Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота

на тему:

«ПРОГРАМНИЙ ЗАСІБ ДЛЯ АНАЛІЗУ ХАРАКТЕРИСТИК ОПЕРАТИВНОЇ ПАМ'ЯТІ»

Виконав: студент 4 курсу , групи 1СП-20б
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Ковальов М.Д.
(прізвище та ініціали)

Керівник: к.т.н., доц.,
Кадук О. В.
(прізвище та ініціали)
«14» 06 2024 р.

Рецензент: д.т.н., проф. каф. ПЗ
Ліщинська Л. Б.
(прізвище та ініціали)
«17» 06 2024 р.

Допущено до захисту
Зав. кафедри ОТ:
д.т.н., проф. Азаров О. Д.

«18» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

_____ 23.04 _____ 2024 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Ковальову Максиму Дмитровичу

1 Тема роботи: «Програмний засіб для аналізу характеристик оперативної пам'яті», керівник роботи Кадук О.В. к.т.н., доц., затверджені наказом вищого навчального закладу від « 11 » березня 2024 року № 80 .

2 Термін подання студентом роботи 20.06.2024

3 Вихідні дані до роботи: засоби — VS Code — це редактор початкового коду, створений Microsoft із Electron Framework. Qt — крос-платформовий інструментарій розробки програмного забезпечення, мова програмування Python 3 — інтерпретована об'єктно-орієнтована мова програмування високого рівня із суворою динамічною типізацією. Передбачено розробити додаток у вигляді прикладного програмного інтерфейсу для аналізу результатів вимірювань та їх обробки.

4 Зміст розрахунково-пояснювальної записки Вступ. Аналіз оперативної пам'яті у сучасних комп'ютерних системах. Проектування програмного засобу для аналізу характеристик оперативної пам'яті. Програмна реалізація засобу для аналізу характеристик оперативної пам'яті. Висновки. Перелік посилань. Додатки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових

креслень) діаграма роботи програми.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти бакалаврської кваліфікаційної роботи.

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1—3	к. т. н., доцент каф. ОТ Кадук О. В.		

7 Дата видачі завдання 12.03.2024р.

8 Календарний план наведено у таблиці 2.

Таблиця 2 — Календарний план.

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Аналіз проблеми, обґрунтування актуальності розробки засобу та постановка задач	09.03.2024	09.03.2024	
2.	Аналіз сучасних технологій оперативної пам'яті	10.03.2024	16.03.2024	
3.	Вибір середовища та мови розробки	17.03.2024	18.03.2024	
4.	Розробка програмного засобу	19.03.2024	22.05.2024	
5.	Оформлення пояснювальної записки	23.05.2024	01.06.2024	
6.	Перевірка якості виконання бакалаврської роботи	02.06.2024	07.06.2024	

Студент _____ Ковальов М.Д.

Керівник роботи _____ Кадук О.В.

АНОТАЦІЯ

УДК 004.4:621.382

Ковальов М. Д. Програмний засіб для аналізу характеристик оперативної пам'яті. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп’ютерна інженерія», освітня програма «Системне програмування». Вінниця: ВНТУ, 2024. 84с.

На укр. мові. Бібліогр.: 22 назв; рис.: 21; табл. 1.

У бакалаврській кваліфікаційній роботі розроблено програмний засіб для аналізу характеристик оперативної пам'яті. Засіб дозволяє покращити обробку та аналіз даних оперативної пам'яті сучасних комп'ютерних систем. У загальній частині роботи розглянуто особливості сучасних типів оперативної пам'яті та обґрунтовано доцільність їх аналізу. У розрахунково-конструкторській частині виконано розробку моделі програмного засобу, описано діаграми Use-Case та обґрунтовано вибір архітектури. У технологічній частині описано процес створення графічного інтерфейсу користувача та основного коду програми.

Графічна частина складається з діаграми роботи програми.

Ключові слова: оперативна пам'ять, програмний засіб, Python, Qt, аналіз даних, ReRAM, графічний інтерфейс користувача.

ABSTRACT

Kovalov M. D. A software tool for analyzing the characteristics of RAM. Bachelor's qualification work in specialty 123 "Computer Engineering", educational program "System Programming". Vinnytsia: VNTU, 2024. 84 p.

In Ukrainian. Bibliography: 22 titles; Figures: 21; Table 1.

In the bachelor's qualification work, a software tool for analyzing the characteristics of RAM was developed. The tool allows to improve the processing and analysis of RAM data of modern computer systems. In the general part of the work, the features of modern types of RAM are considered and the feasibility of their analysis is substantiated. In the design and development part, we develop a model of the software tool, describe the Use-Case diagrams, and justify the choice of architecture. The technological part describes the process of creating a graphical user interface and the main program code.

The graphical part consists of a diagram of the program operation.

Keywords: RAM, software tool, Python, Qt, data analysis, ReRAM, graphical user interface.

ЗМІСТ

ВСТУП 8

1 АНАЛІЗ ОПЕРАТИВНОЇ ПАМ'ЯТІ У СУЧАСНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ 10

1.1 Енергозалежна пам’ять 11

1.2 Енергонезалежна пам'ять 13

1.2.1 Пам’ять EEPROM, FLASH та MLC FLASH 14

1.2.2 STT-MRAM 15

1.2.3 PCRAM 16

1.2.4 FeRAM 17

1.2.5 ReRAM 18

1.3 Огляд відомих програм аналогів 18

1.4 Постановка задач дослідження 21

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ДЛЯ АНАЛІЗУ ХАРАКТЕРИСТИК ОПЕРАТИВНОЇ ПАМ’ЯТІ 22

2.1 Вимоги до проекту розробки 22

2.2 Аналіз вхідних даних програми 23

2.3 Розробка моделі програмного засобу 26

2.4 Діаграма Use-Case для програмного засобу 28

2.5 Обґрунтування вибору архітектури програмного засобу 29

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСОБУ ДЛЯ АНАЛІЗУ ХАРАКТЕРИСТИК ОПЕРАТИВНОЇ ПАМ’ЯТІ 33

3.1 Обґрунтування вибору мови програмування 33

2.6 Обґрунтування вибору середовища програмування 35

2.7 Обґрунтування вибору фреймворку 38

2.8 Особливості програмної реалізації 40

3.4.1 Створення графічного інтерфейсу користувача 40
3.4.2 Створення основного коду програми 44
2.9 Аналіз отриманих результатів 52
ВИСНОВКИ 58
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 59
ДОДАТОК А Технічне завдання 61
ДОДАТОК В Діаграма роботи програми 65
ДОДАТОК Г Лістинг Г.1 — повний код файлу runs_many.py 66
ДОДАТОК Д Лістинг Д.1 — Повний код файлу current_hist_100mV _pos_many.py 75
ДОДАТОК Е Лістинг Е.1 — Повний код файлу conductive_hist_neg _C2C.py 78
ДОДАТОК Є Лістинг Є.1 — Повний код файлу current_hist_100mV _neg_many.py 81
ДОДАТОК Б Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень 84

ВСТУП

У сучасному світі, де інформаційні технології стрімко розвиваються, ефективність та швидкість роботи комп'ютерних систем значною мірою залежить від характеристик оперативної пам'яті. Новітні типи пам'яті стають все більш важливими завдяки своїм унікальним властивостям, таким як висока швидкість запису і зчитування, енергоефективність та тривалість зберігання даних.

Розробка програмного засобу для автоматизації процесу аналізу характеристик оперативної пам'яті дозволить значно підвищити ефективність цього процесу, зменшивши час та зусилля, необхідні для аналізу даних. Такий інструмент сприятиме більш точному та швидкому отриманню результатів, що є важливим для оптимізації технологій і їх подальшого застосування в нових розробках та системах.

Таким чином, **актуальність теми** полягає у необхідності вдосконалення методів аналізу характеристик пам`яті, розробці ефективного програмного засобу для обробки даних та візуалізації результатів, що сприятиме швидшому розвитку та впровадженню перспективних технологій в інформаційні системи.

Метою дослідження є вдосконалення процесу аналізу результатів вимірювання характеристик I-V (струм-напруга) комп'ютерної пам'яті ReRAM. Для досягнення цієї мети були поставлені наступні **завдання**:

— розробка коду для обробки та аналізу даних пам'яті ReRAM;

— реалізація програмного засобу з графічним інтерфейсом користувача (GUI) для зручного використання;

— інтеграція функцій для побудови графіків на основі вхідних даних у форматі Excel;

— проведення тестування програмного засобу для оцінки його роботи;

— оцінка результатів аналізу для визначення переваг та недоліків розробленого засобу;

Об’єкт дослідження — процес аналізу характеристик I-V (струм-

напруга) комп'ютерної пам'яті ReRAM.

Предмет дослідження — методи та програмні засоби для обробки та аналізу даних пам'яті ReRAM, а також побудова відповідних графіків на основі цих даних.

У процесі дослідження використовувались такі **методи**:

— методи математичного аналізу для обробки характеристик I-V;

— алгоритми побудови графіків та візуалізації даних;

— тестування програмного забезпечення для перевірки його працездатності та точності результатів;

— методи розробки графічного інтерфейсу користувача для забезпечення зручності використання програмного засобу.

Наукова новизна одержаних результатів полягає у вдосконаленні методу аналізу характеристик оперативної пам'яті шляхом автоматизації побудови графіків, що дозволило досягти більш точної та швидкої обробки даних, важливої для подальшого розвитку та оптимізації технології оперативної пам'яті.

1 АНАЛІЗ ОПЕРАТИВНОЇ ПАМ'ЯТІ У СУЧАСНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

Як згадувалося у вступі, розвиток технологій пам'яті є ключовим процесом, що свідчить про безперервну еволюцію комп'ютерних систем. Пам'ять з довільним доступом (ОЗП), що є невід'ємною частиною цих систем, відіграє фундаментальну роль у забезпеченні швидкості та продуктивності електронних пристроїв. У цьому розділі, присвяченому сучасним запам'ятовуючим пристроям, представлено огляд різних технологій, які домінують на ринку зберігання даних у сучасному цифровому світі.

Перший критерій поділу пам'яті, як показано на рисунку 1.1, — це поділ на енергозалежну та енергонезалежну пам'ять. Енергонезалежна пам'ять зберігає збережені дані навіть при відключенні живлення, що робить її ідеальною для довготривалого зберігання. З іншого боку, енергозалежна пам'ять вимагає постійного оновлення, а отже, і живлення, оскільки дані будуть втрачені при відключенні живлення. Як правило, найшвидші типи пам'яті є саме енергозалежними, в той час як енергонезалежна пам'ять здатна зберігати дані протягом більш тривалих періодів часу [1,2,3].

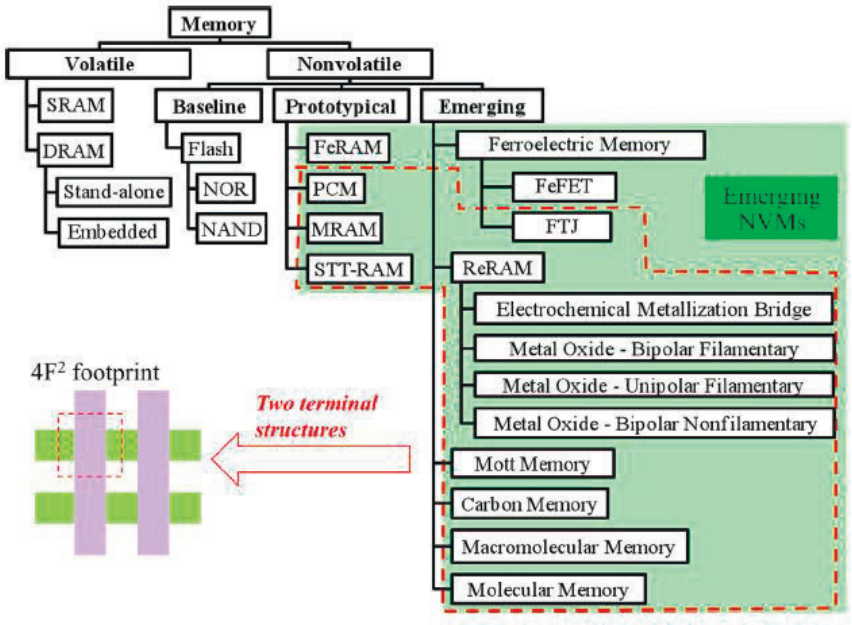


Рисунок 1.1 — Ієрархія поділу пам'яті комп'ютера

Оперативну пам'ять можна поділити на статичну SRAM (статична пам'ять з довільним доступом) та динамічну DRAM (динамічна пам'ять з довільним доступом). Як правило, вони повинні розташовуватися близько до процесора через критичні вимоги до низького часу доступу [4]. SRAM, зокрема, зазвичай вбудовується в процесор і використовується як кеш-пам'ять. DRAM, яка є оперативною пам'яттю більшої щільності, слугує основною оперативною пам'яттю і зазвичай підключається до процесора ззовні. Вона також характеризується гіршими часовими характеристиками порівняно з кеш-пам'яттю SRAM.

Слід також згадати про пам'ять NAND Flash і NOR Flash, які знайшли застосування в системах зберігання даних, таких як твердотільні накопичувачі та флеш-пам'ять. Ці технології пропонують високу ємність і низьке енергоспоживання, але, як правило, менш придатні для завдань, що вимагають швидкого доступу до даних, порівняно з традиційною оперативною пам’яттю. Слід також зазначити, що тривимірна структура флеш-пам'яті NAND дозволяє постійно знижувати вартість гігабайта ємності, що є однією з головних переваг цієї технології [5].

1.1 Енергозалежна пам’ять

DRAM — динамічна пам'ять з довільним доступом є класичним прикладом ефемерної пам'яті. Як випливає з назви, комірки пам'яті повинні динамічно оновлюватися, щоб зберігати дані. Комірки пам'яті DRAM зберігають інформацію за допомогою механізму перевантаження конденсатора, який піддається процесу саморозряду. Саме тому необхідні циклічні операції оновлення [5]. Безсумнівною перевагою DRAM є висока ефективна щільність упаковки, що досягається завдяки надзвичайно простій структурі однієї комірки. Їх також можна охарактеризувати дуже високою пропускною здатністю, що досягається завдяки інноваційним рішенням, які об'єднують процесор і HBM DRAM [6] з надширокою шиною даних (High Bandwidth Memory) в одному модулі (рис. 1.2).

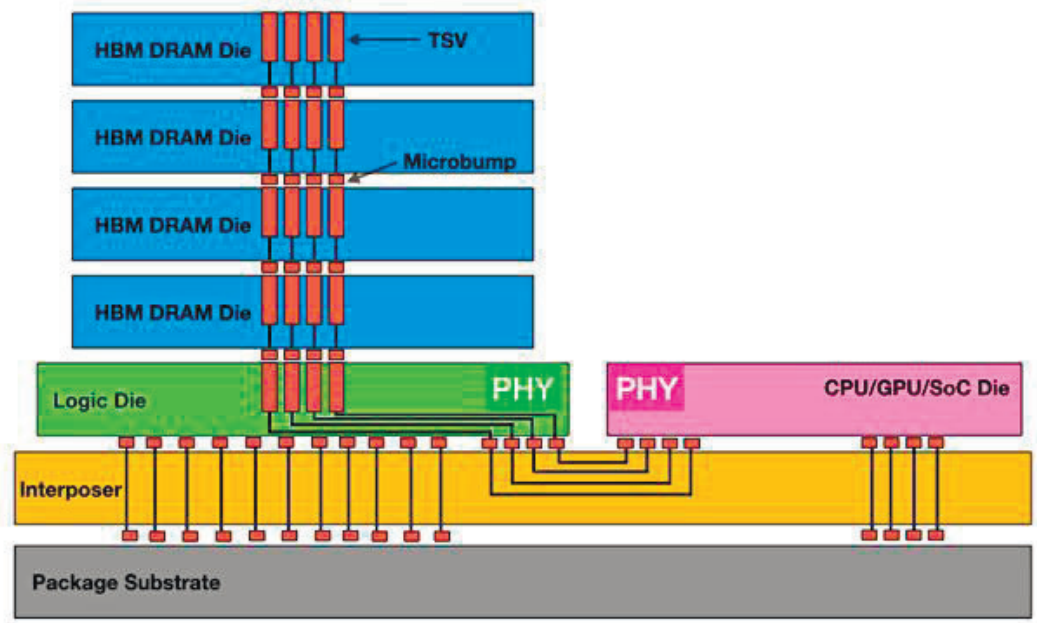


Рисунок 1.2 — Спрощена структурна схема системи пам'ять-процесор в комірках HBM DRAM

Сучасні DRAM, такі як згадані вище мікросхеми HBM, пропонують чудову продуктивність і швидкість доступу до даних. Однак, незважаючи на те, що вони інтегровані в один корпус з процесорною мікросхемою, це все ще дискретне рішення, тобто мікросхема пам'яті виготовляється за іншим технологічним процесом, ніж процесор. Виробництво DRAM за тим самим технологічним процесом, що й процесор, вимагало б додаткових етапів, що значно підвищило б його вартість. Крім того, інтеграція призвела б до значного збільшення кінцевого розміру мікросхеми, що спричинило б проблеми з виходом продукції, виробничими спредами і, як наслідок, подальше збільшення кінцевої ціни виробництва продукту. Тому, як правило, на ринку комерційних пристроїв домінують дискретні рішення, в яких процесор і пам'ять, щонайбільше, інтегровані в одному корпусі. Інтеграція DRAM у сучасні пристрої дозволяє досягти високої продуктивності та швидкості доступу до даних. Незважаючи на ці переваги, важливо відзначити, що виробництво DRAM в одному корпусі з процесором може призвести до складнощів і високої вартості. Інтеграція потребує додаткових технологічних

етапів та може збільшити кінцевий розмір мікросхеми. Це може вплинути на виробничі процеси та зробити продукт менш конкурентоспроможним на ринку. Тим не менш, технологія DRAM продовжує розвиватися і, безумовно, буде широко використовуватися в якості оперативної пам'яті в майбутніх поколіннях комп'ютерних систем, де розмір/ємність пам'яті даних є ключовим параметром [4,7].

Static Random Access Memory (SRAM) — ще одна категорія енергозалежної пам'яті, яка використовується там, де вимоги до високошвидкісного доступу до даних є критичними. Однією з областей її використання є кеш-пам'ять в процесорах. Принцип роботи комірок пам'яті SRAM суттєво відрізняється від раніше згаданої DRAM. В його основі лежить схема замикання, реалізована у вигляді закільцьованих КМОП-інверторів. Її складність більша, ніж у комірок пам'яті DRAM, але цей тип пам'яті можна виготовляти в тому ж технологічному процесі, що і процесор [4,8]. Її характеристиками є дуже низька затримка і майже миттєвий доступ до інформації.

Масштабування технологій пам'яті SRAM і DRAM, які є поширеними технологіями, що використовуються в сучасних комп'ютерних системах, стає все більш обмеженим. Проблема полягає у зростанні статичного енергоспоживання (комірки SRAM) та збільшенні струмів витоку (комірки DRAM). На сьогоднішній день це є основними викликами для розробників сучасних мікросхем в технології інтегральних схем [4].

1.2 Енергонезалежна пам'ять

До енергонезалежної пам'яті належать як пам'ять програм, так і пам'ять даних. Як згадувалося раніше, на відміну від енергозалежної пам'яті, енергонезалежна пам'ять дозволяє зберігати інформацію протягом тривалого часу. Пам'ять тільки для читання ROM (англ. Read Only-Memory), як правило, використовується для зберігання програм у комп'ютерних системах. Ці

програми постійно зберігаються в енергонезалежній пам'яті і запускаються при запуску системи. Це гарантує, що важливі інструкції не будуть втрачені при вимкненні живлення, що має вирішальне значення для надійності та функціональності комп'ютерної системи. З іншого боку, енергонезалежна пам'ять також використовується для зберігання інформації, яка повинна бути доступною при перезавантаженні системи. Наприклад, конфігурації системи, дані користувача, ліцензійні ключі або налаштування пристрою. Зараз популярним типом енергонезалежної пам'яті є флеш-пам'ять, яка зазвичай використовується в USB-накопичувачах, картах пам'яті та твердотільних накопичувачах. Існують й інші застарілі рішення, такі як EEPROM (англ. Electrically Erasable Programmable Read-Only Memory), але на сьогоднішній день вони майже всі замінені флеш-пам'яттю [9].

1.2.1 Пам’ять EEPROM, FLASH та MLC FLASH

EEPROM і флеш-пам'ять засновані на одному і тому ж принципі роботи. У транзистор вбудований плаваючий затвор. Зарядка або розрядка, тобто зміна стану, здійснюється на основі тунельних явищ Фаулера-Нордгейма і/або надбар'єрної провідності шляхом прикладання відповідного електричного поля до керуючого затвора. У старих типах EPROM розряд здійснюється за рахунок внутрішніх фотоелектричних явищ і поглинання ультрафіолету в потенціальній ямі. На рисунку 1.3 показано процес заряджання та розряджання комірки пам'яті [10].

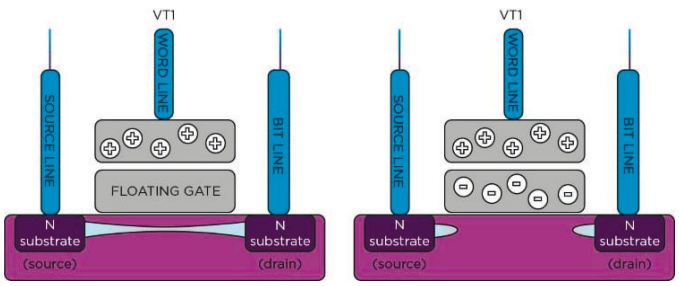


Рисунок 1.3 — Процес завантаження та розвантаження флеш-пам'яті .

Однією з перспективних технологій є MLC Flash (англ. multi-level cell), яка дозволяє зберігати кілька бітів в одній комірці пам'яті, але вимагає набагато складніших схем для виконання процесів запису та зчитування [11].

Однак існують альтернативні технології пам'яті, які використовують нові матеріали та механізми для зберігання інформації. До них відносяться сучасна магнітна STT-MRAM (англ. Spin Torque Transfer MRAM), фазозмінна оперативна пам'ять (англ. Phase-change RAM), резистивна оперативна пам'ять (англ. Resistive RAM), фероелектрична оперативна пам'ять (англ. ferroelectric RAM) та інші. Ці технології поєднують в собі швидкість SRAM, щільність DRAM і енергонезалежність флеш-пам'яті і тому стають дуже привабливими як потенційні альтернативи існуючій пам'яті в майбутніх комп'ютерних системах. Прогнозується, що вони зроблять революцію на ринку пам'яті, а енергоефективність може стати вирішальним фактором. Варто зазначити, що мікросхеми, засновані на вищезгаданих технологіях, вже сьогодні є комерційно доступними.

1.2.2 STT-MRAM

STT-MRAM — це новий тип магнітної пам'яті, який відрізняється високою швидкістю читання/запису, високою витривалістю та низьким споживанням струму при читанні. Зберігання або програмування STT-MRAM базується на властивостях магнітного тунельного переходу MTJ (англ. magnetic tunneling junction), в якому тонкий тунельний діелектрик, наприклад, MgO, затиснутий між двома феромагнітними шарами (рис. 1.4). Один феромагнітний прошарок, який називається опорним, має постійну намагніченість, тоді як намагніченість іншого вільного шару можна змінювати під час запису. Питомий опір тонкого діелектрика низький, коли намагніченість вільного і опорного шарів спрямована в одному напрямку, але коли напрямки намагніченості протилежні, питомий опір діелектрика високий. STT-MRAM є однією з перспективних технологій для застосування у

вбудованих системах, високопродуктивних комп'ютерах та інших пристроях, де вимоги до швидкодії, надійності та енергоефективності є критичними. Ця технологія може бути особливо корисною для розробки мобільних пристроїв, які потребують довгих інтервалів автономної роботи та швидкого доступу до даних. Також варто відзначити, що STT-MRAM має потенціал замінити традиційні види пам'яті, такі як DRAM та NAND Flash, завдяки своїм перевагам у швидкості, енергоефективності та тривалості. Розвиток цієї технології відкриває нові перспективи для розвитку електроніки та комп'ютерних систем у майбутньому.

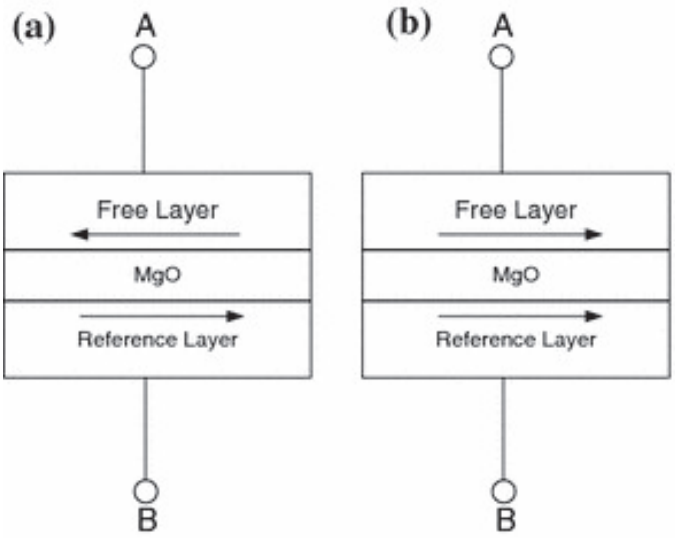


Рисунок 1.4 — Структура магнітного тунельного переходу. а) Високоомний стан. б) Низькоомний стан.

1.2.3 PCRAM

PCRAM — це пам'ять, робота якої заснована на фазових змінах халькогенідного сплаву GeSbTe (GST), з якого вона виготовлена — рис.1.5. Здатність зберігати дані досягається за рахунок різниці опорів між високоомною аморфною фазою та низькоомною кристалічною фазою. Фазовий перехідний матеріал кристалізується шляхом контрольованого нагрівання електричним імпульсом. Значення опору комірки PCRAM можна значною мірою контролювати, що призводить до здатності зберігати декілька

стабільних станів [13]. Таким чином, комірки PCRAM мають потенціал для більш високої ефективної щільності упаковки інформації, ніж звичайна двійкова пам'ять. PCRAM характеризується сумісністю з технологією CMOS, високою швидкістю і великою максимальною кількістю циклів читання/запису. Поточні дослідження цього типу пам'яті зосереджені на використанні механізму фазового переходу в 22 нм технологічному процесі [12].

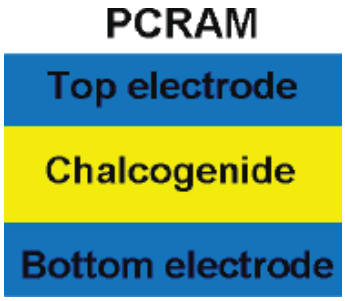


Рисунок 1.5 — Структура комірки пам'яті PCRAM.

1.2.4 FeRAM

Сегнетоелектрична пам'ять з довільним доступом, FeRAM або FRAM (рис. 1.6) — це енергонезалежна пам'ять, яка зберігає інформацію у вигляді поляризаційного стану активного шару сегнетоелектричного матеріалу. Сегнетоелектрик має бістабільні стани поляризації, які зберігаються за відсутності зовнішнього електричного поля. Подача напруги відповідної полярності на зовнішні електроди викликає зміну порядку доменів (локальних областей спонтанної електричної поляризації) в структурі, що призводить до зміни стану комірки [14,15]. Висока максимальна кількість циклів читання/запису, низьке енергоспоживання, малий час доступу та енергонезалежність є основними перевагами структур FeRAM. Пам'ять на основі фероелектриків є комерційно доступною вже більше 10 років, але сфера її застосування обмежена через недоліки цієї технології (розмір комірки і несумісність з технологією CMOS).

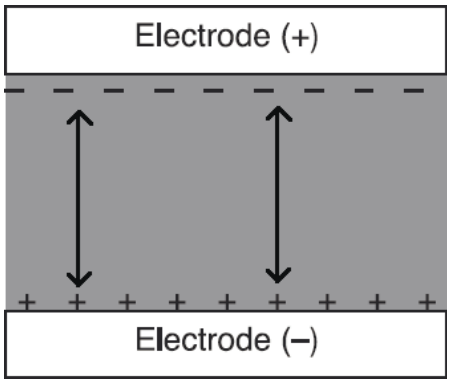


Рисунок 1.6 — Структура комірки пам'яті FeRAM

1.2.5 ReRAM

RRAM або ReRAM — це тип високошвидкісної, енергонезалежної пам'яті, комірки якої складаються з верхнього, зазвичай металевого, електрода, активного/діелектричного шару і нижнього електрода, виготовленого з металевого або напівпровідникового шару [16]. Поперечний переріз комірки такої пам'яті представлений на рисунку 1.7. Вона працює за принципом зміни питомого опору діелектрика, що відбувається під впливом прикладеної напруги [16]. В даний час ряд компаній, таких як Fujitsu, Sharp, Samsung та інші, працюють над розробкою пам'яті RRAM [4]. Технологія виробництва цього типу пам'яті сумісна з технологією CMOS, що є безумовною перевагою.

Рисунок — 1.7 Структура комірки пам'яті ReRAM.

1.3 Огляд відомих програм аналогів

Для аналізу характеристик оперативної пам'яті та інших електронних

компонентів існують кілька відомих програмних засобів, таких як TCAD Sentaurus та Keithley 4200 Software. Ці програми мають свої унікальні характеристики, переваги та недоліки, які будуть розглянуті нижче.

TCAD Sentaurus (рисунок 1.8) є передовим програмним засобом для моделювання напівпровідникових пристроїв, розробленим компанією Synopsys. Ця програма забезпечує повний цикл моделювання, включаючи фізичне моделювання процесів виготовлення пристроїв та електричне моделювання їх характеристик.

Основні характеристики TCAD Sentaurus:

— TCAD Sentaurus надає максимальний функціонал для моделювання різних напівпровідникових пристроїв, включаючи транзистори, діоди, пам'ять та інші компоненти, він підтримує широкий спектр фізичних моделей та параметрів, що дозволяє точно прогнозувати поведінку пристроїв під різними умовами;

— для ефективного використання TCAD Sentaurus потрібні глибокі знання фізики напівпровідників та навичок моделювання, користувачі повинні бути знайомі з складними фізичними моделями та алгоритмами, що використовуються у програмі;

— вартість ліцензії на TCAD Sentaurus є надзвичайно високою, що робить цей інструмент доступним переважно для великих компаній та наукових установ з великим бюджетом;

— через високу ціну та складність використання, доступ до TCAD Sentaurus обмежений. Програму зазвичай використовують у спеціалізованих науково-дослідних лабораторіях та великих корпораціях.

Keithley 4200 Software — це внутрішнє програмне забезпечення для керування приладами серії Keithley 4200A-SCS, що використовуються для вимірювання електричних характеристик напівпровідників, матеріалів та пристроїв. Ця програма забезпечує комплексний набір інструментів для аналізу даних, зібраних під час вимірювань (рисунок 1.9).

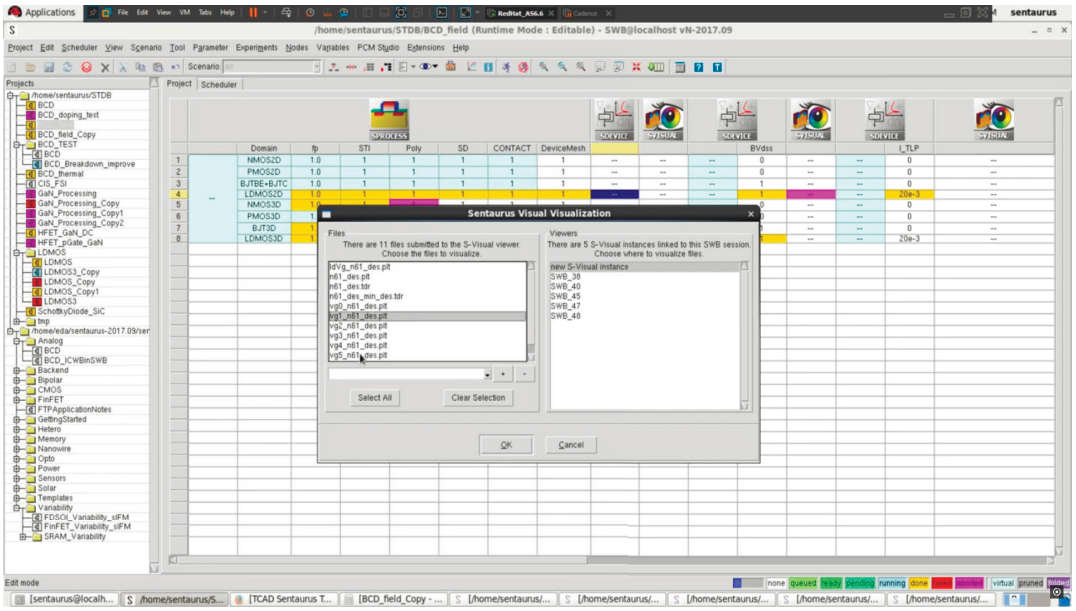


Рисунок 1.8 — Програма TCAD Sentaurus

Основні характеристики Keithley 4200 Software:

— Keithley 4200 Software є частиною апаратно-програмного комплексу приладів Keithley 4200A-SCS і призначене для управління та аналізу вимірювань безпосередньо на цих приладах;

— це програмне забезпечення не доступне для використання поза межами самих приладів Keithley. Це означає, що користувачі можуть виконувати аналіз тільки при наявності відповідного обладнання;

— функціонал Keithley 4200 Software обмежений в порівнянні з більш потужними програмами для моделювання.

Він здебільшого зосереджений на зборі та первинному аналізі даних, отриманих з вимірювань, і не надає розширених можливостей для моделювання та прогнозування поведінки пристроїв.

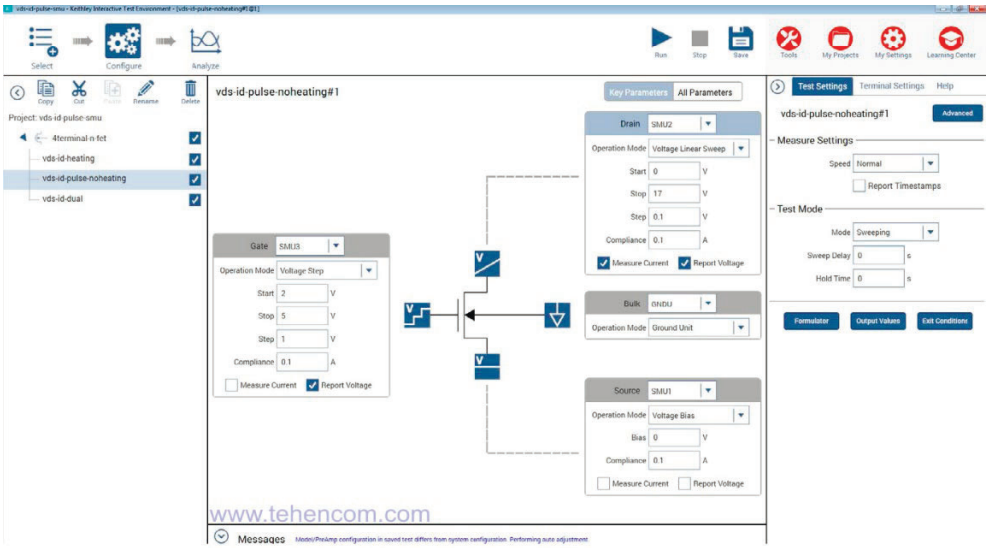


Рисунок 1.9 — Keithley 4200 Software

1.4 Постановка задач дослідження

Завершальним етапом аналітичного дослідження є формулювання завдань для проекту розробки на підставі аналізу існуючих аналогів та інструментальних засобів розробки для подібних проектів.

Постановка завдання:

— сформулювати основні функціональні вимоги до проекту, враховуючи аналіз існуючих аналогів та потреб користувачів;

— створити структуру додатку та розробити користувацький інтерфейс за допомогою фреймворку Qt, відповідно до вимог та специфікацій;

— провести розробку основного функціоналу, обробку файлів та створення графіків. Для цього використовувати середовище розробки Visual Studio Code та мову програмування Python 3;

— після завершення тестування проекту провести аналіз результатів та описати можливості подальшого розвитку додатку.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ДЛЯ АНАЛІЗУ ХАРАКТЕРИСТИК ОПЕРАТИВНОЇ ПАМ'ЯТІ

2.1 Вимоги до проекту розробки

Вимоги до проекту розробки визначають фундаментальні параметри та очікування, які додаються до програмного продукту. Це ключовий етап у процесі створення будь-якого програмного забезпечення, оскільки від якості та повноти вимог залежить успішність проекту. Вимоги відображають потреби користувачів, бізнес-цілі та технічні обмеження, які повинні бути враховані під час розробки.

Визначення вимог дозволяє створити чіткий план дій, встановити межі та напрямок для всього процесу розробки. Крім того, це допомагає забезпечити однозначне розуміння завдання як з боку розробників, так і з боку замовника. Недоліки у вимогах можуть призвести до непорозумінь, затримок у розробці та недоліків у фінальному продукті.

Вимоги можуть бути розділені на функціональні, які визначають функції та можливості програмного забезпечення, та нефункціональні, які описують якість, надійність, ефективність та інші аспекти системи. Під час визначення вимог важливо врахувати потреби користувачів, вимоги бізнесу, технічні обмеження та можливості розробки.

Виконання вимог є критичним кроком для успішної реалізації проекту. Команда розробників повинна ретельно аналізувати вимоги, визначити стратегії виконання та постійно забезпечувати відповідність продукту вимогам протягом усього процесу розробки. Це допоможе забезпечити успішну поставку продукту, задоволення потреб замовника та створення продукту відповідно до вимог і очікувань користувачів.

Програма повинна приймати вхідні дані у форматі Excel, забезпечуючи коректне зчитування та обробку різних форматів даних. Вона має розраховувати та будувати графіки звичайної характеристики I-V (струм-напруга), гістограми струмів, опору, динамічної провідності, а також функції

розподілу (Cumulative Distribution Function). Програма автоматично будує різні типи графіків на основі розрахованих даних з можливістю налаштування параметрів графіків. Інтерфейс користувача (GUI) повинен бути інтуїтивно зрозумілим та зручним, передбачаючи наявність бічної панелі для зручної навігації, анімацію елементів інтерфейсу та налаштування кольорової схеми. Програма забезпечує можливість зберігання результатів аналізу у зручних для подальшого використання форматах.

Інтерфейс повинен бути привабливим, забезпечуючи зручність та інтуїтивну зрозумілість для користувача. Важливо використовувати єдину кольорову схему для всіх елементів інтерфейсу та високу контрастність тексту і елементів керування. Інтерфейс має бути чітко структурованим, забезпечуючи легку навігацію, дотримуючись сучасних підходів до проектування інтерфейсу, таких як Material Design або Flat Design. Він повинен бути адаптивним, підтримуючи різні розміри екранів та роздільні здатності, забезпечуючи комфортне використання на різних пристроях.

Для реалізації обрано мову програмування, яка забезпечує широкі можливості для наукових обчислень та обробки даних. Як середовище розробки вибрано платформу, яка надає розширені можливості для налагодження та керування проектами. Для створення графічного інтерфейсу використовується фреймворк, який забезпечує потужні засоби для розробки інтуїтивно зрозумілих та привабливих інтерфейсів. Для обробки даних застосовуються сучасні бібліотеки, які забезпечують ефективну роботу з великими обсягами інформації та створення високоякісних графіків.

2.2 Аналіз вхідних даних програми

Аналіз вхідних даних є ключовим етапом у розробці програмного засобу, оскільки від якості цього процесу залежить точність та достовірність результатів аналізу. Важливо ретельно дослідити формати даних, їх структуру та характеристики, щоб забезпечити правильну обробку та інтерпретацію

інформації.

Першим кроком є визначення типів даних, які будуть використовуватися в програмі. Це можуть бути числа, текстові дані, дати, графіки тощо.

Потім слід ретельно вивчити структуру вхідних даних, їхній формат та способи представлення. Наприклад, якщо дані надходять у форматі Excel, необхідно врахувати особливості цього формату та забезпечити їх коректне зчитування.

Крім того, важливо розібратися в можливих варіантах обробки помилок та аномалій, які можуть зустрітися у вхідних даних. Це допоможе побудувати ефективні механізми валідації та очищення даних перед їх подальшою обробкою.

Наступним етапом є визначення потреб у додатковій підготовці даних для аналізу, такі як агрегація, фільтрація, перетворення тощо. Це може включати створення нових ознак на основі наявних даних або групування даних за певними критеріями.

Для аналізу характеристик оперативної пам’яті ReRAM розглянемо вхідні дані, отримані з пристрою Keithley 4200. Ці дані містять важливу інформацію для побудови графіків та проведення аналізу характеристик.

Основними колонками у вхідному файлі є Time, AI (Current) та AV (Voltage).

Колонка Time містить час вимірювання, представлений у форматі наукової нотації, що дозволяє легко обробляти та аналізувати часові ряди даних. Колонка AI (Current) містить виміряний струм, представлений у амперах, що є основним параметром для аналізу електричних характеристик. Колонка AV (Voltage) містить виміряну напругу, представлена у вольтах, що є ключовим параметром для побудови графіків I-V та аналізу поведінки ReRAM під різними умовами.

Приклад вхідних даних наведено у таблиці 3.

Таблиця 3 — Фрагмент набору вхідних даних

Time	AI	AV
10,2453E+0	4,4160E-6	000,0000E+0
10,5490E+0	-188,7380E-6	-10,0000E-3
10,6773E+0	-374,1548E-6	-20,0000E-3
10,9591E+0	-559,8964E-6	-30,0000E-3
11,0876E+0	-745,7309E-6	-40,0000E-3
11,2162E+0	-931,6441E-6	-50,0000E-3
11,3449E+0	-1,1175E-3	-60,0000E-3

Колонка Time містить часові мітки, які вказують момент часу, коли було проведено вимірювання. Це дозволяє відстежувати зміну характеристик струму та напруги у часі.

Колонка струму AI (Current) містить значення, які показують силу струму у певний момент часу. Ці значення є ключовими для аналізу характеристик I-V (струм-напруга) та подальшого розрахунку гістограм струмів, опору, динамічної провідності тощо.

Колонка напруги AV (Voltage) містить значення напруги у певний момент часу. Аналіз змін напруги дозволяє будувати функцію розподілу та інші залежності.

Вхідні дані використовуються для побудови графіків I-V, розрахунку гістограм та функцій розподілу. Графіки струм-напруга (I-V) допомагають візуалізувати залежність між струмом і напругою, що є основою для подальшого аналізу. Це дозволяє зрозуміти поведінку ReRAM-комірок під різними напругами. Гістограми струмів, опору та динамічної провідності забезпечують детальнішу інформацію про розподіл цих параметрів у часі, виявляючи тенденції та аномалії. Функція розподілу (Cumulative Distribution Function) дає можливість побачити, як розподіляються значення струмів та напруги, і визначити ймовірність їх появи в певних діапазонах. Це допомагає оцінити стабільність і надійність пам'яті ReRAM, а також ідентифікувати критичні точки в її роботі.

Аналіз вхідних даних є критично важливим етапом для розробки

програми. Вхідні дані, отримані з пристрою Keithley 4200, містять необхідну інформацію для проведення детального аналізу характеристик оперативної пам’яті ReRAM. Правильна обробка та візуалізація цих даних дозволяє отримати точні результати та зробити обґрунтовані висновки щодо характеристик пам’яті.

2.3 Розробка моделі програмного засобу

UML (Unified Modeling Language) є стандартною мовою моделювання, яка використовується в розробці програмного забезпечення для візуалізації, специфікації, конструювання та документування програмних систем. Вона надає стандартизовані засоби для вираження архітектури, структури та поведінки програмних систем.

UML дозволяє розробникам створювати різноманітні типи діаграм для моделювання різних аспектів програмного забезпечення, включаючи структурні, поведінкові та діаграми взаємодії. До найпоширеніших типів UML-діаграм відносяться діаграми класів, діаграми взаємозв'язків, діаграми діяльності, діаграми взаємодії та багато інших.

UML дозволяє представити складні структури та процеси програмного забезпечення у вигляді зрозумілих візуальних моделей, що полегшує розуміння та комунікацію між розробниками. За допомогою UML можна визначити та уточнити вимоги до програмного продукту, виражаючи їх у вигляді діаграм, таких як діаграми варіантів використання та діаграми активності. UML дозволяє створювати діаграми класів та компонентів, які допомагають в проектуванні архітектури програмного забезпечення та його компонентів. UML може служити засобом для створення документації проекту, описуючи його структуру, функції та взаємодію між компонентами [19].

Використання UML-діаграм, таких як діаграми діяльності, допомагає у кращому розумінні функціональності системи та полегшує процес реалізації

програмного забезпечення. Це дозволяє розробникам ефективніше співпрацювати та забезпечує високу якість результуючого продукту. Розробка моделі програмного засобу (рисунок 2.1) у вигляді UML діаграми є важливим кроком у проектуванні програми. Вона дозволяє чітко визначити послідовність дій та взаємодію користувача, що сприяє ефективній розробці та вдосконаленню програмного забезпечення. Використання UML діаграми діяльності забезпечує краще розуміння функціональності системи та полегшує процес її реалізації.

У діаграмі подані основні етапи роботи системи, такі як завантаження вхідних даних, попередня обробка даних, аналіз даних, побудова графіків і відображення результатів. Спочатку програма отримує вхідні дані у форматі Excel, які містять інформацію про час, струм та напругу. На етапі попередньої обробки проводиться валідація та нормалізація даних для забезпечення їхньої коректності та узгодженості. Після цього програма виконує різноманітні розрахунки, включаючи побудову графіків I-V характеристик, розрахунок гістограм струмів, опору та динамічної провідності.

Результати аналізу відображаються у вигляді графіків, що дозволяють користувачу візуалізувати залежності між різними параметрами. Графічний інтерфейс користувача дозволяє зручно переглядати результати аналізу, зокрема звичайну характеристику I-V, гістограми та функції розподілу.

Діаграма UML допомагає візуалізувати взаємодію між компонентами системи та послідовність їх виконання. Вона створює загальне уявлення про роботу системи та сприяє кращому розумінню її функціональності. На діаграмі відображені такі компоненти та їх взаємодія: зчитування вхідних даних із файлу Excel та передача їх у систему, виконання операцій з обробки даних, проведення розрахунку та аналіз вхідних даних, генерація необхідних графіків та гістограм, а також відображення результатів аналізу у вигляді графіків та іншої візуальної інформації. Діаграма роботи програми наведена у додатку В.

2.4 Діаграма Use-Case для програмного засобу

Use-Case діаграма є одним із основних інструментів моделювання в об'єктно-орієнтованому аналізі та проектуванні. Вона використовується для опису функціональних вимог системи та ілюструє взаємодію між користувачами (акторами) та системою. Кожен юз-кейс представляє окрему функцію або послугу, яку надає система з точки зору користувача. Актори можуть бути як реальними користувачами, так і іншими системами, що взаємодіють з моделюваною системою.

Основна мета Use-Case діаграми — забезпечити наочне уявлення про те, які функції повинні бути реалізовані в системі, а також про те, як користувачі взаємодіють з цими функціями. Це дозволяє всім учасникам проекту (розробникам, замовникам, аналітикам) мати спільне розуміння вимог до системи (рисунок 2.1).

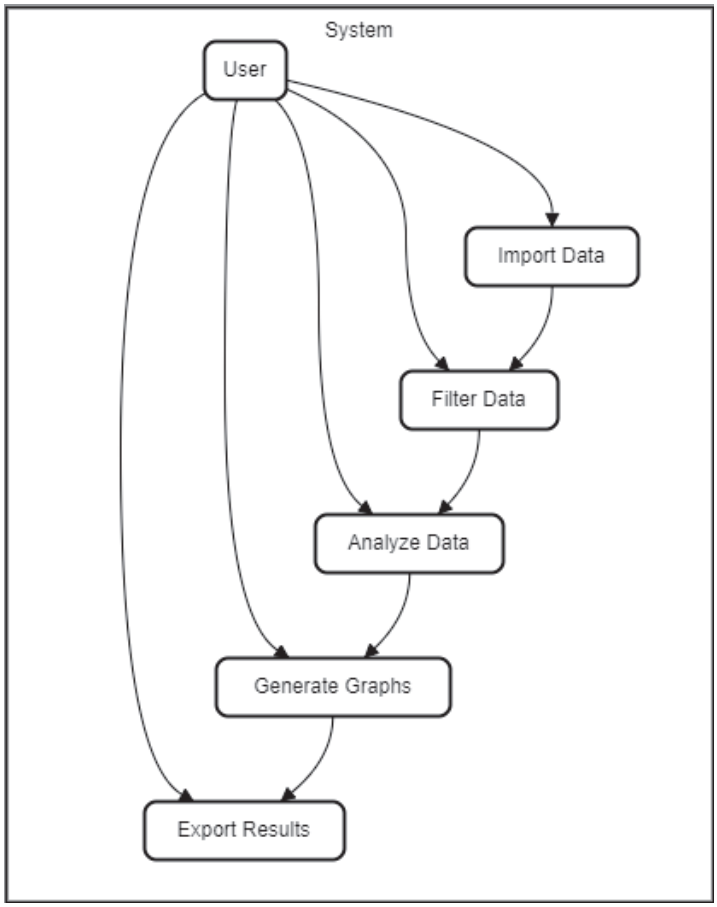


Рисунок 2.1 — Use-Case діаграма

2.5 Обґрунтування вибору архітектури програмного засобу

Розробка десктопного застосунку вимагає ретельного вибору архітектури, яка забезпечить ефективну та надійну роботу системи. Важливо враховувати специфіку десктопних програм, зокрема, відсутність веб-технологій та необхідність забезпечення високої продуктивності і стабільності. Нижче наведено обґрунтування вибору архітектури для нашого десктопного застосунку, враховуючи його специфіку та вимоги.

Продуктивність і відгук системи є критично важливими аспектами для десктопних застосунків. Вони повинні забезпечувати швидку відповідь на дії користувача та миттєву обробку даних. Обрана архітектура має мінімізувати затримки та забезпечити високу швидкість виконання. Відсутність необхідності комунікації через мережу дозволяє уникнути мережевих затримок, що є важливою перевагою для десктопних застосунків. Крім того, десктопні застосунки можуть ефективно використовувати ресурси комп'ютера, такі як процесор і оперативна пам'ять, для забезпечення оптимальної продуктивності

Стабільність і надійність також є ключовими факторами. Застосунок повинен працювати стабільно без збоїв та помилок. Архітектура повинна включати механізми обробки винятків та відновлення після помилок, що забезпечить безперервну роботу системи. Відсутність передачі даних через мережу знижує ризики, пов'язані з безпекою, оскільки всі дані зберігаються локально, що підвищує захищеність інформації. Важливо також передбачити регулярне резервне копіювання даних для запобігання їх втраті у випадку збоїв або несправностей обладнання.

Розширюваність і гнучкість є важливими для забезпечення довгострокової життєздатності застосунку. Архітектура повинна підтримувати модульний підхід, що дозволяє легко додавати нові функції та розширювати можливості застосунку без значних змін у базовому коді. Це також дає можливість швидко реагувати на зміни вимог користувачів та ринку, вносячи

необхідні зміни та оновлення в систему. Гнучка архітектура дозволяє інтегрувати нові технології та інструменти, що можуть з’явитися в майбутньому, з мінімальними зусиллями.

Супровід і розвиток застосунку повинні бути зручними та ефективними. Архітектура повинна бути зрозумілою і легкою для супроводу, що спрощує процес внесення змін, виправлення помилок та оновлення системи. Важливо забезпечити належну документацію архітектури, щоб розробники могли швидко зрозуміти структуру системи та принципи її роботи. Добре документована архітектура сприяє швидшій адаптації нових членів команди розробників і покращує загальну ефективність процесу розробки.

Монолітна архітектура є традиційним підходом, де весь застосунок виконується як єдиний блок. Цей підхід добре підходить для невеликих та середніх десктопних програм, які не потребують складного масштабування. Вона дозволяє легко управляти залежностями між компонентами та спрощує процес розгортання, оскільки всі функції зібрані в одному виконуваному файлі. Однак, для великих проектів, що потребують високої гнучкості та масштабованості, може бути доцільніше розглянути альтернативні архітектурні підходи, такі як мікросервіси або шарова архітектура.

Монолітна архітектура має такі переваги: весь код зберігається в одному місці, що полегшує процес розробки та супроводу. Відсутність потреби в мережевих запитах забезпечує високу швидкість роботи. Управління даними спрощене, оскільки всі дані зберігаються в єдиній базі. Проте, є й недоліки: важко масштабувати окремі частини системи. Крім того, зміни в одній частині можуть впливати на інші частини системи, що може ускладнювати підтримку та оновлення програмного забезпечення (рисунок 2.2).

Архітектура з розділенням на шари забезпечує організацію коду у вигляді окремих шарів, кожен з яких виконує свою роль. Цей підхід дозволяє чітко розділити логіку бізнесу, доступ до даних та користувацький інтерфейс (рисунок 2.3).

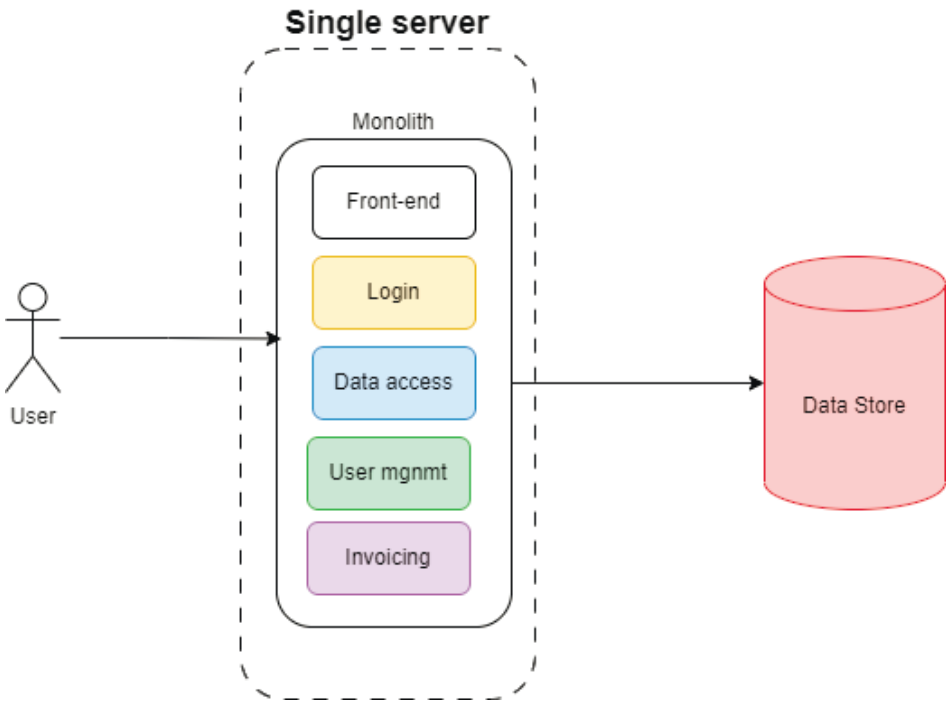


Рисунок 2.2 — Схематична ілюстрація монолітної архітектури

Архітектура з розділенням на шари має такі переваги: кожен шар відповідає за свою частину функціоналу, що полегшує супровід та розвиток системи. Це дозволяє легко додавати нові функції, змінювати або розширювати існуючі. Окремі компоненти системи легше тестувати, що підвищує надійність і якість програмного забезпечення. Проте, є і недоліки: така архітектура може бути складнішою у розробці та налагодженні порівняно з монолітною архітектурою. Крім того, при неправильній реалізації можуть виникати затримки через обробку запитів між шарами, що може вплинути на загальну продуктивність системи.

Для розроблюваного застосунку, який не використовує веб-технології та потребує високої продуктивності, надійності та легкості супроводу, оптимальним вибором є архітектура з розділенням на шари.

Вона забезпечує необхідну гнучкість та модульність, дозволяючи легко додавати нові функції та підтримувати систему в довгостроковій перспективі.

Цей підхід також спрощує тестування та внесення змін, що є важливим для стабільної та безперервної роботи застосунку.

На цьому етап проектування закінчено. Було визначено ключові вимоги до проекту розробки програмного засобу для аналізу характеристик оперативної пам’яті.

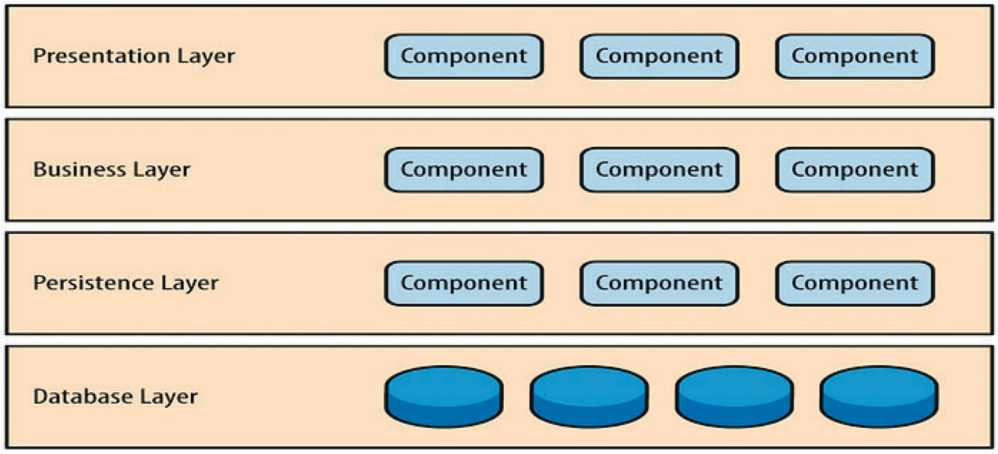


Рисунок 2.3 — Схематична ілюстрація багаторівневої архітектури

Описано, які функції повинні бути реалізовані для забезпечення ефективного аналізу та візуалізації даних. Було проведено детальний аналіз вхідних даних, отриманих з пристрою, що включають час, струм та напругу. На основі цього аналізу розроблено модель програмного засобу у вигляді UML діаграми, яка візуалізує взаємодію між основними компонентами програми.

Цей підхід дозволив чітко визначити послідовність дій та взаємодію між модулями системи, що сприяє кращому розумінню її функціональності та ефективності подальшої реалізації. UML діаграма діяльності допомагає створити загальне уявлення про роботу програмного засобу та забезпечує основу для його подальшої розробки та вдосконалення.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСОБУ ДЛЯ АНАЛІЗУ ХАРАКТЕРИСТИК ОПЕРАТИВНОЇ ПАМ’ЯТІ

3.1 Обґрунтування вибору мови програмування

Для реалізації програмного засобу для аналізу характеристик оперативної пам'яті було обрано мову програмування Python.

Python — це високорівнева мова програмування загального призначення, розроблена Гвідо ван Россумом і вперше випущена в 1991 році. За роки свого існування Python став однією з найпопулярніших мов програмування у світі, завдяки своїй простоті, ефективності та широкому спектру застосувань. Нижче наведено основні характеристики, які роблять Python особливо привабливим для розробників:

Читабельний синтаксис: Python має чіткий і лаконічний синтаксис, що робить код легким для читання і написання. Це дозволяє розробникам швидко розуміти та змінювати код, знижуючи ймовірність помилок.

Інтерпретована мова: Python є інтерпретованою мовою, що означає, що код виконується рядок за рядком. Це спрощує налагодження та тестування програм.

Кросплатформенність: Python працює на всіх основних операційних системах, таких як Windows, macOS та Linux, що дозволяє розробникам створювати кросплатформенні рішення.

Величезна стандартна бібліотека: Python має велику стандартну бібліотеку, яка надає широкий спектр функціональних можливостей для виконання різноманітних завдань без потреби у додаткових бібліотеках.

Активна спільнота: Python має одну з найбільших спільнот розробників, що забезпечує велику кількість ресурсів для навчання, документації та підтримки.

Pandas — це потужна бібліотека для аналізу та обробки даних у Python, створена для полегшення роботи з табличними даними [22]. Вона надає гнучкі та ефективні засоби для маніпуляції даними, що робить її незамінною для

багатьох завдань у сфері обробки даних. Основні особливості Pandas включають:

DataFrame та Series: Pandas вводить дві основні структури даних — DataFrame та Series. DataFrame є двовимірною структурою даних, аналогічною таблиці в базі даних або електронній таблиці, а Series є одномірною структурою даних.

Інтеграція з іншими бібліотеками: Pandas легко інтегрується з іншими бібліотеками для наукових обчислень, такими як NumPy та SciPy, що дозволяє ефективно проводити числові розрахунки та статистичний аналіз.

Обробка відсутніх даних: Pandas надає інструменти для роботи з відсутніми даними, що включають заповнення, видалення та інші методи обробки відсутніх значень.

Групування та агрегація даних: Pandas дозволяє легко групувати дані за певними критеріями та виконувати різноманітні агрегації, що є важливим для аналізу великих наборів даних.

Читання та запис даних: Pandas підтримує читання та запис даних з різних форматів, включаючи CSV, Excel, SQL бази даних та інші.

Matplotlib — це популярна бібліотека для створення статичних, анімаційних та інтерактивних візуалізацій у Python [20]. Вона надає безліч інструментів для створення різноманітних типів графіків, що робить її важливою для візуалізації даних. Основні можливості Matplotlib включають:

Широкий вибір графіків: Matplotlib дозволяє створювати різноманітні графіки, включаючи лінійні графіки, гістограми, розсіювання, коробкові діаграми та багато інших.

Висока налаштовуваність: Matplotlib надає можливість детально налаштовувати зовнішній вигляд графіків, включаючи кольори, шрифти, маркери та інші елементи.

Інтеграція з іншими бібліотеками: Matplotlib легко інтегрується з іншими бібліотеками для візуалізації та обробки даних, такими як Pandas та Seaborn, що дозволяє створювати більш складні та інформативні графіки.

Підтримка інтерактивних графіків: Matplotlib дозволяє створювати інтерактивні графіки, які можна використовувати в Jupyter Notebook для детального аналізу даних.

xlrd — це бібліотека для читання даних з файлів Excel у форматах .xls та .xlsx у Python [21]. Вона забезпечує простий та ефективний спосіб доступу до даних з електронних таблиць, що робить її корисною для роботи з вхідними даними у форматі Excel. Основні можливості xlrd включають:

Читання даних з різних форматів Excel: xlrd підтримує читання даних з файлів у форматах .xls та .xlsx, що забезпечує гнучкість у роботі з різними версіями Excel.

Підтримка великих наборів даних: xlrd дозволяє ефективно обробляти великі набори даних, що є важливим для аналізу великих обсягів інформації.

Легке інтегрування з іншими бібліотеками: xlrd легко інтегрується з іншими бібліотеками для обробки даних, такими як Pandas, що дозволяє швидко завантажувати та аналізувати дані з Excel-файлів.

Гнучкість у доступі до даних: xlrd надає можливість гнучко доступати до даних у електронних таблицях, включаючи можливість читання окремих комірок, рядків та стовпців.

2.6 Обґрунтування вибору середовища програмування

Середовищем програмування для розробки програмного засобу для аналізу характеристик оперативної пам'яті обрано Visual Studio Code (VS Code) (рисунок 3.1). Це інтегроване середовище розробки (IDE), створене компанією Microsoft [18]. VS Code є популярним середовищем серед розробників завдяки своїм широким можливостям, зручному інтерфейсу та підтримці численних мов програмування, включаючи Python.

VS Code має численні функції та можливості, які полегшують роботу розробників:

VS Code має потужний редактор коду з підсвічуванням синтаксису,

автодоповненням, перевіркою помилок та багатьма іншими функціями, що роблять написання коду швидшим та зручнішим. Додаткові можливості редактора включають інтелектуальне автодоповнення: завдяки підтримці IntelliSense, VS Code пропонує контекстно-залежні підказки, автодоповнення та документацію прямо під час написання коду. Підсвічування синтаксису для різних мов програмування допомагає розробникам швидше орієнтуватися в коді. Інтерактивний дебагер дозволяє налагоджувати код з можливістю встановлення точок зупину, перегляду значень змінних та виконання крокування по коду, що значно покращує процес виявлення та виправлення помилок.

VS Code дозволяє легко створювати, керувати та налаштовувати проекти Python. Серед можливостей керування проектами виділяються інтеграція з Git, що забезпечує вбудовану підтримку систем контролю версій і дозволяє зручно працювати з Git, виконувати коміти, злиття та інші операції безпосередньо в IDE. Підтримка віртуальних середовищ Python, таких як virtualenv, дозволяє ізолювати залежності проектів та уникати конфліктів між ними. Крім того, VS Code підтримує численні інструменти для розробки, такі як Docker, Jupyter, та інші, через офіційні та сторонні розширення.

VS Code інтегрується з різними інструментами та фреймворками, що використовуються в екосистемі Python, дозволяючи легко налаштовувати середовище розробки під конкретні потреби проекту. Наприклад, офіційне розширення для Python надає всі необхідні інструменти для розробки, налагодження та тестування Python-коду. Крім того, VS Code підтримує інтерактивні Jupyter Notebooks, що дозволяє виконувати та візуалізувати код безпосередньо в середовищі VS Code, забезпечуючи зручний та ефективний процес розробки.

VS Code використовує статичний аналіз коду для виявлення помилок, порушень стилів та інших проблем. Він надає підказки та рекомендації для покращення якості коду. Основні функції аналізу коду включають: вбудовану підтримку лінтерів (наприклад, pylint, flake8), що допомагають знаходити

синтаксичні та логічні помилки, інструменти для статичного аналізу, що дозволяють знаходити потенційні проблеми в коді, такі як невикористовувані змінні або неправильні виклики функцій.

VS Code надає широкі можливості для навігації по проекту та переходу до визначень функцій, класів та змінних. Серед основних можливостей є перехід до визначень, що дозволяє швидко переходити до визначень функцій, класів та змінних за допомогою простих команд або кліків.

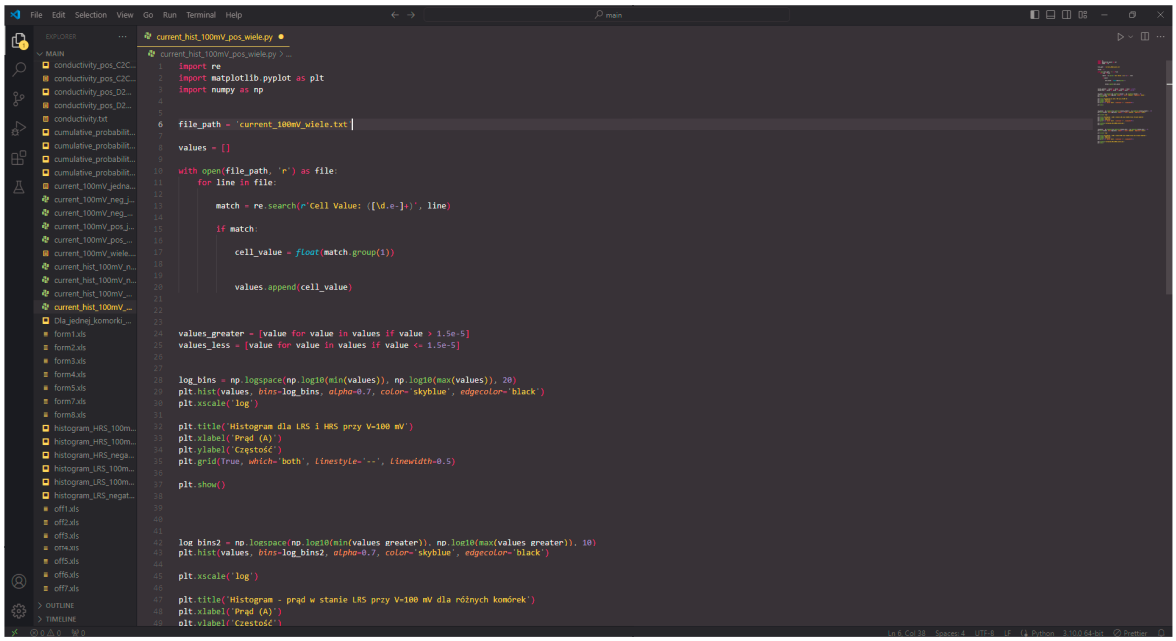


Рисунок 3.1 — Інтерфейс Visual Studio Code

Крім того, символічна навігація забезпечує швидкий доступ до всіх символів у файлі або проекті через інтегровану панель навігації.

VS Code дозволяє розширювати свої можливості за допомогою плагінів, що означає можливість додавання нових функцій, інтеграції зовнішніх інструментів та налаштування робочого середовища під свої потреби. Наприклад, розширення для Python надає інструменти для редагування, налагодження та тестування Python-коду. Додатково, розширення Jupyter додає підтримку Jupyter Notebooks, що дозволяє працювати з ними прямо в VS Code. Інше популярне розширення - Docker - забезпечує інтеграцію з Docker для керування контейнерами та образами.

VS Code є потужним та зручним інструментом для розробки Python-програм, надаючи розширені можливості для редагування коду, налагодження, керування проектами та інтеграції з іншими інструментами.

Це середовище є ідеальним вибором для розробки програмного засобу для аналізу характеристик оперативної пам'яті, забезпечуючи ефективну роботу з мовою Python та відповідними бібліотеками.

2.7 Обґрунтування вибору фреймворку

Для розробки графічного інтерфейсу користувача (GUI) програмного засобу для аналізу характеристик оперативної пам'яті було обрано фреймворк Qt. Qt є потужним кросплатформенним інструментарієм для створення GUI, що забезпечує високу продуктивність і широкий набір функціональних можливостей [17]. Вибір Qt обумовлений декількома ключовими перевагами, які наведені нижче.

Однією з головних переваг Qt є його кросплатформенність. Qt дозволяє розробляти програми, що працюють на різних операційних системах, включаючи Windows, macOS, Linux та інші. Це забезпечує широку аудиторію користувачів та спрощує процес розгортання програмного забезпечення на різних платформах без значних змін у коді.

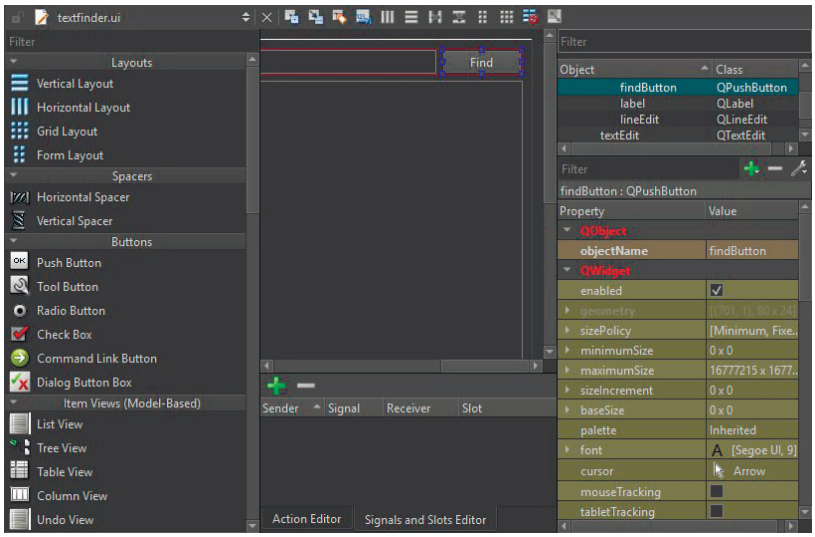


Рисунок 3.2 — Інтерфейс Qt Designer

Qt надає розробникам багатий набір інструментів для створення інтуїтивно зрозумілих та привабливих графічних інтерфейсів. Однією з основних можливостей є гнучкий конструктор інтерфейсів: Qt Designer, який є інструментом для візуального проєктування інтерфейсів, що дозволяє легко створювати та налаштовувати компоненти GUI. Також Qt надає широкий набір вбудованих віджетів, таких як кнопки, поля вводу, таблиці, графіки тощо, що дозволяє створювати функціональні інтерфейси з мінімальними зусиллями. Додатково, Qt підтримує стилізацію, що дозволяє налаштовувати вигляд додатка за допомогою стилів і тем, забезпечуючи можливість створення унікальних та привабливих дизайнів.

Qt забезпечує високу продуктивність додатків завдяки оптимізованим механізмам рендерингу графіки та обробки подій. Це дозволяє створювати швидкі та реагуючі інтерфейси, що є особливо важливим для програм, які обробляють великі обсяги даних або виконують складні розрахунки в реальному часі.

Хоча Qt спочатку був розроблений для мови програмування C++, він також має підтримку для інших мов, включаючи Python через бібліотеку PyQt або PySide. Це дозволяє розробникам використовувати потужні можливості Qt разом із зручністю та простотою мови Python.

Qt має добре документовану базу знань та активну спільноту розробників, що забезпечує швидкий доступ до ресурсів, прикладів коду та підтримки. Це значно спрощує процес навчання та розв'язання проблем, що можуть виникати під час розробки.

Qt дозволяє легко інтегруватися з іншими технологіями та бібліотеками, що є корисним для створення комплексних додатків. Наприклад, інтеграція з бібліотеками для обробки даних (Pandas), візуалізації (Matplotlib) та роботи з Excel (xlrd) забезпечує повний цикл обробки та аналізу даних у межах одного додатка.

Таким чином, вибір фреймворку Qt для розробки графічного інтерфейсу користувача програмного засобу для аналізу характеристик оперативної

пам'яті є виправданим і доцільним. Qt забезпечує кросплатформенність, потужний набір інструментів для створення GUI, високу продуктивність, підтримку мов програмування та широкий набір можливостей для інтеграції з іншими технологіями. Це дозволяє створити зручний, ефективний та привабливий інтерфейс користувача, що задовольняє всі вимоги проекту.

2.8 Особливості програмної реалізації

У цьому підрозділі розглянуто основні аспекти створення графіків для аналізу характеристик оперативної пам'яті, а також процес створення графічного інтерфейсу користувача з використанням фреймворку Qt та Qt Designer.

3.4.1 Створення графічного інтерфейсу користувача

Для створення зручного та функціонального графічного інтерфейсу користувача (GUI) було використано фреймворк Qt разом з інструментом Qt Designer (рисунок 3.1). Цей підрозділ описує процес розробки інтерфейсу, зокрема створення бічної панелі, анімацію її елементів, а також налаштування кольорів вікна та його компонентів.

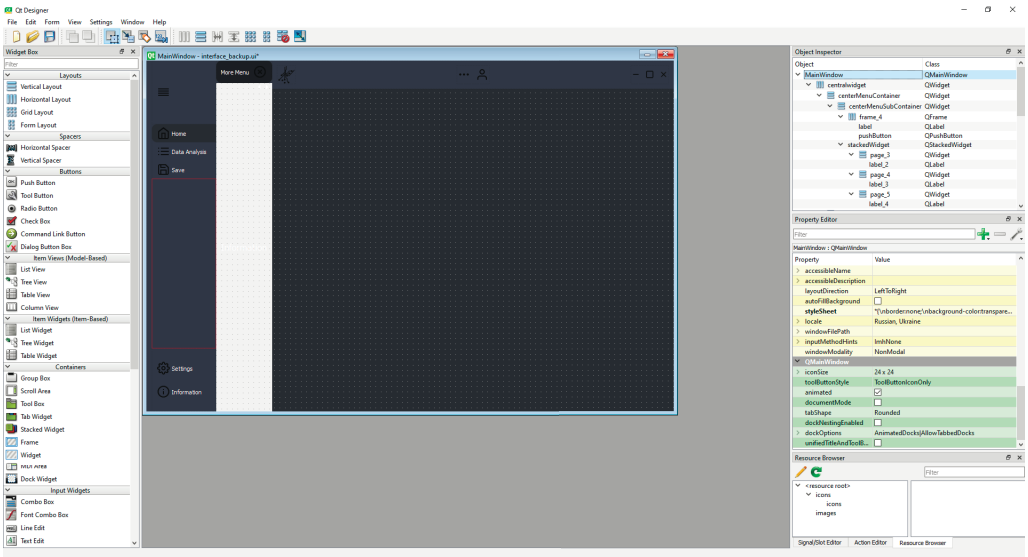


Рисунок 3.3 — Приклад роботи з Qt Designer

В процесі створення графічного інтерфейсу основну увагу було приділено бічній панелі, яка забезпечує зручну навігацію між різними розділами програми. Також було покращено візуальний дизайн та додано нові інтерактивні елементи. Бічна панель має інтуїтивно зрозумілу структуру, що дозволяє користувачам швидко знаходити потрібні розділи. Це сприяє підвищенню загальної зручності користування та ефективності роботи з програмою.

У Qt Designer було створено новий проект, де до головного вікна додано контейнер для sidebar за допомогою QFrame. Цей контейнер розміщено з лівого боку головного вікна (рисунок 3.3).

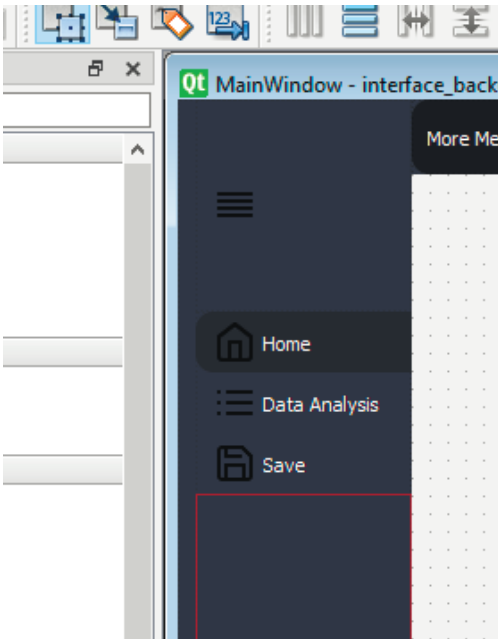


Рисунок 3.4 — Створення sidebar

В контейнері розміщено кілька кнопок, таких як QPushButton, для забезпечення зручного доступу до різних розділів програми, зокрема до розділів "Головна", "Аналіз даних" та "Налаштування" (рисунок 3.3).

Кнопки було організовано за допомогою QVBoxLayout, що забезпечує їх вертикальне розташування. Це дозволяє зберегти гармонійний вигляд інтерфейсу і забезпечити зручний доступ до всіх функцій програми.

Кожна кнопка налаштована таким чином, щоб забезпечити зручне і

інтуїтивно зрозуміле використання, включаючи відповідні розміри та положення.

Для покращення користувацького досвіду було додано анімацію елементів sidebar, що забезпечує плавність переходів і підвищує загальну привабливість інтерфейсу.

Анімацію було реалізовано за допомогою Qt Property Animation Framework. Зокрема, для кнопок налаштовано плавне з'явлення і зникнення при відкритті або закритті sidebar.

Встановлено тривалість анімації та інші параметри, такі як тип інтерполяції (easing curve), що дозволяє створити природний вигляд анімації (рисунок 3.5).

```
    }],
    "menuTransitionAnimation": [{
        "animationDuration": 2000,
        "animationEasingCurve": "Linear",
        "whenCollapsing": [{
            "animationDuration": 500,
            "animationEasingCurve": "Linear"
        }],
        "whenExpanding": [{
            "animationDuration": 500,
            "animationEasingCurve": "OutInBack"
        }]
    }],
    "menuContainerStyle": [{
        "whenMenuIsCollapsed": [
            "border: 2px solid transparent"
        ],
        "whenMenuIsExpanded": [
            "border: 2px solid rgb(9, 5, 13); background-color: rgb(9, 5, 13)"
        ]
```

Рисунок 3.5 — Налаштування анімації меню

Особливу увагу приділено налаштуванню кольорів інтерфейсу, що забезпечує гармонійний та привабливий зовнішній вигляд програми.

QSS, або Qt Style Sheets, це механізм, який надає можливість змінювати вигляд та оформлення елементів інтерфейсу в програмах, що використовують фреймворк Qt. За допомогою QSS можна змінювати такі параметри, як колір, шрифт, розмір, відступи, а також застосовувати різноманітні ефекти, такі як

тінь, округлення кутів і т. д. Це дає розробникам значну гнучкість у визначенні вигляду своєї програми та адаптації до користувацьких вимог. Завдяки QSS можна легко змінювати стиль програми без необхідності вносити зміни у вихідний код, що полегшує розробку та підтримку програмного продукту. Також QSS дозволяє застосовувати стилі до не тільки стандартних елементів Qt, але й до власних, що дає додаткові можливості для творчості в оформленні інтерфейсу користувача.

Колір фону головного вікна та sidebar налаштовано за допомогою Qt Style Sheets (QSS). Це дозволяє створити єдиний стиль для всіх елементів інтерфейсу (рисунок 3.5).

Наприклад, для фону головного вікна використано сірий колір (#1f232a), що забезпечує приємний і ненав'язливий вигляд.

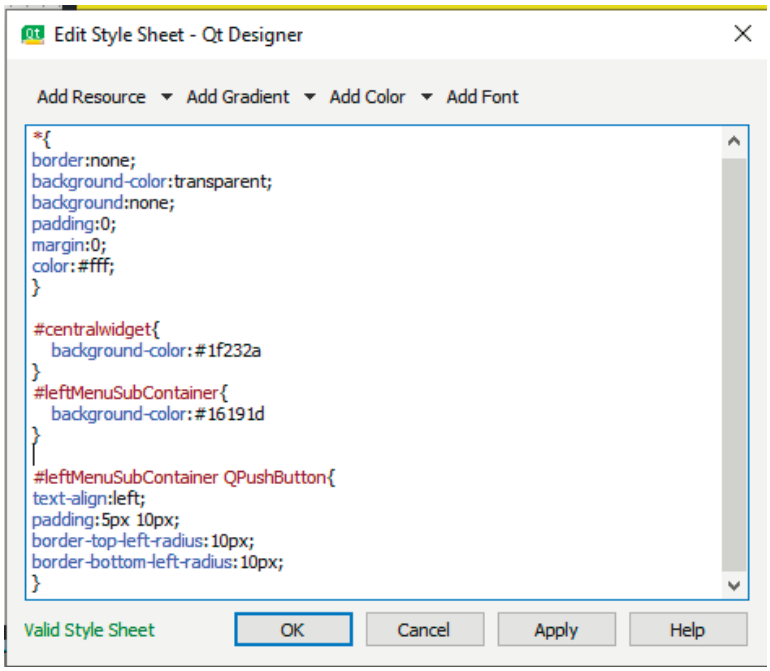


Рисунок 3.5 — Налаштування стилю вікна

Для кнопок sidebar встановлено кольори, що відповідають загальному дизайну програми. Використано QSS для налаштування кольорів тексту, фону, рамок та інших властивостей.

Наприклад, кнопки sidebar мають сірий фон (#1f232a) і білий текст (#ffffff), що забезпечує високу контрастність і зручність використання.

Дизайн інтерфейсу розроблено таким чином, щоб він був адаптивним і добре виглядав на різних розмірах екранів. Використано Layout менеджери та властивості QSS для створення гнучкого інтерфейсу, який автоматично підлаштовується під розмір вікна.

3.4.2 Створення основного коду програми

У цьому підрозділі буде розглянуто основний код програми, яка здійснює обробку даних з файлів Excel та побудову графіків на основі отриманих даних. Програма призначена для читання та обробки даних з декількох Excel файлів, збереження цих даних, а також для побудови графіків, які відображають залежність струму від напруги з логарифмічною шкалою по осі y.

Перша частина коду включає функцію process_file, яка займається обробкою файлів Excel та збереженням даних з окремих листів у словнику, вона наведена в лістингу 3.1. Ця функція приймає шлях до файлу та список листів, які слід пропустити, як аргументи. Вона використовує бібліотеки pandas та xlrd для читання даних з файлів Excel. Імена листів формуються автоматично і включають як листи під назвами "Append1", "Append2", ..., "Append40", так і лист під назвою "Data". Функція намагається зчитати дані з кожного листа, а також обробляє можливі помилки, пов'язані з відсутністю даних або листів.

Лістинг 3.1 — Функція process_file

```
def process_file(file_path, sheets_to_skip=[]):
    sheet_names = ["Append" + str(i) for i in range(1, 41)]  # Предполагается, что листы называются "Append1", "Append2", ..., "Append40"
    sheet_names.append("Data")
    data_dict = {}

    for sheet_name in sheet_names:
        try:
```

```
            if sheet_name in sheets_to_skip:
                continue
             df = pd.read_excel(file_path, sheet_name=sheet_name, header=None, names=['Current',
'Voltage'], skiprows=1)

            data_dict[sheet_name] = df
        except (pd.errors.EmptyDataError, ValueError, xlrd.biffh.XLRDError):
            pass

    return data_dict
```

У наступному блоці коду виконується читання даних з декількох файлів Excel, використовуючи функцію process_file. Для кожного файлу вказується свій шлях, а також список листів, які потрібно пропустити під час обробки. Функція повертає словник з даними для кожного файлу, що дозволяє зберігати результати обробки для подальшого використання в аналізі та побудові графіків.

Лістинг 3.2 — Читання файлів функцією process_file:

```
file_path1 = 'form1.xls'
data_dict1 = process_file(file_path1, sheets_to_skip=["", ""])

file_path8 = 'on2.xls'
data_dict8 = process_file(file_path8, sheets_to_skip=["", "", ""])

file_path15 = 'off1.xls'
data_dict15 = process_file(file_path15, sheets_to_skip=[" ", "", ""])
```

Одне з завдань програми полягає у побудові графіків I-V характеристик на основі зібраних даних. Використовується бібліотека matplotlib.pyplot для візуалізації залежності струму від напруги. Дані з різних файлів об'єднуються на одному графіку, при цьому застосовується логарифмічна шкала для осі y. Для кожного листа у словниках даних здійснюється фільтрація, щоб залишити лише ті рядки, де значення напруги менше або дорівнює 5 В, після чого

будується графік з використанням методу semilogy, приклад наведено у лістингу нижче.

Лістинг 3.3 — Використання методу semilogy:

```
for sheet_name, df in data_dict1.items():
    filtered_data = df[df['Voltage'] <= 5]
    plt.semilogy(filtered_data['Voltage'], filtered_data['Current'], label=sheet_name + ' - File 1', color='green', markersize=0, linewidth=0.1)

for sheet_name, df in data_dict2.items():
    filtered_data = df[df['Voltage'] <= 5]
    plt.semilogy(filtered_data['Voltage'], abs(filtered_data['Current']), label=sheet_name + ' - |File 2|', color='green', markersize=0, linewidth=0.1)
```

Після побудови графіків здійснюється їхнє налаштування та збереження у файл, це можна побачити у лістингу нижче. Встановлюються назви осей, розміщується легенда графіка за межами самого графіка, а також налаштовується логарифмічна шкала для осі y. Додається сітка для кращої читабельності графіка та горизонтальна лінія для позначення певного значення струму. Готова легенда зберігається у окремий файл, а сам графік відображається на екрані.

Лістинг 3.4 — Налаштування графіків:

```
    plt.xlabel('Voltage [V]')
    plt.ylabel('Current [A]')

    legend = plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    legend_labels = [
        'Forming', 'Enabling', 'Disabling'
    ]
    legend_colors = ['green', 'red', 'blue']
    legend_handles = [plt.Line2D([0], [0], color=color, linewidth=2, linestyle='-') for color in legend_colors]
    plt.legend(legend_handles, legend_labels)
```

Продовження лістингу 3.4:

```
legend.set_visible(True)

plt.yscale('log')
plt.yticks([1e-12, 1e-10, 1e-8, 1e-6, 1e-4, 1e-2, 1e0])
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.axhline(y=0.001, color='gray', linestyle='--', label='(CC)', xmin=0.5, xmax=1, fillstyle='right', )

legend_file_path = 'AllIV.png'
legend.figure.savefig(legend_file_path, bbox_inches='tight', dpi=500)

plt.show()
```

Далі розглянемо код для побудови гістограм струмів на основі даних, отриманих з файлів Excel. Код складається з двох основних частин: перша частина займається копіюванням даних з файлів Excel у текстовий файл, друга частина будує гістограми на основі цих даних.

Спершу нам потрібно зчитати дані з кількох Excel файлів. Ми використовуємо бібліотеки pandas та xlrd для обробки даних з Excel. Функція process_file зчитує дані з зазначених листів кожного файлу та зберігає значення, що відповідають певним критеріям, у словнику.

Функція process_file приймає два аргументи: шлях до файлу та список листів, які необхідно пропустити, її представлено у лістингу 3.5 Вона зчитує дані з кожного листа, що не входить у список sheets_to_skip, фільтруючи значення, що знаходяться у діапазоні від -0.95 до -1.05. Отримані дані зберігаються у словнику.

Лістинг 3.5 — Використання функції process_file:

```
def process_file(file_path, sheets_to_skip=[]):
    sheet_names = ["Append" + str(i) for i in range(1, 41)]
```

```
    sheet_names.append("Data")
```

Продовження лістингу 3.5:

```
    data_dict = {}

    for sheet_name in sheet_names:
        try:
            if sheet_name in sheets_to_skip:
                continue

            df = pd.read_excel(file_path, sheet_name=sheet_name, header=None, skiprows=1)

            filtered_values = abs(df.loc[(df.iloc[:, 2] >= -0.105) & (df.iloc[:, 2] <= -0.095), df.columns[1]])

            if not filtered_values.empty:
                data_dict[sheet_name] = filtered_values.tolist()
        except (pd.errors.EmptyDataError, ValueError, xlrd.biffh.XLRDError, IndexError) as e:
            print(f"Error processing sheet {sheet_name} in file {file_path}: {e}")

    return data_dict
```

Після зчитування даних з Excel файлів, ми зберігаємо ці дані у текстовий файл для подальшого використання при побудові гістограм. Це здійснюється шляхом запису значень з кожного листа у відповідний текстовий файл, лістинг наведено нижче.

Лістинг 3.6 — Збереження даних в текстовий файл:

```
output_file_path = 'rozrzut_100mV_negative_wiele.txt'
with open(output_file_path, 'w') as output_file:
    for sheet_name, cell_12_values in data_dict1.items():
        if cell_12_values:
```

Продовження лістингу 3.6:

```
            for value in cell_12_values:
                output_file.write(f'File: off1.xls, Sheet: {sheet_name}, Cell Value: {value}\n')


    for sheet_name, cell_12_values in data_dict2.items():
        if cell_12_values:
            for value in cell_12_values:
                output_file.write(f'File: off2.xls, Sheet: {sheet_name}, Cell Value: {value}\n')
```

На основі збережених даних ми створюємо гістограми, що відображають розподіл струмів для різних станів (LRS і HRS) при напрузі -100 мВ. Ми використовуємо бібліотеку matplotlib для візуалізації даних. Прочитавши дані з текстового файлу, ми зберігаємо їх у список values та будуємо гістограми з логарифмічною шкалою для осі x.

Лістинг 3.7 – Створення гістограм:

```
with open(file_path, 'r') as file:
    for line in file:
        match = re.search(r'Cell Value: ([\d.e-]+)', line)

        if match:
            cell_value = float(match.group(1))

            values.append(cell_value)

values_greater = [value for value in values if value > 3e-6]
values_less = [value for value in values if value <= 3e-6]
log_bins = np.logspace(np.log10(min(values)), np.log10(max(values)), 20)
plt.hist(values, bins=log_bins, alpha=0.7, color='skyblue', edgecolor='black')
plt.xscale('log')
```

Далі розглянемо кроки для обчислення провідності та побудови гістограм для різних станів (LRS і HRS) при напрузі -100 мВ. Провідність

обчислюється на основі даних струму та напруги, отриманих з файлів Excel.

Для обчислення провідності ми скористаємося середнім значенням струму з фільтрованих даних. Фільтрація проводиться в діапазоні напруги від -0.105 до -0.095 В. Середнє значення струму використовується для обчислення провідності за допомогою закону Ома (провідність = струм/напруга).

Лістинг 3.8 — Створення гістограм провідності:

```
def process_file(file_path, sheets_to_skip=[]):
    sheet_names = ["Append" + str(i) for i in range(1, 41)]
    sheet_names.append("Data")
    data_dict = {}

    for sheet_name in sheet_names:
        try:
            if sheet_name in sheets_to_skip:
                continue

            df = pd.read_excel(file_path, sheet_name=sheet_name, header=None, skiprows=1)

            filtered_values = df.loc[(df.iloc[:, 2] >= -0.105) & (df.iloc[:, 2] <= -0.095), df.columns[1]]

    if not filtered_values.empty:
                data_dict[sheet_name] = filtered_values.tolist()
                for i in range(len(filtered_values.index)):
                    index = filtered_values.index[i]
                    if index > 0 and index < len(df) - 1:
                        x1, y1 = df.iloc[index - 1, 2], df.iloc[index - 1, 1]
                        x2, y2 = df.iloc[index, 2], filtered_values.iloc[i]
                        x3, y3 = df.iloc[index + 1, 2], df.iloc[index + 1, 1]

                        average_value = (abs((y2 - y1) / (x2 - x1)) + abs((y3 - y2) / (x3 - x2))) / 2
```

Продовження лістингу 3.8:

```
                    conductivity = y2 / x2 if x2 != 0 else 0

                    with open('conductivity_neg_C2C.txt', 'a') as output_file:
                        output_file.write(f'File: {file_path}, Sheet: {sheet_name}, Cell Value: {filtered_values.iloc[i]}, Average Value: {average_value}, Conductivity: {conductivity}\n')

            except (pd.errors.EmptyDataError, ValueError, xlrd.biffh.XLRDError, IndexError) as e:
                print(f"Error processing sheet {sheet_name} in file {file_path}: {e}")

        return data_dict
```

На основі збережених даних у текстовому файлі ми створимо гістограми, що відображають розподіл провідності для різних станів (LRS і HRS) при напрузі -100 мВ. Гістограми дозволяють візуалізувати розподіл провідності та визначити характерні властивості станів. Для побудови гістограм ми використовуємо бібліотеку matplotlib, лістинг наведено нижче.

Спершу ми читаємо дані з текстового файлу і зберігаємо значення провідності у список. Далі ми створюємо гістограми з використанням логарифмічної шкали для кращого відображення розподілу провідності.

Лістинг 3.9 — Відображення гістограм провідності:

```
with open(file_path, 'r') as file:
    for line in file:
        match = re.search(r'Conductivity: ([\d.e-]+)', line)

        if match:
            cell_value = float(match.group(1))

            values.append(cell_value)

values_greater = [value for value in values if value > 1e-4]
```

Продовження лістингу 3.9:

```
values_less = [value for value in values if value <= 1e-4]


log_bins = np.logspace(np.log10(min(values)), np.log10(max(values)), 15)
plt.hist(values, bins=log_bins, alpha=0.7, color='skyblue', edgecolor='black')
plt.xscale('log')
plt.title('Провідність для LRS і HRS при V=-100 мВ')
plt.xlabel('Провідність (S)')
plt.ylabel('Частота')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```

2.9 Аналіз отриманих результатів

Після виконання коду для аналізу та візуалізації характеристик оперативної памяті, було побудовано графік залежності струму від напруги для різних етапів роботи пристрою. Графік, представлений на рисунку 3.6, дозволяє провести детальний аналіз поведінки пристроїв на різних етапах: формування (Forming), активація (Enabling) та деактивація (Disabling).

Графік відображає залежність струму (в логарифмічній шкалі) від напруги для декількох файлів, кожен з яких містить дані для конкретного етапу роботи пристрою.

Forming (зелені лінії) — цей етап відображає процес початкового формування провідних ниток в пристрої ReRAM.

Enabling (червоні лінії) — цей етап показує поведінку пристрою при активації, коли провідні нитки вже сформовані.

Disabling (сині лінії) — цей етап демонструє процес вимкнення або деактивації провідних ниток в пристрої.

Forming:

— на графіку зеленими лініями позначено процес формування. Видно, що на цьому етапі струм різко змінюється при досягненні певної напруги, що

свідчить про початок формування провідних ниток;

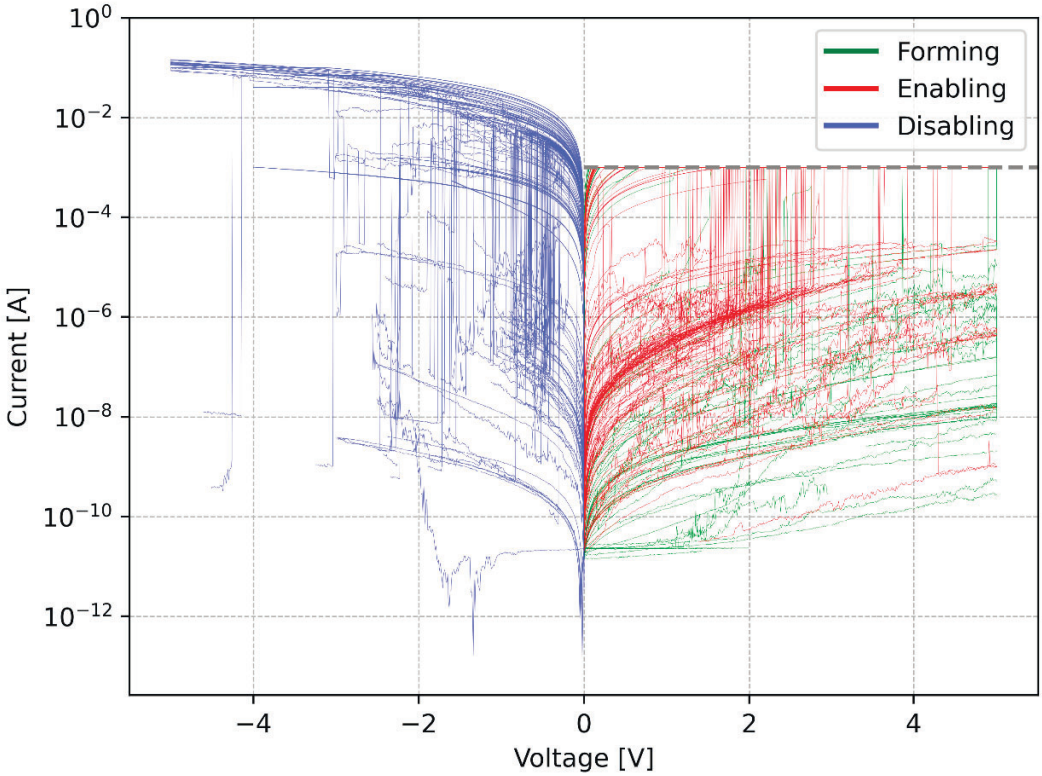


Рисунок 3.6 — Графік струм-напруга

— початкові значення струму є низькими, але при підвищенні напруги спостерігається різке зростання струму, що вказує на утворення провідних шляхів—це відповідає процесу формування структури пам'яті.

Enabling:

— червоні лінії на графіку відображають процес активації, тут видно більш стабільний характер залежності струму від напруги;

— після формування провідних ниток струм проходить через пристрій при менших напругах порівняно з етапом формування — це свідчить про те, що провідні нитки залишаються стабільними і дозволяють протікати струму через пристрій.

Disabling:

— сині лінії показують процес деактивації, відзначається значне зниження струму при зниженні напруги, що відповідає руйнуванню провідних ниток;

— на цьому етапі струм падає на декілька порядків при зниженні напруги, що вказує на ефективне руйнування провідних ниток — це є необхідним для перемикання пристрою у стан високого опору.

Аналіз отриманих графіків дозволяє зробити кілька важливих висновків щодо поведінки пристрою, дані з якого використовувались для тестування:

— процес формування чітко виділяється на графіку, що підтверджує ефективність початкового створення провідних ниток;

— етап активації демонструє стабільну поведінку пристрою при проходженні струму, що є важливим для забезпечення надійної роботи пам'яті;

— процес деактивації чітко вказує на ефективне руйнування провідних ниток, що є ключовим для переключення пристрою у стан високого опору.

На основі отриманих результатів та проведеного аналізу, наступними кроками можуть бути:

— подальша оптимізація процесів формування та деактивації для підвищення надійності та ефективності пристроїв;

— проведення додаткових досліджень для оцінки довгострокової стабільності провідних ниток та їх впливу на загальну продуктивність пристроїв;

— використання отриманих даних для створення моделей та симуляцій, що допоможуть передбачати поведінку пристроїв у різних умовах експлуатації.

Ці кроки сприятимуть подальшому розвитку та вдосконаленню технології ReRAM, що може мати значний вплив на майбутні досягнення у галузі пам'яті та обчислювальної техніки.

На рисунках, представлених нижче, зображено гістограми струмів для різних станів пам'яті (HRS та LRS) при напрузі 100 мВ.

Гістограма показує розподіл струмів в стані HRS (High Resistance State) при напрузі 100 мВ. Як видно з графіку, більшість значень струму знаходяться в діапазоні від 10^{-10} до 10^{-8} ампер, що відповідає високому опору пам'яті в цьому стані (рисунок 3.8).

Ця гістограма ілюструє розподіл струмів в стані LRS (Low Resistance State) при тій же напрузі. В цьому випадку значення струмів знаходяться в діапазоні від 10^{-4} до 10^{-3} ампер, що відповідає низькому опору пам'яті в цьому стані (рисунок 3.8).

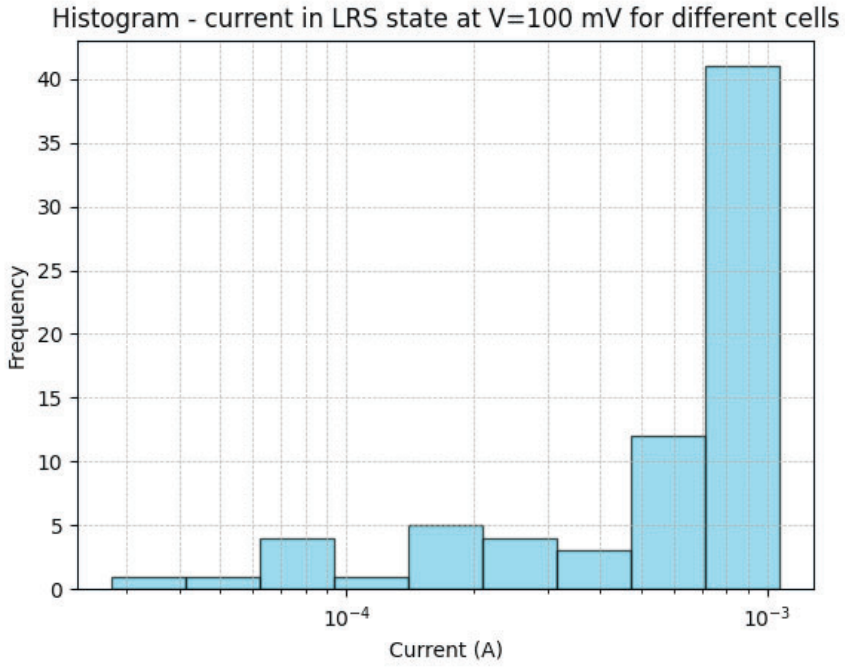


Рисунок 3.7 — Отриманий графік гістограми струму в стані LRS

Гістограми дозволяють детально вивчити поведінку резистивної пам'яті в різних станах при конкретних напругах. Вони є важливим інструментом для аналізу надійності та стабільності елементів пам'яті, що є критично важливим для їх застосування в сучасних комп'ютерних системах.

Зокрема, аналіз гістограм розподілу струмів у станах HRS та LRS дозволяє оцінити розкид параметрів між різними осередками пам'яті. Це може бути використано для оптимізації процесу виготовлення та покращення характеристик пам'яті.

Висновки, зроблені на основі отриманих результатів, можуть бути використані для подальшого вдосконалення технології резистивної пам'яті та розробки нових методів її тестування та оцінки.

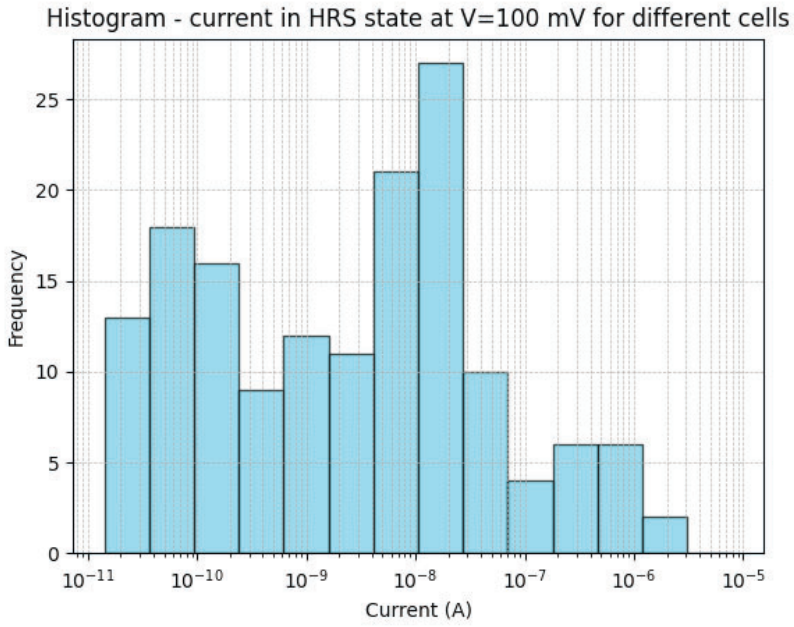


Рисунок 3.8 — Отриманий графік гістограми струму в стані HRS

Ця програма розроблена з метою забезпечити дослідників і інженерів інструментами для зручного та ефективного аналізу характеристик резистивної пам'яті. Вона дозволяє автоматично обробляти великі масиви даних, що надходять з різних осередків пам'яті, та візуалізувати їх у вигляді гістограм.

Основною перевагою використання цієї програми є її здатність швидко і точно ідентифікувати різні стани пам'яті (HRS та LRS) і відповідні струми. Завдяки логарифмічній шкалі, гістограми, згенеровані програмою, дозволяють більш чітко бачити розподіл струмів в широкому діапазоні значень, що є критично важливим для розуміння поведінки пам'яті в різних режимах роботи. Це особливо важливо при розробці нових матеріалів та структур для резистивної пам'яті, оскільки точний аналіз дозволяє виявити слабкі місця та потенційні проблеми на ранніх стадіях.

Крім того, програма забезпечує зручність у використанні завдяки простому інтерфейсу для введення даних та гнучким можливостям для налаштування параметрів гістограм. Це дозволяє дослідникам легко адаптувати її під конкретні потреби свого дослідження, що значно підвищує

ефективність роботи. Наприклад, можливість розділяти дані на дві групи за певним пороговим значенням струму (LRS та HRS) забезпечує більш детальний аналіз і дозволяє краще зрозуміти різницю між цими станами.

Загалом, програма є потужним інструментом для дослідження та аналізу резистивної пам'яті, який може бути використаний як в академічних дослідженнях, так і в промислових додатках. Вона сприяє більш точному та швидкому аналізу даних, що є ключовим фактором для вдосконалення існуючих технологій пам'яті та розробки нових, більш ефективних рішень.

На цьому процес програмної реалізації засобу для аналізу характеристик оперативної пам’яті завершено. Обґрунтовано вибір мови програмування, середовища розробки та фреймворку для створення графічного інтерфейсу. Детально описано процес створення графічного інтерфейсу користувача, включаючи розробку бічної панелі, анімацію її елементів та налаштування кольорів. Окрім того, розглянуто створення основного коду програми для обробки та аналізу даних. Ці кроки забезпечили створення зручного та практичного програмного засобу, що значно покращує користувацький досвід при роботі з програмою.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмний засіб для аналізу характеристик оперативної пам'яті типу ReRAM. Було проаналізовано основні підходи до проектування та розробки архітектури аналогів, визначено їх особливості, переваги та недоліки конкретно для обраної теми.

Обрано мову програмування Python, середовище розробки VS Code та бібліотеки Pandas, Matplotlib, та NumPy, PySide2 а також описано їх доцільність в даному проекті разом із основними використаними технологіями.

Були вирішені наступні задачі:

— визначено архітектурну модель та підхід до розробки;

— розроблено модуль зчитування та фільтрації даних з файлів Excel;

— розроблено модуль обробки даних для розрахунку провідності;

— розроблено модуль візуалізації даних за допомогою побудови графіків та гістограм;

—проведено тестування програмного продукту для перевірки його ефективності та точності аналізу.

Розроблено основні модулі програми. Аналіз отриманих результатів роботи програми довів повну працездатність даного програмного продукту та відповідність поставленому технічному завданню.

Результати проекту оформлені відповідно до вимог та зведені в даній дипломній записці.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Ye Zhou, "Advanced Memory Technology: Functional Materials and Devices", pp. 2-3.

2 An Chen, A review of emerging non-volatile memory (NVM) technologies and applications, Solid-State Electronics, Volume 125, 2016, pp. 25-38, ISSN 0038-1101. URL: https://doi.org/10.1016/j.sse.2016.07.006

3 Computer data storage [Електронний ресурс] — URL: https://pl.wikipedia.org/wiki/Pami%C4%99%C4%87_komputerowa

4 Yuan Xie, "Emerging Memory Technologies: Design, Architecture, and Applications", Springer Science+Business Media New York 2014. URL: https://doi.org/10.1007/978-1-4419-9551-3

5 Wen Siang Lew, Gerard Joseph Lim, Putu Andhita Dananjaya, "Emerging Non-Volatile Memory Technologies: Physics, Engineering, and Applications", Springer Nature Singapore Pte Ltd. 2021. URL: https://doi.org/10.1007/978-981-15-6912-8

6 Choosing The Correct High-Bandwidth Memory [Електронний ресурс] – URL: https://semiengineering.com/choosing-the-correct-high-bandwidth-memory/

7 John Poulton, "An Embedded DRAM for CMOS ASICs", Department of Computer Science University of North Carolina at Chapel Hill.

8 Réda Boumchedda. Ultra-low voltage and energy efficient SRAM design with new technologies for IoT applications. Micro and nanotechnologies/Microelectronics. Université Grenoble Alpes, 2019. English. NNT: 2019GREAT128.

9 Read-only memory [Електронний ресурс] — URL: https://en.wikipedia.org/wiki/Read-only_memory

10 Linux Flash for Newbies: Flash Memory Basics [Електронний ресурс] — URL: https://www.coresecurity.com/core-labs/articles/linux-flash-newbies-flash-memory-basics

11 Flash memory [Електронний ресурс] — URL:

https://en.wikipedia.org/wiki/Flash_memory

12 Chen, Y.C. & Rettner, Charles & Raoux, Simone & Burr, Geoffrey & Chen, S.H. & Shelby, R.M. & Salinga, Martin & Risk, W.P. & Happ, T.D. & Mcclelland, G.M. & Breitwisch, M. & Schrott, Alejandro & Philipp, J.B. & Lee, Min-Hee & Cheek, R. & Nirschl,T. & Lamorey, M. & Chen, C.F. & Joseph, Eric & Lam, Chung. (2007). Ultra-Thin Phase- Change Bridge Memory Device Using GeSb. IEDM Tech. Dig. 1 - 4. 10.1109/IEDM.2006.346910.

13 PCRAM [Електронний ресурс] — URL: https://research.tsmc.com/english/research/memory/pcram/publish-time-1.html

14 T. Eshita, T. Tamura, Y. Arimoto, 14 - Ferroelectric random access memory (FRAM) devices, Advances in Non-volatile Memory and Storage Technology, Woodhead Publishing, 2014, pp. 434-454, ISBN 9780857098030, URL: https://doi.org/10.1533/9780857098092.3.434.

15 Meena, J.S., Sze, S.M., Chand, U. et al. Overview of emerging nonvolatile memory technologies. Nanoscale Res Lett 9, 526 (2014). URL: https://doi.org/10.1186/1556-276X-9-526

16 Sandeep Munjal and Neeraj Khare, "Advances in resistive switching based memory devices", 2019 J. Phys. D: Appl. Phys. 52 433002.

17 Qt Designer Manual [Електронний ресурс] — URL: https://doc.qt.io/qt-6/qtdesigner-manual.html

18 Documentation for Visual Studio Code [Електронний ресурс] — URL: https://code.visualstudio.com/docs

19 Unified Modeling Language [Електронний ресурс] — URL: https://uk.wikipedia.org/wiki/Unified_Modeling_Language

20 Matplotlib 3.9.0 documentation [Електронний ресурс] — URL: https://matplotlib.org/stable/index.html

21 xlrd — xlrd 2.0.1 documentation [Електронний ресурс] — URL: https://xlrd.readthedocs.io/en/latest/

22 Pandas User Guide [Електронний ресурс] — URL: https://pandas.pydata.org/docs/user_guide/index.html

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____ проф., д.т.н. О. Д. Азаров

"_06_" травня_____ 2024р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання дипломної роботи

«Програмний засіб для аналізу характеристик оперативної памяті».

08-54.БДР.009.00.000 ПЗ

Науковий керівник к.т.н., доц. каф. ОТ

_____ Кадук О. В.

Студент групи 1СП-20б

_____ Ковальов М. Д.

Вінниця 2024

1 Підставою для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Важливим є актуальність дослідження у напрямку бакалаврської роботи, яка обумовлена тим, що в даний час все більше користувачів та дослідників стикаються із завданням аналізу характеристик оперативної пам'яті в сучасних комп'ютерних системах. З розвитком новітніх технологій, таких як STT-MRAM, PCRAM, FeRAM та ReRAM, зростає потреба у створенні інструментів, які дозволяють автоматизувати процес аналізу та візуалізації даних про оперативну пам'ять. Це необхідно для підвищення ефективності обробки даних, що, в свою чергу, сприяє вдосконаленню методів тестування та розвитку нових технологій пам'яті. Виконання даної бакалаврської роботи забезпечить дослідників зручним інструментом для автоматизованого аналізу струмів при різних напругах та допоможе у виявленні закономірностей і аномалій в характеристиках пам'яті;

1.2 Наказ №_80__від 11.03.2024 р. про затвердження теми БКР.

2 Мета БКР і призначення розробки

2.1 Мета роботи — підвищення ефективності аналізу та візуалізації характеристик оперативної пам'яті шляхом створення програмного забезпечення, яке дозволяє автоматично обробляти дані струмів при конкретних напругах і представляти їх у вигляді графіків для подальшого аналізу;

2.2 Призначення розробки — розробка методів та засобів для автоматизованої обробки та візуалізації даних про струми в різних станах пам'яті, що дозволить дослідникам більш ефективно виявляти закономірності та аномалії в характеристиках пам'яті, а також вдосконалювати технології пам'яті та методи їх тестування.

3 Вихідні дані для виконання БКР

3.1 Проведення аналізу існуючих типів оперативної пам'яті та новітніх технологій, таких як STT-MRAM, PCRAM, FeRAM та ReRAM.

3.2 Розробка моделі програмного засобу для автоматизованого аналізу характеристик оперативної пам'яті.

3.3 Створення графічного інтерфейсу користувача для візуалізації результатів аналізу та обробки даних про струми при різних напругах.

3.4 Виконання тестування програмного засобу на реальних даних для перевірки його ефективності та надійності.

3.5 Надання рекомендацій щодо використання розробленого програмного засобу для подальшого розвитку та вдосконалення технологій пам'яті.

4 Вимоги до виконання БКР

Засіб має володіти ясними функціями, які працюють за інтуїтивно зрозумілими та заздалегідь узгодженими правилами. Він повинен надійно функціонувати, забезпечувати цілісність даних та операцій, а також відповідати встановленим вимогам.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 – Етапи БКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз проблеми, та постановка задач	09.03.2024		Аналітичний огляд, задачі досліджень,
2	Аналіз сучасних технологій оперативної пам'яті	10.03.2024	16.03.2024	Розділ 1
3	Вибір середовища та мови розробки	17.03.2024	18.03.2024	Розділ 2
4	Розробка програмного засобу	19.03.2024	22.05.2024	Розділ 3
5	Оформлення пояснювальної записки	23.05.2024	01.06.2024	ПЗ, графіч. матеріал
6	Перевірка якості виконання бакалаврської роботи	02.06.2024	07.06.2024	Оформлені документи

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлювання БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

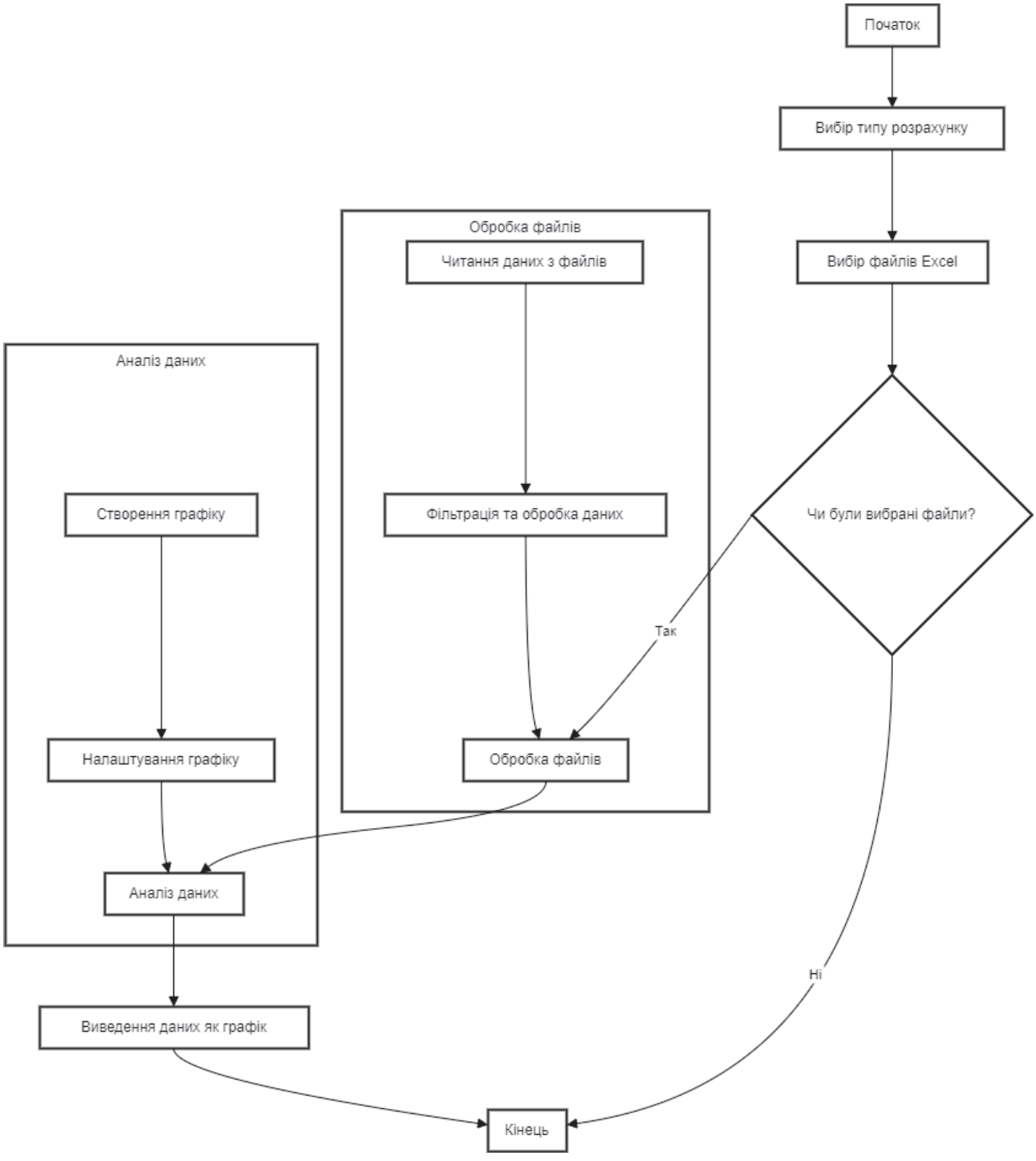
— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – «Комп'ютерна інженерія» (освітня програма «Системне програмування»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК В

Діаграма роботи програми



ДОДАТОК Г

Лістинг Г.1 — повний код файлу runs_many.py

```
import pandas as pd
import matplotlib.pyplot as plt
import xlrd

def process_file(file_path, sheets_to_skip=[]):
    sheet_names = ["Append" + str(i) for i in range(1, 41)] sheet_names.append("Data")
    data_dict = {}

    for sheet_name in sheet_names:
        try:
            if sheet_name in sheets_to_skip:
                continue

            df = pd.read_excel(file_path, sheet_name=sheet_name, header=None, names=['Current', 'Voltage'], skiprows=1)
            data_dict[sheet_name] = df
        except (pd.errors.EmptyDataError, ValueError, xlrd.biffh.XLRDError):
            pass

    return data_dict

file_path1 = 'form1.xls'
data_dict1 = process_file(file_path1, sheets_to_skip=["", ""])

file_path2 = 'form2.xls'
```

```
        data_dict2 = process_file(file_path2, sheets_to_skip=["Append100",
"Append105", "Append300"])

        file_path3 = 'form3.xls'
        data_dict3 = process_file(file_path3, sheets_to_skip=["Append700",
"Append200", "Append300"])

        file_path4 = 'form4.xls'
        data_dict4 = process_file(file_path4, sheets_to_skip=["Append700",
"Append200", "Append300"])

        file_path5 = 'form5.xls'
        data_dict5 = process_file(file_path5, sheets_to_skip=["Append700",
"Append200", "Append300"])

        file_path6 = 'form7.xls'
        data_dict6 = process_file(file_path6, sheets_to_skip=["Append700",
"Append200", "Append300"])

        file_path7 = 'form8.xls'
        data_dict7 = process_file(file_path7, sheets_to_skip=["Append700",
"Append200", "Append300"])

        file_path8 = 'on2.xls'
        data_dict8 = process_file(file_path8, sheets_to_skip=["Append700",
"Append200", "Append300"])

        file_path9 = 'on3.xls'
        data_dict9 = process_file(file_path9, sheets_to_skip=["Append700",
"Append200", "Append300"])
```

```
file_path10 = 'on4.xls'
data_dict10 = process_file(file_path10, sheets_to_skip=["Append700",
"Append200", "Append300"])

file_path11 = 'on5.xls'
data_dict11 = process_file(file_path11, sheets_to_skip=["Append700",
"Append200", "Append300"])

file_path12 = 'on6.xls'
data_dict12 = process_file(file_path12, sheets_to_skip=["Append700",
"Append200", "Append300"])

file_path13 = 'on7.xls'
data_dict13 = process_file(file_path13, sheets_to_skip=["Append700",
"Append200", "Append300"])

file_path14 = 'on8.xls'
data_dict14 = process_file(file_path14, sheets_to_skip=["Append700",
"Append200", "Append300"])

file_path15 = 'off1.xls'
data_dict15 = process_file(file_path15, sheets_to_skip=["Append700",
"Append200", "Append300"])

file_path16 = 'off2.xls'
data_dict16 = process_file(file_path16, sheets_to_skip=["Append700",
"Append200", "Append300"])

file_path17 = 'off3.xls'
```

```
    data_dict17 = process_file(file_path17, sheets_to_skip=["Append700",
"Append200", "Append300"])

    file_path18 = 'off4.xls'
    data_dict18 = process_file(file_path18, sheets_to_skip=["Append700",
"Append200", "Append300"])

    file_path19 = 'off5.xls'
    data_dict19 = process_file(file_path19, sheets_to_skip=["Append700",
"Append200", "Append300"])

    file_path20 = 'off6.xls'
    data_dict20 = process_file(file_path20, sheets_to_skip=["Append700",
"Append200", "Append300"])

    file_path21 = 'off7.xls'
    data_dict21 = process_file(file_path21, sheets_to_skip=["Append700",
"Append200", "Append300"])

    file_path22 = 'off8.xls'
    data_dict22 = process_file(file_path22, sheets_to_skip=["Append700",
"Append200", "Append300"])


    for sheet_name, df in data_dict1.items():
        filtered_data = df[df['Voltage'] <= 5]
plt.semilogy(filtered_data['Voltage'], filtered_data['Current'], label=sheet_name + ' -
File 1', color='green', markersize=0, linewidth=0.1)  #Используем semilogy

    for sheet_name, df in data_dict2.items():
```

```
            filtered_data = df[df['Voltage'] <= 5]
plt.semilogy(filtered_data['Voltage'], abs(filtered_data['Current']),
label=sheet_name + ' - |File 2|', color='green', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict3.items():
            filtered_data = df[df['Voltage'] <= 5]
            plt.semilogy(filtered_data['Voltage'], abs(filtered_data['Current']),
label=sheet_name + ' - |File 3|', color='green', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict4.items():
            filtered_data = df[df['Voltage'] <= 5]
            plt.semilogy(filtered_data['Voltage'], abs(filtered_data['Current']),
label=sheet_name + ' - |File 4|', color='green', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict5.items():
            filtered_data = df[df['Voltage'] <= 5]
            plt.semilogy(filtered_data['Voltage'], abs(filtered_data['Current']),
label=sheet_name + ' - |File 5|', color='red', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict6.items():
            filtered_data = df[df['Voltage'] <= 5]
            plt.semilogy(filtered_data['Voltage'], abs(filtered_data['Current']),
label=sheet_name + ' - |File 6|', color='green', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict7.items():
            filtered_data = df[df['Voltage'] <= 5]
            plt.semilogy(filtered_data['Voltage'], abs(filtered_data['Current']),
label=sheet_name + ' - |File 7|', color='green', markersize=0, linewidth=0.1)

        #for sheet_name, df in data_dict8.items():
```

```
        #     filtered_data = df[df['Voltage'] <= 5]
        #      plt.semilogy(filtered_data['Voltage'], abs(filtered_data['Current']),
label=sheet_name + ' - |File 8|', color='red', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict9.items():
            filtered_data = df[df['Voltage'] <= 100]
            plt.semilogy(filtered_data['Voltage'],          abs(filtered_data['Current']),
label=sheet_name + ' - |File 9|', color='red', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict10.items():
            filtered_data = df[df['Voltage'] <= 100]
            plt.semilogy(filtered_data['Voltage'],          abs(filtered_data['Current']),
label=sheet_name + ' - |File 10|', color='red', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict11.items():
            filtered_data = df[df['Voltage'] <= 100]
            plt.semilogy(filtered_data['Voltage'],          abs(filtered_data['Current']),
label=sheet_name + ' - |File 11|', color='red', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict12.items():
            filtered_data = df[df['Voltage'] <= 100]
            plt.semilogy(filtered_data['Voltage'],          abs(filtered_data['Current']),
label=sheet_name + ' - |File 12|', color='red', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict13.items():
            filtered_data = df[df['Voltage'] <= 100]
            plt.semilogy(filtered_data['Voltage'],          abs(filtered_data['Current']),
label=sheet_name + ' - |File 13|', color='red', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict14.items():
```

```
            filtered_data = df[df['Voltage'] <= 100]
            plt.semilogy(filtered_data['Voltage'],     abs(filtered_data['Current']),
label=sheet_name + ' - |File 14|', color='red', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict15.items():
            filtered_data = df[df['Voltage'] >= -5]
            plt.semilogy(filtered_data['Voltage'],              abs(filtered_data['Current']),
label=sheet_name + ' - File 15', color='blue', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict16.items():
            filtered_data = df[df['Voltage'] >= -5]
            plt.semilogy(filtered_data['Voltage'],              abs(filtered_data['Current']),
label=sheet_name + ' - |File 16|', color='blue', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict17.items():
            filtered_data = df[df['Voltage'] >= -5]
            plt.semilogy(filtered_data['Voltage'],              abs(filtered_data['Current']),
label=sheet_name + ' - |File 17|', color='blue', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict18.items():
            filtered_data = df[df['Voltage'] >= -5]
            plt.semilogy(filtered_data['Voltage'],              abs(filtered_data['Current']),
label=sheet_name + ' - |File 18|', color='blue', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict19.items():
            filtered_data = df[df['Voltage'] >= -5]
            plt.semilogy(filtered_data['Voltage'],              abs(filtered_data['Current']),
label=sheet_name + ' - |File 19|', color='blue', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict20.items():
```

```
            filtered_data = df[df['Voltage'] >= -5]
            plt.semilogy(filtered_data['Voltage'],     abs(filtered_data['Current']),
label=sheet_name + ' - |File 20|', color='blue', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict21.items():
            filtered_data = df[df['Voltage'] >= -5]
            plt.semilogy(filtered_data['Voltage'],              abs(filtered_data['Current']),
label=sheet_name + ' - |File 21|', color='blue', markersize=0, linewidth=0.1)

        for sheet_name, df in data_dict22.items():
            filtered_data = df[df['Voltage'] >= -5]
            plt.semilogy(filtered_data['Voltage'],              abs(filtered_data['Current']),
label=sheet_name + ' - |File 22|', color='blue', markersize=0, linewidth=0.1)



        plt.xlabel('Voltage [V]')
        plt.ylabel('Current [A]')

        legend = plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
        legend_labels = [
           'Forming', 'Enabling', 'Disabling'
        ]
        legend_colors = ['green', 'red', 'blue']
        legend_handles = [plt.Line2D([0], [0], color=color, linewidth=2, linestyle='-')
for color in legend_colors]
        plt.legend(legend_handles, legend_labels)
        legend.set_visible(True)

        plt.yscale('log')
        plt.yticks([1e-12, 1e-10, 1e-8, 1e-6, 1e-4, 1e-2, 1e0])
```

```
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.axhline(y=0.001, color='gray', linestyle='--', label='(CC)', xmin=0.5, xmax=1, fillstyle='right', )


legend_file_path = 'AllIV.png'
legend.figure.savefig(legend_file_path, bbox_inches='tight', dpi=500)

plt.show()
```

ДОДАТОК Д

Лістинг Д.1 — Повний код файлу
current_hist_100mV_pos_many.py

```
import re

import matplotlib.pyplot as plt
import numpy as np



file_path = 'current_100mV_wiele.txt'

values = []

with open(file_path, 'r') as file:
    for line in file:

        match = re.search(r'Cell Value: ([\d.e-]+)', line)

        if match:

            cell_value = float(match.group(1))



            values.append(cell_value)



values_greater = [value for value in values if value > 1.5e-5]
values_less = [value for value in values if value <= 1.5e-5]
```

```
log_bins = np.logspace(np.log10(min(values)), np.log10(max(values)), 20)
plt.hist(values, bins=log_bins, alpha=0.7, color='skyblue', edgecolor='black')
plt.xscale('log')

plt.title('Histogram for LRS and HRS at V=100 mV')
plt.xlabel('Current (A)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.show()
```

```
log_bins2 = np.logspace(np.log10(min(values_greater)), np.log10(max(values_greater)), 10)
plt.hist(values, bins=log_bins2, alpha=0.7, color='skyblue', edgecolor='black')

plt.xscale('log')

plt.title('Histogram - current in LRS state at V=100 mV for different cells')
plt.xlabel('Current (A)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.savefig('histogram_LRS_100mV_wiele.png')
plt.show()
```

```
log_bins1 = np.logspace(np.log10(min(values_less)), np.log10(max(values_less)), 15)
plt.hist(values, bins=log_bins1, alpha=0.7, color='skyblue', edgecolor='black')

plt.xscale('log')

plt.title('Histogram - current in HRS state at V=100 mV for different cells')
plt.xlabel('Current (A)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.savefig('histogram_HRS_100mV_wiele.png')
plt.show()
```

ДОДАТОК Е

Лістинг Е.1 — Повний код файлу conductive_hist_neg_C2C.py

```
import re
import matplotlib.pyplot as plt
import numpy as np

file_path = 'conductivity_neg_C2C.txt'

values = []

with open(file_path, 'r') as file:
    for line in file:
        match = re.search(r'Average Value: ([\d.e-]+)', line)
        if match:
            cell_value = float(match.group(1))
            values.append(cell_value)

values_greater = [value for value in values if value > 1e-4]
values_less = [value for value in values if value <= 1e-4]

log_bins = np.logspace(np.log10(min(values)), np.log10(max(values)), 15)
plt.hist(values, bins=log_bins, alpha=0.7, color='skyblue', edgecolor='black')
plt.xscale('log')

plt.title('Conductance for LRS and HRS at V=-100 mV')
plt.xlabel('Conductance (S)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
#plt.savefig('histogram_LRS_and_HRS_negative.png')
```

```
plt.show()

log_bins2 = np.logspace(np.log10(min(values_greater)), np.log10(max(values_greater)), 12)
plt.hist(values, bins=log_bins2, alpha=0.7, color='skyblue', edgecolor='black')

plt.xscale('log')

plt.title('Conductance in the LRS state at V=-100 mV for a selected cell')
plt.xlabel('Conductance (S)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.savefig('histogram_LRS_negative.png')

plt.show()

log_bins1 = np.logspace(np.log10(min(values_less)), np.log10(max(values_less)), 7)
plt.hist(values, bins=log_bins1, alpha=0.7, color='skyblue', edgecolor='black')

plt.xscale('log')

plt.title('Conductance in the HRS state at V=-100 mV for a selected cell')
plt.xlabel('Conductance (S)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.savefig('histogram_HRS_negative.png')
```

```
plt.show()
```

ДОДАТОК Є

Лістинг Є.1 — Повний код файлу

current_hist_100mV_neg_many.py

```
import re

import matplotlib.pyplot as plt
import numpy as np


file_path = 'rozrzut_100mV_negative_wiele.txt'


values = []


with open(file_path, 'r') as file:
    for line in file:

        match = re.search(r'Cell Value: ([\d.e-]+)', line)

        if match:

            cell_value = float(match.group(1))


            values.append(cell_value)


# Create a histogram with logarithmically distributed bins
```

```
values_greater = [value for value in values if value > 3e-6]
values_less = [value for value in values if value <= 3e-6]

log_bins = np.logspace(np.log10(min(values)), np.log10(max(values)), 20)
plt.hist(values, bins=log_bins, alpha=0.7, color='skyblue', edgecolor='black')
plt.xscale('log')

plt.title('Histogram for LRS and HRS at V=-100 mV')
plt.xlabel('Current (A)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
#plt.savefig('histogram_LRS_and_HRS_negative.png')

plt.show()


log_bins2 = np.logspace(np.log10(min(values_greater)), np.log10(max(values_greater)), 10)
plt.hist(values, bins=log_bins2, alpha=0.7, color='skyblue', edgecolor='black')

plt.xscale('log')
# Graph settings
plt.title('Histogram - current in LRS state at V=-100 mV for different cells')
plt.xlabel('Current (A)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.savefig('histogram_LRS_negative.png')

plt.show()
```

```
log_bins1 = np.logspace(np.log10(min(values_less)), np.log10(max(values_less)), 7)
plt.hist(values, bins=log_bins1, alpha=0.7, color='skyblue', edgecolor='black')

plt.xscale('log')
# Graph settings
plt.title('Histogram - current in HRS state at V=-100 mV for different cells')
plt.xlabel('Current (A)')
plt.ylabel('Frequency')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.savefig('histogram_HRS_negative.png')

plt.show()
```

ДОДАТОК Б

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційний ресурс з налаштувань системних параметрів комп'ютера

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 97,8% Схожість 2,2%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Ковальов М.Д.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Кадук О.В.
(підпис) (прізвище, ініціали)