НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ім. Ігоря Сікорського
Навчально-науковий комплекс «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
"___"_____2017 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 Комп'ютерні науки
(код і назва)

на тему: Програма випробування тестів комбінаційних схем

Виконав: студент 4 курсу, групи ДА-31
(шифр групи)

Петрик Іван Романович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник старший викладач Британов О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Економічний доцент Рощина Н. В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль старший викладач Британов О.А. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____
(підпис)

Київ – 2017 року

Національний технічний університет України
«Київський політехнічний інститут»
ім. Ігоря Сікорського

Інститут (факультет) ННК «Інститут прикладного системного аналізу
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050101 Комп'ютерні науки
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2017 р.

ЗАВДАННЯ
на дипломну роботу студенту
Петрику Івану Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програма випробування тестів комбінаційних схем
керівник роботи старший викладач Британ О. А. _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «10» 05 2017 р. № 1477-с

2. Термін подання студентом роботи 14.06.2017

3. Вихідні дані до роботи Типи тестів: контролюючі і діагностичні. Об'єкт тестування: комбінаційна схема. Обмеження на кількість входів, виходів, елементів: обмежено оперативною пам'яттю. Наявність графічного інтерфейсу користувача. Програма повинна перевіряти ефективність тестів, розроблених під час лабораторних та курсових робіт.

4. Зміст роботи 1) Виконати аналіз задачі; 2) Обрати інструментальні засоби для розробки програми; 3) Виконати проектування програми; 4) Розробити програму згідно проекту; 5) Протестувати програму, використовуючи розроблені в рамках лабораторних робіт тести.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Діаграма варіантів використання (Use Case Diagram) – плакат

2. Діаграма класів (UML Class Diagram) – плакат

3. Формат опису вхідних даних – плакат

4. Приклади роботи програми – плакат

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	доцент Рощина Н. В.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання		
2	Аналіз задачі		
3	Вибір засобів вирішення задачі		
4	Проектування програми		
5	Розробка програми		
6	Тестування програми		
7	Оформлення дипломної роботи		
8	Отримання допуску до захисту та подача роботи в ДЕК		

Студент _____ І. Р. Петрик
(підпис) (ініціали, прізвище)

Керівник роботи _____ О. А. Бритов
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломної роботи.

АНОТАЦІЯ

бакалаврської дипломної роботи Петрика Івана Романовича

на тему «Програма випробування тестів комбінаційних схем»

Дипломна робота присвячена розробці комп'ютерної програми для випробування тестів, призначених для тестування комбінаційних схем.

Метою розробки є перевірка ефективності тестів, розроблених в рамках лабораторних та курсових робіт.

В ході роботи проаналізовано необхідні інструментальні засоби для розробки програми, розроблено формат опису комбінаційної схеми та завдання на її тестування, розроблено архітектуру програми та прототип графічного інтерфейсу користувача. Розглянуто способи моделювання комбінаційних схем та методи побудови тестів для них.

Результатом роботи є відлагоджена програма з графічним інтерфейсом користувача, яка дозволяє моделювати будь-які комбінаційні схеми в справному стані, задавати несправності для схеми та виконувати її тестування на заданому тестовому наборі задля перевірки ефективності розроблених тестів, а також будувати тести для схем програмним шляхом.

Загальний обсяг роботи – 81 сторінка, 1 додаток на 3 сторінках, 42 рисунка, 25 таблиць, 10 посилань.

Ключові слова: комбінаційна схема, моделювання, несправність, тестування, контролюючий тест, діагностичний тест.

АННОТАЦИЯ

бакалаврской дипломной работы Петрика Ивана Романовича

на тему «Программа испытания тестов для комбинационных схем»

Дипломная работа посвящена разработке компьютерной программы для испытания тестов, предназначенных для тестирования комбинационных схем.

Целью разработки является проверка эффективности тестов, разработанных в рамках лабораторных и курсовых работ.

В ходе работы были проанализированы необходимые инструментальные средства для разработки программы, разработан формат описания комбинационной схемы и задания на ее тестирование, разработаны архитектура программы и прототип графического интерфейса пользователя. Были рассмотрены способы моделирования комбинационных схем и методы построения тестов для них.

Результатом работы является хорошо отлаженная программа с графическим интерфейсом пользователя, которая позволяет моделировать любые комбинационные схемы в исправном состоянии, задавать неисправности для схемы и выполнять ее тестирование на заданном тестовом наборе для проверки эффективности разработанных тестов, а также строить тесты для схем программным путем.

Общий объем работы - 81 страница, 1 приложение на 3 страницах, 42 рисунка, 25 таблиц, 10 ссылок.

Ключевые слова: комбинационная схема, моделирование, неисправность, тестирование, контролирующий тест, диагностический тест.

ANNOTATION

of a bachelor’s degree work by Petryk Ivan Romanovich

on the topic of «The program tests for testing combinational circuits»

Degree work is devoted to the development of a computer program for testing tests designed for testing combinational circuits.

The purpose of the development is to test the effectiveness of tests developed in the framework of laboratory and coursework.

The work analyzed the necessary tools to develop applications, designed format for combinational circuit descriptions and tasks of testing, developed architecture of application and a prototype of a graphical user interface. Methods for modeling combinational circuits and methods for building tests for them are considered.

The result of the work is well debugged program with a graphical user interface, that allows you to simulate any combinational circuit in correct state, set malfunctions for the circuit and perform its test on a given test set in order to test the effectiveness of the developed tests, and build tests for circuits programmatically.

The total volume of work – 81 pages, 1 Appendix on 3 pages, 42 picture, 25 tables, 10 references.

Keywords: combinational circuit, simulation, malfunction, testing, controlling test, diagnostic test.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

Комбінаційна схема (КС) - схема, комбінація сигналів на виході якої в будь-який момент часу однозначно визначається комбінацією сигналів на її вході.

Контролюючий тест (КТ) – тест, мета якого перевірити наявність помилок в схемі.

Діагностичний тест (ДТ) – тест, мета якого встановити тип помилки в схемі.

Повний КТ – тест, що виявляє всі несправності, які впливають на функціонування схеми

Повний ДТ – тест, що відрізняє всі несправності, які по різному впливають на функціонування схеми

EBNF (Extended Backus–Naur form) – формальна система визначення синтаксису, в якій одні синтаксичні категорії послідовно визначаються через інші

GUI (Graphical User Interface) – графічний інтерфейс користувача

Токен – група символів, що відповідає певному шаблону

Лексичний аналіз – це процес перетворення послідовності символів в послідовність токенів, та визначення їх типів.

Синтаксичний аналіз – це процес аналізу вхідної послідовності токенів, з метою розбору граматичної структури згідно із заданою формальною граматикою.

ВСТУП

Пристрої на основі КС є широко розповсюджені в цифровій електроніці. Вони застосовуються в обчислювальних цілях для формування вхідних сигналів і для підготовки даних, які підлягають збереженню. Найпоширеніші пристрої на основі КС: шифратори, дешифратори, перетворювачі кодів, суматори, мультиплексори, демультиплексори.

При виробництві, експлуатації, ремонті і зберіганні пристроїв на основі КС потрібно здійснювати перевірку їх справності, працездатності, правильності функціонування і шукати несправності. Типовими несправностями пристроїв на основі КС є обрив і замикання з’єднань компонентів схеми. Ці несправності можна описати як постійне значення логічної 1 або 0 в місці несправності.

Вважають, що пристрій на основі КС знаходиться в справному стані, якщо він задовольняє всім технічним вимогам, що пред'являються до нього в даний період його життя (виробництва, експлуатації, ремонту, зберігання). В іншому випадку він знаходиться в несправному стані і називається несправним.

При виникненні несправності в пристрої в процесі його виготовлення, налагодження, експлуатації та зберігання можливі два випадки:

- функціонування пристрою з несправністю відрізняється від функціонування справного пристрою (випадок несправностей, що функціонально проявляються);
- функціонування пристрою з несправністю не відрізняється від функціонування справного пристрою.

Другий випадок має місце в надлишкових пристроях, наприклад в пристроях з резервуванням елементів.

Для контролю і діагности КС розрізняють два основні методи: функціональний і тестовий. У першому випадку стан пристрою перевіряється в процесі його використання за прямим призначенням. При цьому перевіряється правильність реалізації пристроєм тієї чи іншої функції. Така перевірка

здійснюється додатковими або вбудованими апаратурними засобами.

При тестовому контролі і діагностиці на пристрій подаються спеціально сформовані вхідні (тестові) набори, правильна реакція на які заздалегідь відома. Стан пристрою визначається за отриманою реакцією на задані тестові набори.

В даній роботі розглядається тестовий контроль і діагностика КС. Для однієї ж і тієї схеми може бути побудовано кілька варіантів тестів, які можуть відрізнятися за кількістю тестових наборів та за ефективністю. При побудові даних тестів людиною, можуть виникати помилки, що призводять до неефективності даного тесту або надлишковості деяких наборів тесту. Тому перед практичним застосуванням розроблених тестів, важливим етапом є перевірка їх ефективності.

Задачею даної дипломної роботи є розробка програми, яка виконує тестування схеми з заданим переліком несправностей за допомогою розроблених КТ та ДТ.

Метою дипломної роботи є перевірка ефективності тестів розроблених в рамках лабораторних та курсових робіт. Критерієм для оцінки ефективності КТ є кількість несправностей, що виявляє даний КТ. Критерієм оцінки ефективності ДТ є кількість вірно діагностованих несправностей.

Вимогами до програми є наявність графічного інтерфейсу, стабільна робота на будь-яких вхідних даних, а при наявності помилок, повідомлення про їх місце та тип, можливість збереження вхідних даних у вигляді файлу для повторного використання, зручний спосіб введення вхідних даних, інформативність результатів тестування, можливість програмної побудови тестів для порівняння з розробленими тестами.

Результатом розробки повинна бути програма з графічним інтерфейсом користувача, яка дозволяє користувачу ввести вхідні дані (схему, перелік несправностей, тести) та виконати тестування. Результати тестування повинні містити інформацію про несправності, що не було виявлено/діагностовано, їх кількість та результати моделювання схеми на заданому тесті з несправностями та без них.

1 АНАЛІЗ ЗАДАЧІ І ВИБІР ЗАСОБІВ ЇЇ ВИРІШЕННЯ

1.1 Аналіз задачі

1.1.1 Загальний аналіз

Задачею дипломної роботи є написання програми для випробування тестів, призначених для КС. Метою цієї розробки є перевірка ефективності тестів розроблених в рамках лабораторних та курсових робіт. В рамках лабораторних та курсових робіт розроблялися два види тестів: контролюючі та діагностичні. Метою КТ є перевірка того, чи є в схемі несправність, а метою ДТ – знаходження типу цієї несправності. Однією з вимог до програми є наявність графічного інтерфейсу. Для виконання тестування необхідно попередньо реалізувати алгоритм моделювання схем. Програма повинна моделювати схеми в справному та несправному станах. Щоб моделювати схему, її попередньо потрібно представити в пам’яті комп’ютера. Для опису схеми, несправностей та тестів необхідно розробити власний формат вхідних даних та парсер цього формату. Додатковим завданням є реалізація алгоритмів побудови тестів для схем. Метою їх реалізації є порівняння розроблених тестів з побудованими програмним шляхом.

1.1.2 Характеристики випробуваних тестів

КТ та ДТ призначені для тестування константних несправностей типу 0 або 1. В контексті лабораторних та курсових робіт несправність в схемі може бути призначена входу схеми або виходу елемента схеми. Значення компонента схеми з несправністю буде рівна значенню константної несправності незалежно від значень вхідних сигналів.

КТ та ДТ призначені для тестування одиничних несправностей. Вони не розроблені для тестування схем, в яких одночасно можуть бути присутні декілька несправностей.

КТ та ДТ схеми будується для певного переліку можливих несправностей схеми, який задається на етапі побудови. Для одного й того ж переліку несправностей можливе отримання різних КТ та ДТ, що не відрізняються в плані ефективності. Детально методика побудови КТ та ДТ розглянута в [1, 2].

1.1.3 Аналіз задачі випробування КТ

Результатом побудови КТ є набір вхідних сигналів і значення виходів справної схеми при подачі кожного з цих сигналів на її вхід. Для схеми зображеної на рисунку 1.1 у таблиці 1.1 наведено КТ для наступних несправностей: 10, 20, 30, 40, 50,60, 70, x10, x20, x30, x40, x50, 11, 21, 31. Цифра або літера і цифра на початку позначають назву елемента з несправним виходом або несправний вхід схеми відповідно, цифра в кінці – значення константної несправності.

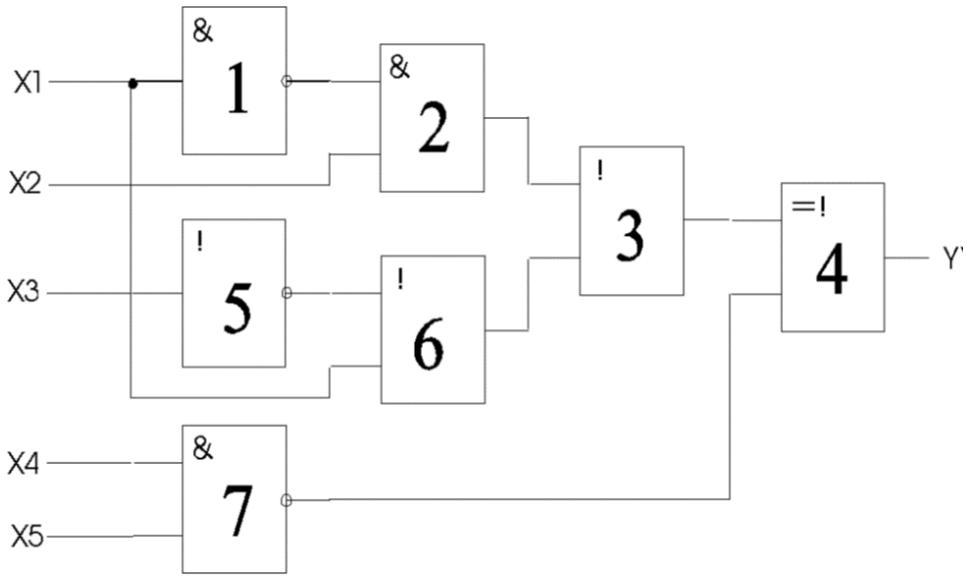


Рисунок 1.1 – Комбінаційна схема для КТ

Для контролю схеми необхідно послідовно подавати на входи схеми тестові набори КТ (див. табл. 1.1) і порівнювати значення виходів схеми зі значеннями виходів вказаними в КТ. Якщо вони відрізняються, то це означає, що в схемі присутня несправність, якщо ні – схема справна або тест не виявляє несправності. Наприклад, константна несправність 31 (вихід 3-го елемента

схеми, значення несправності 1) виявляється набором №5, оскільки вихід справної схеми рівний 1, а вихід несправної – 0.

Таблиця 1.1 – КТ для схеми на рис. 1.1

№ набору	Входи					Виходи
	X1	X2	X3	X4	X5	Y
4	0	0	0	1	1	1
5	0	0	1	0	0	1
13	0	1	1	0	0	0

Приведений в табл. 1.1 КТ не виявляє дві несправності x10 та 10. Несправність 10 не виявляється тестовим набором, оскільки вона не впливає на роботу схеми і схема функціонує правильно навіть з цією несправністю. В той же час, несправність x10 не виявляється тестовим набором, хоча впливає на роботу схеми. Наприклад, вхідний сигнал 10100 в справній схемі дає значення виходу 0, а в несправній – 1. Таким чином, можна зробити висновок, що КТ в табл. 1.1 є не повним, оскільки він не виявляє несправності x10, яка впливає на функціонування схеми.

Отже, задачею щодо КТ є пошук несправностей, які тест не виявляє та перевірка повноти тесту. Для пошуку несправностей, які тест не виявляє, потрібно промоделювати на тестовому наборі несправну схему по кожній несправності з переліку та порівняти значення вихідних сигналів справної та несправної схем. Якщо вони одинакові, то дану несправність даний КТ не виявляє. Для з’ясування повноти тесту, необхідно знати які несправності з переліку несправностей не впливають на функціонування схеми. З’ясувати ці несправності можна промоделювавши справну та несправну схему по кожній не виявленій даним тестом несправності на всіх можливих вхідних сигналах та порівнявши результати моделювання справної та несправних схем. Якщо результати відрізняються, то тест не виявляє несправності, інакше – несправність не впливає на функціонування. Якщо є відомим будь-який повний

КТ, то для перевірки повноти достатньо порівняти несправності, які не виявляє розроблений КТ з несправностями, які не виявляє повний КТ.

1.1.4 Аналіз задачі випробування ДТ

Результатом побудови ДТ є набір вхідних і вихідних сигналів для кожної несправності схеми. Значення вихідного сигналу справної схеми для діагностичного тесту не потрібно, оскільки його метою є знаходження типу несправності у схемі з несправністю, а не перевірка схеми на її наявність. Для схеми зображеної на рисунку 1.2 в таблиці 1.2 наведено ДТ для наступних несправностей: 31, 41, 51, 61, 71, 91.

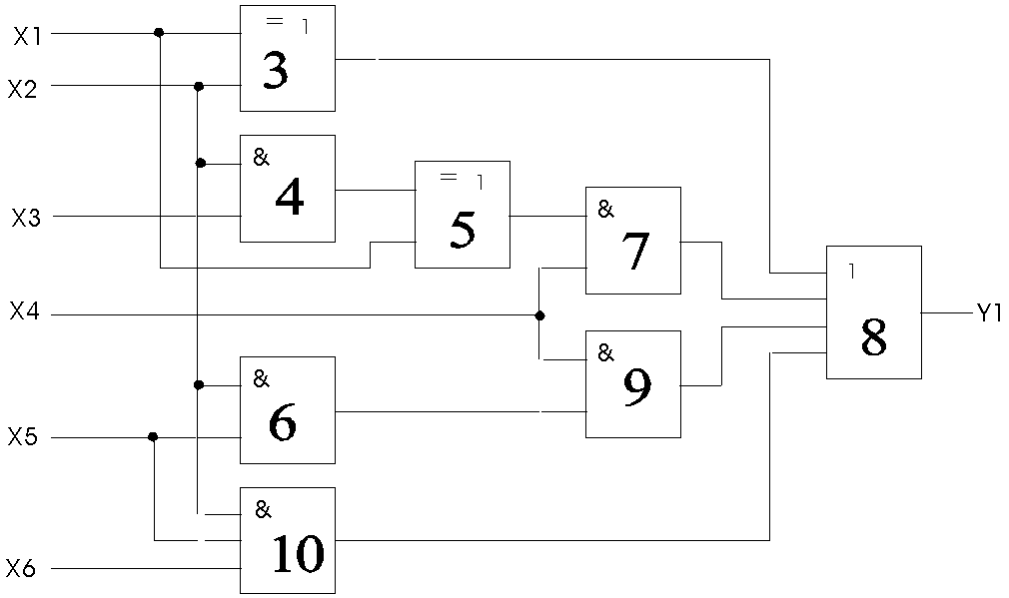


Рисунок 1.2 – Комбінаційна схема для ДТ

Таблиця 1.2 – ДТ для схеми на рис. 1.2

№ набору	Входи						Несправності					
	X1	X2	X3	X4	X5	X6	31	41	51	61	71	91
							Y	Y	Y	Y	Y	Y
1	0	0	0	0	0	0	1	0	0	0	1	1
53	1	1	0	1	0	0	1	0	1	1	1	1

Для діагностики схеми необхідно послідовно подавати на входи схеми тестові набори ДТ і зафіксувати значення вихідних сигналів схеми. Наприклад, при наявності в схемі несправності 51 отримаємо послідовність вихідних сигналів 01. Проте послідовності 01 також відповідає несправність 61, тому не можна однозначно сказати, яка несправність присутня в схемі: 51 чи 61. Таким чином, даний ДТ не розрізняє дані несправності.

Насправді, не можливо побудувати ДТ який би розрізняв несправності 51 та 61, оскільки функціонування схеми на всіх можливих вхідних сигналах при наявності однієї з них не відрізняється. Таким чином, даний ДТ можна назвати повним, оскільки всі несправності, які по різному впливають на функціонування схеми він відрізняє. Тим не менше, можливі варіанти неповних ДТ, які не будуть розрізняти ще більшу кількість несправностей, ніж даний.

В першу чергу, задачею щодо ДТ є перевірка коректності даного ДТ. ДТ буде вважатися коректним, якщо він вірно діагностує несправності. Тобто отримання послідовності 01 на виході можливе лише при наявності в схемі несправності 51 або 61 з переліку несправностей. Якщо в схемі наявна інша несправність з переліку заданого при побудові, наприклад 31, і на виході отримана послідовність 01, то даний ДТ є не коректним.

Другою задачею щодо ДТ є перевірка його ефективності або повноти, тобто порівняння кількості несправностей з переліку заданих несправностей, які розрізняє тест, і кількості несправностей, які можна відрізнити при моделюванні схеми на всіх можливих вхідних сигналах.

Для вирішення першої задачі необхідно моделювати схему в несправних станах по кожній несправності з переліку на заданому ДТ і порівнювати послідовності вихідних сигналів схеми з очікуваними послідовностями вихідних сигналів вказаними в ДТ. Якщо всі отримані послідовності еквівалентні послідовностям в ДТ, то тест коректний.

Якщо вже є відомим повний ДТ, то для вирішення другої задачі достатньо порівняти несправності які розрізняє повний ДТ і розроблений ДТ, інакше потрібно з’ясувати, які несправності однаково впливають на

функціонування схеми. Для цього необхідно промоделювати схему в несправному стані на всіх можливих вхідних сигналах по кожній групі несправностей, що не розрізняє даний ДТ. Якщо для певної групи несправностей, є несправності з різними результатами моделювання, то даний ДТ не є повним.

1.2 Аналіз інструментальних засобів необхідних для розробки програми

При виборі інструментальних засобів для розробки програми необхідно враховувати різні аспекти, найбільш важливим з яких є мова програмування, оскільки саме вона визначає інші доступні засоби. Якщо програма повинна бути оснащена графічним інтерфейсом користувача, то для його розробки потрібна GUI-бібліотека, яка надає готові елементи інтерфейсу, такі як кнопки, поля введення.

Сьогодні існує сотні мов програмування, кожна з яких має свою сферу застосування, а також переваги та недоліки над іншими мовами у даній сфері. Тому, перш за все, мову необхідно вибирати згідно сфери застосування програми, а вже потім порівнювати її з іншими.

Оскільки розроблювана програма призначена для використання на ПК, то вибір мови програмування повинен бути здійснений з огляду на найбільш популярні мови програмування додатків для ПК. Найпопулярнішою на сьогоднішній день є мова C++ [3, 4], на другому місці C# [5], а на третьому Java [6]. Причиною такого рейтингу є те, що розробка настільних додатків не є основною сферою Java, а основною є розробка Android-додатків і бек-енду корпоративних веб-додатків. Невеликі настільні додатки, що не вимогливі до пам’яті і швидкості можна розробляти на Java з використанням GUI бібліотеки JavaFX. C# широко розповсюджений для розробки настільних додатків, але він з його GUI бібліотекою WPF обмежений платформою Windows, через що програє C++. C++ є де-факто стандартом для розробки настільних додатків.

Перш за все, він широко застосовується для розробки ігор та додатків, що вимогливі до пам’яті та швидкості. Крос-платформними GUI бібліотеками для C++ є Qt, GTK+.

C# і Java є чистими об’єктно-орієнтованими мовами програмування, C++ підтримує також процедурне програмування. Для створення глобальних змінних або функцій в Java та C# необхідно створювати фіктивні класи та оголошувати їх з модифікатором static. Головну функцію навіть самої простої програми необхідно поміщати в такий клас.

Для керування пам’яттю в Java і C# використовується збирач сміття, який автоматично вивільняє невикористовувану пам'ять. Це дуже зручно, але за це приходиться платити великим споживанням пам’яті і низькою продуктивністю, оскільки блоки не будуть звільнені до наступного виклику збирача мусора, періодичність роботи якого залежить від використовуваної реалізації. На додачу до цього, процес збирача мусора вимагає додаткових системних ресурсів, що знижує ефективність виконання програм і може призводити до «замороження» програми на кілька секунд.

В C++, навпаки, керування пам’яттю повністю здійснюється програмістом, тобто виділення і звільнення пам’яті виконується вручну. Це з однієї сторони робить програми швидкими, не вимогливими до пам’яті, а з іншої – може призводити до «витоків пам’яті». Виявлення місця «витоку пам’яті» є досить складною задачею і потребує багато часу. Тому зазвичай програми на C++ більш нестабільні і схильні до помилок, але хороші знання, інструментарій і досвід можуть значно зменшити цей ризик.

З іншої сторони, в C++ є засоби, які дозволяють звільняти використовувану пам'ять, як тільки на неї втрачено останній вказівник. Це зменшує продуктивність роботи і збільшує використання пам’яті, але є зручним компромісом, коли це дійсно необхідно. Таким чином, C++ є більш гнучкою мовою з точки зору керування пам’яттю.

В мові Java та C# відсутні деструктори, натомість присутні методи-фіналізатори, які викликаються перед видаленням об’єкту збирачем сміття, але

через невизначеність моменту його виклику, вони не можуть використовуватися для звільнення ресурсів зайнятих об’єктом, що примушує створювати додаткові методи для звільнення ресурсів і викликати їх явно або створювати об’єкти в конструкції try-with-resource в мові Java або using в мові C# для неявного закриття ресурсів. C++ дозволяє використовувати принцип «захвату ресурсів шляхом ініціалізації» (RAII), згідно якого об’єкт отримує ресурс в конструкторі, а звільняє в деструкторі, що в цілому є більш зручним і ефективним способом управління ресурсами.

Мови Java, C# на відміну від C++ не підтримують множинного наслідування класів, але в них було введено інтерфейси, які підтримують множинне наслідування і вирішують проблему ромбовидного наслідування. В C++ відсутня синтаксична конструкція для інтерфейсів, але їх механізм реалізується за допомогою абстрактних класів та множинного наслідування.

Код, написаний на Java і C# компілюється в байт-код – проміжний код, який потім інтерпретується JVM (Java Virtual Machine) та CLR (Common Language Runtime) відповідно. Це звичайно призводить до втрати продуктивності, але має певні переваги. Так, JVM дозволяє запускати програми на будь-якій платформі, для якої існує реалізація JVM. На відміну від JVM, CLR тісно пов’язана з операційною системою Microsoft Windows і головним чином призначена для інтеграції компонентів, розроблених на .NET сумісних мовах.

Код, написаний на C++ компілюється одразу в машинний код, що робить виконання програм на C++ швидшим. Мова C++ є кросплатформеною на рівні мови – один і той самий код компілюється під кожну платформу спеціальним компілятором, призначеним для цієї платформи. Існує велика кількість як кросплатформених, так і платформо-залежних бібліотек, тому кросплатформеність основним чином визначається бібліотеками, що використовуються.

З точки зору швидкості розвитку мов, C# розвивається найшвидше, слабко обмежуючи себе в додаванні нових проблемно-орієнтованих

можливостей. Повільніше розвивається Java – нові можливості додаються більш обережно і повільно. Зовсім повільно розвивається C++ - останнє значне оновлення мови відбулося в 2011 році, а наступне зовсім невелике в 2014.

Популярність мови визначається не лише її потужністю, швидкістю, гнучкістю і т. п., а й кількістю бібліотек, що написані для неї. Адже чим більше бібліотек має мова, тим більше задач можна вирішити з її допомогою. Якщо брати до уваги стандартні бібліотеки, що йдуть в комплекті, то у Java і C# їх на порядок більше і їх можливості значно ширші, ніж у C++. Тому розробляючи програму на C++ необхідно залучати більшу кількість сторонніх бібліотек. У свою чергу, у C++ більша кількість сторонніх бібліотек, аніж у Java і C#. Але багато з них мають дуже різну, часто навіть архаїчну архітектуру, часто не об’єктний, а структурно-процедурний інтерфейс. Крім того, багато C++ бібліотек створює і визначає нові базові типи, заводить свої типи контейнерів, рядків, що вносить додаткову заплутаність і, як наслідок, зменшує зручність використання. Тим не менш, існує і велика кількість добре структурованих бібліотек для C++.

Популярним є крос-платформний фреймворк Qt [7] – який включає в себе всі основні класи, які можуть знадобитися при розробці прикладного програмного забезпечення, починаючи від елементів графічного інтерфейсу і закінчуючи класами для роботи з мережею, базами даних і XML. Добре продуманий набір класів Qt покриває майже всі потреби програміста та надає дуже високий рівень абстракції. За своїми можливостями і багатством бібліотека не поступається .NET Framework або системі класів Java 2 EE.

Qt є повністю об'єктно-орієнтованим, легко розширюваним і підтримує техніку компонентного програмування. Він надає програмісту не тільки зручний набір бібліотек класів, а й певну модель розробки додатків, певний каркас їх структури. Дотримання принципів і правил «гарного стилю програмування на C ++/Qt» істотно знижує частоту таких важко відловлювати помилок в додатках, як «витік пам'яті», необроблені виключення, незакриті файли або не звільнені дескриптори ресурсних об'єктів, чим нерідко

страждають програми, написані «на голому C ++» без використання бібліотеки Qt.

Настільні додатки на Java розробляються з використанням бібліотеки Java SE, яка складається з наступних пакетів:

- java.lang – містить фундаментальні класи і інтерфейси, близько прив'язані до мови і системи під час виконання. Сюда входять кореневі класи, основні виключення, математичні функції, класи багатопоточності, функції безпеки, а також класи, що дозволяють отримати інформацію щодо операційної системи, на якій виконується програма;
- java.io – містить класи для забезпечення файлового введення-виведення інформації, кілька класів абстракції введення-виведення, а також набір класів для обробки інформації, що вводиться;
- java.math – містить класи для обчислень над великими цілими числами і над десятковими дробами довільної точності, а також методи для операцій над числами;
- java.net – містить класи, що дозволяють працювати з мережею, надаючи абстракції для мережевих адрес, з'єднань, реалізацію сокетів і т. д;
- java.text – містить набір класів і утиліт, що дозволяють організувати роботу з різного роду форматами даних, застосовувати наявні або створювати власні шаблони форматування;
- java.util – містить класи колекцій, дати і часу, і різні службові класи;
- java.swing – містить набір графічних компонентів, що дозволяють створювати графічні інтерфейси, що працюють по можливості однаково на всіх платформах;
- java.sql – містить класи для роботи з БД.

Мова C# в якості базових бібліотек використовує бібліотеку класів платформи .NET Framework, що являє собою бібліотеку класів, інтерфейсів і типів значень, які забезпечують доступ до функціональних можливостей системи. В таблиці 1.3 наведено основні простори імен платформи.

Таблиця 1.3 – Основні простори імен платформи

Простір імен	**Опис**
System	містить фундаментальні і базові класи, події і їх обробники, інтерфейси, атрибути і обробку винятків
System.Collections	містить типи, що визначають різні стандартні, спеціальні і універсальні об'єкти колекцій
System.Data	містить класи для доступу до даних з різних джерел і для управління цими даними
System.IO	містить типи, що підтримують введення і виведення, включаючи можливості читання і запису даних в потоках, як синхронно, так і асинхронно, стиснення даних в потоках, створення і використання ізольованих сховищ
System.Net	містить класи, що забезпечують простий інтерфейс програмування для різних мережевих протоколів
System.Numerics	містить числові типи, що доповнюють числові типи-примітиви, такі як Byte, Double і Int32
System.Runtime	містить типи, що підтримують взаємодію з середовищем CLR
System.Text	містить типи для роботи з кодуваннями символів і для управління рядками, дозволяє обробляти текст з використанням регулярних виразів.
System.Threading	містить типи, що забезпечують можливості багатопотокового програмування, спрощують задачу написання паралельного і асинхронного коду.
System.Windows	містить типи, використовувані в додатках Windows Presentation Foundation (WPF), включаючи клієнти анімації, елементи управління інтерфейсу користувача, прив'язку даних і перетворення типів

З огляду мов C++, Java, C# та їх бібліотек видно, що розроблювану програму можна розробити на будь-якій з них. Але була обрана мова C++ та бібліотека Qt з наступних причин:

- кросплатформенність;
- швидкість;
- широкий спектр бібліотек;
- зручна IDE Qt Creator;
- хороша документація бібліотеки Qt;
- економія часу розробки (розробник вже володіє даними інструментами).

1.3 Висновки

В даному розділі виконано аналіз задачі та засобів для її вирішення. В результаті аналізу сформовані вимоги до програми:

- наявність графічного інтерфейсу;
- вхідні дані: опис схеми, несправності, тести;
- вихідні дані: результати тестування за допомогою КТ та ДТ, результати побудови КТ та ДТ.

Враховуючи вимоги, розроблені наступні завдання по розробці програми:

- розробка власного формату для введення даних та його парсеру;
- розробка модуля для моделювання схем;
- розробка модуля для тестування схем;
- розробка модуля для побудови тестів;
- розробка GUI.

За результатами аналізу інструментальних засобів для розробки програми була обрана мова C++ та бібліотека Qt.

2 ПРОЕКТУВАННЯ ПРОГРАМИ

2.1 Діаграма варіантів використання (Use Case)

На рисунку 2.1 приведена Use Case діаграма розроблюваної програми, на якій зображено основні варіанти використання розроблюваної програми:

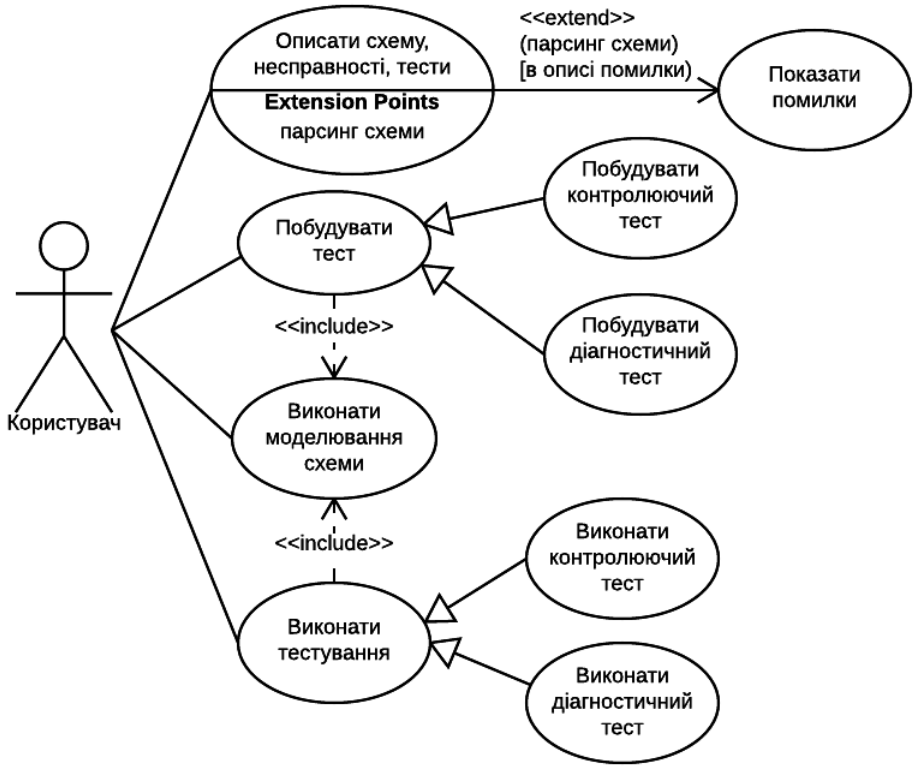


Рисунок 2.1 – Use Case Diagram

Користувач програми має змогу описати схему, несправності та тести. При парсингу схеми, якщо є помилки, то виконується варіант використання «Показати помилки». Варіант використання «Побудувати тест» дає можливість будувати або КТ, або ДТ. Варіант використання «Виконати тестування» дає можливість виконати КТ або ДТ для схеми. Варіанти використання «Побудувати тест» та «Виконати тестування» включають в себе виконання варіанту «Виконати моделювання схеми».

2.2 Основні класи програми та їх взаємодія. Діаграма класів

В програмі необхідно розробити наступні класи та структури даних:

- класи для представлення компонентів схеми;
- клас для представлення схеми і її моделювання;
- клас для виконання тестування;
- клас для побудови тестів;
- класи для парсингу вхідних даних;
- структури даних, що містять результати парсингу, тестування, побудови тестів;
- класи GUI.

Спрощена UML діаграма основних класів та структур, що необхідно розробити і їх взаємодія зображена на рисунку 2.2. На цій діаграмі не зображено класів, що відповідають за GUI та деякі допоміжні класи.

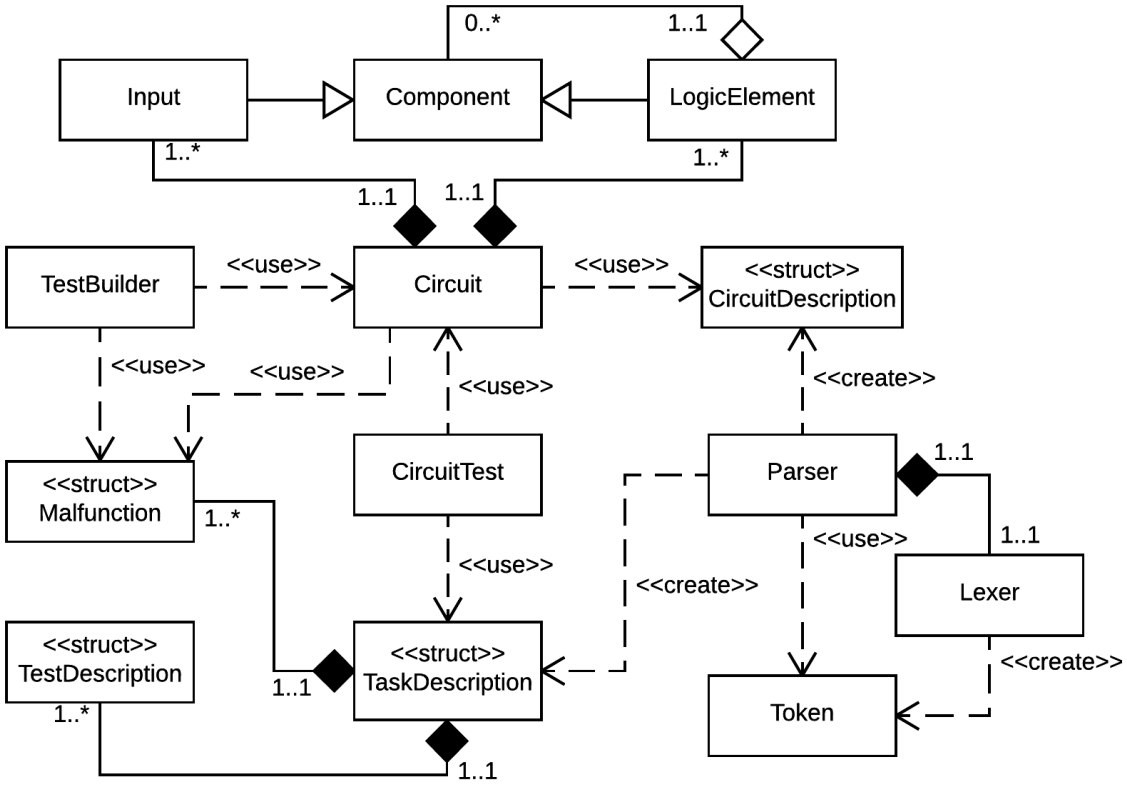


Рисунок 2.2 – Діаграма основних класів і структур та їх взаємодія

Відповідність між назвами класів і структур та їх призначенням описана в таблиці 2.1.

Таблиця 2.1 – Основні класи і структури та їх призначення

Назва класу або структури	Призначення
Component	Слугує базовим класом для компонентів схеми: вхід і логічний елемент. Задає спільні властивості, такі як можливість мати несправності
Input	Представляє вхід схеми, містить значення сигналу
LogicElement	Представляє логічний елемент схеми: містить інформацію про елементи або входи схеми до яких підключений та виконує задану операцію над вхідними сигналами
Circuit	Представляє схему. Створює компоненти схеми (входи, виходи, елементи) та встановлює зв’язки між ними використовуючи структуру CircuitDescription. Дозволяє моделювати схему
CircuitDescription	Містить назви компонентів схеми (входів, виходів, елементів) та інформацію про структуру схеми. Слугує для створення класу Circuit
Token	Представляє лексему при парсингу вхідних даних. Містить інформацію про її тип, позицію у вхідному тексті та її значення
Lexer	Виконує лексичний аналіз вхідних даних. Під час аналізу повертає структуру Token
Parser	Виконує синтаксичний та семантичний аналіз вхідних даних. Результатом роботи є структури CircuitDescription та TaskDescription.

Таблиця 2.1 (закінчення)

1	2
TaskDescription	Містить опис завдання тестування (несправності, тести)
TestDescription	Містить тести (КТ та ДТ)
Malfunction	Описує несправність (назва компонента та значення несправності)
CircuitTest	Виконує тестування схеми Circuit по наданому завданню TaskDescription
TestBuilder	Будує тести для схеми Circuit по наданим несправностям Malfunction

2.3 Прототип GUI

Головне вікно програми зображене на рисунку 2.3.

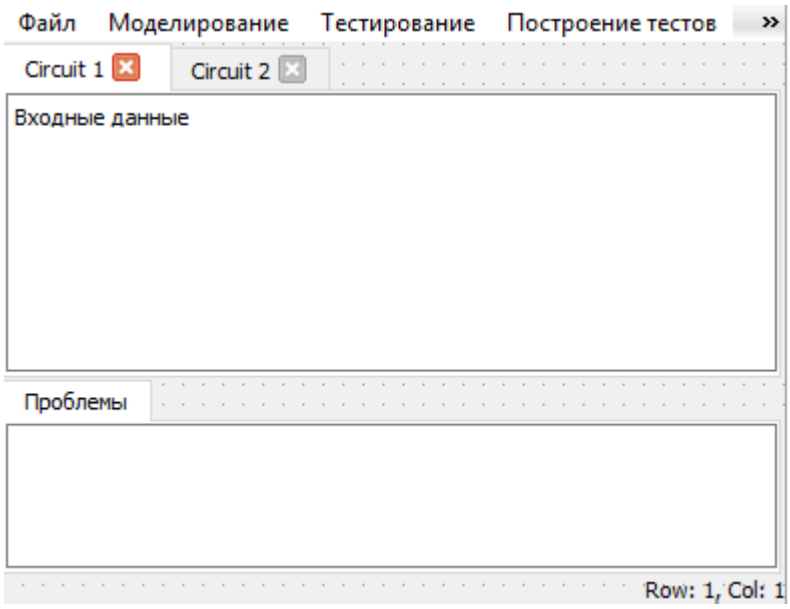


Рисунок 2.3 – Головне вікно програми

В центрі вікна буде розташована текстова область для вхідних даних. За рахунок наявності вкладок можна буде переключатись між різними вхідними даними. Нижче розташовано вікно («Проблемы»), що буде відображати

повідомлення про помилки у вхідних даних. В нижньому правому кутку відображатиметься поточна позиція курсору в текстовій області.

Меню головного вікна міститиме наступні пункти та підпункти:

- «Файл»
  - «Новый»
  - «Открыть»
- «Моделирование»
  - «Моделирование исправной схемы»
- «Тестирование»
  - «Контроль»
  - «Диагностика»
- «Построение тестов»
  - «Для контроля»
  - «Для диагностики»

Результати моделювання справної схеми будуть відображатися у вигляді таблиці так як це зображено на рисунку 2.4. Тут x1, x2, x3 входи схеми, y – вихід схеми, а в клітинках позначені їх значення.

	x1	x2	x3	y
1	0	0	0	1
2	0	0	1	1
3	0	1	0	0
4	0	1	1	0
5	1	0	0	0
6	1	0	1	0
7	1	1	0	0
8	1	1	1	0

Рисунок 2.4 – Таблиця результатів моделювання справної схеми

Результати контролю схеми (виконання КТ) будуть відображатися у вікні, макет якого зображено на рисунку 2.5. Назви стовпців таблиці відповідають назвам виходів схеми. Назви рядків відповідають типам несправностей схеми

(перший рядок – справна схема). У клітинках таблиці відображатимуться значення виходів схеми, отримані в результаті моделювання схеми, в тій послідовності, в якій подавались вхідні сигнали. Нижче відображатиметься перелік несправностей, які не було виявлено. Також, їх буде підсвічено червоним кольором в таблиці. Внизу вікна відображатиметься статистика по кількості виявлених несправностей, що загалом дасть змогу зробити висновок про ефективність КТ.

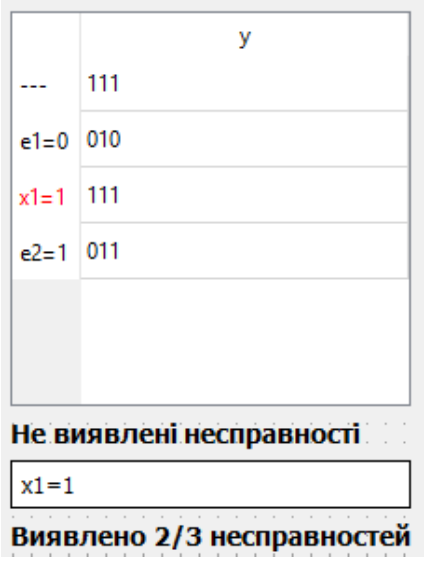


Рисунок 2.5 – Макет вікна, що відображатиме результати контролю

На рисунку 2.6 зображено макет вікна для відображення результатів діагностичного тесту. В клітинках стовпців, назви яких в кінці містять символ «'», відображена очікувана послідовність значень виходів схеми. В клітинках стовпців, назви яких не містять цього символу відображена послідовність значень вихідних сигналів, отримана в результаті моделювання схеми, яка відповідає послідовності подання вхідних сигналів. Нижче відображатимуться невірно визначені несправності, тобто ті, результати моделювання яких не рівні очікуваним результатам. Вони також підсвічуватимуться червоним кольором у таблиці. Внизу вікна відображатиметься статистика по кількості вірно діагностованих несправностей, що дасть смогу оцінити коректність і ефективність тесту.

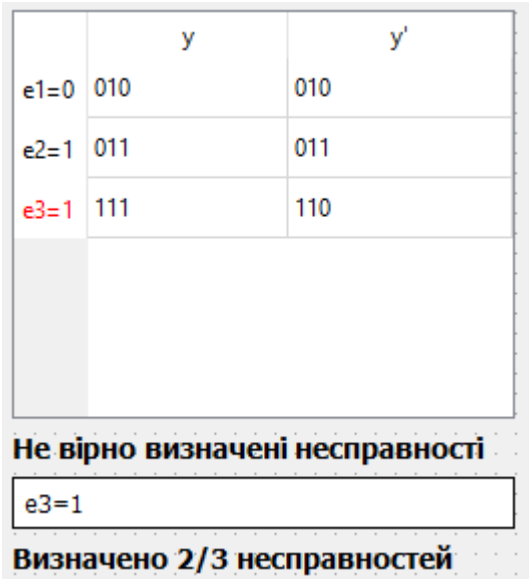


Рисунок 2.6 – Макет вікна, що відображатиме результати діагностики

Макет вікна, що відображатиме результати побудови КТ зображено на рисунку 2.7.

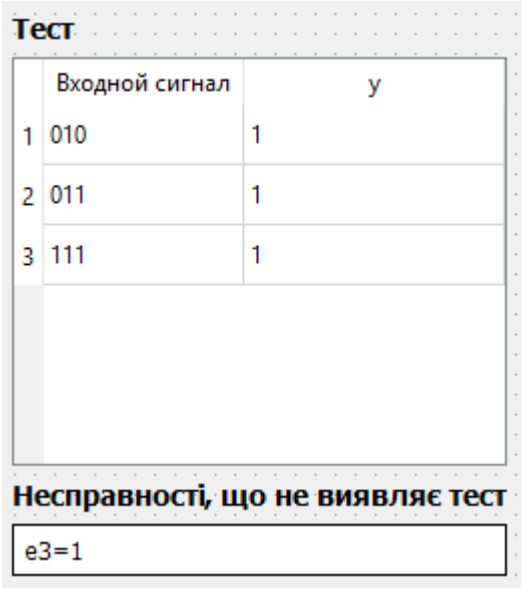


Рисунок 2.7 – Макет вікна, що відображатиме побудований КТ

В першому стовпці будуть відображатися вхідні сигнали, які необхідно подавати на схему, у всіх наступних – очікувана реакція виходу в справній схемі на цей сигнал. Внизу вікна відображатимуться несправності, які не виявлятиме тест, тобто ті несправності, які не впливатимуть на функціонування

схеми.

Результати побудованого ДТ, відображатимуться так як зображено на рисунку 2.8.

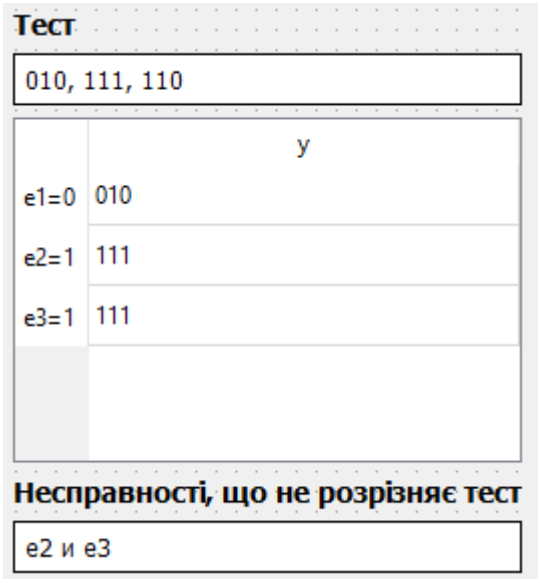


Рисунок 2.8 – Макет вікна, що відображатиме побудований ДТ

В верхній частині вікна відображатиметься побудований тест – послідовність вхідних сигналів, які необхідно подавати на входи схеми в описаному порядку. В клітинках таблиці відображатиметься послідовність значень вихідних сигналів, які очікується отримати при наявності в схемі несправності, тип якої зазначений в назві рядка. В нижній частині вікна відображатимуться несправності, які не розрізняє тест, тобто ті, які однаково впливатимуть на функціонування схеми.

2.4 Розробка формату вхідних даних

Вхідними даними програми буде опис схеми та опис завдання. Опис схеми повинен містити назви входів схеми, назви виходів схеми, назви елементів схеми та їх логічний тип, опис з’єднань елементів або виходів схеми з іншими елементами або входами схеми. Опис завдання повинен містити опис несправностей в схемі та опис тестів. Отже, формат вхідних даних повинен

бути розроблений виходячи з вищенаведених вимог.

Синтаксис розроблюваного формату вхідних даних у вигляді EBNF приведений в Додатку А.1. Нотація використовуваної EBNF описана в [8]. Синтаксичні діаграми розроблюваного формату приведені в Додатку А.2.

2.5 Висновки

В даному розділі за допомогою діаграми варіантів використання (Use Case) була спроектована взаємодія користувача з розроблюваною програмою. Розроблено перелік основних класів, що необхідно розробити, описано їх призначення, та за допомогою діаграми класів було відображено їх взаємодію. Розроблено прототип GUI та описано зміст вікон з результатами моделювання, тестування, побудови тестів. Розроблено формат вхідних даних, а його опис наведено у вигляді EBNF та синтаксичних діаграм.

.

3 РОЗРОБКА ПРОГРАМИ

3.1 Розробка класів компонентів схеми

UML діаграма розроблених класів компонентів зображена на рисунку 3.1.

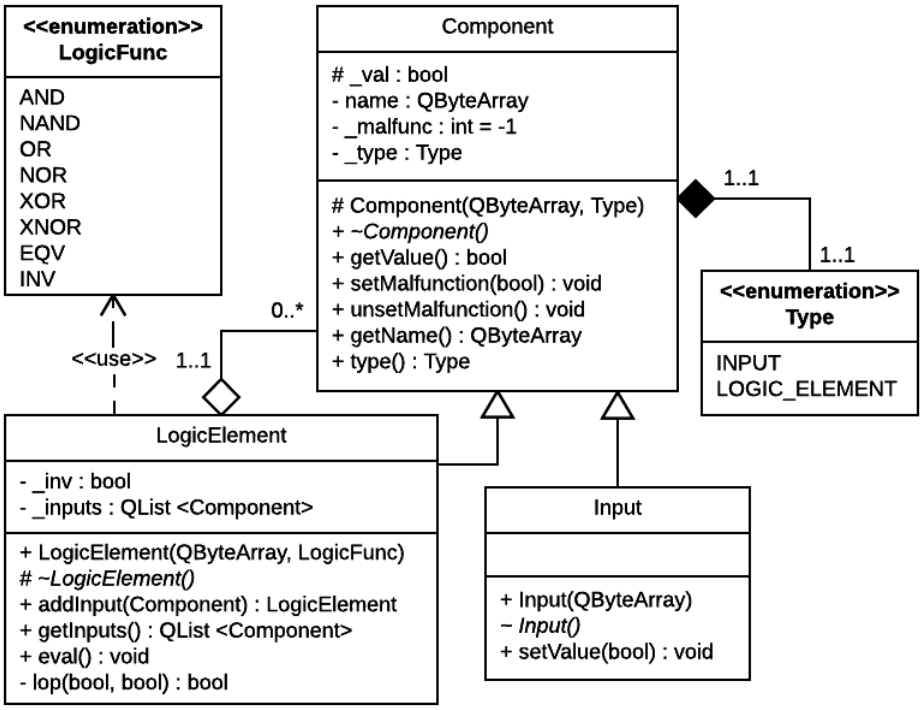


Рисунок 3.1 – UML діаграма класів компонентів схеми

Базовий клас Component містить поле _val, яке зберігає значення сигналу, що реалізує компонент в справному стані. Поле _malfunc зберігає значення несправності компонента (0 або 1) або -1 якщо несправність відсутня. Несправність встановлюється та вилучається за допомогою методів setMalfunction та unsetMalfunction, код яких приведено нижче:

```
    void setMalfunction(bool malfunction) { _malfunc = malfunction; }
    void unsetMalfunction() { _malfunc = -1; }
```

Ключовою функцією є getValue, яка повертає значення сигналу, що реалізує компонент, якщо він у справному стані, а інакше значення

несправності:

```
    bool getValue() { return _malfunc != -1 ? _malfunc : _val; }
```

Клас Input додає метод setValue, що дозволяє задати значення сигналу для входу схеми.

Клас LogicElement представляє логічний елемент схеми, який реалізує певну логічну функцію. Логічні функції описані в LogicFunc. В списку _inputs зберігаються вказівники на компоненти схеми, до яких підключений даний елемент. Вказівник на функцію lop вказує на функцію, що реалізує операцію, яку необхідно виконати над вхідними сигналами елементу. Він може вказувати на одну з наступних функцій, якщо елемент має два і більше входів:

```
bool _and(bool a, bool b) { return a && b; }
bool _or(bool a, bool b) { return a || b; }
bool _xor(bool a, bool b) { return  a != b; }
```

Поле _inv визначає чи є вихід елемента інверсним, тобто чи потрібно інвертувати сигнал.

В конструкторі класу, в залежності від логічної функції, що реалізує елемент, визначається логічна операція над входами та інверсія виходу. Лістинг конструктора приведено нижче:

```
LogicElement::LogicElement(const QByteArray &name, LogicFunc lf) :
Component(name, LOGIC_ELEMENT)
{
    if (lf == NAND || lf == NOR || lf == XNOR || lf == NOT)
        _inv = true;
    else _inv = false;

    if (lf == AND || lf == NAND)
        lop = &_and;
    else if (lf == OR || lf == NOR)
        lop = &_or;
    else if (lf == XOR || lf == XNOR)
        lop = &_xor;
    else
        lop = nullptr;
}
```

Якщо елемент реалізує логічну функцію EQV або NOT, то цей елемент має один вхід і не виконує логічної операції над вхідними сигналами. В цьому випадку, вказівнику lop привласнюється nullptr.

Функція eval виконує обчислення сигналу, що реалізує елемент в справному стані. ЇЇ лістинг наведено нижче:

```
void LogicElement::eval()
{
    bool val = _inputs.first()->getValue();
    if (lop != nullptr)
        for (int i = 1; i < _inputs.size(); i++)
            val = lop(val, _inputs.at(i)->getValue());

    _val = val != _inv;
}
```

В локальній змінній val записується значення сигналу одного з компонентів, до якого підключений даний елемент. Якщо елемент реалізує логічну операцію над вхідними сигналам, то обчислюється значення цієї операції. Після цього, якщо вихід елемента інверсний, то значення інвертується.

3.2 Розробка класу схеми

Лістинг визначення класу схеми наведено нижче:

```
class Circuit
{
public:
    Circuit(const CircuitDescription &cd);
    QMap <QByteArray, QByteArray> simulate(const Malfunction &malfunc);
    QMap <QByteArray, QByteArray> simulate(const QList <QByteArray>
&inputsValues, const Malfunction &malfunc);
    QMap <QByteArray, QByteArray> simulate();
    QMap <QByteArray, QByteArray> simulate(const QList<QByteArray>
&inputsValues);
    ~Circuit();
private:
    QList <Input*> inputs;
    QList <LogicElement*> elements;
    QMap <QByteArray, Component*> outputs;
    QHash <QByteArray, Component*> components;

    void setInputsValue(const QByteArray &inputsValue);
    void setInputsValue(int inputsValue);
    void eval();
    void doRanging();
};
```

Об’єкт класу схеми створюється з структури CircuitDescription, отриманої в результаті парсингу вхідних даних. Об’єкти компонентів схеми – Input та LogicElements створюються динамічно та вказівники на них зберігаються в списках inputs та elements відповідно. Також, заради зручності внесення

несправностей, вказівники на них зберігаються в хеш-таблиці QHash, де ключем є назва компонента. Словник outputs містить назву виходу та вказівник на компонент схеми, що зв’язаний з виходом схеми.

3.2.1 Алгоритм моделювання схеми

Перед розробкою класів схеми та класів компонентів, було розглянуто два алгоритми для моделювання схеми:

1) Ітеративний алгоритм;
2) Рекурсивний алгоритм.

В ітеративному алгоритмі весь розрахунок схеми зводиться до послідовного обчислення значень виходів елементів схеми. Щоб обчислення були вірними, необхідно попередньо ранжувати елементи схеми – розмістити елементи в порядку появи на їх входах всіх вхідних сигналів. В ітеративному алгоритмі кожен елемент розраховується лише один раз, але необхідно зберігати в пам’яті значення виходу кожного елементу. Якщо в схемі є несправний елемент, то елементи, до яких він підключений, все рівно будуть розраховані, навіть якщо до їх виходів не підключено жодних інших елементів, незважаючи на те, що схему можна було б розрахувати й без розрахунку значень виходів цих елементів.

В рекурсивному алгоритмі виходи схеми розраховуються за рахунок виклику рекурсивної процедури, яка в свою чергу, викликає рекурсивну процедуру для розрахунку виходів тих елементів, до яких підключений вихідний елемент. Рекурсія продовжується до входів схеми, які вернуть значення вхідних сигналів. Цей спосіб є більш повільним і надлишковим, адже елемент буде розрахований рівно стільки раз, скільки елементів підключено до його виходу. Крім того, якщо схема має більше одного виходу, то для кожного виходу необхідно викликати процедуру рекурсивного розрахунку. З іншої сторони, не потрібно зберігати в пам’яті розраховане значення виходу кожного елементу. А також, якщо в схемі є несправний елемент, то рекурсія завершується на цьому елементі і наступні елементи не розраховуються

(справедливо лише для тих елементів, які мають один вихід і підключені ним до несправного елементу, або до елементів, що підключені до нього). Крім того, реалізація рекурсивного розрахунку є компактнішою, адже не потрібно ранжувати елементи схеми.

З огляду на переваги та недоліки обох алгоритмів, а також зважаючи на те, що з точки зору моделювання більш правильним є ітераційним алгоритм, він і був обраним для моделювання.

Клас схеми містить чотири перевантажені методи для моделювання схеми, які повертають словник, ключем якого є назва виходу, а значенням – послідовність вихідних сигналів даного виходу, отримана в результаті моделювання. Методи simulate() та simulate(Malfunction) моделюють справну та несправну схему для заданої несправності відповідно на всіх можливих комбінаціях вхідних сигналів. Методи simulate(QList <QByteArray>) та simulate(QList <QByteArray>, Malfunction) моделюють справну та несправну схему для заданої несправності відповідно на заданих в списку значення вхідних сигналів. Загальний алгоритм розроблених методів моделювання наступний:

1) Якщо схема моделюється з несправністю, то задається несправність для компонента схеми.

2) За допомогою методу setInputsValue задаються значення входів схеми.

3) Виконується моделювання (розрахунок) схеми за допомогою методу eval, який послідовно викликає метод eval для кожного елемента схеми.

4) З кожного компоненту схеми, асоційованого з виходом схеми, за допомогою методу getValue витягується значення сигналу його виходу. Отримані значення записуються в словник з результатами.

5) Якщо схема була промодельована не на всіх значеннях вхідних сигналів, повернення до пункту 2.

6) Якщо схема моделювалася з несправністю, несправність усувається;

7) Результати моделювання повертаються у вигляді словника (ключ – назва виходу, значення – послідовність значень даного виходу).

3.2.2 Алгоритм ранжування елементів схеми

Алгоритм ранжування елементів схеми реалізований в методі doRanging і викликається в конструкторі класу.

Для ранжування елементів використовуються допоміжні структури:

1) Множина currentElements – містить елементи, ранг яких, на даному етапі ранжування, невідомий, але є відомим як мінімум один вхідний сигнал.

2) Список rangedElements – містить елементи, ранги яких, на даному етапі ранжування, відомі. Елементи в списку розміщені порядку появи на їх входах всіх вхідних сигналів.

Алгоритм роботи методу можна описати наступним чином:

1) Для кожного елемента схеми перебираються компоненти, з якими з'єднані його входи. При цьому, якщо його вхід з'єднаний з входом схеми, то елемент додається в множину currentElements. Інакше, його вхід з'єднаний з виходом елемента схеми, і він додається в список виходів елемента, з яким він з'єднаний.

2) Для кожного елемента з множини currentElements перебираються компоненти, з якими з'єднані його входи. При цьому, якщо його вхід не з'єднаний з входом схеми, то перевіряється, чи елемент схеми, з виходом якого з'єднаний вхід даного елемента, є в списку rangedElements. Якщо його немає в цьому списку, то розглядається наступний елемент множини. Якщо всі елементи, до виходів яких під'єднані входи даного елемента, є в списку rangedElements, то даний елемент додається в кінець списку rangedElements і видаляється з множини currentElements, а всі елементи, входи яких з'єднані з виходом даного елемента, додаються в множину currentElements.

3) Крок 2 повторяється до тих пір, поки множина currentElements не буде пустою. Якщо за 1000 ітерацій цього не буде досягнуто, робиться висновок, що схема не комбінаційна і кидається виключення.

4) Список елементів схеми, в якому елементи розташовані в довільному порядку, заміняється на список rangedElements.

3.3 Розробка класу для тестування схеми

Лістинг визначення класу для тестування схеми наведено нижче:

```
class CircuitTest
{
public:
    static QMap<QByteArray, QMap<QByteArray, ControlTestResult> >
    doControlTest(Circuit &c, const QMap<QByteArray, TaskDescription> &t);
    static QMap <QByteArray, QMap <QByteArray, DiagnosticTestResult> >
    doDiagnosticTest(Circuit &c, const QMap<QByteArray, TaskDescription> &t);
};
```

Клас містить два статичні методи doControlTest та doDiagnosticTest, які відповідно виконують КТ та ДТ. Першим параметром передається схема, що підлягає тестуванню, а другим словник, ключем якого є назва завдання, а значенням – опис завдання тестування (структура TaskDescription). В структурі TaskDescription, описані несправності схеми і КТ та ДТ, що потрібно виконати. Лістинг визначення TaskDescription наведено нижче:

```
struct TaskDescription {
    QList <Malfunction> malfunctions;
    QMap <QByteArray, TestDescription> tests;
};
```

Список malfunctions містить несправності схеми, а словник tests, ключем якого є назва тесту, – опис тестів (структура TestDescription).

Лістинг визначення Malfunction та TestDescription наведено нижче:

```
struct TestDescription {
    …
    QList <QByteArray> test;
    QMap <QByteArray, QList <QByteArray> > diagnostic;
};

struct Malfunction {
    …
    QByteArray name;
    bool val;
    …
};
```

В структурі Malfunction поле name визначає назву компонента схеми з несправністю, а val – значення константної несправності. Список test в структурі TestDescription містить значення вхідних сигналів, що є тестовими

для схеми. Якщо тест діагностичний, то в словнику diagnostic, ключами якого є назви виходів схем, описуються очікувані послідовності значень вихідних сигналів для всіх несправностей в списку malfunctions структури TaskDescription.

Результати контролю і діагностики схеми повертаються у вигляді словника, ключами якого є назви завдань, а значеннями – словники, ключами яких є назви тестів, а значеннями – результати тестування у вигляді структур ControlTestResult та DiagnosticTestResult відповідно. Лістинг визначення цих структур наведено нижче:

```
struct ControlTestResult {
    QMap <QByteArray, QByteArray> correct;
    QMap <Malfunction, QMap <QByteArray, QByteArray> > incorrect;
    QList <Malfunction> undetected;
};

struct DiagnosticTestResult {
    QMap <Malfunction, QMap <QByteArray, QByteArray> > simulated;
    QMap <Malfunction, QMap <QByteArray, QByteArray> > expected;
    QList <Malfunction> undefined;
};
```

В структурі ControlTestResult поле correct містить результати моделювання справної схеми, поле incorrect містить словник, ключами якого є несправності, а значеннями – результати моделювання схеми з даними несправностями, поле undetected містить список не виявлених несправностей, тобто тих, результати моделювання схеми з якими, не відрізняються від результатів моделювання справної схеми.

В структурі DiagnosticTestResult поле simulated містить словник, ключами якого є несправності, а значеннями – результати моделювання схеми з даними несправностями, поле expected містить словник, ключами якого є несправності, а значеннями – очікувані результати моделювання схеми з даними несправностями, поле undefined містить список несправностей, що діагностовані невірно – результати моделювання схеми з несправністю відрізняються від очікуваних результатів моделювання (ДТ не діагностує дану несправність).

3.4 Розробка класу для побудови тестів

Лістинг визначення класу для побудови тестів наведено нижче:

```
class TestBuilder
{
public:
    static ControlTestInfo buildControlTest(Circuit &circuit, const QList
<Malfunction> &malfuncs);
    static DiagnosticTestInfo buildDiagnosticTest(Circuit &circuit, const
QList <Malfunction> &malfuncs);
…
};
```

Клас містить два статичні методи buildControlTest та buildDiagnosticTest, які будують КТ та ДТ відповідно. Першим параметром передається схема, а другим – перелік несправностей схеми, для яких будується тест. Алгоритми генерації тестів, що реалізовані в цих методах, описані в «Методичних вказівках до лабораторних робіт з КіДІС», а також в [1, 2].

Лістинг визначення структур, які повертають дані методи, наведено нижче:

```
struct ControlTestInfo {
    QList <QByteArray> test;
    QMap <QByteArray, QByteArray> outputs;
    QList <Malfunction> undetected;
};

struct DiagnosticTestInfo {
    QList <QByteArray> test;
    QMap <Malfunction, QMap <QByteArray, QByteArray> > outputs;
    QList <QPair <Malfunction, Malfunction> > indistinguishable;
};
```

Поле test обох структур містить побудовані тестові набори вхідних значень. В структурі ControlTestInfo поле outputs містить результати моделювання справної схеми, а в структурі DiagnosticTestInfo – словник, значеннями якого є результати моделювання несправної схеми для несправностей, що відповідають ключам словника. Поле undetected структури ControlTestInfo містить несправності, що не виявляє тест, а поле indistinguishable структури DiagnosticTestInfo містить пари несправностей, що не розрізняє тест.

3.5 Розробка класів для парсингу формату вхідних даних

Для парсингу формату вхідних даних, використовуючи принципи [9, 10], був розроблений клас Parser, який:

- виконує синтаксичний аналіз вхідних даних згідно правил синтаксису, наведених в Додатку А.1;
- виконує семантичний аналіз вхідних даних.

Результатом роботи парсера є структури CircuitDescription (опис схеми) і TaskDescription (опис завдання). Структура CircuitDescription використовується для створення об’єкта класу схеми (Circuit). В ній описана інформація про назви входів, виходів, елементів та їх типів і зв’язки між ними. Структура TaskDescription вже була описана в підрозділі 3.3.

Алгоритм роботи парсера досить прямолінійний: він послідовно, в порядку слідування ключових нетерміналів (INPUTS, OUTPUTS, ELEMENTS і т. д.), викликає методи для їх парсингу. Якщо при парсингу виникає синтаксична або семантична помилка, метод зупиняє свою роботу кидаючи виключення.

У своїй роботі парсер використовує клас Lexer, який аналізує послідовність символів в тексті з вхідними даними і повертає парсеру класифіковані лексеми – токени, у вигляді структур Token. Лістинг визначення структури Token наведено нижче:

```
struct Token {
    int row;
    int col;
    QByteArray value;
    enum Type {SECTION, ELEMENT_TYPE, DELIMITER, IDENTIFIER, BINARY_CONSTANT,
END} type;
…
};
```

В полях row, col міститься інформація про позицію лексеми в тексті, поле value містить значення лексеми, поле type містить тип лексеми. В таблиці 3.1 наведена відповідність між типом лексеми і її можливим значенням. При лексичному аналізі за цією таблицею з тексту виділяється лексема та

ідентифікується її тип. Якщо ідентифікувати тип лексеми не вдається, то кидається виключення з описом помилки.

Таблиця 3.1 – Відповідність між типом лексеми і можливим значенням

Тип	Значення
SECTION	INPUTS, OUTPUTS, ELEMENTS, LINKS, TASK, MALFUNCTIONS, TESTS, DIAGNOSTIC
ELEMENT_TYPE	AND, NAND, OR, NOR, XOR, XNOR, EQV, NOT
DELIMITER	Один з символів всередині лапок «(),;:=[]»
IDENTIFIER	Задовольняє регулярному виразу [A-Za-z][\w]*
BINARY_CONSTANT	Задовольняє регулярному виразу [01]+
END	Службовий тип, без значення

3.6 Висновки

В даному розділі було розроблено та описано основні моменти і прийняті рішення щодо розробки. Було проаналізовано два підходи до розрахунку КС та вибрано ітераційний алгоритм. Для правильної роботи ітераційного алгоритму потрібно попередньо ранжувати елементи схеми, тому було розроблено алгоритм ранжування. Для зручної взаємодії з результатами тестування та побудови тестів було розроблено відповідні структури, у які ці результати записуються. Ці структури, в цілому, містять дублюючу інформацію. В подальшому їх можна буде оптимізувати з точки зору споживаної пам’яті, але це призведе до менш зручної взаємодії з цими структурами. Для представлення результатів парсингу вхідних даних також були розроблені відповідні структури. Завдяки виконанню синтаксичного та семантичного аналізу вхідних даних, дані цих структур будуть коректними та придатними для створення об’єкту схеми та виконання тестування чи побудови тестів, а у разі виникнення помилок буде виведено місце та причина помилки, що загалом спрощує її знаходження та виправлення.

4 ОПИС ПРОГРАМИ

4.1 Алгоритм взаємодії користувача з програмою

Для ілюстрації алгоритму взаємодії користувача з програмою була обрана схема, що зображена на рисунку 4.1.

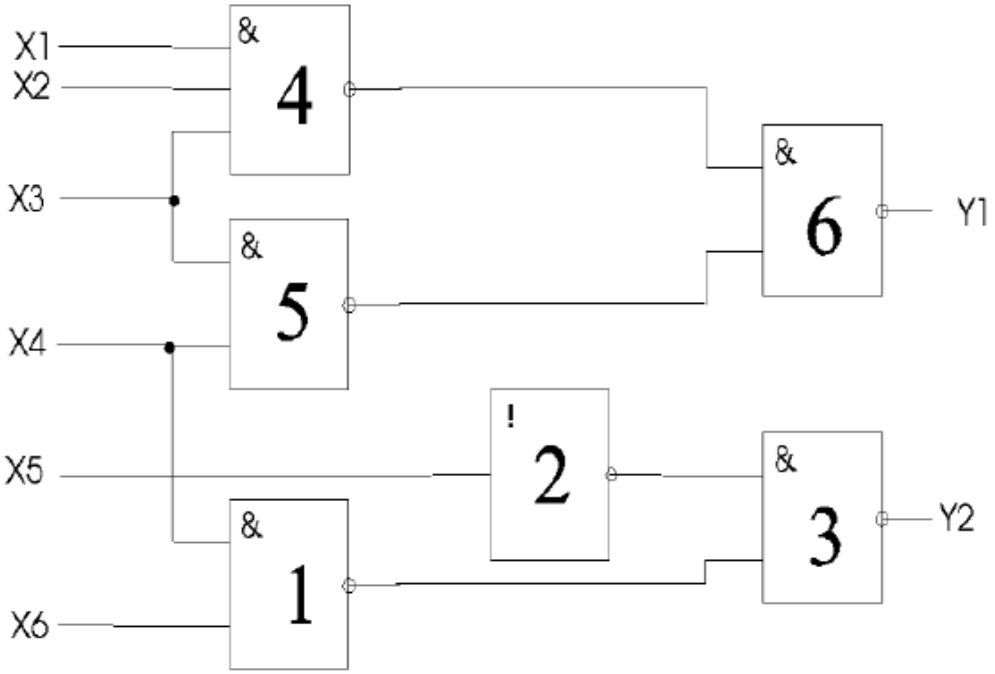


Рисунок 4.1

В таблиці 4.1 наведено випробуваний КТ, що був побудований для несправностей: 10, 20, 30, 40, 50, 60, x10, x20, 11, 21, 31, 41, 51, 61, x11, x21.

Таблиця 4.1 – КТ для схеми на рис. 4.1

Входи						Виходи	
X1	X2	X3	X4	X5	X6	Y1	Y2
1	1	1	0	0	0	1	0
0	1	1	0	0	0	0	0
1	0	1	0	0	0	0	0

В таблиці 4.2 наведено випробуваний ДТ, що був побудований для несправностей: 11, 31, 41, 51, 61, x11.

Таблиця 4.2 – ДТ для схеми на рис. 4.1

Входи						Несправності											
X1	X2	X3	X4	X5	X6	11		31		41		51		61		x11	
						y1	y2	y1	y2	y1	y2	y1	y2	y1	y2	y1	y2
0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0
0	0	1	1	0	0	1	0	1	1	1	0	0	0	1	0	1	0

Після запуску програми, в текстовій області користувач описує схему, несправності та тести згідно формату наведеного в Додатку А. На рисунку 4.2 зображено опис схеми, несправностей та тестів для розглядуваного нами випадку.

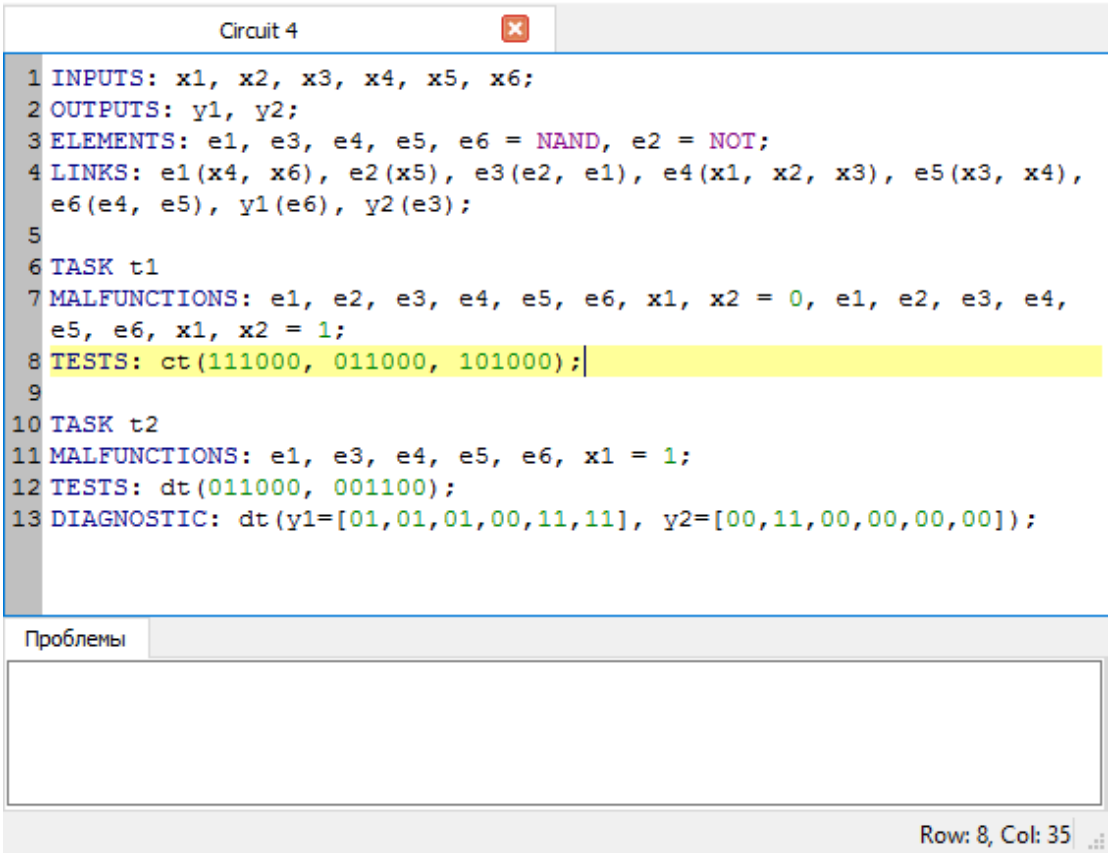


Рисунок 4.2 – Приклад вхідних даних

Після введення вхідних даних, користувач може виконати тестування схеми. Для цього необхідно перейти в пункт меню «Тестирование» і вибрати «Контроль» або «Діагностика» для виконання КТ або ДТ відповідно. Результати контролю та діагностики для розглядуваного випадку зображені на рисунку 4.3.

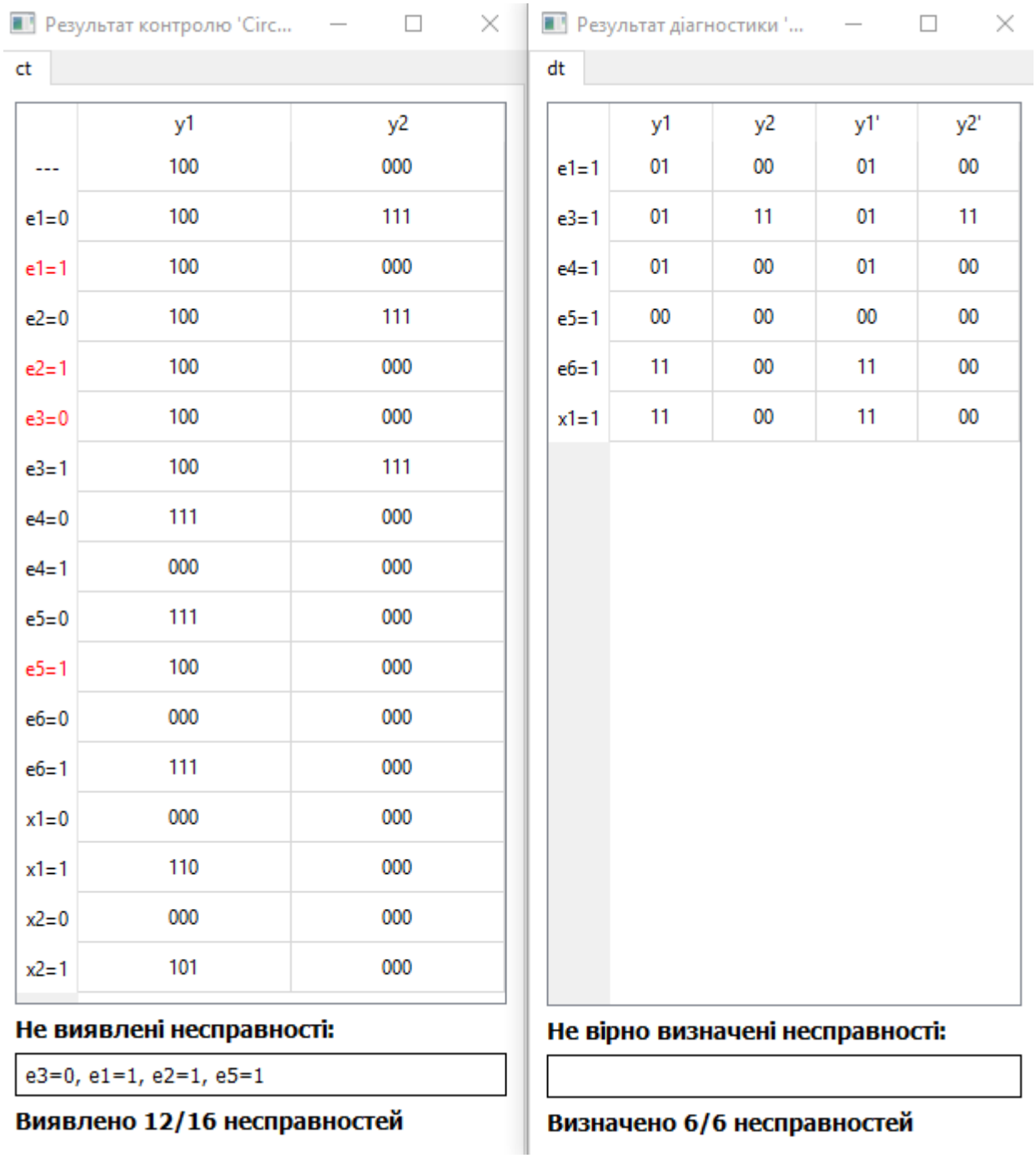


Рисунок 4.3 – Результати контролю та діагностики

Для побудови КТ або ДТ з допомогою програми, необхідно перейти в пункт меню «Построение тестов» і вибрати, відповідно, «Для контроля» або «Для диагностики». Результати побудови тестів зображені на рисунку 4.4.

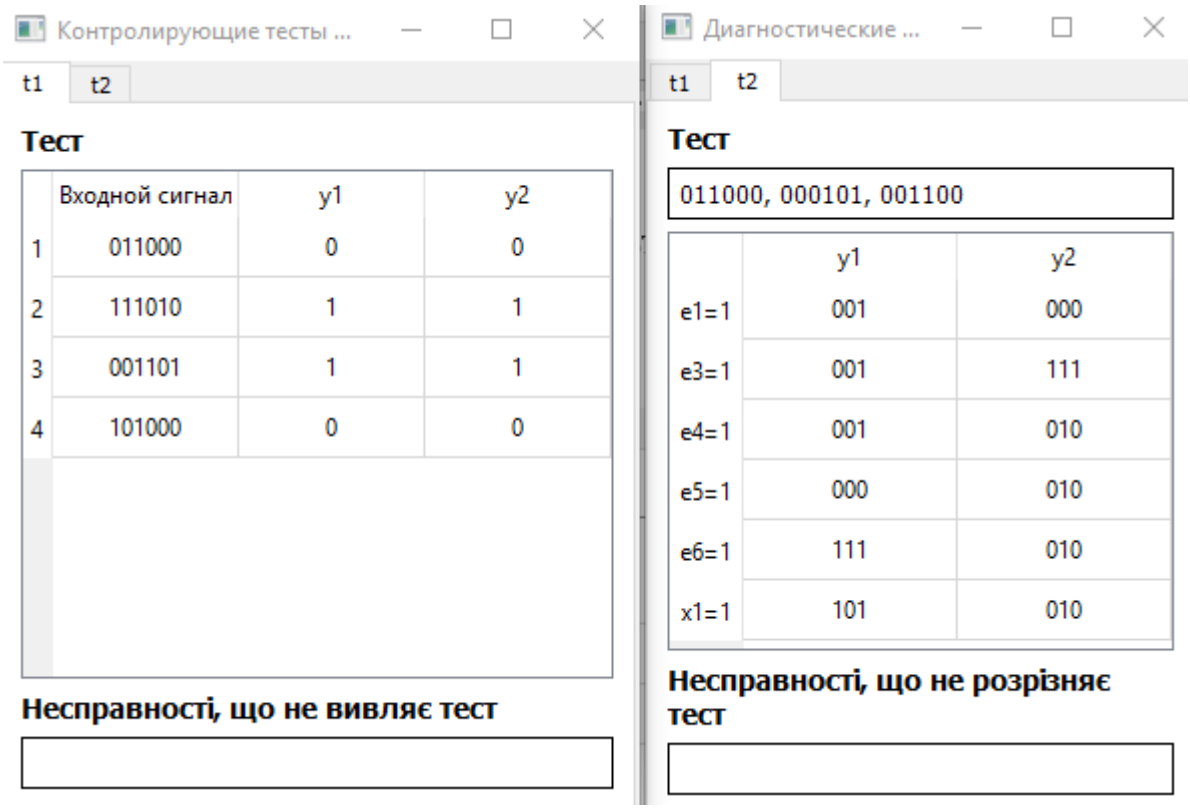


Рисунок 4.4 – Результати побудови КТ та ДТ

Уся інформація по ефективності побудованих тестів вже наведена у вікнах з результатами, та при бажанні їх також можна випробувати. Змінимо опис завдання для схеми, так, як зображено на рисунку 4.5.

```
TASK t1
MALFUNCTIONS: e1, e2, e3, e4, e5, e6, x1, x2 = 0, e1, e2, e3,
 e4, e5, e6, x1, x2 = 1;
TESTS: ct(111000, 011000, 101000), bct(011000, 111010, 001101,
 101000);

TASK t2
MALFUNCTIONS: e1, e3, e4, e5, e6, x1 = 1;
TESTS: dt(011000, 001100), bdt(011000, 000101, 001100);
DIAGNOSTIC: dt(y1=[01,01,01,00,11,11], y2=[00,11,00,00,00,00]),
bdt(y1=[001, 001, 001, 000, 111, 101], y2=[000, 111, 010, 010,
 010, 010]);
```

Рисунок 4.5 – Завдання з побудованими тестами

Результати випробування тестів, побудованих за допомогою програми, зображені на рисунку 4.6.

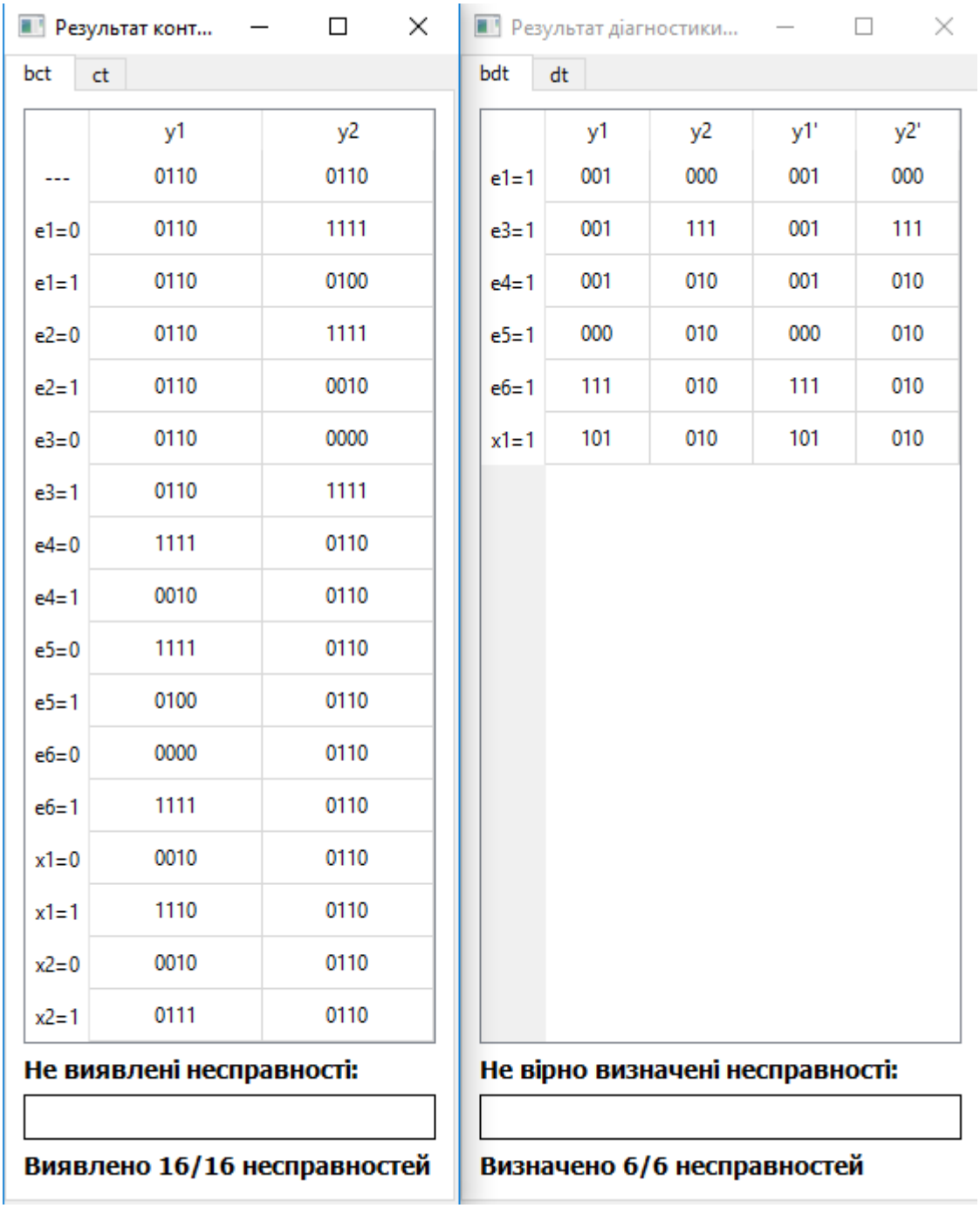
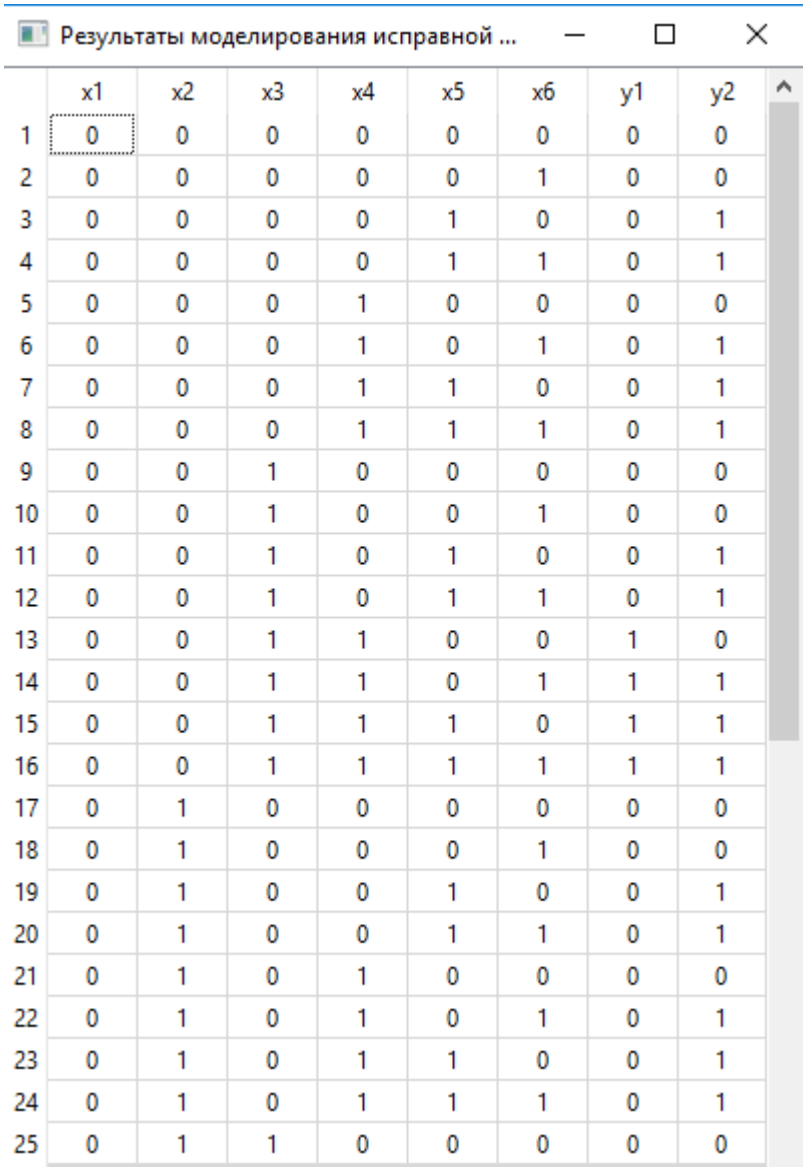


Рисунок 4.6 – Результати випробування тестів, отриманих програмно

Порівнюючи випробувані тести з тестами, отриманими програмно, можемо прийти до висновку, що вони не є ефективними, оскільки не виявляють та не розрізняють усіх помилок, на відміну від тестів, отриманих програмно.

При закритті програми або вкладки, у якій знаходиться опис схеми та завдання, користувачу пропонується зберегти опис у файл. Для відкриття файлу з описом схеми та завдання необхідно перейти в пункт меню «Файл» та вибрати «Открыть». Для відкриття нової вкладки необхідно перейти в пункт меню «Файл» та вибрати «Новый».

Для моделювання схеми на всіх можливих комбінаціях вхідних сигналів необхідно перейти в пункт меню «Моделирование» та вибрати «Моделирование исправной схемы». Результати моделювання зображенні на рисунку 4.7.



	x1	x2	x3	x4	x5	x6	y1	y2
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0
3	0	0	0	0	1	0	0	1
4	0	0	0	0	1	1	0	1
5	0	0	0	1	0	0	0	0
6	0	0	0	1	0	1	0	1
7	0	0	0	1	1	0	0	1
8	0	0	0	1	1	1	0	1
9	0	0	1	0	0	0	0	0
10	0	0	1	0	0	1	0	0
11	0	0	1	0	1	0	0	1
12	0	0	1	0	1	1	0	1
13	0	0	1	1	0	0	1	0
14	0	0	1	1	0	1	1	1
15	0	0	1	1	1	0	1	1
16	0	0	1	1	1	1	1	1
17	0	1	0	0	0	0	0	0
18	0	1	0	0	0	1	0	0
19	0	1	0	0	1	0	0	1
20	0	1	0	0	1	1	0	1
21	0	1	0	1	0	0	0	0
22	0	1	0	1	0	1	0	1
23	0	1	0	1	1	0	0	1
24	0	1	0	1	1	1	0	1
25	0	1	1	0	0	0	0	0

Рисунок 4.7 – Результати моделювання справної схеми

4.2 Формат вхідних даних

Синтаксис формату вхідних даних наведений в Додатку А. Приклад вхідних даних зображений на рис. 4.2, 4.5. Опишемо семантику формату вхідних даних.

В розділі INPUTS описуються назви входів схеми, а в розділі OUTPUTS – назви виходів схеми. В розділі ELEMENTS описуються назви елементів схеми та їх тип (AND, NAND, OR, NOR, XOR, XNOR, EQV, NOT). Назви входів, виходів, елементів повинні бути унікальними, інакше програма повідомить про дублювання ідентифікаторів.

В розділі LINKS задаються з’єднання виходів схеми та входів елементів (за межами дужок) з входами схеми та виходами елементів (усередині дужок). Усі входи схеми повинні бути під’єднанні до входів елементів, усі виходи схеми повинні бути під’єднанні до виходів елементів, входи усіх елементів повинні бути під’єднанні до виходів елементів або до входів схеми та виходи усіх елементів повинні бути під’єднанні до входів елементів або до виходів схеми. Якщо хоча б одна умова не виконується, програма виводить помилку з переліком входів, виходів, елементів, що не підключені. Вихід схеми повинен бути з’єднаний лише з одним елементом. Елемент типу EQV або NOT є одновходовим, тому, за входом, він також повинен бути з’єднаний лише з одним елементом. Елементи іншого типу, за входом, повинні бути з’єднанні з двома і більше елементами. Якщо умова кількості з’єднань не виконується, програма виводить помилку з зазначенням місця помилки і її описом.

В розділі TASK описується назва завдання. В розділі MALFUNCTIONS описуються несправності схеми. Якщо та сама несправність описана двічі або описана несправність для неіснуючого компонента схеми, то програма повідомляє про помилку. В розділі TESTS описуються назви тестів та їх тестові набори вхідних сигналів. Якщо в одному з наборів кількість значень сигналів більша або менша за кількість входів схеми, то програма повідомляє про помилку. Також недопустимими є однакові назви тестів.

Якщо тест діагностичний, то в розділі DIAGNOSTIC описується назва тесту з розділу TESTS, що є діагностичним, і послідовність значень вихідних сигналів, отримувана в результаті подання на входи тестових наборів, для переліку несправностей з розділу MALFUNCTIONS. Можливі помилки: тесту з заданою назвою немає в розділі TESTS; не всі входи описані; не існуючі входи описані; кількість послідовностей не рівна кількості несправностей; кількість вихідних сигналів в послідовності не рівна кількості наборів в тесті. Усі помилки виявляються та повідомляються.

4.3 Висновки

В даному розділі був описаний алгоритм роботи користувача з програмою – послідовність дій, необхідна для отримання певного результату. На прикладі було показано, як потрібно описувати схему та завдання (несправності, тести). Було проведено тестування, побудовано тести та виконано моделювання схеми. Також було описано семантику формату та деякі можливі помилки в описі.

5 РЕЗУЛЬТАТИ ТЕСТУВАННЯ

5.1 Схема №5

На рисунку 5.1 зображена схема №5 для якої було розроблено КТ та ДТ, що наведені в таблицях 5.1, 5.2 відповідно.

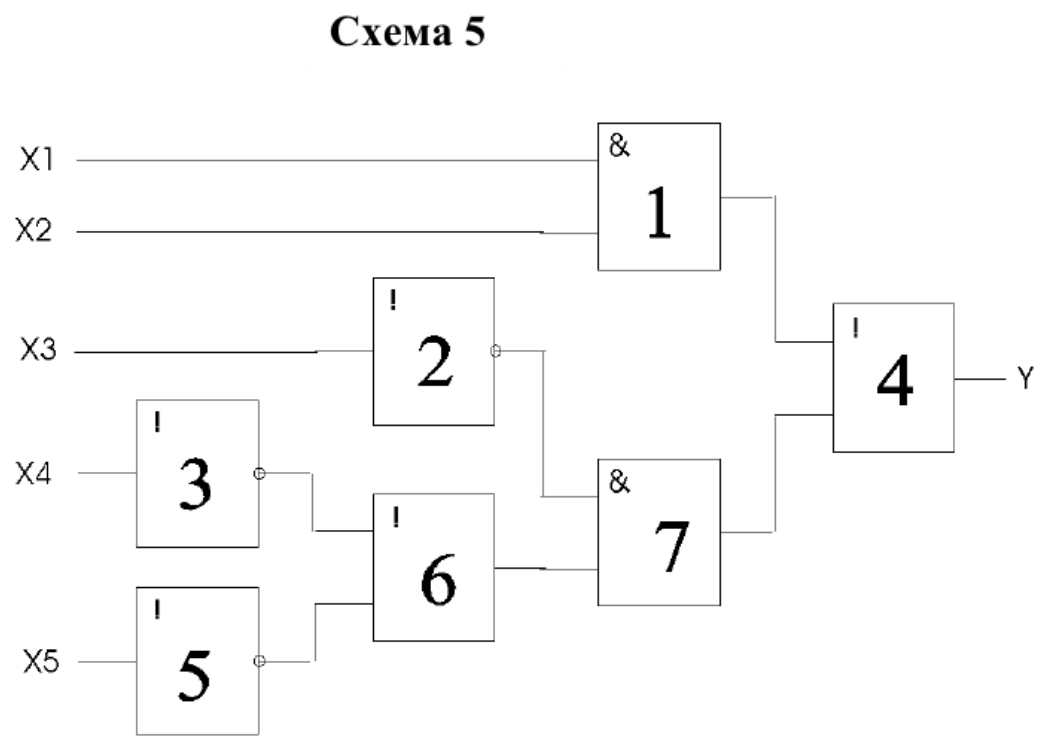


Рисунок 5.1 – КС №5

Таблиця 5.1 – КТ для схеми на рис. 5.1

Несправності: 20, 30, 40, 50, x11, x21, x31, x41, x51, 31, 51, 61, 71					
Входи					Виходи
x1	x2	x3	x4	x5	y
0	0	0	0	1	1
0	0	0	1	0	1
0	1	0	1	1	0
1	1	0	1	1	1

Таблиця 5.2 – ДТ для схеми на рис. 5.1

Входи					Несправності					
x1	x2	x3	x4	x5	20	30	40	50	61	71
					y	y	y	y	y	y
0	0	0	0	0	0	1	0	1	1	1
1	1	0	0	0	1	1	0	1	1	1

Опис схеми та завдання зображений на рисунку 5.2.

Circuit 5

```
1  INPUTS: x1, x2, x3, x4, x5;
2  OUTPUTS: y;
3  ELEMENTS: e1, e7 = AND, e2, e3, e5 = NOT, e4, e6 = OR;
4  LINKS: e1(x1, x2), e2(x3), e3(x4), e4(e1, e7), e5(x5), e6(e3, e5), e7(e2,
   e6), y(e4);
5
6  TASK t1
7  MALFUNCTIONS: e2, e3, e4, e5 = 0, x1, x2, x3, x4, x5, e3, e5, e6, e7 = 1;
8  TESTS: ct(00001, 00010, 01011, 11011);
9
10 TASK t2
11 MALFUNCTIONS: e2, e3, e4, e5 = 0, e6, e7 = 1;
12 TESTS: dt(00000, 11000);
13 DIAGNOSTIC: dt(y = [01, 11, 00, 11, 11, 11]);
```

Рисунок 5.2 – Опис КС №5 та завдання

Результати контролю і діагностики наведені на рисунку 5.3. З результатів контролю бачимо, що одна несправність не виявляється, а з результатів діагностики – всі несправності діагностуються вірно, але несправності з кодами 30, 50, 61, 71 не можливо відрізнити.

Результати побудови тестів з допомогою програми зображені на рисунку 5.4. Бачимо, що тести, побудовані програмно, виявляються та розрізняють усі помилки, на відміну від випробуваних тестів.

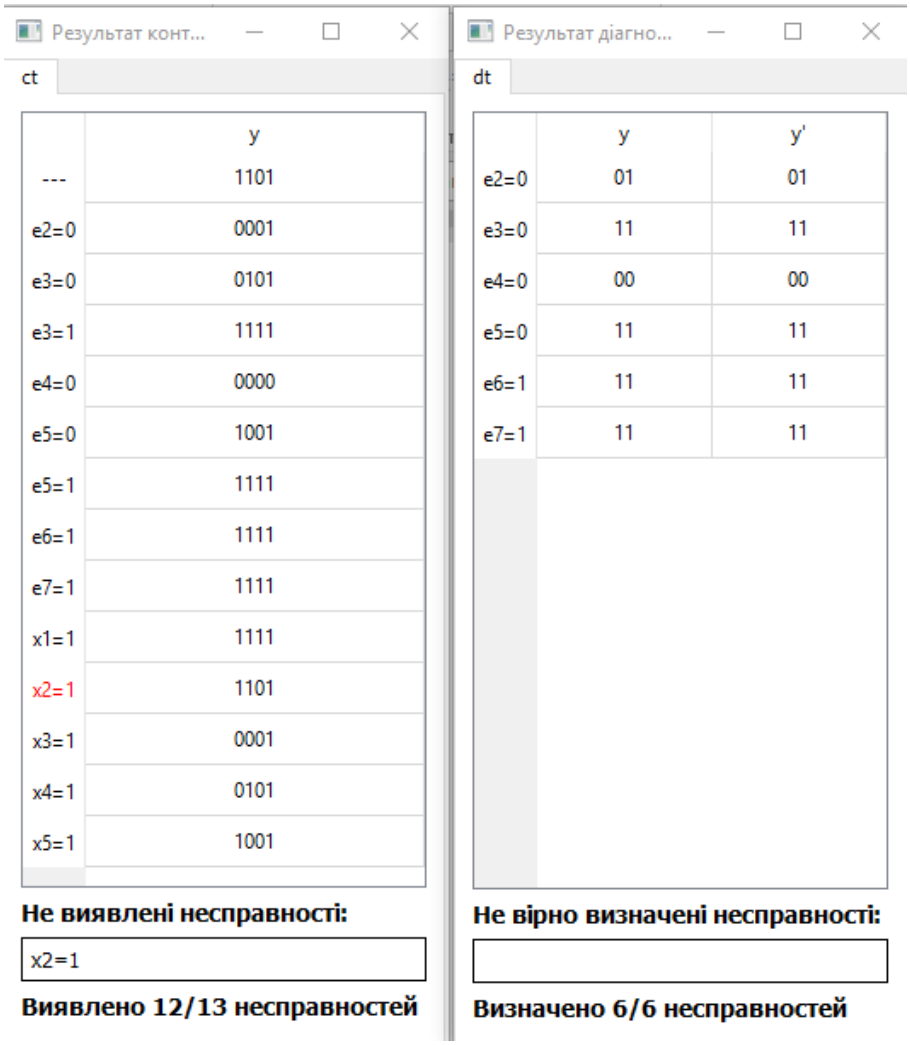


Рисунок 5.3 – Результати контролю і діагностики КС №5

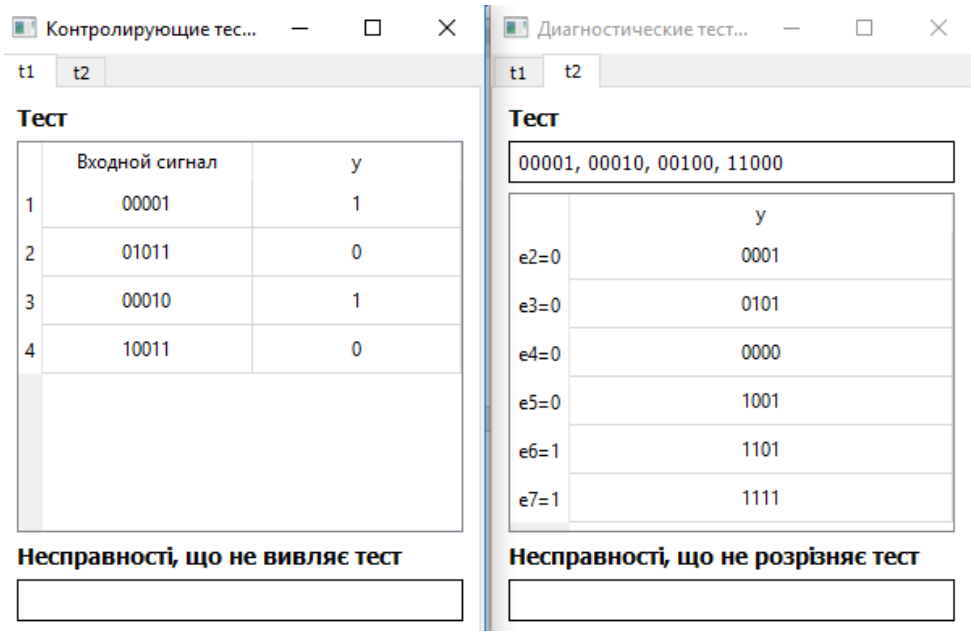


Рисунок 5.4 – Результати побудови КТ та ДТ для КС №5

5.2 Схема №7

На рисунку 5.5 зображена схема №7 для якої було розроблено КТ та ДТ, що наведені в таблицях 5.3, 5.4 відповідно.

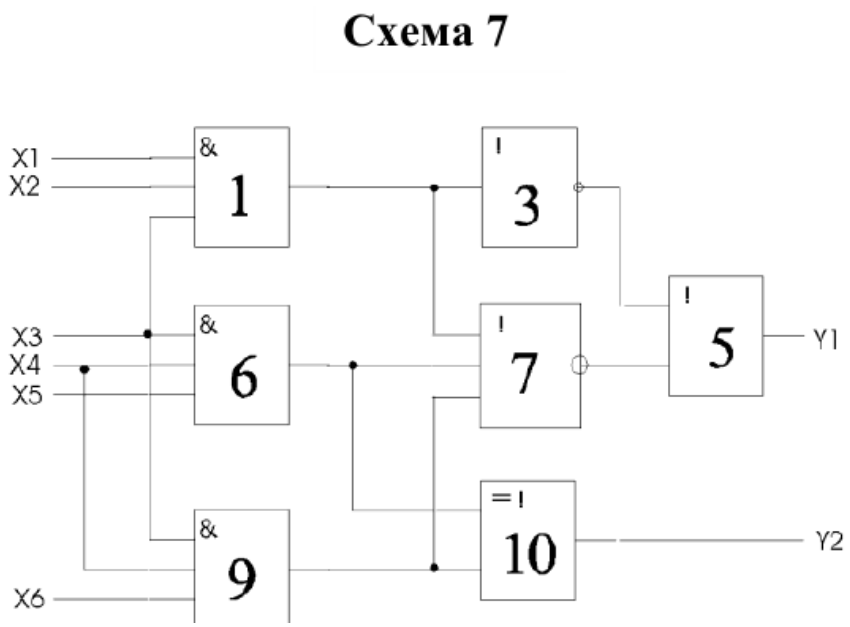


Рисунок 5.5 – КС №7

Таблиця 5.3 – КТ для КС №7

Несправності: 10, 30, 50, 60, 70, 90, 100, x10, x20, x30, x40, x50, x60, 11, 31							
Входи						Виходи	
x1	x2	x3	x4	x5	x6	y1	y2
1	1	1	1	0	1	0	1
0	0	1	1	1	0	1	1

Таблиця 5.4 – ДТ для КС №7

Входи						Несправності											
x1	x2	x3	x4	x5	x6	11		31		51		61		71		91	
						y1	y2	y1	y2	y1	y2	y1	y2	y1	y2	y1	y2
0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	1	1
0	0	1	1	0	1	0	1	1	1	0	1	1	0	1	1	0	1

Опис схеми та завдання зображений на рисунку 5.6. Результати контролю і діагностики наведені на рисунку 5.7.

Circuit 7

```
INPUTS: x1, x2, x3, x4, x5, x6;
OUTPUTS: y1, y2;
ELEMENTS: e1, e6, e9 = AND, e3 = NOT, e7 = NOR, e5 = OR, e10 = XOR;
LINKS: e1(x1, x2, x3), e3(e1), e5(e3, e7), e6(x3, x4, x5), e7(e1, e6, e9), e9(x3,
x4, x6), e10(e6, e9), y1(e5), y2(e10);

TASK t1
MALFUNCTIONS: e1, e3, e5, e6, e7, e9, e10, x1, x2, x3, x4, x5, x6 = 0, e1, e3 = 1;
TESTS: ct(111101, 001110);

TASK t2
MALFUNCTIONS: e1, e3, e5, e6, e7, e9 = 1;
TESTS: dt(000000, 001101);
DIAGNOSTIC: dt(y1=[00, 11, 10, 11, 11, 10], y2 = [01, 01, 01, 10, 01, 11]);
```

Рисунок 5.6 – Опис КС №7 та завдання

Результат контролю '...

ct

	y1	y2
---	01	11
e1=0	11	11
e1=1	00	11
e10=0	01	00
e3=0	00	11
e3=1	11	11
e5=0	00	11
e6=0	01	10
e7=0	01	11
e9=0	01	01
x1=0	11	11
x2=0	11	11
x3=0	11	00
x4=0	01	00

Не виявлені несправності:

e7=0

Виявлено 14/15 несправностей

Результат діагностики...

dt

	y1	y2	y1'	y2'
e1=1	00	01	00	01
e3=1	11	01	11	01
e5=1	11	01	10	01
e6=1	11	10	11	10
e7=1	11	01	11	01
e9=1	11	11	10	11

Не вірно визначені несправності:

e5=1, e9=1

Визначено 4/6 несправностей

Рисунок 5.7 – Результати контролю і діагностики КС №7

З результатів контролю бачимо, що несправність з кодом 70 не виявляється, а з результатів діагностики – несправності з кодами 51 і 91 діагностуються не вірно. Отже, діагностичний тест не є коректним.

Результати побудови тестів з допомогою програми зображені на рисунку 5.8. Бачимо, що КТ, побудований програмно, не виявляє ту ж саму несправність, отже ця несправність не впливає на функціонування схеми. ДТ, побудований програмно, спроможний діагностувати несправності з кодами 51 і 91, але не розрізняє несправностей з кодами 31, 51, 71.

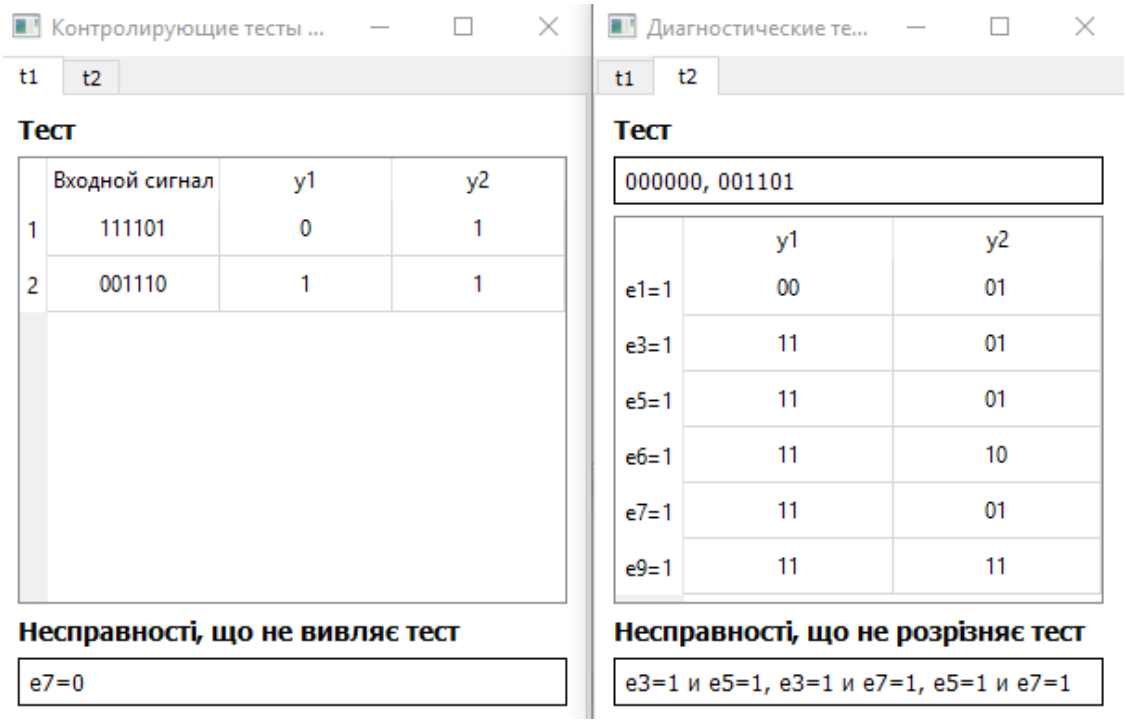


Рисунок 5.8 – Результати побудови КТ та ДТ для КС №7

5.3 Схема №16

На рисунку 5.9 зображена схема №16 для якої було розроблено КТ та ДТ, що наведені в таблицях 5.5, 5.6 відповідно. Опис схеми та завдання зображений на рисунку 5.10.

Результати контролю і діагностики зображені на рисунку 5.11. З результатів контролю бачимо, що усі несправності виявляються, а з результатів діагностики – тест коректний та розрізняє всі несправності.

Схема 16

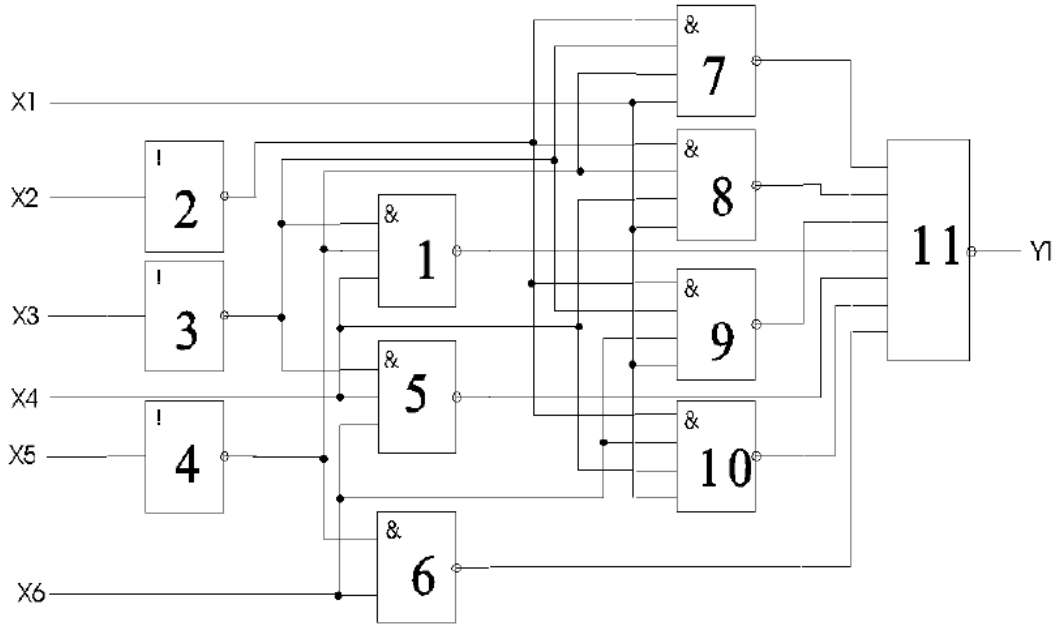


Рисунок 5.9 – КС №16

Таблиця 5.5 – КТ для КС №16

Несправності: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, x10, x20						
Входи						Виходи
x1	x2	x3	x4	x5	x6	y
1	1	0	0	0	0	0
1	0	0	0	0	0	1

Таблиця 5.6 – ДТ для КС №16

Входи						Несправності					
x1	x2	x3	x4	x5	x6	51	61	71	81	91	101
						y	y	y	y	y	y
0	0	0	0	0	1	1	0	1	1	1	1
0	0	0	1	1	1	0	1	1	1	1	1
1	0	0	0	0	0	1	1	0	1	1	1
1	0	0	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	0

Circuit 16

```
1 INPUTS: x1, x2, x3, x4, x5, x6;
2 OUTPUTS: y1;
3 ELEMENTS: e2, e3, e4 = NOT, e1, e5, e6, e7, e8, e9, e10, e11 = NAND;
4 LINKS: e2(x2), e3(x3), e4(x5), e1(e3, e4, x4), e5(e3, x4, x6), e6(e4, x6),
  e7(x1, e4, e3, e2), e8(e2, e4, x4, x1), e9(x6, x1, e3, e2), e10(e2, x6, x4,
  x1), e11(e7, e8, e9, e1, e5, e10, e6), y1(e11);
5
6 TASK t1
7 MALFUNCTIONS: e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, x1, x2 = 0;
8 TESTS: ct(110000, 100000);
9
10 TASK t2
11 MALFUNCTIONS: e5, e6, e7, e8, e9, e10 = 1;
12 TESTS: dt(000001, 000111, 100000, 100011, 101111);
13 DIAGNOSTIC: dt(y1=[10111, 01111, 11011, 11111, 11101, 11110]);
```

Рисунок 5.10 – Опис КС №16 та завдання

Результат контролю...

ct

	y1
---	01
e1=0	11
e10=0	11
e11=0	00
e2=0	00
e3=0	00
e4=0	00
e5=0	11
e6=0	11
e7=0	11
e8=0	11
e9=0	11
x1=0	00
x2=0	11

Не виявлені несправності:

Виявлено 13/13 несправностей

Результат діагностики ...

dt

	y1	y1'
e10=1	11110	11110
e5=1	10111	10111
e6=1	01111	01111
e7=1	11011	11011
e8=1	11111	11111
e9=1	11101	11101

Не вірно визначені несправності:

Визначено 6/6 несправностей

Рисунок 5.11 – Результати контролю і діагностики КС №16

Результати побудови тестів з допомогою програми зображені на рисунку 5.12. Отримані тести не відрізняються за ефективністю від випробуваних, однак, в ДТ, побудованого програмним шляхом, відрізняється останній набір.

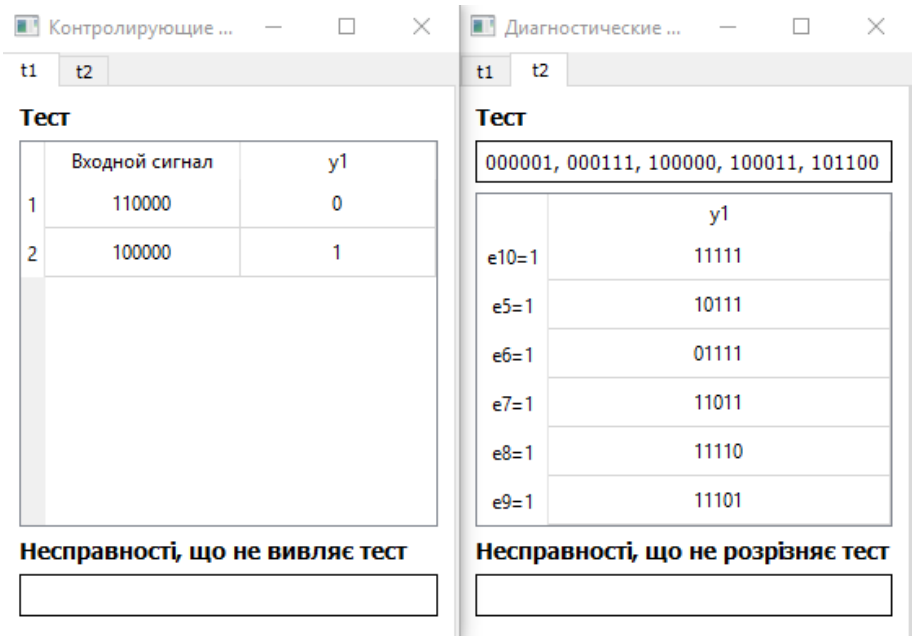


Рисунок 5.12 – Результати побудови КТ та ДТ для КС №16

5.4 Схема №26

На рисунку 5.13 зображена схема №26 для якої було розроблено КТ та ДТ, що наведені в таблицях 5.7, 5.8 відповідно.

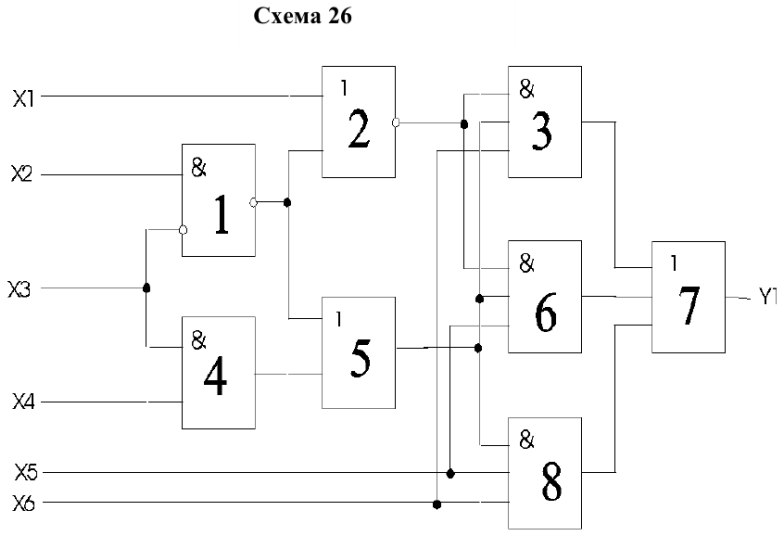


Рисунок 5.13 – КС №26

Таблиця 5.7 – КТ для КС №26

Несправності: 10, 20, 30, 40, 50, 60, 70, 80, 11, 21, 31, 41, 51						
Входи						Виходи
x1	x2	x3	x4	x5	x6	y
0	0	0	0	1	1	1
0	0	1	1	0	1	0
0	1	1	0	1	1	0
0	1	1	1	0	1	1
0	1	1	1	1	0	1

Таблиця 5.8 – ДТ для КС №26

Входи						Несправності						
x1	x2	x3	x4	x5	x6	10	20	30	40	60	11	21
						y	y	y	y	y	y	y
0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	1	1	0	1	1	1	1	1	1
0	1	1	0	1	1	0	0	0	0	0	1	0
0	1	1	1	0	1	1	0	0	0	1	0	1
0	1	1	1	1	0	1	0	1	0	0	0	1
1	1	1	1	1	1	1	1	1	0	1	1	1

Опис схеми та завдання зображений на рисунку 5.14.

```
1  INPUTS: x1, x2, x3, x4, x5, x6;
2  OUTPUTS: y1;
3  ELEMENTS: e1 = NAND, e2 = NOR, e3, e4, e6, e8 = AND, e5, e7 = OR;
4  LINKS: e1(x2, x3), e2(x1, e1), e3(e2, e5, x6), e4(x3, x4), e5(e1, e4), e6(e2,
   e5, x5), e7(e3, e6, e8), y1(e7), e8(e5, x5, x6);
5
6  TASK t1
7  MALFUNCTIONS: e1, e2, e3, e4, e5, e6, e7, e8 = 0, e1, e2, e3, e4, e5 = 1;
8  TESTS: ct(000011, 001101, 011011, 011101, 011110);
9
10 TASK t2
11 MALFUNCTIONS: e1, e2, e3, e4, e6 = 0, e1, e2 = 1;
12 TESTS: dt(000010, 000011, 011011, 011101, 011110, 111111);
13 DIAGNOSTIC: dt(y1 = [000111, 010001, 010011, 010000, 010101, 011001, 110111]);
```

Рисунок 5.14 – Опис КС №26 та завдання

Результати контролю і діагностики зображено на рисунку 5.15.

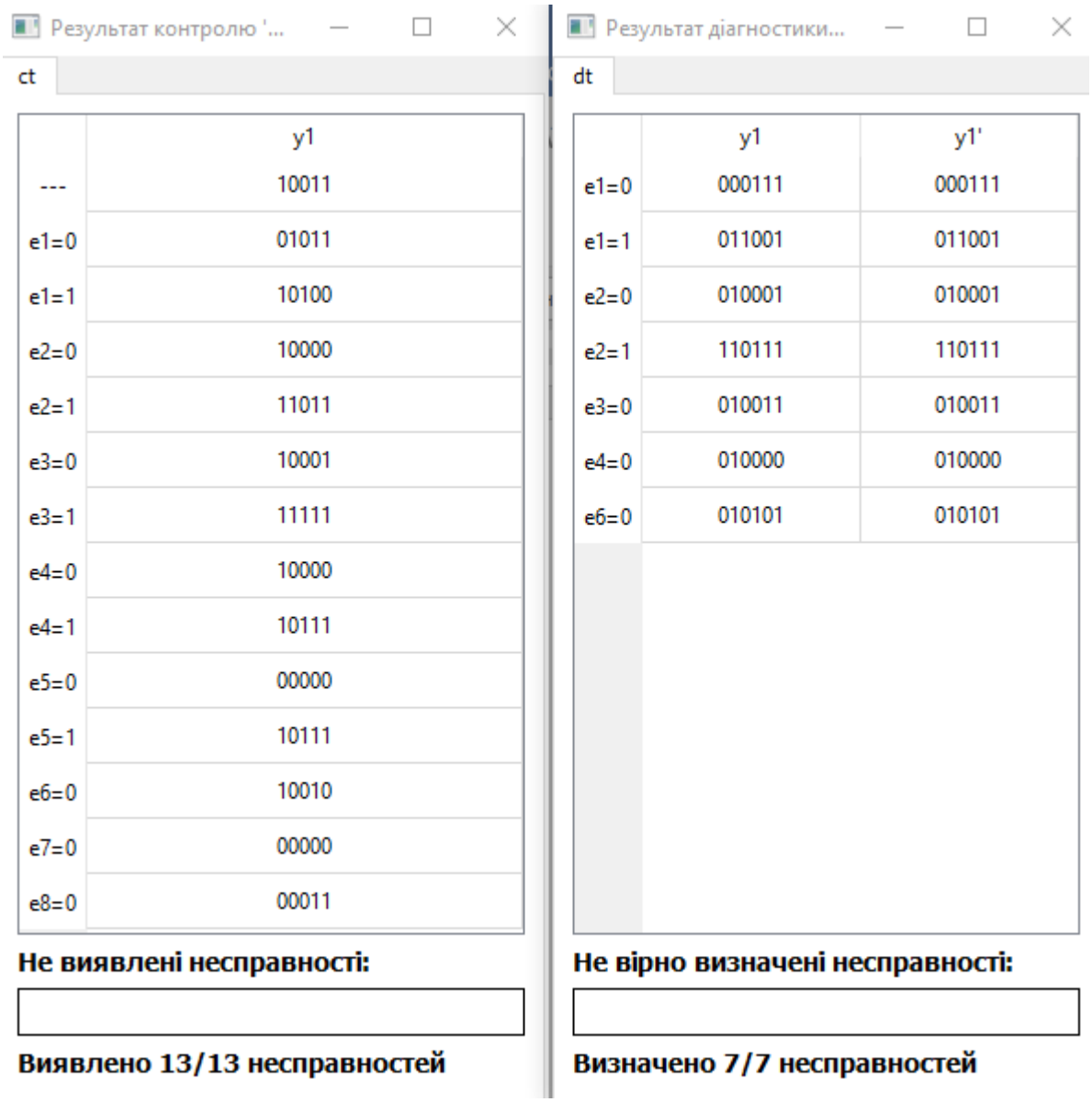


Рисунок 5.15 – Результати контролю і діагностики КС №26

З результатів контролю видно, що усі несправності виявляються та діагностуються вірно. Усі несправності розрізняються.

5.5 Схема №27

На рисунку 5.16 зображена схема №27 для якої було розроблено КТ та ДТ, що наведені в таблицях 5.9, 5.10 відповідно.

Схема 27

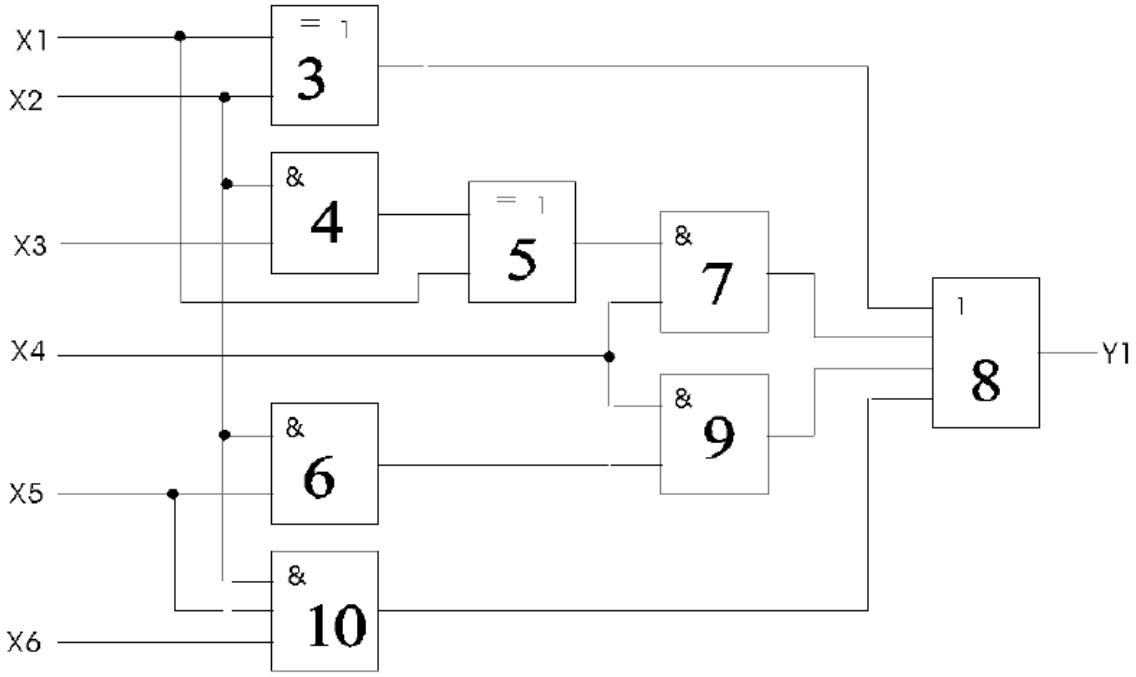


Рисунок 5.16 – КС № 27

Таблиця 5.9 – КТ для КС №27

Несправності: 31, 41, 51, 61, 71, 81, 91, 101, x11, x21, x31, x41, x51, x61						
Входи						Виходи
x1	x2	x3	x4	x5	x6	y
0	0	0	1	0	0	0
1	1	0	1	0	0	1
1	1	0	0	1	0	0

Таблиця 5.10 – ДТ для КС №27

Входи						Несправності					
x1	x2	x3	x4	x5	x6	31	41	51	61	71	91
						y	y	y	y	y	y
0	0	0	0	0	0	1	0	0	0	1	1
1	1	0	1	0	0	1	0	1	1	1	1

Опис схеми та завдання зображений на рисунку 5.17. Результати контролю і діагностики зображені на рисунку 5.18.

Circuit 27

```
1  INPUTS: x1, x2, x3, x4, x5, x6;
2  OUTPUTS: y1;
3  ELEMENTS: e3, e5= XOR, e4, e6, e7, e9, e10 = AND, e8 = OR;
4  LINKS: e3(x1, x2), e4(x2, x3), e5(e4, x1), e6(x2, x5), e7(e5, x4), e8(e3,
   e7, e9, e10), e9(x4, e6), e10(x2, x5, x6), y1(e8);
5
6  TASK t1
7  MALFUNCTIONS: e3, e4, e5, e6, e7, e8, e9, e10, x1, x2, x3, x4, x5, x6 = 1;
8  TESTS: ct(000100, 110100, 110010);
9
10 TASK t2
11 MALFUNCTIONS: e3, e4, e5, e6, e7, e9 = 1;
12 TESTS: dt(000000, 110100);
13 DIAGNOSTIC: dt(y1 = [11, 00, 01, 01, 11, 11]);
```

Рисунок 5.17 – Опис КС №27 та завдання

Результат контролю '...

ct

	y1
---	010
e10=1	111
e3=1	111
e4=1	100
e5=1	110
e6=1	110
e7=1	111
e8=1	111
e9=1	111
x1=1	110
x2=1	110
x3=1	000
x4=1	011
x5=1	010
x6=1	011

Не виявлені несправності:

x5=1

Виявлено 13/14 несправностей

Результат діагностики...

dt

	y1	y1'
e3=1	11	11
e4=1	00	00
e5=1	01	01
e6=1	01	01
e7=1	11	11
e9=1	11	11

Не вірно визначені несправності:

Визначено 6/6 несправностей

Рисунок 5.18 – Результати контролю та діагностики КС №27

Результати побудови тестів з допомогою програми зображені на рисунку 5.19.

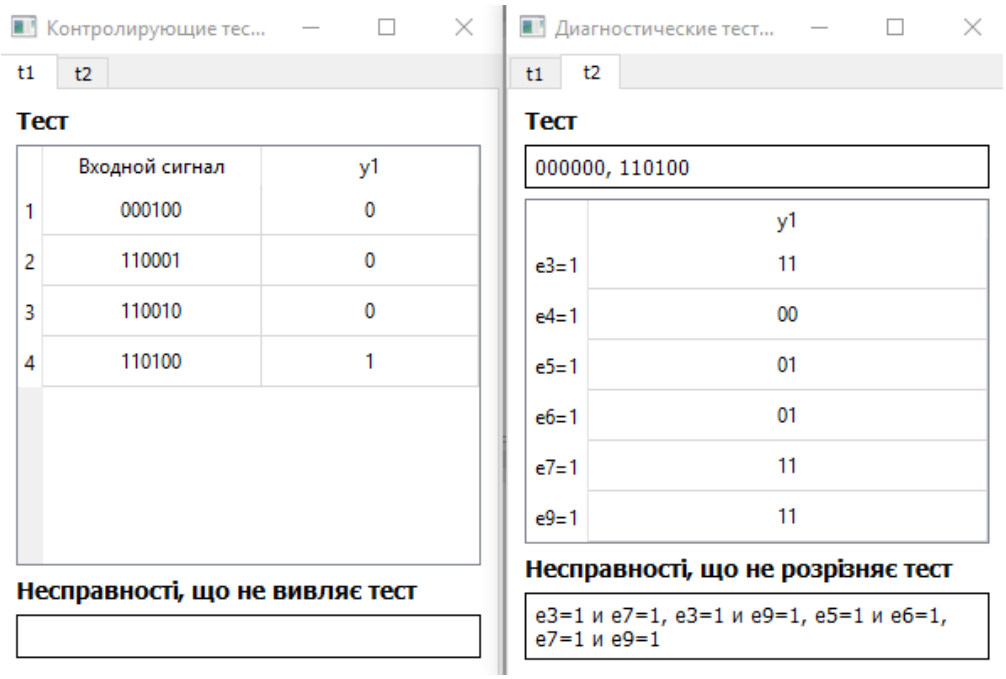


Рисунок 5.19 – Результати побудови КТ та ДТ для КС №27

Порівнюючи результати, видно, що випробуваний КТ, на відміну від отриманого програмним шляхом, не виявляє несправності з кодом x51. Щодо ДТ результати одинакові.

5.6 Висновки

В даному розділі дипломної роботи було випробувано кілька КТ та ДТ, що були розроблені в рамках лабораторних робіт. Також КТ та ДТ було побудовано з допомогою програми. Вони використовувались для порівняння з випробуваними тестами для визначення їх ефективності. За результатами випробувань та порівнянь, деякі з розроблених тестів виявилися неефективними, оскільки виявляли меншу кількість несправностей, ніж побудований КТ, або розрізняли меншу кількість несправностей, ніж побудований ДТ. Також були виявлені некоректні ДТ.

6 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для випробування тестів КС, що були розроблені в рамках лабораторних та курсових робіт. Програма була розроблена за допомогою мови програмування C++ з використанням можливостей останнього стандарту C++11 та з використанням бібліотеки Qt 5.8. Розробка велась у середовищі розробки Qt Creator 4.2.1. Інтерфейс користувача створений за допомогою модулю Qt Widgets бібліотеки Qt.

Програмний продукт призначено для використання на персональних комп’ютерах під управлінням ОС Windows, але є можливість його використання під Unix-подібними ОС та macOS.

6.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;
- забезпечувати стабільну роботу, а при некоректних вхідних даних чи у разі виникненні помилок в процесі роботи програми надавати інформацію про їх походження;
- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;
- передбачати мінімальні витрати на впровадження програмного продукту.

6.2 Обґрунтування функцій програмного продукту

6.2.1 Формування варіантів функцій

Головна функція F0 – розробка програмного продукту, який випробовує тести для КС. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F1 – мова програмування

F2 – бібліотека GUI;

F3 – формат вхідних даних;

F4 – моделювання схем;

Кожна з основних функцій може мати декілька варіантів реалізації:

Функція F1:

a) C++;
b) C#;
c) Java.

Функція F2:

a) Qt;
b) WPF;
c) JavaFX.

Функція F3:

a) використання формату JSON;
b) розробка власного формату;

Функція F4:

a) ітераційний алгоритм;
b) рекурсивний алгоритм.

6.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 6.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 6.1).

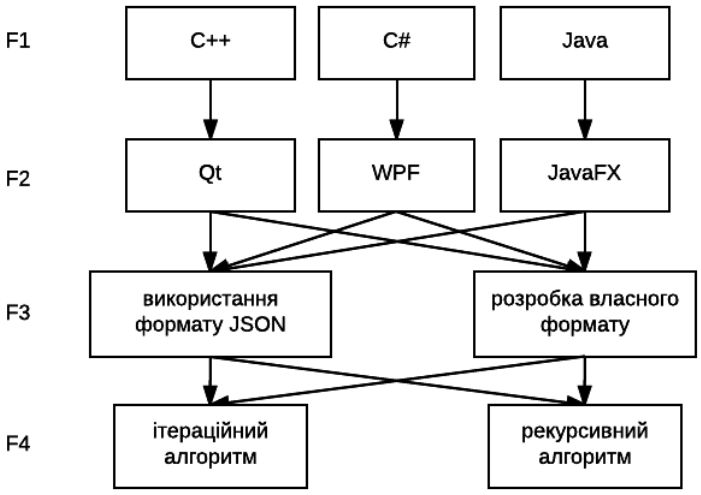


Рисунок 6.1 – Морфологічна карта системи

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, як такі, що вони не відповідають поставленим перед програмним продуктом технічним вимогам.

Функція F1.

Варіанти b) та c) вимагають наявності в користувача попередньо встановлених компонентів, тому ці варіанти мають бути відкинуті, оскільки вони не задовольняють технічним вимогам.

Функція F2.

Оскільки варіанти b) та c) функції F1 були відкинуті, то варіанти b) та c) функції F2 не є гідними для використання.

Функція F3

Варіанти a) та b) вважаємо гідними розгляду.

Функція F4

Варіант a) та b) вважаємо гідними розгляду.

Таблиця 6.1 – Позитивно-негативна матриця варіантів основних функцій

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	a	Швидка мова, низьке споживання пам’яті	Складність мови, повільний розвиток
	b	Безпечний код, мова швидко розвивається, простота мови	Високе споживання пам’яті, для виконання потрібний .NET Framework
	c	Крос-платформність, безпечний код, простота мови	Високе споживання пам’яті, для виконання необхідна JVM
F2	a	Зручний API, крос-платформність, наявність візуального редактора, хороша документація	Бібліотеки споживають багато пам’яті
	b	Простота використання, наявність візуального редактора	Не крос-платформний, необхідний .NET Framework
	c	Крос-платформність, наявність візуального редактора, простота	Молодість
F3	a	Потрібно лише виконати перевірку вхідних даних	Незручність опису вхідних даних
	b	Зручність опису вхідних даних	Потрібно розробляти повноцінний парсер з перевіркою вхідних даних
F4	a	Висока швидкість моделювання	Більш складна реалізація
	b	Проста реалізація	Низька швидкість моделювання

Таким чином, будемо розглядати такі варіанти реалізаціїї ПП:

1. F1a – F2a – F3a – F4a
2. F1a – F2a – F3a – F4b
3. F1a – F2a – F3b – F4a
4. F1a – F2a – F3b – F4b

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

6.3 Обґрунтування системи параметрів ПП

Для характеристики розроблюваної програми можна використати такі параметри:

– *X1* – швидкодія мови програмування, Оп/мс;

– *X2* – об’єм пам’яті, потрібної для роботи програми, Мб;

– *X3* – об’єм опису вхідних даних, кількість символів;

– *X4* – час моделювання, мс.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 6.2.

Таблиця 6.2 – Основні параметри ПП

Назва параметра	**Позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Швидкодія мови програмування	X1	Оп/мс	2000	11000	19000
Об’єм пам’яті, потрібної для роботи програми	X2	Мб	32	16	8
Об’єм опису вхідних даних	X3	к-сть симв.	700	400	200
Час моделювання	X4	мс	100	20	5

За даними таблиці 6.2 будуються графіки залежності бальної оцінки параметра від його основного значення – рис. 6.2 – рис. 6.5.

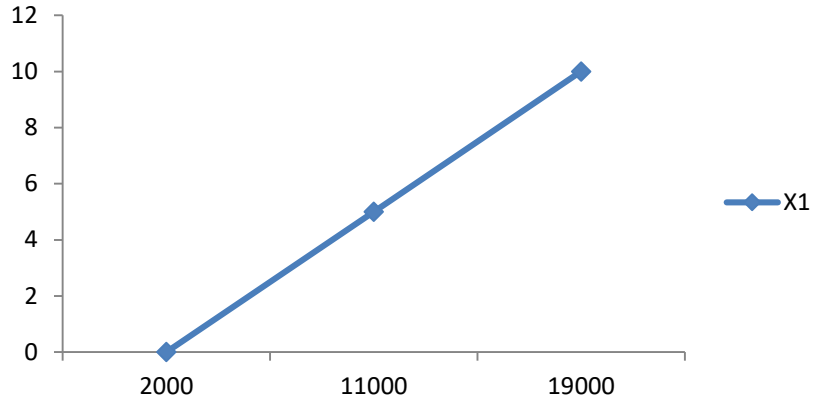


Рисунок 6.2 – X1, швидкодія мови програмування

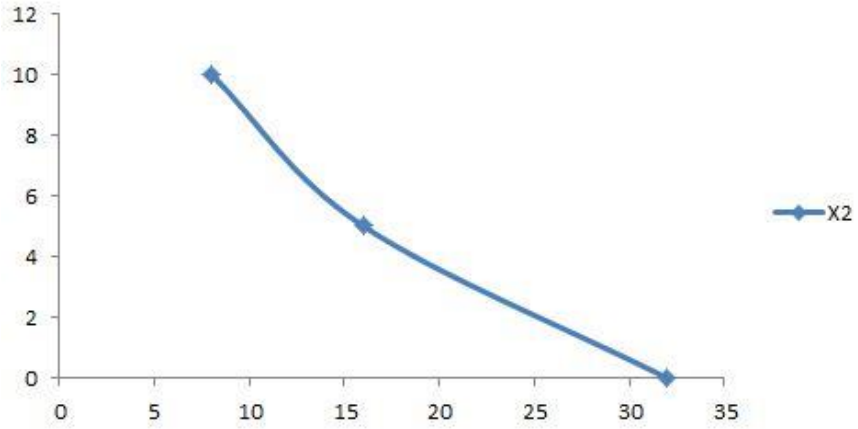


Рисунок 6.3 – X2, об’єм пам’яті, потрібної для роботи програми

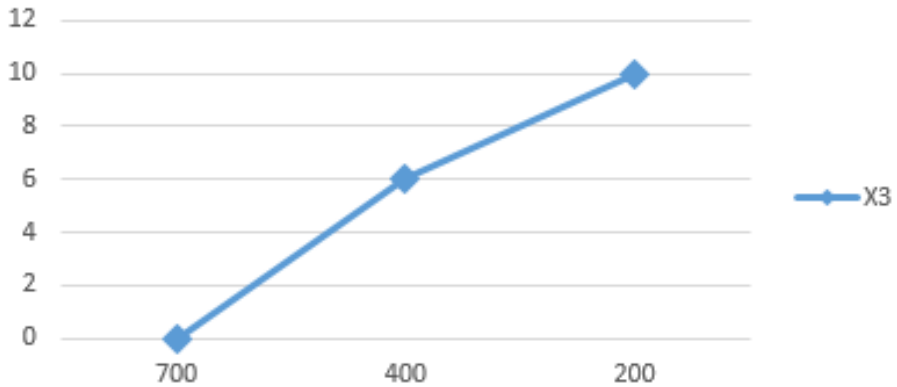


Рисунок 6.4 – X3, об’єм опису вхідних даних

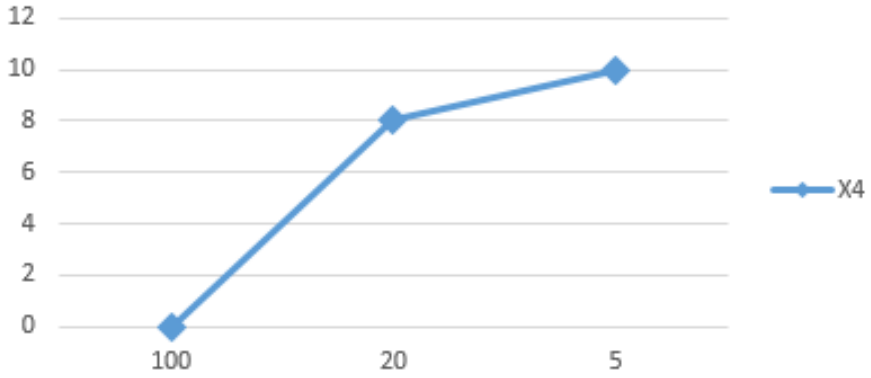


Рисунок 6.5 – X4, час моделювання

6.4 Аналіз експертного оцінювання параметрів

Результати експертного ранжування наведені у таблиці 6.3.

Таблиця 6.3 – Результати ранжування параметрів

Параметр	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
	1	2	3	4	5	6	7			
X1	3	3	3	4	4	4	3	24	6,5	42,25
X2	4	4	4	3	3	3	4	25	7,5	56,25
X3	1	2	1	2	2	1	1	10	-7,5	56,25
X4	2	1	2	1	1	2	2	11	-6,5	42,25
Разом	10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{l=1}^{N} r_{il},$$

$$R = \sum_{i=1}^{n} R_i = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n}R = 17{,}5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 197}{7^2(4^3 - 4)} = 0{,}8 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 6.4.

Таблиця 6.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	<	<	<	>	>	1,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	>	<	>	<	<	>	>	>	1,5

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$, де $b_i = \sum_{i=1}^{N} a_{ij}$.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}$,де $b_i' = \sum_{i=1}^{N} a_{ij} b_j$.

Як видно з таблиці 6.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 6.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	0,5	0,5	3,5	0,219	12,25	0,208	44.875	0,208
X2	0,5	1,0	0,5	0,5	2,5	0,156	9,25	0,157	34.125	0,158
X3	1,5	1,5	1,0	1,5	5,5	0,344	21,25	0,360	77.875	0,361
X4	1,5	1,5	0,5	1,0	4,5	0,281	16,25	0,275	59.125	0,274
Всього:					16	1	59	1	216	1

6.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо. Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так:

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів; $K_{вi}$– коефіцієнт вагомості i–го параметра; B_i – оцінка i–го параметра в балах.

Результати розрахунків зведено в табл. 6.6.

Таблиця 6.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	a	15000	7.5	0,208	1.56
F2	a	24	2.5	0,158	0.395
F3	a	500	4	0,361	1.444
	b	300	8	0,361	2.888
F4	a	10	9.3	0,274	2.548
	b	50	5	0,274	1.37

За даними з таблиці 6.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 1,56 + 0,395 + 1.444 + 2.548= 5.947

K_{K2} = 1,56 + 0,395 + 1.444 + 1.37 = 4.769

K_{K3} = 1,56 + 0,395 + 2.888 + 2.548 = 7.383

K_{K4} = 1,56 + 0,395 + 2.888 + 1.37 = 6.213

Як видно з розрахунків, кращим є третій варіант, для якого коефіцієнт технічного рівня має найбільше значення.

6.6 Економічний аналіз варіантів розробки ПП

6.6.1 Визначення трудомісткості

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості. Вихідні дані для розрахунку трудомісткості за варіантами реалізації наведено в табл. 6.7.

Таблиця 6.7 – Вихідні дані для розрахунку трудомісткості

Основні функції	Варіант реалізації	К-сть вх. інф.	К-сть вих. інф.	Ступінь новизни задач	Складність алг.	Складність контролю інформації		Використовувана інформація		
						Вхідна	Вихідна	Змінна	Довідкова	Банк даних
F1	a	1	1	Г	3	11	21	-	-	1
F2	a	1	1	Г	3	11	21	-	-	1
F3	a	4	2	В	2	12	22	-	1	-
	b	4	2	Б	2	12	22	-	1	-
F4	a	4	1	В	1	12	22	-	1	-
	b	4	1	В	1	12	22	-	1	-

Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість розробки ПП; $K_П$ – поправочний коефіцієнт; $K_{СК}$ – коефіцієнт на складність вхідної інформації; $K_М$ – коефіцієнт рівня мови програмування; $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення. Результати розрахунку трудомісткості зведено в таблицю 6.8.

Таблиця 6.8 – Розрахунок трудомісткості

Функція	Варіант	T_p	$K_п$	$K_{ск}$	$K_м$	$K_{ст}$	$K_{ст.м}$	$T_о$
F1	a	8	0,3	1,16	1	1	1	2,784
F2	a	8	0,3	1,16	1	1	1	2,784
F3	a	19	0,72	1	1	0,6	1	8,208
	b	27	1,08	1	1	1	1	29,16
F4	a	43	0,81	1	1	1	1	34,83
	b	43	0,81	1	1	1	1	34,83

Складемо трудомісткість відповідних завдань для кожного з чотирьох обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

T_I = (2,784 + 2,784 + 8,208 + 34,83) · 8 = 388,848 людино-годин;

T_{II} = (2,784 + 2,784 + 8,208 + 34,83) · 8 = 388,848 людино-годин;

T_{III} = (2,784 + 2,784 + 29.16 + 34.83) · 8 = 556,464 людино-годин;

T_{IV} = (2,784 + 2,784 + 29.16 + 34.83) · 8 = 556,464 людино-годин.

Найбільш високу трудомісткість мають варіанти III і IV.

6.6.2 Розрахунок витрат на оплату праці

У розробці бере участь один програміст з окладом 10000 грн. Визначимо зарплату програміста за годину:

$$C_Ч = \frac{10000}{21 \cdot 8} = 59{,}52 \text{ грн.}$$

Зарплата розробника за варіантами становить:

I, II: $C_{ЗП}$ = 59,52 * 388,848 · 1,2 = 27773,1 грн.

III, IV: $C_{ЗП}$ = 59,52 * 556,464 · 1,2 = 39744,9 грн.

Відрахування на єдиний соціальний внесок становить 22%:

I, II: $C_{ВІД}$ = $C_{ЗП}$ * 0.22 = 6110,08 грн.

III, IV: $C_{ВІД}$ = $C_{ЗП}$ * 0.22 = 8743,88 грн.

6.6.3 Розрахунок вартості машинного часу

Так як одна ЕОМ обслуговує одного програміста з окладом 10000 грн з коефіцієнтом зайнятості 0,2, то для однієї машини отримаємо:

$C_Г$ = 12 · 10000 · 0,2 = 24000 грн.

З урахуванням додаткової заробітної плати:

$C_{ЗП}$ = 24000 * 1,2 = 28800 грн.

Відрахування на єдиний соціальний внесок:

$C_{ВІД}$ = 28800 * 0,22 = 6336 грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 6000 грн:

C_A = 1,15 * 0,25 * 6000 = 1725 грн.

Витрати на ремонт та профілактику:

C_P = 1,15 * 6000 * 0,05 = 345 грн.

Ефективний годинний фонд часу ПК за рік:

$T_{ЕФ}$ = (365 – 104 – 8 – 16) * 8 * 0.9 = 1706,4 годин.

Витрати на оплату електроенергії:

$C_{ЕЛ}$ =1706,4 * 0,156 * 1,93819 = 515,943 грн.

Накладні витрати:

$C_Н$ = 6000 * 0.67 = 4020 грн.

Річні експлуатаційні витрати:

$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_A + C_P + C_{ЕЛ} + C_Н$ = 28800 + 6336 + 1725 + 345 + 515,943 + 4020 = 41741,9 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$C_{М-Г} = C_{ЕКС} / T_{ЕФ}$ = 41741,9 / 1706,4 = 24,46 грн/год

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_М = C_{М-Г} * T$$

I, II: $C_М$ = 24,46 * 388,848 = 9511,22 грн.

III, IV: $C_М$ = 24,46 * 556,464 = 13611,1 грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} * 0,67$$

I, II: $C_Н$ = 27773,1· 0,67 = 18608 грн.

III, IV: $C_Н$ = 39744,9 · 0,67 = 26629,1 грн.

6.6.4 Розрахунок вартості розробки ПП

Вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н$$

I, II: $C_{ПП}$ = 27773,1 + 6110,08 + 388,848 + 18608 = 52880 грн.

III, IV: $C_{ПП}$ = 39744,9 + 8743,88 + 556,464 + 26629,1 = 75674,3 грн.

6.7 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{ППj},$$

$К_{ТЕР1} = 5.947 / 52880 = 11,24 * 10^{-5}$;

$К_{ТЕР2} = 4.769 / 52880 = 9,02 * 10^{-5}$;

$К_{ТЕР3} = 7.383 / 75674,3 = 9,76 * 10^{-5}$;

$К_{ТЕР4} = 6.213 / 75674,3 = 8,21 * 10^{-5}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 11,24 * 10^{-5}$.

6.8 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Після виконання функціонально-вартісного аналізу, можна зробити висновок, що з альтернатив, які залишились оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 11,24 * 10^{-5}$. Проте, ПП було реалізовано за третім варіантом реалізації, у якого найвищий коефіцієнт технічного рівня, а показник техніко-економічного рівня якості є другим за рейтингом і рівний $К_{ТЕР} = 9,76 * 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – C++;

– бібліотека GUI – Qt;

– розробка власного формату;

– ітераційний алгоритм моделювання.

Даний варіант виконання ПП дає користувачу зручний інтерфейс, зручний спосіб введення даних, непоганий функціонал і швидкодію.

ВИСНОВКИ

Дана дипломна робота була присвячена розробці комп’ютерної програми для випробування тестів, призначених для тестування КС.

Результатом роботи є добре відлагоджена програма з GUI, яка дозволяє моделювати будь-які КС в справному стані на всіх можливих комбінаціях вхідних сигналів, задавати несправності для компонентів КС та виконувати її тестування на заданому тестовому наборі задля перевірки ефективності розроблених тестів, а також будувати тести для схем програмним шляхом. Програму було розроблено на мові C++ з використанням GUI бібліотеки Qt.

Для зручного введення вхідних даних (схеми, несправностей, тестів) був розроблений формат вхідних даних та парсер для цього формату. В парсері реалізована перевірка помилок у вхідних даних, як синтаксичних, так і семантичних, з повідомленням їх місця та причини, що дозволяє користувачу моментально їх виправляти.

У майбутньому в програмі можна реалізувати наступний функціонал:

- тестування схеми за допомогою діагностичного тесту з елементами контролюючого;
- побудова діагностичного тесту з елементами контролюючого;
- графічний редактор схеми;
- відображення результатів моделювання несправної схеми на всіх можливих вхідних наборах;
- задання наборів вхідних значень для виконання моделювання (а не тільки тестування) з відображенням результатів моделювання.

Основною галуззю застосування розробленої програми є навчальний процес, зокрема, виконання лабораторних робіт з курсу КіДІС, в ході яких необхідно перевірити ефективність розроблених тестів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Чегис И. А. Логические способы контроля работы электрических схем / И. А. Чегис, С. В. Яблонский // Сборник статей по математической логике и ее приложениям к некоторым вопросам кибернетики / И. А. Чегис, С. В. Яблонский. – М.: АН СССР, 1958. – (Тр. МИАН СССР; 51). – С. 270–360

2. Теория тестирования логических устройств / В. Б.Кудрявцев, Э. Э. Гасанов, О. А. Долотова, Г. Р. Погосян. ; за ред. В. А. Садовничий – М.: ФИЗМАТЛИТ, 2006. – 160 с.

3. Страуструп Б. Язык программирования C++ / Бьерн Страуструп. – М.: Бином, 2015. – 1136 с.

4. Дейтел Х. Как программировать на C++ / Х. Дейтел, П. Дейтел. – М.: Бином-Пресс, 2008. – 1456 с.

5. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# / Дж. Рихтер. – СПб.: Питер, 2017. – 896 с. – (Мастер-класс).

6. Шилдт Г. Java 8. Полное руководство / Герберт Шилдт. – М.: Вильямс, 2017. – 1376 с.

7. Шлее М. Qt 5.3. Профессиональное программирование на C++ / Макс Шлее. – СПб.: БХВ-Петербург, 2015. – 928 с.

8. XQuery 1.0: An XML Query Language (Second Edition) [Електронний ресурс] Режим доступу: https://www.w3.org/TR/xquery/#id-grammar. – Дата доступу: 05.06.2017

9. Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. – М.: Вильямс, 2016. – 1184 с.

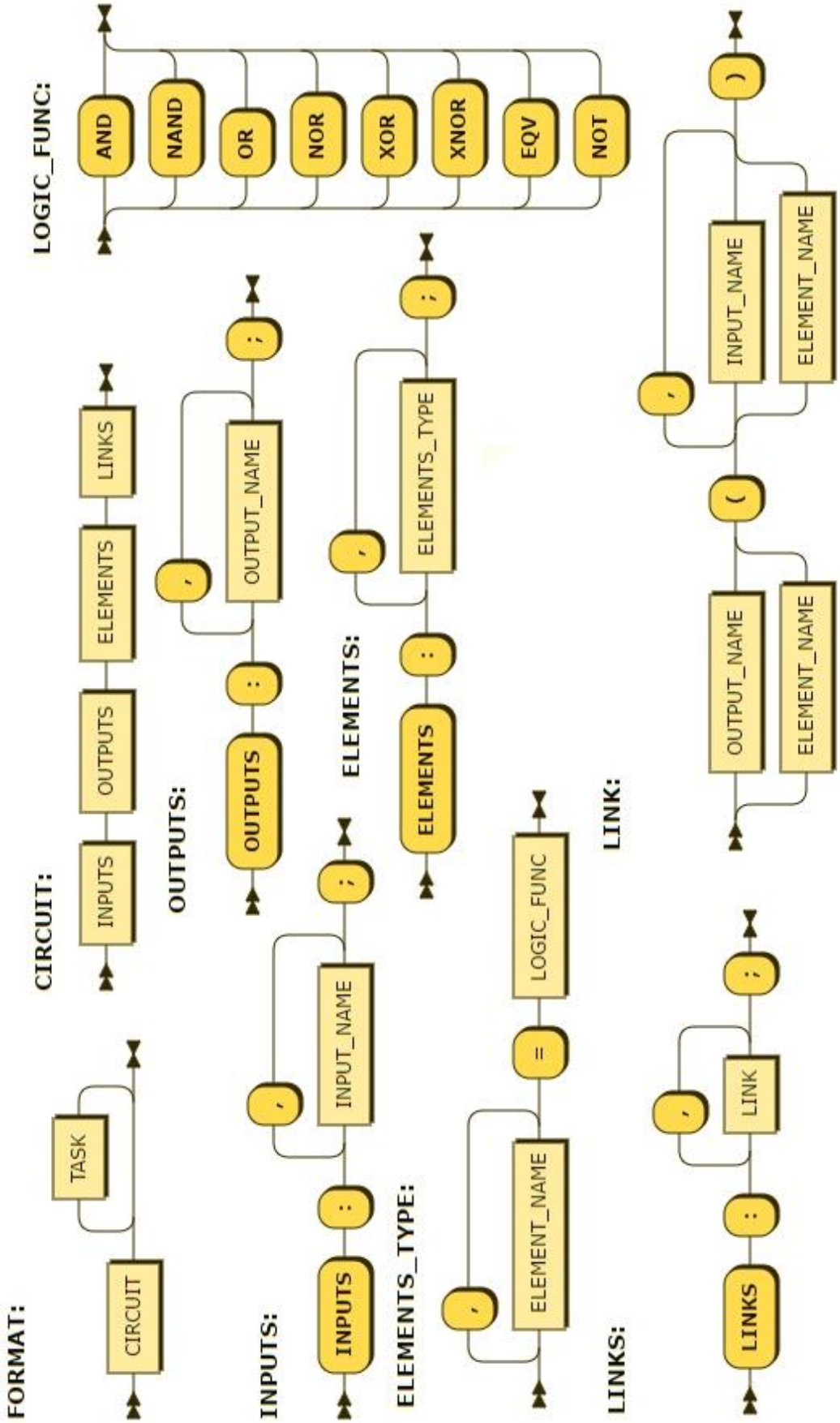
10. Хантер Р. Основные концепции компиляторов / Робин Хантер. – М.: Вильямс, 2002. – 256 с.

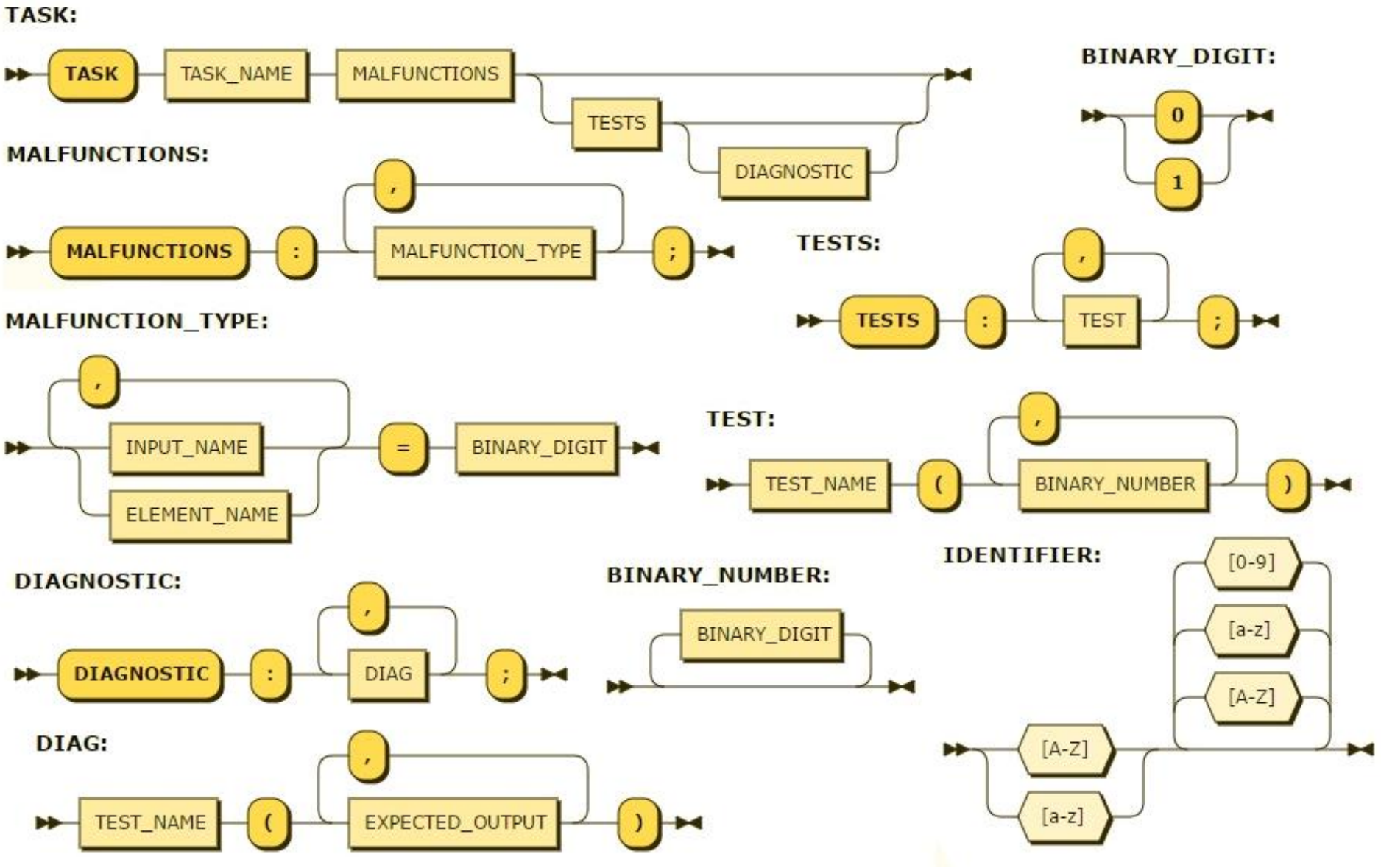
ДОДАТОК А

A.1 EBNF формату вхідних даних

```
FORMAT ::= CIRCUIT TASK*

CIRCUIT ::= INPUTS OUTPUTS ELEMENTS LINKS
INPUTS ::= 'INPUTS' ':' INPUT_NAME (',' INPUT_NAME)* ';'
OUTPUTS ::= 'OUTPUTS' ':' OUTPUT_NAME (',' OUTPUT_NAME)* ';'
ELEMENTS ::= 'ELEMENTS' ':' ELEMENTS_TYPE (',' ELEMENTS_TYPE)* ';'
ELEMENTS_TYPE ::= ELEMENT_NAME (',' ELEMENT_NAME)* '=' LOGIC_FUNC
LOGIC_FUNC ::= 'AND' | 'NAND' | 'OR' | 'NOR' | 'XOR' | 'XNOR' | 'EQV' |
'NOT'
LINKS ::= 'LINKS' ':' LINK (',' LINK)* ';'
LINK ::= (OUTPUT_NAME | ELEMENT_NAME) '(' (INPUT_NAME | ELEMENT_NAME)
(',' (INPUT_NAME | ELEMENT_NAME)) *  ')'

TASK ::= 'TASK' TASK_NAME MALFUNCTIONS (TESTS DIAGNOSTIC?)?
MALFUNCTIONS    ::=    'MALFUNCTIONS'    ':'    MALFUNCTION_TYPE    (','
MALFUNCTION_TYPE)* ';'
MALFUNCTION_TYPE  ::=  (INPUT_NAME  |  ELEMENT_NAME)  (','  (INPUT_NAME  |
ELEMENT_NAME))* '=' BINARY_DIGIT
TESTS ::= 'TESTS' ':' TEST (',' TEST)* ';'
TEST ::= TEST_NAME '(' BINARY_NUMBER (',' BINARY_NUMBER)* ')'
DIAGNOSTIC ::= 'DIAGNOSTIC' ':' DIAG (',' DIAG)* ';'
DIAG ::= TEST_NAME '(' EXPECTED_OUTPUT (',' EXPECTED_OUTPUT)* ')'
EXPECTED_OUTPUT   ::=   OUTPUT_NAME   '='   '['   BINARY_NUMBER   (','
BINARY_NUMBER)* ']'
BINARY_DIGIT ::= '0' | '1'
BINARY_NUMBER::= BINARY_DIGIT*

IDENTIFIER ::= [A-Za-z] ([A-Za-z] | [0-9])*
INPUT_NAME ::= IDENTIFIER
OUTPUT_NAME ::= IDENTIFIER
ELEMENT_NAME ::= IDENTIFIER
TEST_NAME ::= IDENTIFIER
TASK_NAME ::= IDENTIFIER
```

А.2 Синтаксичні діаграми




TASK:
TASK
TASK_NAME
MALFUNCTIONS
TESTS
DIAGNOSTIC
BINARY_DIGIT:
0
1
MALFUNCTIONS:
MALFUNCTIONS
:
,
MALFUNCTION_TYPE
;
TESTS:
TESTS
:
,
TEST
;
MALFUNCTION_TYPE:
,
INPUT_NAME
ELEMENT_NAME
=
BINARY_DIGIT
TEST:
TEST_NAME
(
,
BINARY_NUMBER
)
DIAGNOSTIC:
DIAGNOSTIC
:
,
DIAG
;
BINARY_NUMBER:
BINARY_DIGIT
IDENTIFIER:
[0-9]
[a-z]
[A-Z]
[A-Z]
[a-z]
DIAG:
TEST_NAME
(
,
EXPECTED_OUTPUT
)