатизация образования. – 2007. – № 9. – С. 152-155.

4. Белова Е.В., Яникова З.М., Хожаева Т.С. Автоматизация организационно-управленческой деятельности на разных уровнях системы общего образования // Вестник МГПУ. Серия: Информатика и информатизация образования. – 2008. – № 15. – С. 22-24.

5. Бова В.В. Процессо-ориентированный подход к автоматизации деятельности образовательных учреждений // Перспективные информационные технологии и интеллектуальные системы. – 2007. – № 1. – С. 50-55.

6. Говорков А.С. Автоматизация организационно-управленческих аспектов научной деятельности вуза // Университетское управление: практика и анализ. – 2009. – № 6. – С. 13-18.

7. Денисова А.Б. Средства автоматизации организационно-управленческого процесса внеучебной деятельности // Вестник РУДН. Серия: Информатизация образования. – 2013. – № 1. – С. 126-132.

8. Носова Л.С. Автоматизация деятельности учителя по разработке системы уроков // Ученые записки ИИО РАО. – 2006. – № 20. – С. 184-187.

9. Charles H. Roth, Jr. Digital Systems Design Using VHDL. – Boston: PWS Publishing Company, 1998. – 470 с

10. Свейгарт, Эл. Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих. – М.: ООО «И.Д. Вильямс», 2017. – 592с.

11. Ashenden, Peter J. The designer's guide to VHDL. – San Francisco, Calis.: California: Morgan Kaufmann Publishers, Inc, 1996. – 688 с.

12. Суворова Е. А., Шейнин Ю. Е. Проектирование цифровых систем на VHDL. – СПб.: БХВ-Петербург, 2003. – 576 с.: ил.

13. Бибило П.Н. Синтез логических схем с использованием языка VHDL. – М.: Соломон-Р, 2002. – 384 с.

14. Семенец В.В, Хаханова И.В., Хаханов В.И. Проектирование цифровых систем с использованием языка VHDL, Харьков, 2003. – 510 с.

15. Хаханов В.И., Хаханова И.В. – VHDL+Verilog = синтез за минуты. – Харьков: Смит.– 2006.– 264 с.

16. Г. В. Табунщик, Т.І. Каплієнко, О.О. Каплієнко, Д. Ван Мероде Верифікація та валідація цифрових систем управління, Запоріжжя: Дике Поле, 2017. – 150 с.

17. ALINT-PRO [Електронний ресурс] – Режим доступу: http://aldec.ru/products/alint-pro/

18. Лавров А.А., Лопатина А.А., Хаханова И.В., Разработка автоматизированной системы тестирования большого объема проектов [Текст] // XXIV Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», 7 – 9 квітня 2020 р. Харків: 2020. – С. 40-41