

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗАЦІЇ
ПЕРЕВІРКИ ПРОГРАМНОГО КОДУ
SOFTWARE TO AUTOMATE THE VERIFICATION
OF A PROGRAM CODE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-119

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

СТРІЛЬЧУК Д.В.

(ПРИЗВИЩЕ та ініціали)

Керівник КОЦУР М.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СТРІЛЬЧУКА Дмитра Віталійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення автоматизації перевірки програмного коду. Software to Automate the Verification of a Program Code

керівник проєкту (роботи) к.т.н., доцент, КОЦУР Михайло Ігорович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “04” квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 01 червня 2023 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Керівництво програміста. 5. Керівництво користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	КОЦУР М.І., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання «04» квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділи 4, 5
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Дмитро СТІЛЬЧУК
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Михайло КОЦУР
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 72 с., 1 табл., 49 рис., 2 дод., 19 джерел.

ГРАФІЧНЕ ПРОГРАМУВАННЯ, UNIT-ТЕСТУВАННЯ, МОДУЛЬНЕ ТЕСТУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ЯКІСТЬ.

Об'єкт дослідження – процес перевірки програмного коду.

Предмет дослідження – засоби перевірки програмного коду.

Мета роботи – розробка моделі програмного забезпечення для перевірки програмного коду.

Матеріали, методи та технічні засоби: графічна мова програмування «G», яка заснована на архітектурі потоків даних, персональний комп'ютер з процесором Intel Core 7 під управлінням операційної системи Microsoft Windows 10, середовище графічного програмування LabView.

Результати. Створено програмну реалізацію засобів верифікації в LabView.

Висновки. Розроблено програмну реалізацію системи для підвищення якості функціонування інформаційних моделей з використанням програмно-апаратного комплексу LabView, за рахунок верифікації.

Галузь використання – програмування.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 72 pages, 1 tables, 49 figures, 2 appendixes, 19 sources.

GRAPHIC PROGRAMMING, UNIT TESTING, MODULAR TESTING, SOFTWARE, QUALITY.

The object of research is the process of checking the software code.

The subject of the research is software code verification tools.

The purpose of this work is to develop the model for testing the software code.

Materials, methods, and tools: the graphical programming language "G" is based on a data stream architecture, personal computer with Intel Core 7 processor running Microsoft Windows 10 operating system, Labview graphical programming environment.

Results. A software implementation of verification tools in LabView has been created.

Conclusions. Developed software implementation of the system to improve the quality of information models using the software and hardware LabView, through verification.

The field of use is programming.

ЗМІСТ

	С.
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Поняття моделі та їх класифікація	12
1.2 Розгляд існуючих засобів перевірки програмного коду	14
2 Матеріали і методи.....	18
2.1 Основні методи моделювання	18
2.2 Дослідження методів верифікації програмних моделей	20
2.2.1 Експертиза.....	22
2.2.2 Статичний аналіз	23
2.2.3 Динамічні методи	23
2.2.4 Формальні методи	24
2.2.5 Синтетичні методи	24
2.3 Верифікація в LabView	27
2.3.1 Обґрунтування вибору середовища розробки.....	27
2.3.2 Верифікація правильності роботи програми.....	28
2.3.3 Перевірка вірності часу виконання	31
2.3.4 Перевірка завантаженості пам'яті.....	33
2.3.5 Відстеження виконання	34
2.4 Модульне тестування.....	35
3 Опис програми.....	37
3.1 Розробка програмної моделі	37
3.2 Модульне тестування.....	40
3.3 Структура проєкту	44
4 Керівництво програміста.....	50
4.1 Призначення застосунку.....	50
4.2 Програмне та апаратне забезпечення.....	50
4.3 Структура проєкту	52
5 Керівництво користувача	55
5.1 Призначення програми	55
5.2 Умови виконання програми	55

5.3 Виконання програми.....	55
5.4 Завершення роботи	57
Висновки	58
Перелік джерел посилань	59
Додаток А Графічне подання програмного забезпечення, розробленого за допомогою графічної мови програмування G та середовища LabVIEW	62
Додаток Б Слайди презентації	65

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BDD	– Binary Decision Diagrams;
Labview	– Laboratory Virtual Instrumentation Engineering Workbench;
SMV	– Symbolic Model Verifier;
VCEGAR	– Verilog Couter Example Guided Abrstaction Refinement;
БД	– база даних;
ВП	– віртуальний прилад;
ПЗ	– програмні засоби;
ПК	– персональний комп'ютер;
ПРЧ	– прилад реального часу.

ВСТУП

Останнім часом все більшої популярності набувають інформаційні моделі та алгоритми їх реалізації. Також, зростає складність інформаційних моделей, які розробляються [1]-[2]. Практичні застосування потребує все складніших моделей, а технології їх проєктування не можуть забезпечити необхідну якість та надійність [3]-[4]. Інформаційні моделі та алгоритми, на їх основі, можуть тривалий час зберігати і накопичувати помилки, які проявляються на етапі експлуатації. Це обумовлено складною комбінацією численних факторів, зокрема непередбачуваності інформаційних ситуацій [5]-[6] та складної взаємодії сукупності процесів. Таким чином, виникає необхідність верифікації інформаційних моделей подібно верифікації програмного забезпечення [7]-[8]. Особливість цієї технології полягає в тому, що верифікація інформаційних моделей в одних випадках може бути верифікацією алгоритму реалізації, в інших випадках виникає потреба в додатковій верифікації алгоритму, що пов'язано з верифікацією моделі. Інформаційні моделі часто будуються на проєктовані інформаційні системи. Тому верифікація інформаційних моделей забезпечує верифікацію інформаційних систем. Інформаційні системи управління часто будуються з взаємодіючих модулів [9]. Помилки виникають не тільки в модулях, а й в їх взаємодії. Дані помилки можуть бути критичними. Усунення подібних помилок вимагає верифікації моделі системи та самої моделі як реалізації.

Таким чином, верифікація інформаційних моделей частково вирішує завдання алгоритму реалізації або завдання верифікації інформаційної системи, реалізованої з використанням такої моделі. Отже, дослідження верифікації інформаційних моделей є досить актуальним завданням.

Об'єкт дослідження – процес перевірки програмного коду.

Предмет дослідження – засоби перевірки програмного коду.

Мета роботи – розробити модель для перевірки програмного коду.

Відповідно до мети були поставлені такі завдання:

- дослідити засоби перевірки програмного коду;
- дослідити програмні та апаратно-програмні комплекси для побудови засобу верифікації програмного коду;
- розробити засоби верифікації програмного коду з використанням LabView.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Моделювання грає дуже важливу роль в нашому житті та використовується практично у всіх областях людської діяльності: в науці, медицині, економіці, політиці, освіті та ін. Використання різних моделей дозволяє людям дізнатися про те, що представляв собою світ в минулому, зрозуміти причини багатьох явищ в сьогоденні та передбачити події, які можуть статися в майбутньому. А особливо велике значення та перспективи має створення комп'ютерних моделей і їх застосування.

Моделювання можна розглядати як заміщення оригінального об'єкта його умовним образом, описом або іншим об'єктом, що має назву моделі та забезпечує близьку до оригіналу поведінку в рамках деяких припущень і прийнятних похибок. Моделювання зазвичай виконується з метою пізнання властивостей оригіналу шляхом дослідження його моделі, а не самого об'єкта. Але, моделювання виправдано в тому випадку, коли воно простіше створення самого оригіналу або коли останній з якихось причин краще взагалі не створювати.

В наш час все більше використовується моделювання для дослідження об'єктів, процесів або явищ в різних областях. Результати цих досліджень необхідна:

- для визначення та поліпшення характеристик реальних об'єктів і процесів;
- для розуміння суті явищ та вироблення вміння пристосовуватися або керувати ними;
- для підвищення якості, конструювання нових об'єктів або модернізації старих.

1.1 Поняття моделі та їх класифікація

Модель – фізичний або абстрактний об'єкт, властивості якого в певному сенсі схожі з властивостями досліджуваного об'єкта. При цьому вимоги до моделі визначаються розв'язуванням завданням і наявними засобами.

Існує ряд загальних вимог до моделей (рис.1.1):

- адекватність або досить точне відображення властивостей об'єкта;
- повнота – надання отримувачу всієї необхідної інформації про об'єкт;
- гнучкість – можливість відтворення різних ситуацій у всьому діапазоні зміни умов і параметрів;
- трудомісткість розробки повинна бути прийнятною для наявного часу та програмних засобів (ПЗ).

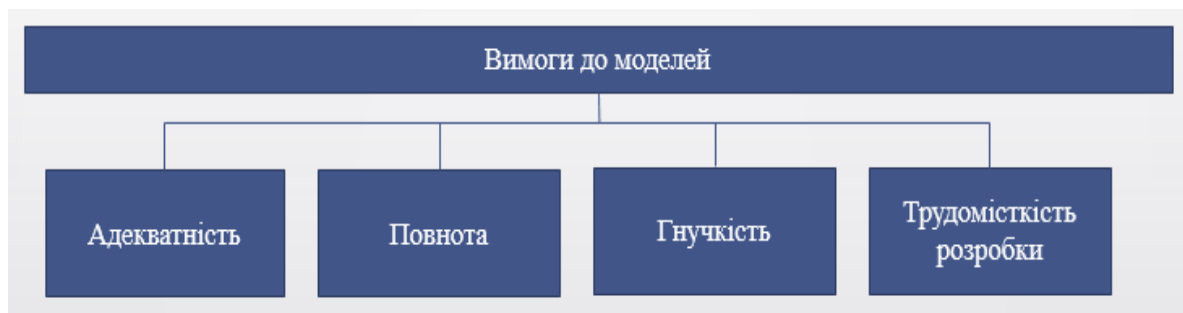


Рисунок 1.1 – Вимоги до моделей

Моделі прийнято класифікувати за різними ознаками

Основні ознаки класифікації моделей представлені на рис. 1.2.



Рисунок 1.2 – Ознаки класифікації

Класифікація за галуззю використання:

- навчальні (тренажери, посібники, навчальні програми)
- натурні моделі - це зменшені або збільшені копії проєктованого об'єкта. Їх використовують для дослідження об'єкта і прогнозування його майбутніх характеристик (моделі корабля, машини);
- науково-технічні моделі – створюють для дослідження процесів і явищ (синхрофазотрон, прилад, що імітує розряд блискавки);
- ігрові (ділові, військові, економічні, спортивні ігри);
- імітаційні моделі – не просто відображають реальність з тим або іншим ступенем точності, а імітують її. Експеримент або багаторазово повторюється, щоб вивчити і оцінити наслідки будь-яких дій на реальну обстановку, або проводиться одночасно з багатьма іншими схожими об'єктами, але поставленими в різні умови. Подібний метод вибору правильного рішення називається методом проб і помилок (нові ліки відчувають на мишах, щоб виявити побічні явища, уточнити дозування).

Класифікація з урахуванням фактору часу:

- статична модель – це одномоментний зріз інформації по об'єкту;
- динамічна модель дозволяє побачити зміни об'єкта в часі.

Класифікація моделей за галуззю знань - це класифікація за галуззю діяльності людини:

- математичні;
- біологічні;
- хімічні;
- соціальні;
- економічні;
- історичні.

Класифікація за способом подання:

- матеріальні моделі або предмети, фізичні. Вони відтворюють геометричні та фізичні властивості оригіналу і завжди мають реальне втілення.

– інформаційні моделі не мають матеріального втілення, тому що вони будуються тільки на інформації. В основі цього методу моделювання лежить інформаційний підхід до вивчення навколишньої дійсності.

Інформаційна модель - сукупність інформації, що характеризує властивості і стан об'єкта, процесу, явища, а також взаємозв'язок із зовнішнім світом.

Однак для підвищення точності створеної моделі необхідно використовувати перевірку моделей або верифікацію. Верифікація моделі – перевірка її істинності, адекватності.

1.2 Розгляд існуючих засобів перевірки програмного коду

В наш час існує безліч різних як комерційних, так і вільно розповсюджуваних систем верифікації для програмних і апаратних систем. Велике число верифікаторів розробляється через відсутність задовільного для промисловості універсального методу перевірки правильності розроблюваних систем. Різні верифікатори вирішують вузькоспеціалізовані проблеми, наприклад, перевірку властивостей, специфічних для мов реалізації систем. Як правило, верифікатори мають власні спеціальні мови специфікацій і методи перевірки моделей (рис.1.3).

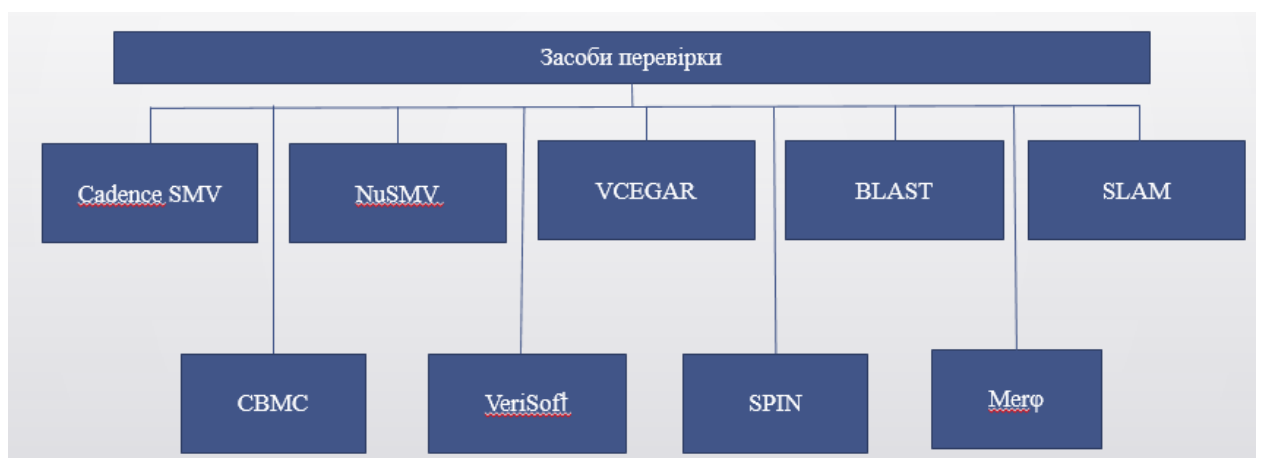


Рисунок 1.3 – Засоби перевірки

Нижче наведені короткі описи найбільш популярних систем верифікації:

- Cadence SMV (Symbolic Model Verifier) [1] – розробка Cadence Berkeley Laboratories для символічної верифікації моделей. Перевіряє вимоги коректності, виражені в CTL. SMV забезпечений двома мовами моделювання - Verilog і розширеним SMV. Для формального завдання систем переходів використовується структура Крипке - модель, яка має кінцеву безліч станів та переходів між ними, причому кожен стан позначений для безліччі дійсних в цьому стані предикатів. Для стиснення перевірки безлічі станів використовуються символічні способи подання моделей - діаграми (BDD, Binary Decision Diagrams). Алгоритми верифікації моделей за допомогою BDD можуть бути значно ефективніше явного перебору станів в тому випадку, коли BDD системи переходів моделі і проміжних результатів залишаються компактними [2];

- NuSMV (New Symbolic Model checker [4]). NuSMV - це оновлена версія SMV, був розроблений в якості системи з відкритою архітектурою для верифікації моделей, яка може бути використана для перевірки промислових розробок, як ядро складових верифікаційних інструментів і як основа для тестування інших технологій формальної верифікації. NuSMV здійснює верифікацію комбінуючи техніки використання BDD і перевірку моделей, засновану на SAT (SAT-based model checking). У порівнянні з SMV забезпечує такі можливості як взаємодія кінцевих автоматів, аналіз інваріантів, реалізація методів декомпозиції;

- VCEGAR (Verilog Counter Example Guided Abstraction Refinement [5]). Призначений для перевірки моделей апаратного забезпечення; використовує мову Verilog для опису поведінки моделі. Реалізує метод абстракції і уточнень предикатів, який розроблений для перевірки моделей ПЗ, і реалізований в SLAM. Розробляється за підтримки General Motors Collaborative Research Lab at CMU, Naval Research Laboratory, Semiconductor Research Corporation. Використовує Cadence SMV, NuSMV;

- BLAST (Berkeley Lazy Abstraction Software Verification Tool [1]).
Верификатор використовує метод абстракції предикатів. Будує абстракції швидко та з необхідною точністю;
- SLAM – розвиток BLAST, розробка Microsoft Research [3].
виробляє верифікацію коду, написаного на мові C. Вимоги коректності задаються на спеціалізованій мові SLIC. продукт орієнтований на верифікацію драйверів. Також необхідно відзначити розробку тієї ж групи розробників Terminator, яка призначена для перевірки зациклення програм;
- CBMC [5] – верифікатор, який забезпечує можливість обмеженою перевірки моделей (Bounded Model Checker) для мов ANSI-C і C ++. Він здійснює такі перевірки, як вихід за межі масивів (переповнення буфера), безпеку покажчиків, виключення і затвердження, сформульовані користувачем;
- VeriSoft [5] розроблений для перевірки властивостей паралельно працюють процесів. Його відмінною рисою є використання так званого пошуку «без станів» (stateless), запам'ятовуються тільки переходи поточної траси. При цьому використовуються техніка вибіркового пошуку (selective search), в основу якого покладено метод часткового порядку. Так як стану не запам'ятовуються, і як наслідок, цикли не виявляються, накладаються обмеження на глибину пошуку. Верифікатор успішно застосовувався в тестуванні цільового коду ряду промислових проєктів;
- SPIN [5] – інструмент формальної верифікації розподілених систем. Використовує прямий перебір, перевіряючи властивості моделі в оперативному режимі. Розроблявся в Bell Labs в групі в Computing Sciences Research Center, починаючи з 1980. З 1991 року перебуває у вільному поширенні. SPIN – це популярний інструмент з відкритим кодом, який використовується в багатьох організаціях світу, в 2001 році був нагороджений ACM престижною премією System Software Award. SPIN має три основні режими роботи: режим симулятора, що дозволяє здійснювати швидко прототипування з випадковим, керованим або інтерактивним

модельованням; режим верифікатори, здатний строго довести істинність вимог коректності, зазначених користувачем; режим апроксимації, який здатний підтверджувати правильність великих за обсягом протоколів з максимальним покриттям простору станів. SPIN підтримує високорівневу мову PROMELA (PROcess MEta LAnguage), за допомогою якої можна уточняти опис програмних систем. SPIN працює оперативно (onthe-fly), таким чином, уникаючи необхідності побудови глобального графа станів, або структури Крипке, як передумови для верифікації будь-яких властивостей системи. Для оптимізації верифікації використовується техніка часткового порядку (partial order reduction), також техніка злиття станів (statement merging) – яка виробляє злиття внутрішніх (непостережуваних) переходів процесу для редукції кількості досяжних станів моделі. Для ефективного зберігання станів використовується компресія станів (різновид «byte-sharing») та техніка апроксимації bit hashing;

– Мерф [4] є інструментом формальної верифікації розподілених систем, використовує прямий перебір станів моделі, які представлені значеннями всіх атрибутів. Відмінною особливістю є метод використання симетрій при перевірці еквівалентності станів. Метод ефективний при наявності повторюваних компонент (процесів) моделі, число яких заздалегідь невідомо. З огляду на обмежень на розмір оперативної пам'яті, застосовується техніка записи станів моделі на жорсткий диск.

Однак, не зважаючи на велику кількість переваг, розглянуті системи не сумісні з середовищами графічного програмування. Тому завдання розроблення засобів верифікації програмного коду є актуальним завданням.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Основні методи моделювання

Моделювання - це процес побудови моделі об'єкта і дослідження його властивостей шляхом дослідження моделі.

Таким чином, моделювання передбачає кілька основних етапів, на кожному з яких вирішуються різні завдання і використовуються відрізняються по суті методи і засоби.

На практиці застосовують різні методи моделювання. Залежно від способу реалізації, всі моделі можна розділити на класи [5]:

- фізичні;
- математичні;
- напівнатурні.

Математичне моделювання розглядають як засіб дослідження процесів або явищ за допомогою їх математичних моделей.

Під фізичним моделюванням розуміють дослідження об'єктів і явищ на фізичних моделях, коли досліджуваний процес відтворюють зі збереженням його фізичної природи або використовують інше фізичне явище, аналогічне досліджуваному. При цьому фізичні моделі припускають, як правило, реальне втілення тих фізичних властивостей оригіналу, які є істотними в конкретній ситуації. Наприклад, при проектуванні нового літака створюється його макет, що володіє тими ж аеродинамічними властивостями; при плануванні забудови архітектори виготовляють макет, що відображає просторове розташування її елементів. У зв'язку з цим фізичне моделювання називають також макетуванням.

Напівнатурне моделювання – дослідження керованих систем на змодельованих комплексах з включенням до складу моделі реальної апаратури [5]. Поряд з реальною апаратурою в замкнуту модель входять імітатори впливів та перешкод, математичні моделі зовнішнього середовища і процесів, для яких невідомо досить точний математичний опис. Включення

реальної апаратури або реальних систем в контур моделювання складних процесів дозволяє зменшити апріорну невизначеність та досліджувати процеси, для яких немає точного математичного опису. За допомогою напівнатурного моделювання дослідження виконуються з урахуванням малих постійних часу і нелінійностей, властивих реальній апаратурі [6]. При дослідженні моделей з включенням реальної апаратури використовується поняття динамічного моделювання, при дослідженні складних систем і явищ - еволюційного, імітаційного і кібернетичного моделювання.

Ефективність від моделювання може бути отримана при дотриманні двох умов:

- модель повинна забезпечувати коректне (адекватне) відображення властивостей оригіналу, істотних з точки зору досліджуваної операції;
- модель повинна дозволити усунути перераховані вище проблеми, властиві проведенню досліджень на реальних об'єктах.

Однак, при моделюванні дослідник повинен бути впевнений в правильності моделі, відповідно моделі реального прототипу. Точність моделювання залежить від того, наскільки добре математична модель відображає властивості об'єкта [7]. Досліднику важливо знати, з якою похибкою він отримує результат, тому що в разі великої погрішності розрахунок втрачає сенс.

На точність моделювання впливають такі особливості [7]:

- спрощення моделі;
- помилки при побудові моделі;
- використання елементів з низькою точністю, з лінійною апроксимацією;
- наявність в моделі вироджених кінцевих елементів;
- некоректні зв'язку;
- некоректні параметри моделей;
- некоректні властивості елементів;
- некоректні початкові і граничні умови;

- похибки методу розрахунків.

Отже для підвищення точності створюється модель, результати використання якої збігаються з реальними об'єктами із заданою похибкою. А для цього в використовується верифікація моделей. Верифікація моделі – перевірка її істинності, адекватності. Основні методи верифікації розглянемо нижче.

2.2 Дослідження методів верифікації програмних моделей

Методи верифікації моделей призначені для підтвердження фактів відповідності властивостей моделі заявленим вимогам. Такі методи різноманітні та різнорідні, як за своїм призначенням, так і за способами досягнення кінцевого результату та включають як емпіричні, так і формальні докази підтвердження наявності у моделей зумовлених властивостей [1]. Поняття про верифікації моделей та ПЗ розуміється в досить широкому сенсі та мати різні математичні моделі, що служать основою верифікації істотно розрізняються. У процесі верифікації може досягатися ще одна мета, яка полягає в пошуку некоректно реалізованих властивостей, невиконаних вимог і супутньому виявленні помилок [2]. У зв'язку з цим, сучасні методи верифікації можуть включати в себе методи тестування. Методи верифікації в цілому можна розділити на структурні і функціональні, які також мають у своїй основі формальну математичну модель, або залежні від кваліфікації осіб, котрі приймають рішення про коректність роботи моделі.

Можна виділити основні методи верифікації, які виділені на рис.2.1:

- експертиза;
- статичний аналіз;
- динамічні методи;
- формальні методи ;
- синтетичні методи.

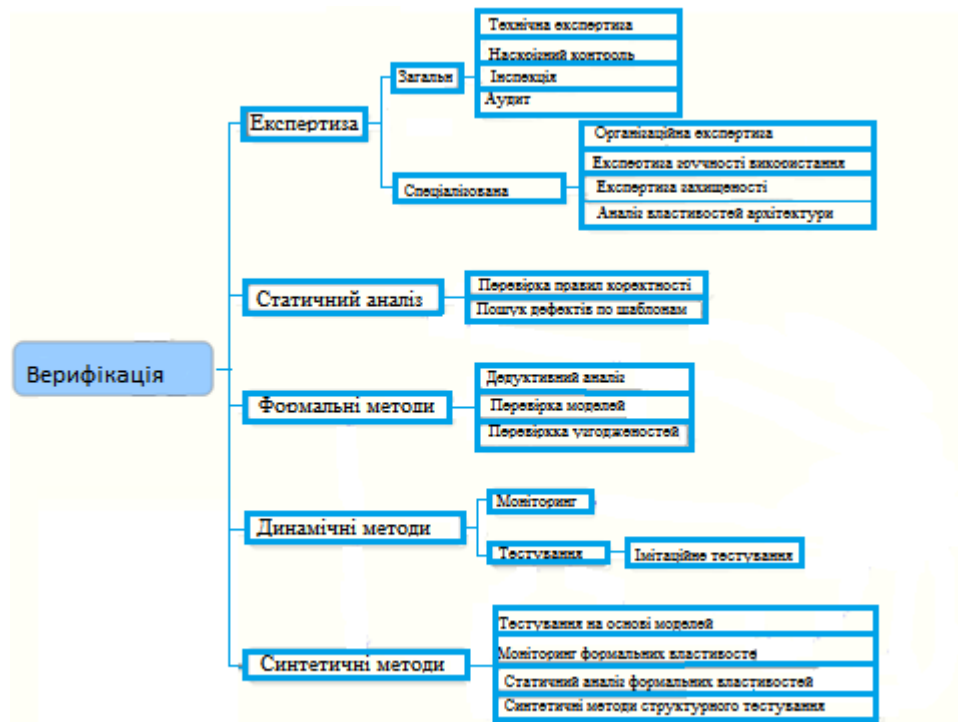


Рисунок 2.1 – Методи верифікації програмних проєктів

Необхідно ввести ряд понять, лежать в основі цієї класифікації методів [2]:

- види методу - опис основних видів розглянутого методу;
- рівень автоматизації - рівень, який визначає в якій мірі можна автоматизувати алгоритми того чи іншого методу верифікації;
- рівень функціональної придатності - визначає наскільки широке коло завдань покриває метод верифікації ПЗ;
- точність - характеристика якості одержуваних вимірів, показує, на скільки похибка методу (оцінка відхилення виміряного значення величини від її справжнього значення) прагне до нуля;
- типи виявлених помилок;
- ефективність - визначає продуктивність методу;
- область застосування - визначає на якому етапі розробки застосуємо той чи інший метод верифікації ПЗ, а також визначає до яких артефактів ПО можна застосувати метод верифікації.

- час виконання - визначає час, який потрібен для верифікації програми;
- спосіб досягнення результату - визначає методи, і алгоритми, за допомогою яких інструменти аналізу здійснюють верифікацію.

Основними видами і характеристиками є [2], стійкість системи в разі недетермінованої поведінки оточення, ефективність використання ресурсів часу та пам'яті, саме за цими параметрами, як правило, проводиться верифікація.

2.2.1 Експертиза

Методи першої групи представляють собою тестування та експертний аналіз властивостей моделей та відповідність деякому зразку.

Відмінною особливістю експертизи при верифікації моделей є наявність ряду міжнародних стандартів (IEEE1074 [8], ISO12207 [9], ISO15504 [10], ISO15288 [11]). В цілому, при такому підході до верифікації, спостерігається орієнтованість на експертні оцінки, які можуть змінюватись в залежності від парадигм програмування. Таким чином, дані методи не можна віднести до універсальних та строго формалізованих. Також експертиза може бути застосована безпосередньо до ПЗ, а не до формальних моделей та результатів роботи. Серед видів експертиз виділяють: організаційні експертизи (management review), технічні експертизи (technical review), наскрізний контроль (walkthrough), інспекції (inspection) та аудити (audit). Різні види експертиз застосовуються на різних етапах проєктування. Своєчасне виявлення дефектів дозволяє знизити до мінімуму ризик некоректної роботи моделей в подальшому.

2.2.2 Статичний аналіз

Методи статичного аналізу проводять порівняння деяких моделей із заздалегідь визначеним шаблоном.

Статичний аналіз коректної побудови моделі або пошук дефектів, які часто зустрічаються за деякими шаблонами [7]. Даний метод добре автоматизований та практично не потребує процедури прийняття рішень людиною. Однак, помилки, що виявляються статичним аналізом, досить вузької направленості. Одним із застосувань шаблонів типових помилок є їх включення в семантичні правила компіляторів мов програмування.

2.2.3 Динамічні методи

Динамічні методи аналізують вже готовий, працюючий продукт і виявляють проблеми з працездатністю.

Динамічні методи верифікації застосовуються для аналізу і оцінки вже результатів роботи програми, або деяких її прототипів і моделей [6].

Прикладами таких методів є тестування ПЗ або імітаційне тестування, моніторинг та профілювання.

Динамічні методи застосовуються на етапі експлуатації ПЗ тому необхідно мати вже працюючу систему або хоча б деякі її працюють компоненти. Такі методи не можна використовувати на стадіях розробки та проєктування ПЗ, зате з їх допомогою можна контролювати характеристики роботи системи в її реальному оточенні, які іноді неможливо точно визначити за допомогою інших підходів [5]-[6]. Динамічні методи дозволяють виявляти в ПЗ тільки помилки, які проявляються при його функціонуванні. Ця обставина обмежує область їх застосування, наприклад, з їх допомогою неможливо знайти помилки ПЗ, пов'язані зі зручністю супроводу програми.

2.2.4 Формальні методи

Формальна верифікація ґрунтується на математичному (логічному) моделюванні [5]-[6]. Основна ідея: створюється модель - ідеалізований опис досліджуваного об'єкта або явища; модель досліджується із застосуванням математичних методів; результати дослідження переносяться на реальний об'єкт або явище. Безумовно, застосовність такого підходу визначається використовуваними моделями - потрібно чітко розуміти умови їх адекватності.

Методи формальної верифікації розрізняються типами моделей програм (кінцеві автомати, мережі Петрі, розмічені системи переходів), типами моделей вимог (програмні контракти, продукційні правила, темпоральні затвердження), відносинами відповідності (еквівалентність, симуляція, включення трас) і техніками перевірки відповідності (дослідження простору станів, символічний аналіз, дедуктивний висновок) [4]-[5].

2.2.5 Синтетичні методи

Синтетичні методи верифікації використовують техніки різних видів за наведеною вище класифікацією, а також комбінують ідеї різних підходів для отримання більшої ефективності верифікації в термінах витрат на її проведення і достовірності отриманих результатів [4]-[6].

На даний момент такі методи відносяться до однієї з наступних груп:

- статичний аналіз передбачає побудову деяких моделей перевіряє системи, найчастіше, у вигляді розмічених графів потоків управління і даних, і аналіз властивостей цих моделей, наприклад, пошук помилок певного роду за відповідними їм шаблонами в потоках даних. Зараз все частіше використовуються специфічні види статичного аналізу, в рамках яких

знаходять застосування формальні моделі та спеціалізовані інструменти вирішення обмежень [1]-[2];

- розширений статичний аналіз (extended static checking) [3]-[4] перевіряє відповідність моделі вимогам, зазвичай записується теж в коді у вигляді коментарів до його окремих елементів (процедур, типам даних і методам класів). При цьому на основі результатів аналізу коду автоматично будуються формальні моделі його поведінки, виконання вимог для яких перевіряється найчастіше за допомогою дедуктивного аналізу і спеціалізованих рішень (solvers);

- статичний аналіз на базі автоматичної абстракції [5]. В рамках такого підходу на основі результатів статичного аналізу коду автоматично будуються більш абстрактні, а тому більш прості моделі роботи перевіряється ПЗ, які потім піддаються перевірці на виконання певних властивостей за допомогою інструментів перевірки моделей. Зазвичай перевіряються властивості фіксовані для даного інструменту або формулюються в файлі конфігурації. При порушенні вимоги в моделі інструменти цього типу намагаються побудувати відповідний сценарій роботи коду. Якщо це не виходить через спрощення, зроблених при побудові моделі, визначаються елементи коду, що перешкоджають виконанню такого сценарію, і в модель вносяться уточнення, більш акуратно описують роботу саме цих елементів, після уточнена модель знову перевіряється на виконання заданої властивості. В результаті інструмент або підтверджує виконання вимог, або знаходить контрприклад, або завершує роботу після закінчення деякого часу або через вичерпання ресурсів, не приходячи до певних висновків;

- синтетичне структурне тестування [6] при якому після першого випадково обраного тесту інші тести генеруються автоматично так, щоб забезпечити покриття ще не покритих раніше елементів коду. Для вибору відповідних тестових даних використовуються вирішувачі, що враховують символічну інформацію про раніше виконаних тестах (обмеження на дані, що

відокремлюють минулі тести від ще не покритого коду), а для побудови потрібних послідовностей впливів - випадкова генерація, яку направляють як цієї ж символічної інформацією, так і деякими евристичними абстракціями, що зменшують простір станів перевіряється системи;

- тестування на основі моделей (model based testing) [7] поєднує розробку формальних моделей вимог до перевіряється ПО і побудова тестів на базі цих моделей. Структура моделі при цьому служить основою для критерію повноти тестування, а обмеження моделі на коректні результати роботи ПО використовуються в якості тестових оракулів, які оцінюють правильність поведінки ПЗ в ході тестування. В рамках останніх двох підходів (або окремо від них) використовуються спеціальні техніки побудови тестів, самі по собі поєднують різні методи верифікації;

- побудова тестів за допомогою дозволу обмежень [7]. Часто при розробці тестів на основі критеріїв повноти тестування формулюються так звані цілі тестування (test objectives), що представляють собою специфічні ситуації, в яких необхідно перевірити поведінку тестованої системи для досягнення необхідної впевненості в її коректної роботи. Мета тестування формулюється як набір обмежень на прохідні під час тесту стану системи і дані виконуваних дій. Для побудови тесту, що досягає таку мету, можна використовувати спеціалізовані вирішувачі (solvers). Такий вирішувач або автоматично знаходить необхідні дані і послідовність викликів операцій як рішення заданої системи обмежень, або показує, що ця система нерозв'язна, тобто задана мета тестування недосяжна і будувати націлені на неї тести не має сенсу;

- побудова тестів як контрприкладів за допомогою інструментів перевірки моделей [7]. Інший спосіб побудови тестів - сформулювати заперечення обмежень, які задають мета тестування, як властивість, яке можна перевірити чи спростувати за допомогою інструменту перевірки моделей. Якщо це властивість підтверджується, значить, мета тестування

недосяжна, якщо ж воно спростовується, то інструмент будує контрприклад, що є в даному випадку необхідним тестом;

- моніторинг формальних властивостей (runtime verification, passive testing) [6] теж використовує формальні моделі вимог для оцінки правильності поведінки перевіряється системи, але тільки в ході її нормальної роботи, без використання спеціально побудованих тестів.

Як видно, всі синтетичні методи так чи інакше намагаються поєднати переваги різних підходів до верифікації, копіюючи їх недоліки.

Однак для проведення верифікації складних моделей та програм виникає необхідність

- полегшувати зміни, таким чином, щоб вони не порушували існуючу функціональність.
- спростувати інтеграцію, гарантія, що основні блоки функціонують, та можливість відокремити інтеграційні проблеми від внутрішніх проблем модуля;
- документувати, можливість використання результатів верифікації для опису функціональності тестованого коду.

Для задоволення даних потреб використовують модульне тестування.

2.3 Верифікація в LabView

2.3.1 Обґрунтування вибору середовища розробки

Для створення забезпечення для верифікації програмних систем доцільно використовувати програмно-апаратний комплекс LabVIEW [11]. Побудова застосувань для верифікації в LabVIEW вимагають технології програмування відмінні від типового програмування LabVIEW. Дані застосування мають два рівня налагодження – робота застосування та час роботи.

Обґрунтування вибору середовища розробки для створення забезпечення для верифікації програмних систем наведено у табл. 2.1.

Таблиця 2.1 – Обґрунтування вибору середовища розробки для створення забезпечення для верифікації програмних систем

Критерій порівняння мов середовищ розробки	Середовище розробки		
	LabVIEW	myOpenLab	PyLab_Works
Парадигма програмування	Графічне програмування	Графічне програмування	Текстове програмування
Зручність використання	+	+-	+-
Підтримка розробників	+	+-	+
Висока функціональність	+	+-	+-
Застосування в індустрії	+	+-	+-
Зручний графічний інтерфейс користувача	+	+-	+-

Отже, для створення забезпечення для верифікації програмних систем було обрано LabVIEW, що являє собою середовище розробки та платформу для виконання програм на основі графічної мови програмування G.

Проведення верифікації відбувається шляхом виконання одного з наступних критеріїв [12]:

- верифікація логіки;
- верифікація часу виконання;
- перевірка завантаження пам'яті;
- відстеження виконання.

2.3.2 Верифікація правильності роботи програми

Для перевірки застосування, необхідно переконатися в правильності його роботи, для цього можна використовувати інструменти налагодження (debugging tools) LabVIEW, такі як виконання в режимі налагодження

(execution highlighting) і послідовне виконання (single-stepping), доки головний комп'ютер підключений до віддаленого пристрою в реальному часі, для покрокової перевірки коду LabVIEW [13].

Єдиний інструмент який не підтримується LabVIEW Real-Time Module - це Call Chain ring, який з'являється в наборі інструментів блок-діаграми підпрограми при покроковому виконанні.

Не слід використовувати інструменти налагодження LabVIEW для налагодження часу виконання, тому що всі інструменти налагодження впливають на розрахунок часу роботи програми [13].

Коли необхідний віртуальний прилад (ВП) не може бути виконаний, в наборі інструментів на кнопці Run з'явиться «зламана» стрілка. Щоб побачити список помилок та попереджень, необхідно натиснути на зламану стрілку. При натисненні на повідомлення про помилку, щоб побачити «зламаний» об'єкт на блок-діаграмі.

Для налагодження та управління помилками в ВП, використовується error handling (обробку помилок). Обробка помилок повертає код помилок, при наявності помилки в ВП. Коду помилок вказують проблеми, що зустрічаються в ВП. Після формування віддаленого пристрою реального часу, LabVIEW автоматично копіює файл коду помилок, використаний error handler ВП в пристрій [12].

Можна використовувати зібраний користувачем код помилок (custom errors) для ВП запущеного на віддаленому пристрої. Створюється файл помилок, використовуючи Error Code File Editor, для цього необхідно обрати Tools »Advanced» Edit Error Codes. При використанні custom errors з LabVIEW, необхідно прописати розширення .err, створеному файлу, і помістити файл помилок в директорію c: \ ni-rt \ system \ user.lib \ errors або в директорію c: \ ni-rt \ system \ errors віддаленого пристрою реального часу (ПРЧ). При використанні FTP клієнт в MAX або будь-який інший FTP клієнт для передачі файлу помилок інших мережних пристроїв. В LabVIEW Help

можна переглянути інформацію про custom error codes (зібраному користувачем кодів помилок).

Execution highlighting (виконання в режимі налагодження) анімує діаграму та простежує потік даних, що дозволяє бачити проміжні значення. Щоб запустити execution highlighting, в наборі інструментів слід натиснути на лампочку.

Перевірка допомагає бачити значення, що проходять через сегмент сполучної лінії. При натисненні на лінії з інструментом Probe або правої кнопки мишки, щоб встановити probe.

Вибір точок переривання встановлює паузу на обраній позиції. Для цього необхідно натиснути на лінію або об'єкт з інструментом Breakpoint, або правою кнопкою мишки, щоб встановити breakpoints (точки переривання).

Conditional probe – це перевірка, в якій можуть бути встановлені умови переривання перебігу прогону [12]. Наприклад, можна встановити переривання тільки в разі, коли значення в сегменті нижче нуля.

Single stepping дозволяє проходити діаграму від вузла до вузла. Можна отримати доступ до single stepping, використовуючи step functions, що знаходиться в наборі інструментів.

Step Into / Step Over, використовується для початку послідовного виконання:

- Step Into- "Кроки" в вузли. Якщо підпрограма, викликає діаграму підпрограми і запускається покрокове виконання даної підпрограми;
- Step Over- "Кроки" через вузли, візуально покрокове виконання ігнорує вузли;
- Step Out- "Кроки" з вузлів, якщо діаграма завершила виконання; Step Out завершає послідовне виконання.

2.3.3 Перевірка вірності часу виконання

Час виконання критичний в спеціалізованих застосуваннях. Для перевірки вірності часу виконання використовуються наступні методи [14]:

- виконання профілю та інструменти пам'яті;
- часові функції;
- драйвера функціонального призначення;
- осцилограф.

Для перевірки використовують Profile Window (рис. 2.2).

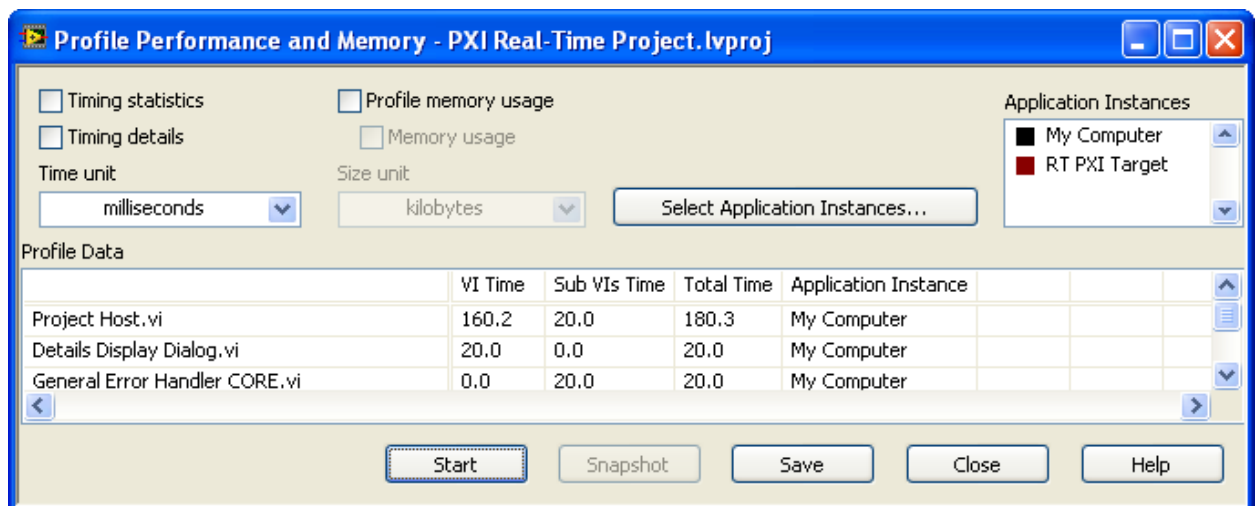


Рисунок 2.2 – Profile Window

Вікно Profile Performance and Memory - це інструмент для статистичного аналізу розподілу застосуванням часу та пам'яті. Вікно Profile Performance and Memory відображає інформацію, яка визначає специфічні ВП або їх частини, які необхідно оптимізувати. Вікно Profile Performance and Memory відображає інформацію для всіх ВП в діалоговому табличному форматі. З вікна Profile Performance and Memory, можна вибрати тип інформації, що дозволить сортувати її за категоріями. Також можна перевірити виконання підпрограм в межах ВП.

Визначення ефективності системи на операціях включає в себе вимір часу на початку операції, виконання операції, і вимір часу в кінці операції.

Зазвичай дана операція виконується кілька разів, що дозволяє отримати середнє значення.

Точність даного методу залежить від точності захоплення часу. В цілому, час вимірюється в мілісекундах з використанням годин в процесорі, лічильника в DAQ-платі, або в мікросекундах, з використанням годин в процесорі віддаленого пристрою. Швидкість годин визначає точність захоплення часу.

Використання функції Tick Count (мс), виконує вимір часу в N ітераціях певної операції. Даний метод має ту ж точністю, що і годинник, який використовуються для функції Tick Count. При використанні даного методу, необхідно враховувати можливість одночасного натискання клавіш.

NI Timestamp VIs, розташований в Real-Time Utilities, читає регістр Time Stamp Counter з процесора віддаленого ПРЧ для кожної ітерації циклу. NI Timestamp Code Timer VI розташований в NI Example Finder для прикладу використання NI Timestamp VIs [13].

Використання програмного забезпечення драйвера

Іноді можна використовувати програмне забезпечення драйвера, що б переконатися, що додаток відповідає бажаним показникам циклу. Наприклад, можна виконати перевірку на помилки після DAQmx Wait for Next Sample Clock VI, що б перевірити чи зберігає цикл роботу в реальному часі.

Використання осцилографа

Функція Tick Count (ms) та функція NI Timestamp VIs дозволяють вивчити відносну синхронізацію секцій LabVIEW ВП та виміряти нестійку синхронізацію в ВП [15]. Іноді потрібно виміряти нестійку синхронізацію всієї системи на апаратному рівні, особливо, коли використовується апаратна синхронізація. Нестійка синхронізація прихована на системному рівні.

Використовуючи зовнішні інструменти, такі як осцилограф, можна вивчити взаємозв'язок між вхідним і вихідним сигналами та виміряти показники циклу, рівні нестійкою синхронізації

2.3.4 Перевірка завантаженості пам'яті

Завантаженість пам'яті можна перевірити у вікні Profile Performance and Memory. Крім того, можна перевірити та визначити завантаженість пам'яті інструментами ПРЧ Engine Info, Show Buffer Allocations, і Real-Time System Manager.

Engine Info відображає адресу поточного цільового пристрою ПРЧ, повну пам'ять цільового пристрою, і доступну вільну пам'ять [16]. Можна періодично використовувати цей інструмент для перегляду статусу цільового пристрою.

Для відображення інструментів розподілу буфера слід обрати Tools »Profile» Show Buffer Allocations, щоб відобразити вікно.

Дане вікно можна використовувати для визначення, де LabVIEW розподіляє пам'ять на блок-діаграмі. Також можна помічати маркерами ті типи даних, які необхідно відслідкувати бачити в буфері.

Проте, відомості вікна Show Buffer Allocations можуть бути не точними [15]. Наприклад, буфер може показувати розподіл в циклі, а насправді розподіл йде на першій ітерації або розподіл на кожній ітерації окремо.

System Manager відображає деталі про ВП, запущений віддалено та забезпечує динамічний показ виконання віддаленого пристрою.

Модуль встановлюється за умовчанням при установці LabVIEW. Це доступно тільки з ETS.

Інтерфейс (RTSM) працює на головному ПК та повідомляє про завантаженість системи. Дані, які відображаються в інтерфейсі беруться з ВП, що працює на тлі віддаленого пристрою. ВП запускається тільки тоді, коли запускається інтерфейс, який надає звіт системи на оновленій частоті.

Якщо віддалений пристрій запитує недоступні CPU ресурси при запуску інтерфейсу, тоді віддалений пристрій не надає ніякої інформації.

З системами FieldPoint і Compact FieldPoint, діаграма використання CPU завжди показує 100%. FP Manager використовує доступну потужність процесора для фонового запуску завдання.

2.3.5 Відстеження виконання

LabVIEW Execution Trace Toolkit необхідний для установки та виконання події в реальному часі, трасуючі інструменти дозволяють захоплювати і відображати час виконання та дані про події.

З мінімальними змінами в коді, ці інструменти графічно відображають виконання багатопоточного коду доки відбувається виділення ланцюга замін, mutexes та розподіл пам'яті [15]. Використовуючи дану інформацію, можна оптимізувати код для більш швидких циклів контролю і більш цільового виконання.

Спеціальні контрольно-вимірювальні інструменти розміщені в застосуванні, запущеному на віддаленому пристрої. Дані ВП управляються початком і кінцем трасування [16]. Коли трасування завершено, інформація повертається на призначений для користувача інтерфейс Trace tool, запущений на головному ПК.

Оснащення ВП включає Start Trace, Stop Trace (початок кінець трасування) і Send (відправка), Stop Trace і Save (зупинка і збереження трасування), Load Trace (завантаження трасування) і Send (відправка) і Log User Events (реєстрація логів).

Обмеження реєстрації трасування пов'язано тільки з обмеженістю пам'яті буфера. Проте можна збільшити розмір буфера, використовуючи Start Trace ВП. Якщо цей буфер заповнений, тоді він діє, як FIFO, містить тільки нові дані і перезапис старших.

Існує два шляхи перегляду виконання системи. Перегляд модулів показує, коли кожен з модулів активний. Також доступні прапори, які можна

конфігурувати. Перегляд ВП записує кожен ВП в пам'ять і точно показує момент його виконання.

Найчастіше необхідно відстежити не тільки запуск і зупинку, а також інші події. Ці події можуть бути включені в менеджер пам'яті (загальний ресурс), сон, ітерації часового циклу, і т.д.

2.3 Модульне тестування

Для складних моделей та програм ще існує такий підхід до верифікації як модульне тестування або unit-тестування.

Unit-тестування або блочне тестування – процес, який дозволяє проводити перевірку окремих модулів вихідного коду програми або моделі, набори з одного або більше програмних модулів разом з відповідними даними, процедурами використання та обробки.

Основна ідея полягає в тому, щоб створювати тести для кожної нетривіальною функції або методу. Даний підхід дозволяє досить швидко перевірити, вплив змін на регресію, тобто до появи помилок в уже протестованих відрізках, також полегшує виявлення і усунення таких помилок [6].

Даний підхід має багато переваг [12]:

- дозволяє знаходити проблеми на ранній стадії та зменшує ймовірність появи помилок;
- після написання модульних тестів регресійне тестування стає дуже простим. Отже, якщо вносяться зміни до якогось коду, запуск модульних тестів перевіряє, чи все працює належним чином та виділяє помилки;
- модульні тести можуть також виконувати функцію документації коду;
- модульне тестування дозволяє продемонструвати охоплення вимог.

Для проведення тестування було обрано середовище графічного програмування LabVIEW в якому є безліч інструментів, а також додаткові інструменти програмного забезпечення для вдосконаленого тестування та аналізу коду. Досить поширеним є Unit Test Framework, інструмент для LabVIEW, який може бути використаний для автоматизації модульного тестування, верифікації на основі вимог та регресійного тестування VI.

3 ОПИС ПРОГРАМИ

3.1 Розробка програмної моделі

Модель проєктується на основі схеми вбудованої системи (рис. 3.1). Розроблення даної схеми було виконано з використанням додаткового програмно-апаратного забезпечення. Виникла необхідність налаштування устаткування таким чином, щоб дані подавалися на послідовний серійний порт [14]. При проєктуванні були використані оператори, які можуть дозволити користувачеві обирати конкретний серійний для використання та встановлювати швидкість обміну даними.

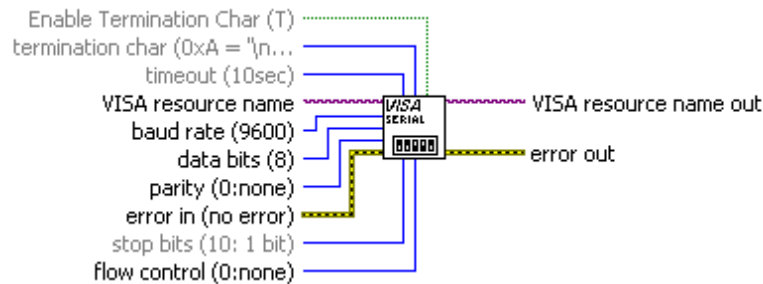


Рисунок 3.1 – VISA Configure Serial Port

Enable Termination Char готує послідовний пристрій до розпізнавання символу. Якщо TRUE (за замовчуванням), атрибут VI_ATTR_ASRL_END_IN встановлений для розпізнавання символу закінчення. Якщо FALSE, для атрибута VI_ATTR_ASRL_END_IN встановлено значення 0 (немає), і послідовний пристрій не розпізнає символ припинення.

Term char вимагає припинення операції зчитування. Операція зчитування закінчується, коли символ закінчення зчитується із послідовного пристрою.

Timeout визначає час у мілісекундах для операцій запису та читання. За замовчуванням 10000.

VISA resource name визначає ресурс, який потрібно відкрити. Елемент керування іменем ресурсу VISA також визначає сеанс і клас.

Baud rate- швидкість передачі. За замовчуванням - 9600.

Data bits – це кількість бітів у вхідних даних. Значення бітів даних становить від п'яти до восьми. Значення за замовчуванням - 8.

Parity визначає парність, яка використовується для кожного кадру, що передається або приймається.

Error in описує умови помилок, які виникають до запуску цього вузла. Цей вхід забезпечує стандартну помилку у функціональності.

Stop bits визначає кількість стоп-бітів, що використовуються для позначення кінця кадру.

Flow control встановлює тип управління, що використовується механізмом передачі.

Ім'я ресурсу VISA - це копія імені ресурсу VISA, яке повертають функції VISA.

error out містить інформацію про помилку. Цей результат забезпечує стандартну функціональність щодо виведення помилок .

Для використання циклу було використано наступну умову: за наявності інформації на серійному порту відбувається її зчитування (рис. 3.2). Для виконання даної умови існує оператор перевірки кількості байт на серійному порту та відбувається порівняння з 0. У випадку, якщо кількість байт більше 0 то починає виконуватися цикл.



Рисунок 3.2 – Блок перевірки наявності інформації на серійному порту

Також виникає необхідність записати дані до буферу, для цього необхідно зробити конвертацію з рядка в число (рис. 3.3), та з'являється можливість побудувати графік (рис. 3.4).



Рисунок 3.3 – Блок конвертації рядка в число

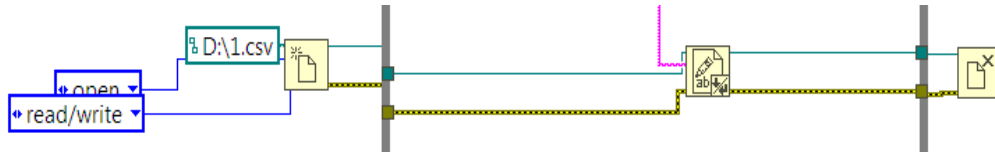


Рисунок 3.4 – Процес запису даних до файлу

Таким чином, необхідна нам схема повинна виглядати наступним чином (рис. 3.5).

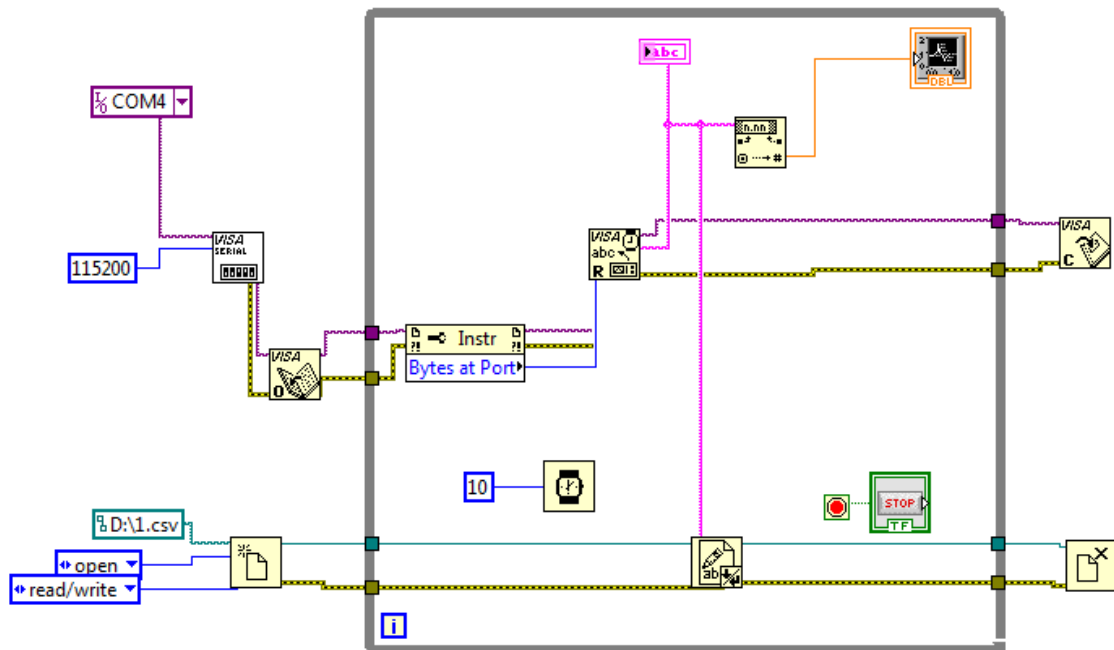


Рисунок 3.5 – Схема проєктованої моделі

Але при проєктуванні моделі виникає необхідність забезпечити належне функціонування блоків програми [14]. Для уникнення помилок на стадії використання готової моделі необхідно провести верифікацію для виявлення помилок на більш ранніх стадіях. Для досягнення даної мети будемо використовувати модульне тестування.

3.2 Модульне тестування

Модульне тестування – це процес, який дозволяє проводити верифікацію найменших частин програми, що дозволить переконатися, в її працездатності [15].

Для успішного впровадження модульного тестування важливо створити код, який можна розділити на тестові одиниці. Чим більше модулів в моделі, тим легше буде її перевірити одиничними блоками [16].

У випадку, коли необхідно до простої схеми додати операції з даними, наприклад (рис. 3.6), то багаторазові обчислення виконуються в одному і тому ж subVI, тому ускладнюється пошук помилок.

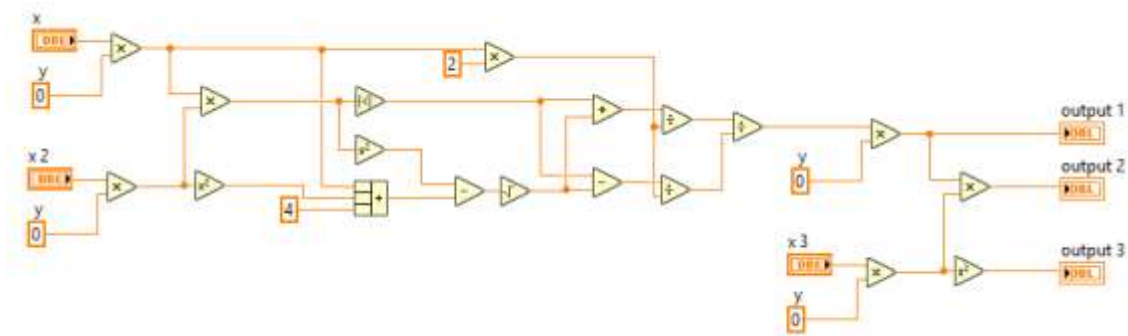


Рисунок 3.6 – Блок операції з даними

Але даний фрагмент можна спростити утворивши два subVI-блоки, які набагато легше перевірити (рис. 3.7).

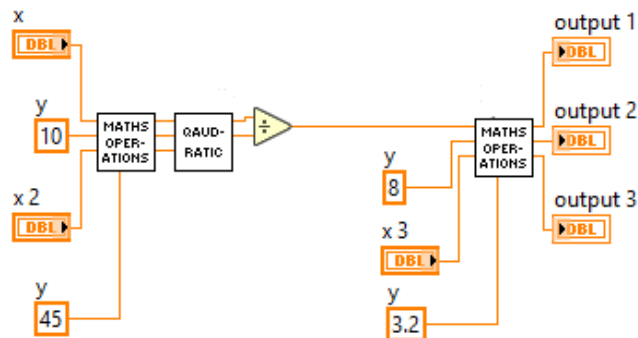


Рисунок 3.7 – Спрощений блок операцій з даними

Даний підхід дозволяє знаходити проблеми на ранній стадії та зменшує ймовірність помилок та якщо вносяться зміни до коду, запуск модульних тестів перевіряє, чи все працює належним чином, та підсвічує місця, де були введені помилки. Для побудови тестів в LabVIEW було використано Frame Unit Framework та Caraya [17].

Дані інструменти забезпечують спосіб написання, виконання та автоматизації модульних тестів (рис. 3.8).

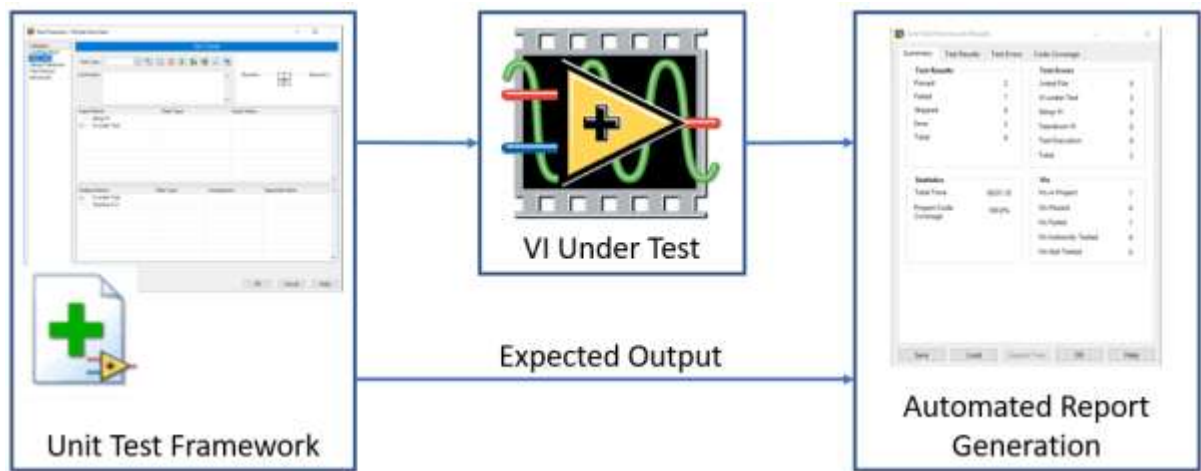


Рисунок 3.8 – Схема тестування

Незважаючи на існування різних способів написання модульних тестів, наприклад, за допомогою модуля Unit Test Framework в Control Software Solutions, однак ефективніше використовувати тести визначені користувачем. Використання тестів визначених користувачем дозволяє писати тестові блоки з використанням VI, а не через властивості модульного тесту [18]. Це дозволяє інтегрувати будь-які умови налаштування та розірвання, які необхідні тестованій VI (наприклад, налаштування FGV, відкриття посилання на файл тощо). Також виникає можливість проволити складні тести включаючи паралельне тестування з декількома умовами тестування.

Під час першого запуску тесту він повинен працювати з помилками. Потім необхідно заповнити тест всередині VI, щоб його пройти. Даний процес зображений на рис. 3.9.

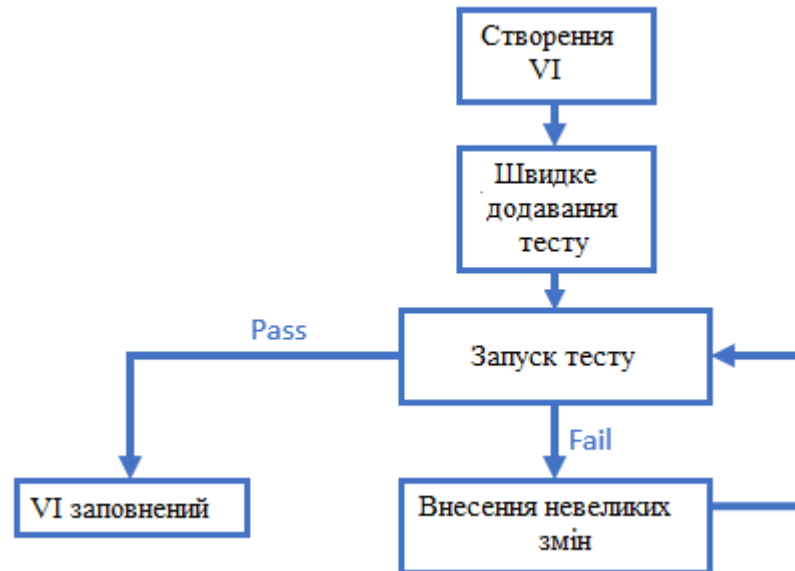


Рисунок 3.9 – Процес тестування

Оскільки є можливість повторно запустити всі модульні тести в проєкті, будь-які зміни в коді не повинні містити помилок.

Очевидно, є деякі накладні витрати на написання, запуск та підтримку всіх цих модульних тестів, і спочатку це може здатися трохи більш трудомістким, однак ми виявили, що це економить багато часу, який в іншому випадку може бути витрачений на переслідування помилок.

Але створення користувацьких тестів може зайняти багато часу, особливо в випадках, коли з'являються різні рівні складності (рис. 3.10).

У випадку показаному на рис. 3.10 виникає проблема, яка полягає в тому, що користувач отримає два варіанта результату "пропуск" або "невдачу". Таким чином, у випадку провалу тесту не буде отримано додаткової інформації (наприклад, який конкретний тестовий випадок не вдався). Дану проблему можна вирішити методами налагодження, проте такий підхід обмежує автоматизацію та об'ємний за часом.

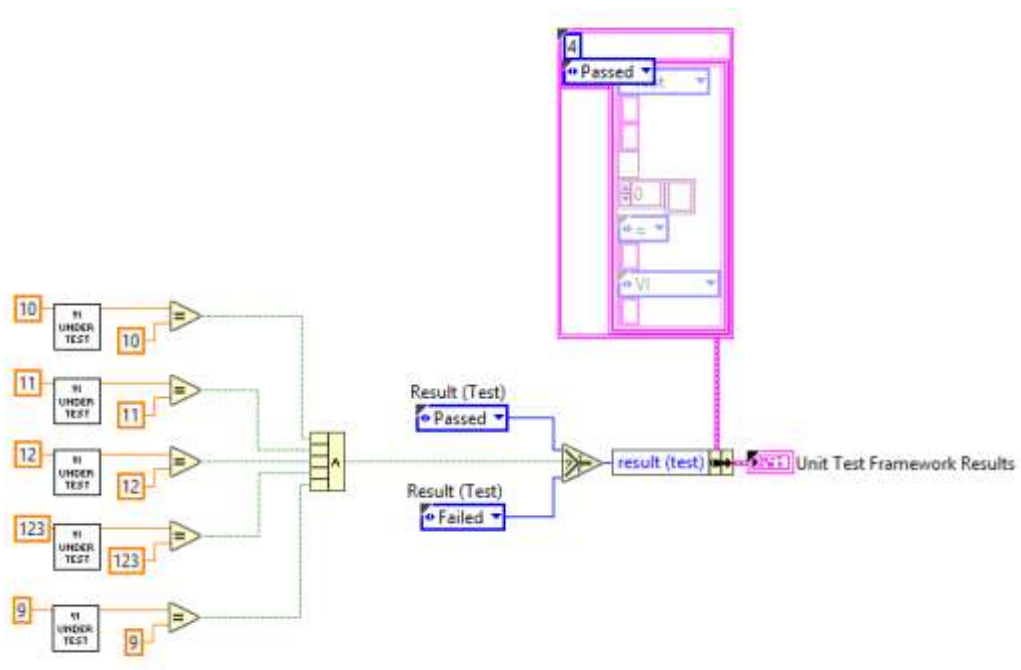


Рисунок 3.10 – Приклад тесту

У Control Software Solutions було зроблено засіб, який спрощує даний процес та показаний на рис. 3.11.

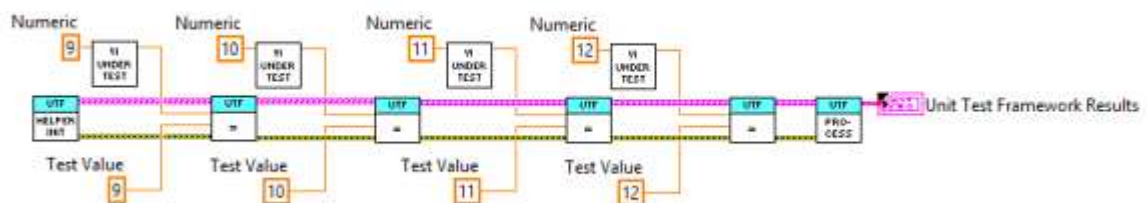


Рисунок 3.11 – Засіб для створення тестів

Даний засіб дозволяє дуже швидко складати дуже складні тестові кейси, використовуючи допоміжні VI.

CSS Unit Test Helper також дозволяє отримати звіт про результат. Якщо окремий компонент тесту не проходить перевірку, то зазнає невдачі весь тест. Однак створений звіт HTML дозволяє швидко визначити, який конкретний компонент вийшов з ладу, як показано на рис. 3.12:

1. Test Case 1: Failed

Repetition	Permutation	VI	In/Out	Control	Comparison	Used Vector Value	Value	Expected Value
1	1	VI under Test	In	Numeric			9	
			Out	Tested Value	=		9	9
				Tested Value	=		10	10
				Tested Value	=		11	77
				Tested Value	=		12	12

Рисунок 3.12 – Звіт тестування

Використання розробленого засобу значно скорочує час програмування, необхідний для створення користувацьких тестів, одночасно збільшуючи інформацію, отриману зі звітування.

3.3 Структура проєкту

Для створення нового тесту, необхідно натиснути на testExample.vit -> New from Template. Всі тести повинні містити префікс / слово test спочатку імені файлу.

При встановленні інструментів, в палітру функцій JKI Toolkits додаються API функції для роботи з VI тестером. Їх використовують для створення тесту, а саме - для визначення умов для успішного завершення тесту.

Спочатку необхідно створити модуль, який буде інплементувати числовий суфікс рядка. Для цього, необхідно дотримуватися наступних умов:

- якщо рядок порожній, необхідно генерувати помилку (5005);
- якщо рядок не містить числового суфіксу на кінці, то його необхідно додати (string -> string_01);
- якщо рядок містить числовий суфікс, його необхідно інкрементувати (string_02 -> string_03).

В такому випадку, коли відомі вимоги можна створити наступні рядки (рис. 3.13).

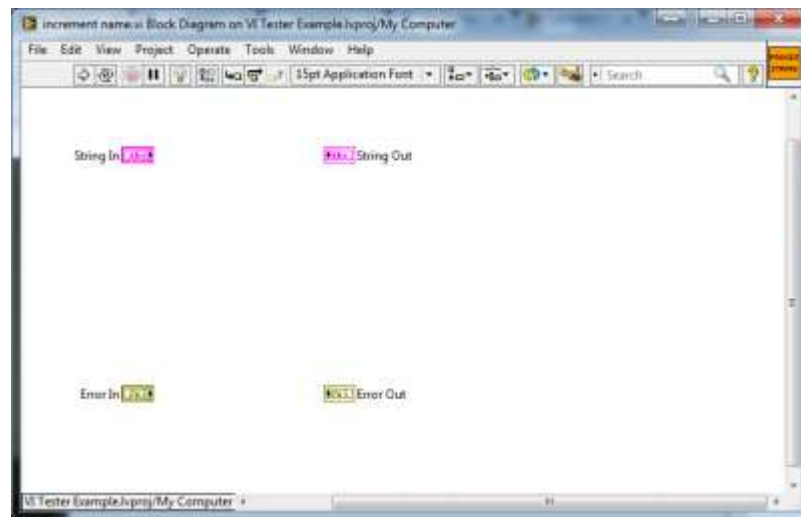


Рисунок 3.13 – Написання тесту

Наступним кроком необхідно запустити сам тестер, для цього необхідно обрати Tools -> VI Tester -> Test VIs... Та отримаємо вікно з тестуванням (рис. 3.14).

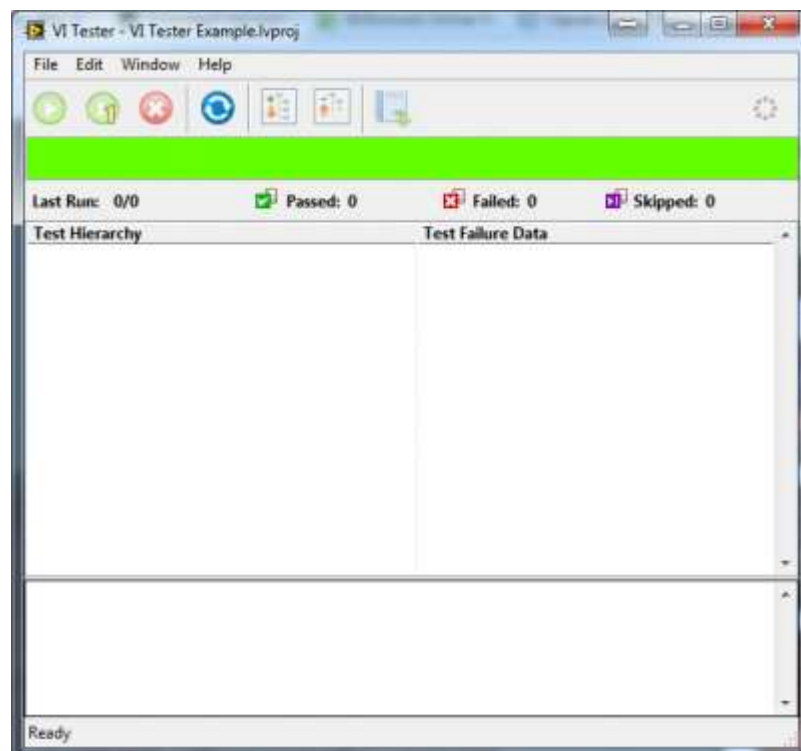


Рисунок 3.14 – Запуск тестера

тести мають статус Fail, це обумовлено тим що модуль ще не імплементований (рис. 3.18).

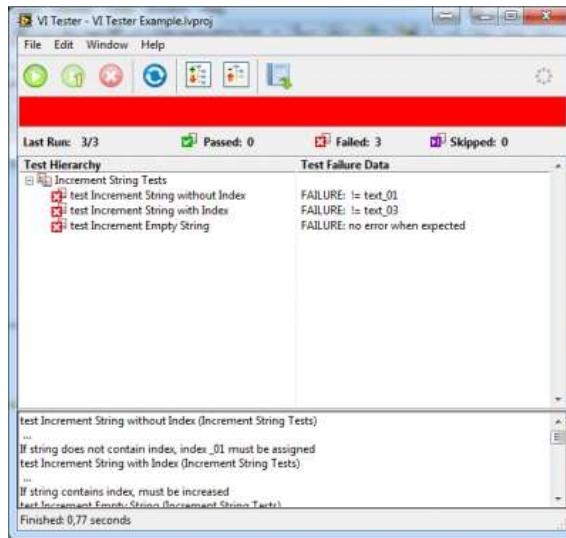


Рисунок 3.18 – Тести

Імплементуємо вимоги. Перше - це перевірка умови порожнього рядка, і генерація помилки (рис. 3.19).

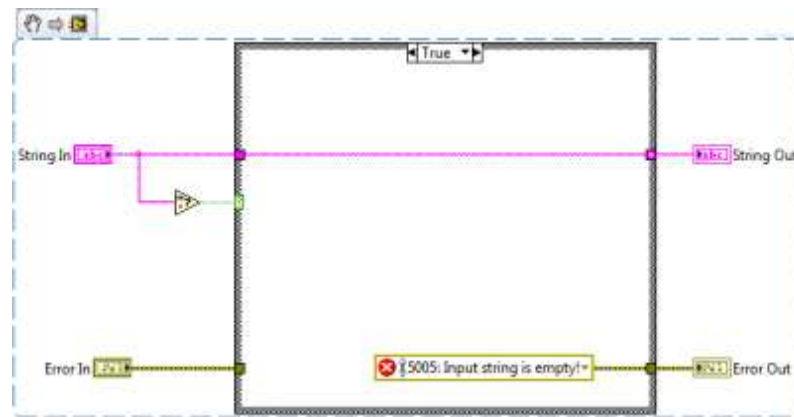


Рисунок 3.19 – Імплементация першої вимоги

Зберігаємо зміни модуля, запускаємо тест. Тепер, один з трьох тестів пройшов успішно (рис. 3.20).

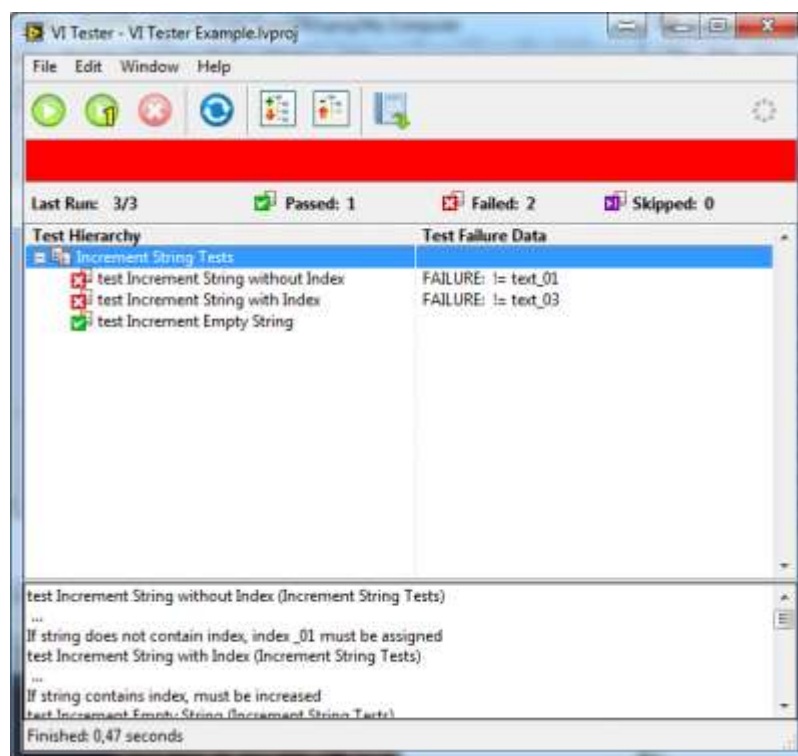


Рисунок 3.20 – Результат першого тестування

Імплементуємо ще дві вимоги (рис. 3.21), і запускаємо тести (рис. 3.22).

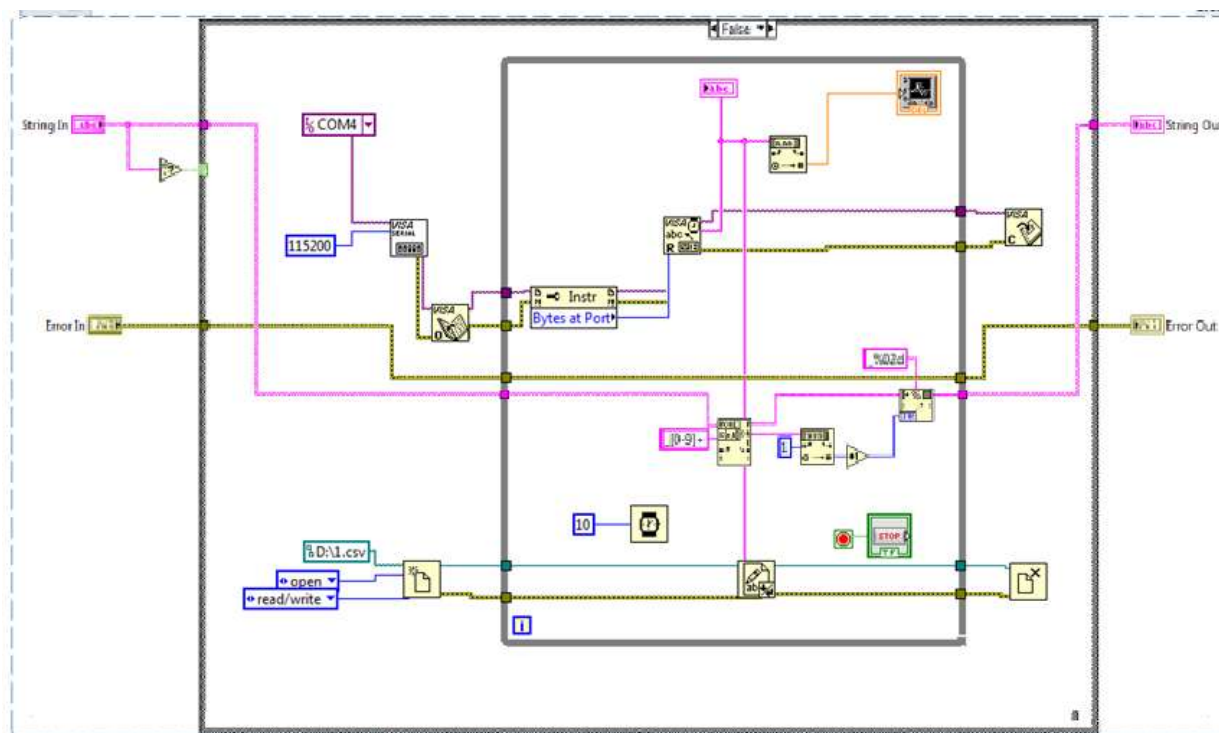


Рисунок 3.21 – Імплементация вимог

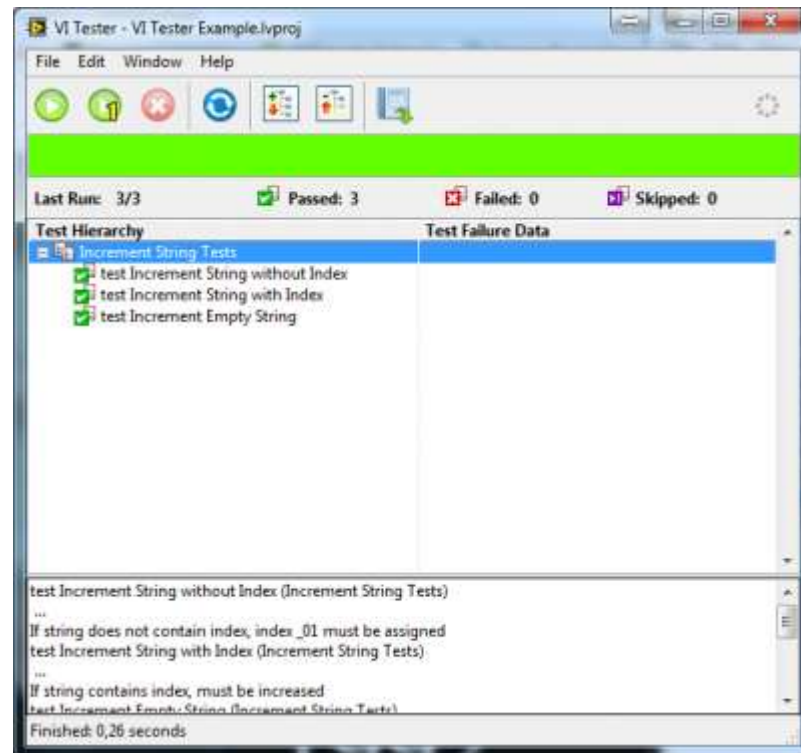


Рисунок 3.22 – Результат імplementації

Всі тести пройшли успішно, а це свідчить про коректність функціоналу проєктованого модуля.

Даний підхід ефективний якщо модуль повинен відповідати великій кількості вимог або якщо імplementація може змінювалась. Також в проєкті може бути безліч модулів для перевірки. тому зміна одного з модулів може нашкодити або вплинути на роботу інших.

Але якщо один раз створити тести, які будуть коректно працювати то потім можна їх використовувати для автоматичного тестування коду, даний підхід може заощадити час на тестування.

Також, за допомогою VI Tester можна створювати код, який буде запускати тестер з проєкту - наприклад, при додаванні коду в бібліотеку, або застосування. За допомогою Pre-Build Action та VI Tester API файли .vi можуть тестуватися автоматично.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення застосунку

Розроблений засіб призначена для перевірки моделей в Labview. За допомогою розробленого засобу можна проводити перевірку моделей та коду, що значно покращує якість проєктування та швидкість створення моделей та програм. Даний засіб дозволяє використовувати розроблені тести, вбудовані інструменти. таким чином засіб перевірено при проєктуванні моделі зчитування даних за допомогою Arduino і отримано звіт про помилки, які виникли в процесі проєктування. Графічне подання розробленого застосунку наведено у додатку А.

4.2 Програмне та апаратне забезпечення

Застосування створено за допомогою наступних програм і інструментів:

- LabVIEW;
- NI.LabVIEW v.16 DSP.Module;
- NI.LabVIEW v.2.0.DSP.Module;
- NI.LabVIEW v.16Microprocessor.SDK.Module;
- NI.LabVIEW v.16.FPGA.Module;
- NI.LabVIEW v.16.Mobile.Module;
- NI.LabVIEW.SignalExpress.v.3.0;
- NI.LabVIEW v.16.Real.Time.Module;
- NI.LabVIEW.TestStand.v.4.0.1;
- NI.LabVIEW.Syste.Identification.Toolkit.v.4.0;
- NI.LabVIEW.Sound and Vibration Toolkit v.4.0;
- LabVIEW.Spectral Measurements Toolkit v.2.1;
- NI.LabVIEW .Vision Builder for Automated Inspection (AI);
- LabVIEW.Simulation Interface Toolkit v.3.0.1;

- LabVIEW.Vision v16 Acquisition Software;
- NI LabVIEW v16 Statechart Module;
- NI.LabVIEW.Vision v16 Development Module;
- LabVIEW Advanced Signal Processing Toolkit V.16;
- NI Motion Assistant v2.3;
- NI Real-Time Execution Trace Toolkit v2.0.1;
- NI.LabVIEW.LabWindows CVI FDS v16;
- NI LabVIEW Adaptive Filter Toolkit v1.0;
- NI LabVIEW v 16 Digital Filter Design Toolkit;
- NI LabVIEW Report Generation Toolkit v1.1.3 for Microsoft Office;
- NI LabVIEW Database Connectivity Toolkit v1.0.2;
- NI LabVIEW VI Analyzer Toolkit v1.1.1;
- NI LabVIEW Simulation Interface Toolkit v5.0;
- NI LabVIEW v 16 Touch Panel Module;
- LabVIEW v 16.PID.Control.Toolkit;
- NI.LabVIEW.Modulation Toolkit V.3.1;
- LabView Embedded Module for ADI Blackfin v2.0 ;
- LabVIEW.Signal Processing Toolset V6.0;
- LabVIEW System Identification Toolkit v3.0.1;
- NI.LabVIEW.DIADEM v11.0;
- NI.LabVIEW.DIADEM INSIGHT v2.09;
- NI.LabVIEW.Digital Waveform Editor v2.0;
- LabVIEW.Calibration Executive v3.2;
- NI.LabVIEW.DAQmx v 16;
- NI.LabVIEW.ECU Measurement and Calibration Toolkit v2.0.1;
- NI.LabVIEW.NI.VI.Logger v2.0.1;
- NI.LabVIEW.Motion v 12;
- NI.LabVIEW.MATRIX.V 12;
- NI.LabVIEW.SoftMotion Controller v2.1;
- NI.LabVIEW.Lookout v6.1;

- NI.LabVIEW v.16.DSC.Module;
- NI.LabVIEW Internet Toolkit v.6.02;
- NI.LabVIEW.Control Design Toolkit v2.1.2;
- NI.LabVIEW Drivers;
- Introduction to LabVIEW 8 in 3 Hours;
- LabVIEW Basics 1 Hands on Cours;
- LabVIEW Example Collect;
- LabVIEW Basics 2 Hands on Cours;
- NI.LabVIEW 15 PDA Module;
- NI.Analog Waveform Editor v.1.1;
- D NI.IMAQ.for IEEE 1394 v.2.01;
- ASYLab v.9.02 English.

Для успішного встановлення та функціонування програмно-апаратного комплексу ПК повинен мати наступні характеристики: Pentium 4M (or equivalent) or later (32-bit), Pentium 4 G1, (64-bit) Windows 10 (version 1909)/ 8.1 Update 1/ 7 SP1, Windows Server 2016 (version 1607)/ 2012 R2 / 2008 R2 SP1 1 GB, RAM 5 GB \ 1,024 x 768 [19].

4.3 Структура проєкту

До структури VI Tester складається з трьох частин:

- групові тести, які складені для тестування власного програмного забезпечення - модульний тест - це лише VI, який написаний для тестування іншого VI;
- графічний користувальницький інтерфейс для запуску тестів або "Графічний пробник";
- об'єктно-орієнтована тестова структура - основна архітектура тестової основи.

Тестер складається з наступних файлів:

- VI Tester Example.lvproj;

- increment name.vi;
- VI Tester Example.aliases;
- VI Tester Example.lvlp;
- Increment String Tests.lvclass;
- setUp.vi;
- tearDown.vi;
- temp_VI_UnderTest.vi;
- test Increment Empty String.vi;
- test Increment String with Index.vi;
- test Increment String without Index.vi;
- testExample.vit.

Принцип роботи Tester досить простий. Можна створювати Test Cases, і Test Suites (які включають в себе Test Cases). Test Cases зазвичай містять тести для конкретних модулів / функціоналу.

При створенні Test Case, в проєкт додається клас, який містить setUp.vi, tearDown.vi, ві-приклад у вигляді - шаблону testExample.vi (рис. 4.1).

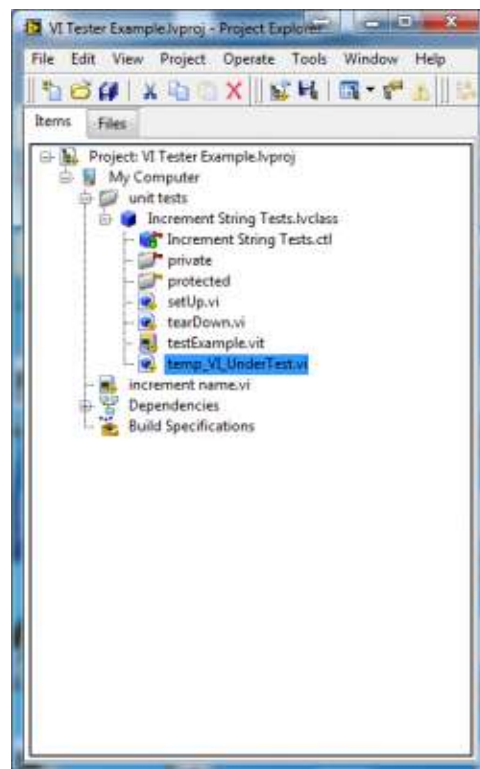


Рисунок 4.1 – Структура тестера

Для створення нового тесту, необхідно натиснути на `testExample.vit` -> `New from Template`. Всі тести повинні містити префікс `test` спочатку імені файлу.

При встановленні інструментів, в палітру функцій JKI Toolkits додаються API функції для роботи з VI тестером. Їх використовують для створення тесту, а саме - для визначення умов для успішного завершення тесту.

Для початку тесту необхідно визначити умови. В нашому випадку наступні:

- якщо рядок порожній, необхідно генерувати помилку (5005);
- якщо рядок не містить числового суфіксу на кінці, то його необхідно додати (`string` -> `string_01`);
- якщо рядок містить числовий суфікс, його необхідно інкрементувати (`string_02` -> `string_03`).

Згідно умов проєктується `.vi` з відповідними умовами.

У файлах `setUp.vi` і `tearDown.vi` можна додавати код, який ініціалізує параметри для тесту, також може відбуватися їх деініціалізація. Таким чином, під час запуску тесту виконується послідовність `setUp.vi` - `test.vi` - `tearDown.vi`.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

5.1 Призначення програми

Засіб призначений для перевірки якості моделей з використанням LabView за рахунок верифікації шляхом модульного тестування.

Основні можливості:

- верифікація моделей;
- перевірка за допомогою графічного програмування;
- можливість створення власних тестів;
- визначення помилок на стадії проєктування.

5.2 Умови виконання програми

Одним з важливих етапів створення моделей є забезпечення їх якості та перевірка на наявність помилок. Основним способом вирішення цієї проблеми є верифікація. Верифікацію можна виконувати шляхом проведення модульного тестування. Для роботи додатку необхідно використовувати програмно-апаратний комплекс LabView 20. Для початку роботи необхідно запустити файл testExample.vi. Після відкриття слід знайти файл .csv в якому містяться дані з результатами роботи моделі або системи. Наступним кроком необхідно на блок панелі написати шлях до даного файлу, це необхідно для тестування. Далі необхідно запустити проєкт для запуску тесту та вказати схему, яку необхідно перевірити, для цього необхідно запустити файл tester.vi Після запуску тестування можна перейти в вікно графічного відображення та переглянути результати роботи.

5.3 Виконання програми

Для початку користувачеві потрібно запустити проєкт VI Tester.lvprog.

Далі буде доступний графічний користувацький інтерфейс для запуску тестів або "Графічний пробник" (рис. 5.1).

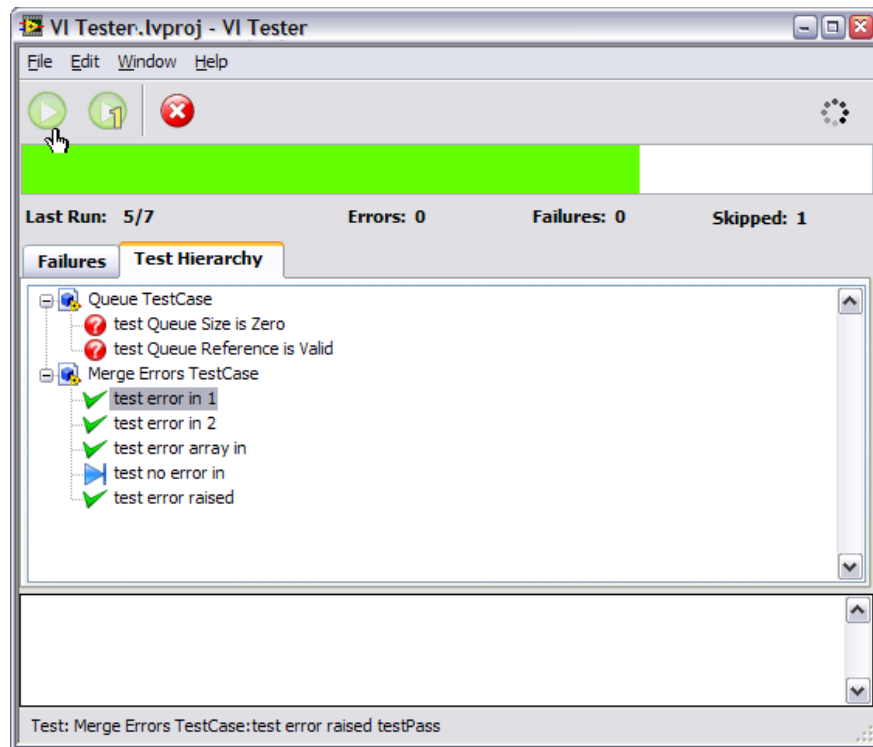


Рисунок 5.1 – Графічний інтерфейс

Допоміжні інструменти можна встановити за допомогою VI Package Manager (рис. 5.2). Після установки, інструмент доступний в меню Tools -> VI Tester.

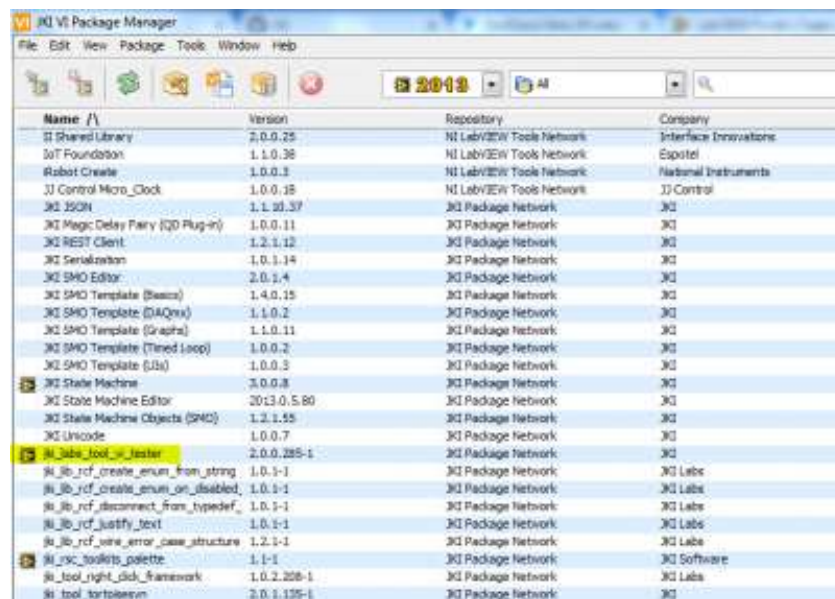


Рисунок 5.2 – VI Package Manager

Далі користувачу потрібно вказати з якого файлу потрібно обрати дані та до якого потрібно записати результат (рис. 5.3).

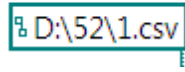


Рисунок 5.3 – Вікно запису шляху до файлу

5.4 Завершення роботи

Після завершення роботи тестера програми необхідно натиснути стандартну кнопку виходу, за замовчуванням користувачеві буде запропоновано зберегти зміни. Користувач сам вирішує чи погоджуватися із внесенням змін.

ВИСНОВКИ

В даній роботі були розглянуті методи верифікації інформаційних моделей, були досліджені можливості модульного тестування для вирішення задач верифікації.

Даний підхід який дозволяє проводити верифікацію найменших частин програми, що дозволить переконатися, в її працездатності.

Розроблений засіб ефективний у випадках, коли модуль повинен відповідати великій кількості вимог або при зміні імплементації. Також в проєкті може бути безліч модулів для перевірки. тому зміна одного з модулів може нашкодити або вплинути на роботу інших. Також даний засіб можна застосовувати для автоматичного тестування коду, даний підхід може заощадити час на тестування. Також, за допомогою інструменту LabView VI Tester можна створювати код, який буде запускати тестер з проєкту - наприклад, при додаванні коду в бібліотеку, або застосування. За допомогою Pre-Build Action та VI Tester API файли .vi можуть тестуватися автоматично.

В результаті роботи були вирішені наступні завдання:

- досліджено методи верифікації інформаційних моделей;
- досліджено програмні та апаратно-програмні комплекси для побудови засобів верифікації моделей;
- розроблено засіб верифікації моделі з використанням LabView.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Е. В. Кузьмин, О дисциплине специализации «Верификация программ» [Текст] / Е. В. Кузьмин, В. А. Соколов Доклады II научно-методической конференции «Преподавание математики в компьютерных науках» Ярославль: ЯрГУ. 2007. С. 91-101.
2. Харченко В. С. Оперативная верификация и коррекция программного обеспечения ИУС космических систем. Критичность функций и этапы верификации / В. С. Харченко, Н. В. Замирец, С. А. Засуха // Системи управління, навігації та зв'язку. – К.: ДП «ЦНДІ НІУ», 2011. – С. 124-130.
3. Symbolic Model Verificator. Carnegie Mellon University. – [Electronic resource]. – 2016 – Access mode: <http://www.cs.cmu.edu/~modelcheck/smv.html>.
4. Bender Marc. Positioning Verification in the Context of Software / System Certification / Marc Bender, Tom Maibaum, Mark Lawford, Alan Wassong // Proceedings of the 11th International Workshop on Automated Verification of Critical Systems (AVoCS 2011). – Newcastle-upon-Tyne. – 2011. – 15 p.
5. Маевский Д. А. Верификация моделей надежности программного обеспечения / Д. А. Маевский, Е. Ю. Маевская, О. П. Жеков // Радиоэлектронні і комп'ютерні системи. – 2014. – № 5. – С. 55-59.
6. Когай, Г. Д. Достоинства и недостатки современных видов тестирования программного обеспечения / Г. Д. Когай, А. Ж. Амиров, Р. В. Стародубенко, К. В. Стародубенко. – Текст : непосредственный // Молодой ученый. – 2016. – № 21 (125). – С. 153-156.
7. Developing a Software Project Life Cycle Process тестирования. – [Electronic resource]. – 2006 – Access mode: https://global.ihs.com/doc_detail.cfm?document_name=IEEE%201074&item_s_key=00130568.

8. ISO/IEC 12207:2008 Systems and software engineering – Software life cycle processes. – [Electronic resource]. – 2008 – Access mode: <https://www.iso.org/standard/43447.html>.
9. ISO/IEC 15504-5:2012 Information technology – Process assessment – Part 5: An exemplar software life cycle process assessment model.– [Electronic resource]. – 2012 – Access mode: <https://www.iso.org/standard/60555.html>.
10. ISO/IEC/IEEE 15288:2015 Systems and software engineering – System life cycle processes. – [Electronic resource]. – 2015 – Access mode: <https://www.iso.org/standard/63711.html>.
11. Кузин, Д. А. Создание модульной системы умного дома на базе аппаратного комплекса National Instruments [Текст] / Д. А. Кузин. – Текст : непосредственный // Молодой ученый. – 2016. – № 19 (123). – С. 87-90.
12. Кожанов, Р. В. Модернизированная виртуальная лаборатория «Комбинационные цифровые устройства. Версия 2» [Текст] / Р. В. Кожанов, К. Ю. Котова, Е. Р. Кожанова. // Исследования молодых ученых : материалы IV Междунар. науч. конф. – 2019. – С. 5-8.
13. Средства для модульного тестирования. – [Электрон. ресурс]. – 2019 – Режим доступа: <http://cunit.sourceforge.net/>.
14. Туллит LabVIEW Unit Test Framework. – [Электрон. ресурс]. – 2019 – Режим доступа: <https://www.ni.com/ru-ru/search.html?pg=1&ps=10&q=780509-35>.
15. Туллит LabVIEW 2020 Unit Test Framework. – [Электрон. ресурс]. – 2020 – Режим доступа: <https://www.ni.com/ru-ru/support/downloads/software-products/download.labview-unit-test-framework-toolkit.html#345603>.
16. Metcalf A. Unit Testing in LabVIEW. – [Electronic resource]. – 2017 – Access mode: <http://controlsoftwaresolutions.com/software-development-utf/>.
17. JKI VI Tester - инструмент для тестирования LabVIEW. – [Электрон. ресурс]. – 2020 – Режим доступа: <https://resources.jki.net/vi-tester>.

18. Caraya Unit Test Framework by JKI - Toolkit for LabVIEW. – [Electronic resource]. – 2020 – Access mode: https://www.vipm.io/package/jki_lib_caraya/.

19. How to Automate LabVIEW Code Validation to Focus on Your Tests (Part 1 of 2). – [Electronic resource]. – 2021 – Access mode: <https://www.synovus.ca/how-to-automate-labview-code-validation-to-focus-on-your-tests-part-1-of-2/>.

ДОДАТОК А

Графічне подання програмного забезпечення, розробленого за допомогою графічної мови програмування G та середовища LabVIEW

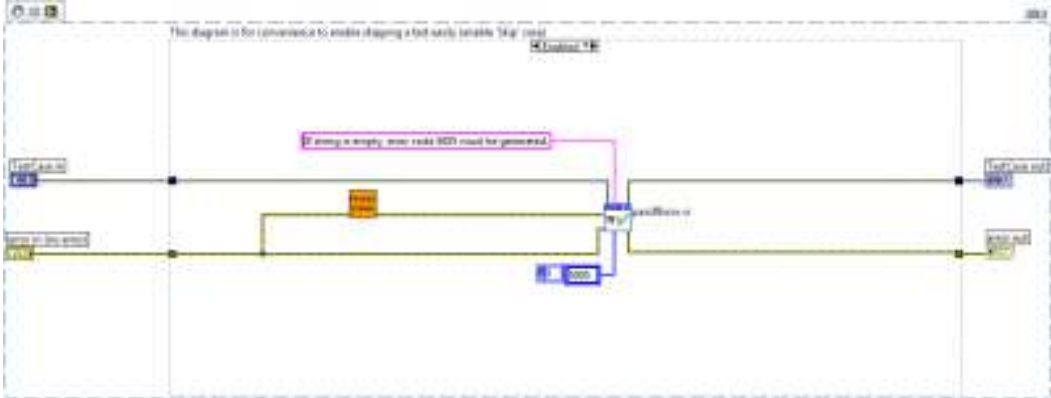


Рисунок А.4 – Третій тест

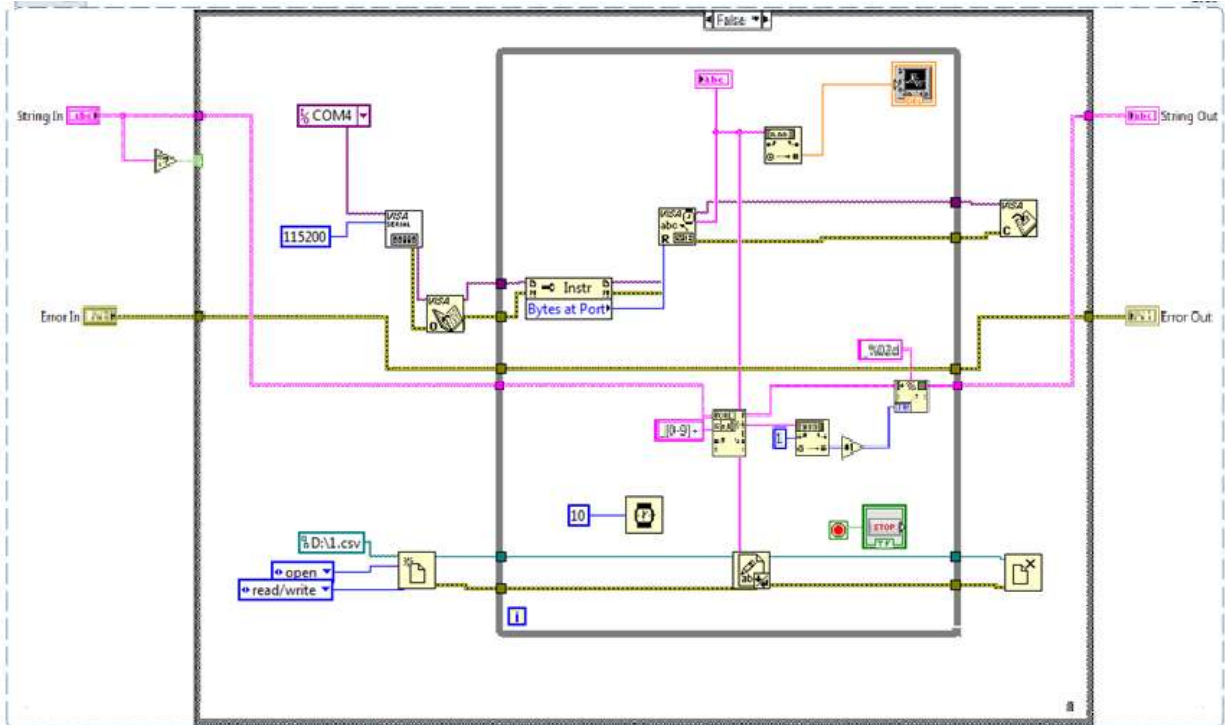


Рисунок А.5 – Імплементация вимог

ДОДАТОК Б
Слайди презентації

Міністерство освіти та науки України
Національний університет «Запорізька політехніка»

Програмне забезпечення автоматизації перевірки програмного коду

СТРІЛЬЧУК Дмитро Віталійович

ст. групи КНТ-119

керівник КОЦУР Михайло Ігорович

к.т.н., доцент НУ «Запорізька політехніка»

Рисунок Б.1 – Слайд №1

Об'єкт, предмет та мета роботи

Об'єкт дослідження – процес перевірки програмного коду.

Предмет дослідження – засоби перевірки програмного коду.

Метою роботи є розробка моделі програмного забезпечення для перевірки програмного коду.

Рисунок Б.2 – Слайд №2

Завдання роботи

Для досягнення поставленої мети у дипломній кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

- дослідити засоби перевірки програмного коду;
- дослідити програмні та апаратно-програмні комплекси для побудови засобу верифікації програмного коду;
- розробити засоби верифікації програмного коду з використанням LabView.

3

Рисунок Б.3 – Слайд №3

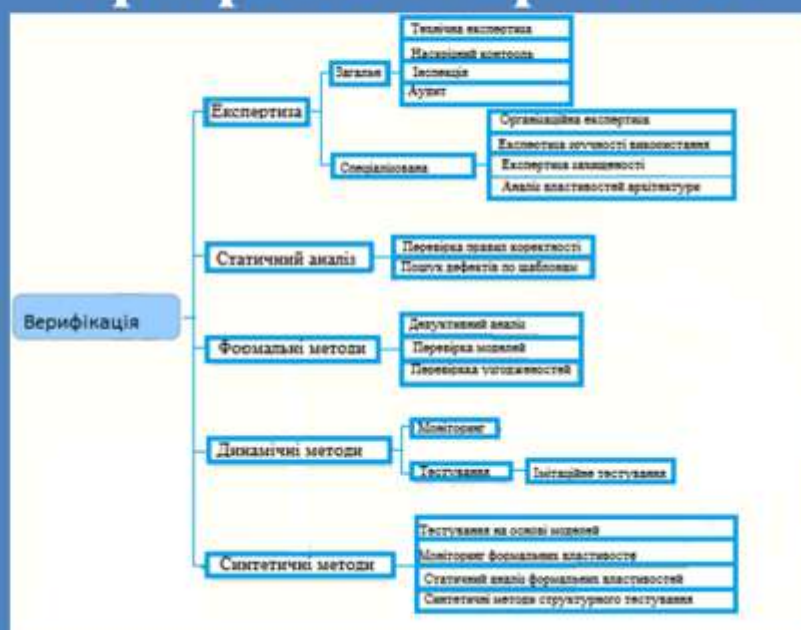
Порівняння існуючих засобів перевірки програмного коду



4

Рисунок Б.4 – Слайд №4

Порівняння методів верифікації програмних проєктів



5

Рисунок Б.5 – Слайд №5

Вибір середовища розробки

Критерій порівняння мов середовищ розробки	Середовище розробки		
	LabVIEW	myOpenLab	PyLab_Works
Парадигма програмування	Графічне програмування	Графічне програмування	Текстове програмування
Зручність використання	+	+-	+-
Підтримка розробників	+	+-	+
Висока функціональність	+	+-	+-
Застосування в індустрії	+	+-	+-
Зручний графічний інтерфейс користувача	+	+-	+-

6

Рисунок Б.6 – Слайд №6

Процес верифікації програмних систем у LabVIEW

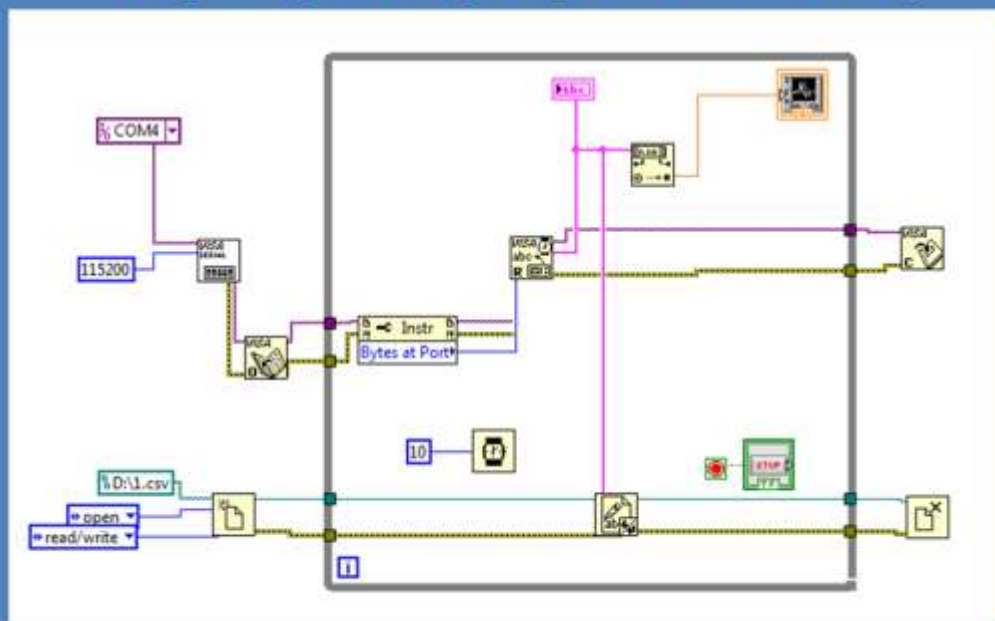
Проведення верифікації відбувається шляхом виконання наступних етапів:

- верифікація логіки;
- верифікація часу виконання;
- перевірка завантаження пам'яті;
- відстеження виконання.

7

Рисунок Б.7 – Слайд №7

Модель програмного забезпечення для перевірки програмного коду



8

Рисунок Б.8 – Слайд №8

Файлова структура програмного забезпечення для перевірки програмного коду

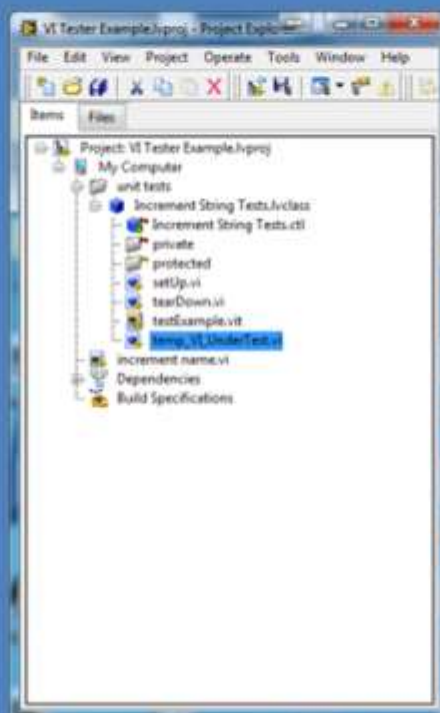


Рисунок Б.9 – Слайд №9

Схема процесу тестування програмного забезпечення

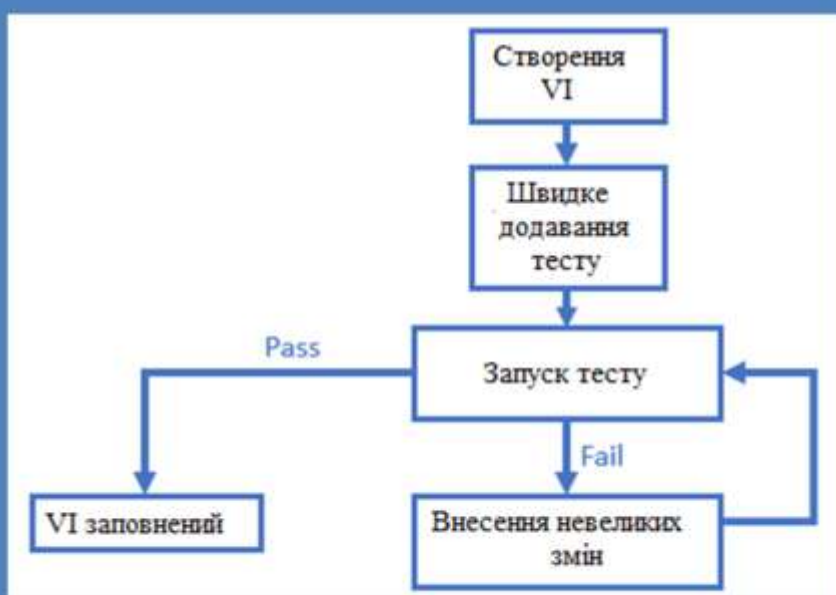


Рисунок Б.10 – Слайд №10

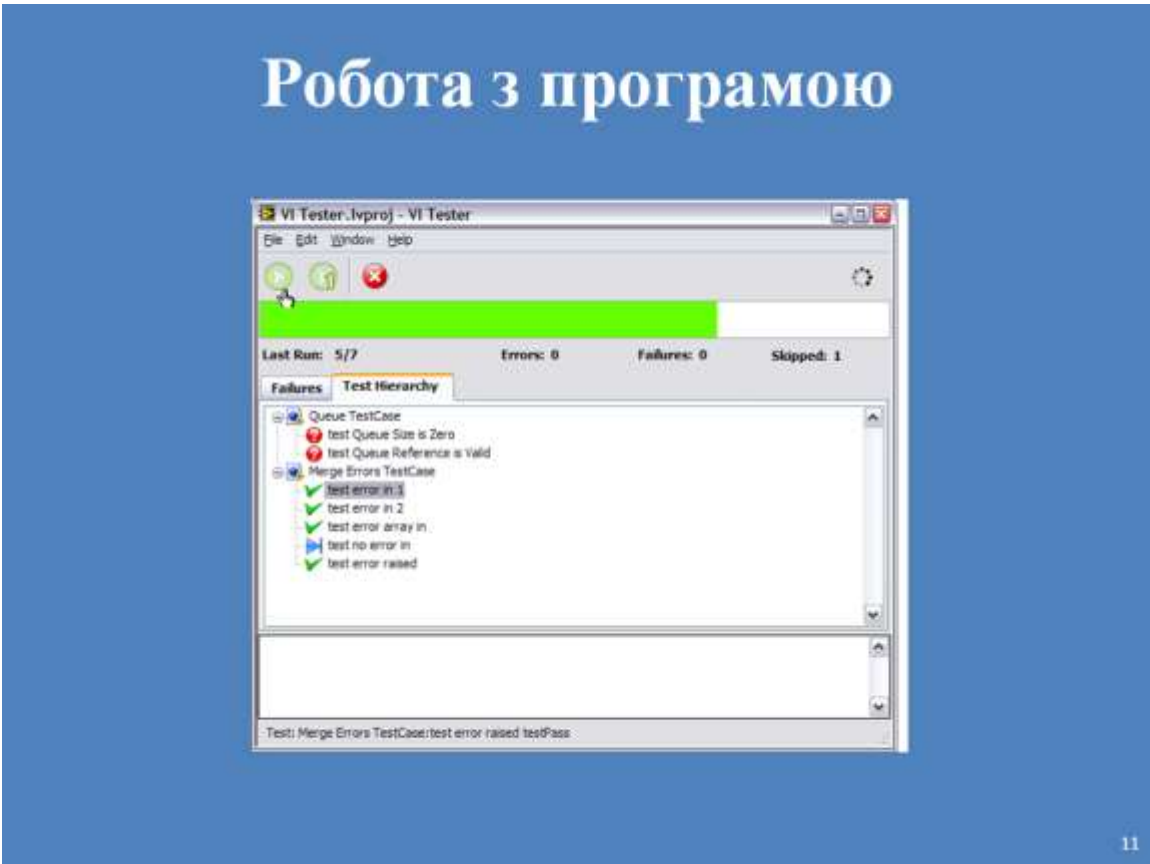


Рисунок Б.11 – Слайд №11

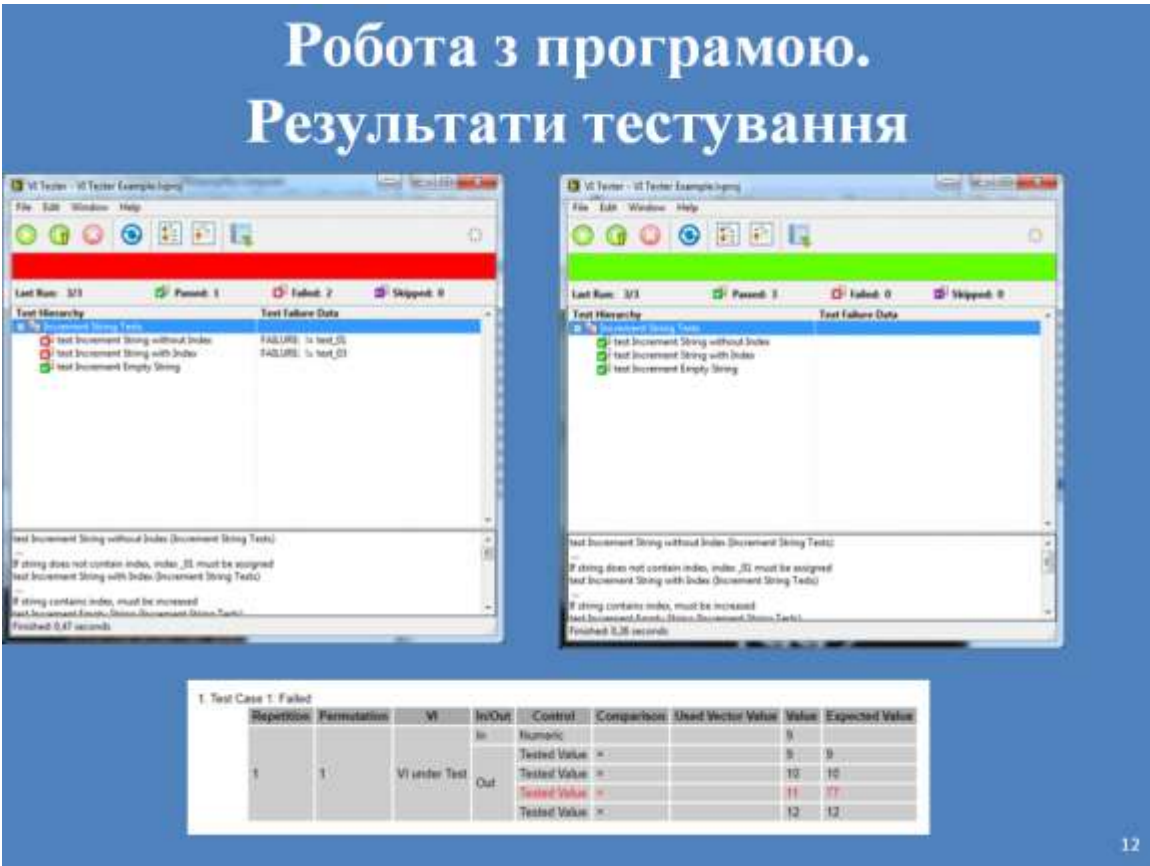


Рисунок Б.12 – Слайд №12

Висновки

В результаті виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процес перевірки програмного коду.

Розроблено програмне забезпечення для перевірки програмного коду, що дозволяє проводити верифікацію частин програми та дозволяє перевірити її працездатність.

Розроблений засіб ефективніший у випадках, коли модуль повинен відповідати великій кількості вимог або при зміні імплементації. Також розроблений засіб можна застосовувати для автоматичного тестування коду, даний підхід може заощадити час на тестування. За допомогою розробленого програмного засобу можна створювати код, який буде запускати тестер з проксту, наприклад, при додаванні коду в бібліотеку, або застосування.

Результати перевірки розробленого програмного забезпечення підтвердили його працездатність.

Усі завдання дипломної кваліфікаційної роботи бакалавра повністю виконано.

Рисунок Б.13 – Слайд №13