Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ПРОГРАМНИЙ ЗАСІБ ТЕСТУВАННЯ ОПЕРАТИВНОЇ ПАМ'ЯТІ КОМП'ЮТЕРА

Виконав: студент 2 курсу, групи 1 КІ-23 м
спеціальності 123 — «Комп'ютерна інженерія»
_____ Стрілковський В.С.
Керівник: к.т.н., доц.каф. ОТ
_____ Колесник І.С.
«____» _____ 2024 р.
Опонент: к.т.н., доц.. каф.ПЗ
Хошаба О.М.
«____» _____ 2024 р.

Допущено до захисту
Завідувач кафедри ОТ
_____ д.т.н., проф. Азаров О. Д.
«___» _____2024 р

ВНТУ 2024

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри обчислювальної техніки

_____ д.т.н., проф. О.Д. Азаров

"__" _____2024 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

Студенту Стрілковському Вадиму Сергійовичу

1 Тема роботи «Програмний засіб тестування оперативної пам'яті комп'ютера» керівник роботи Колесник І.С., к.т.н., доц. каф. ОТ, затверджено наказом вищого навчального від 17.09.2024 року №310.

2 Строк подання студентом роботи 9.12.2024 р.

3 Вихідні дані до роботи: методи дискретної математики для формування тестових послідовностей, методи технічної діагностики, методи цифрової схемотехніки, методи алгебри Буля, методи математичної статистики для здійснення аналізу дефектів компонентів оперативних запам'ятовуючих пристроїв.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):, економічна частина, висновки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): несправності комірок пам'яті, структура програми тестування ОЗП, модель несправності впливу, лістинг програми визначення характеристик пам'яті

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесник Ірина Сергіївна, к.т.н., доцент		
5	Небава Микола Іванович проф., к.е.н		

7 Дата видачі завдання 19.09.2023 .

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз методів та засобів пошуку несправності ОЗП комп'ютера	20.09.2024р.	30.09.2024р.	Розділ 1
2	Розробка засобів пошуку несправності ОЗП комп'ютера	01.10.2024р.	04.10.2024р.	Розділ 2
3	Розробка програми пошуку несправності ОЗП комп'ютера	05.10.2024р.	18.10.2024р.	Розділ 3
4	Перевірка працездатності програмного продукту	19.10.2024р.	01.11.2024р.	Розділ 4
5	Розрахунок економічної доцільності створення програми пошуку несправності ОЗП комп'ютера	02.11.2024р.	15.11.2024р.	Розділ 5
6	Апробація та впровадження результатів дослідження	16.11.2024р.	30.11.2024р.	Тези доповіді
7	Оформлення пояснювальної записки, графічного матеріалу і презентації	02.12.2024р.	10.12.2024р.	ПЗ, графічний матеріал
8	Підготовка і підпис супроводжуючих документів, нормоконтроль та тест на плагіат	10.12.2024р.	12.12.2024р.	Оформлені документи

Студент Стрілковський Вадим Сергійович

Керівник к.т.н., доц. Колесник Ірина Сергіївна

АНОТАЦІЯ

УДК 004

Стрілковський В.С. Програмний засіб тестування оперативної пам’яті комп’ютера. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна інженерія, Вінниця: ВНТУ, 2024 — 119 с. На укр. мові. Бібліогр.: 45 назв; рис.: 25; табл. 7.

У магістерській кваліфікаційній роботі досліджено методи дискретної математики для формування тестових послідовностей, методи технічної діагностики, методи цифрової схемотехніки, методів алгебри Буля, методи математичної статистики для здійснення аналізу дефектів компонентів оперативних запам’ятовуючих пристроїв, використано комп’ютерне моделювання для аналізу і перевірки коректності отриманих теоретичних та практичних результатів.

Експериментальна частина роботи включає розробку алгоритму тестування комірок оперативної пам’яті комп’ютера на основі якого розроблено програмний засіб тестування комірок оперативної пам’яті

Економічна частина дослідження містить комерційний та технологічний аудит науково-технічної розробки, прогнозування витрат та розрахунок потенційної економічної ефективності проекту.

Ключові слова: тестування, оперативна пам'ять, комп'ютерне моделювання, метод, алгоритм.

ABSTRACT

UDC 004

Strilkovskiy V.S. A software tool for testing the computer's RAM. Master's Qualifying Thesis in Specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2024 — 119 p. In Ukrainian. Bibliography: 45 titles; figures: 25; tables: 7.

The master's qualification work investigated the methods of discrete mathematics for the formation of test sequences, methods of technical diagnostics, methods of digital circuit engineering, methods of Boolean algebra, methods of mathematical statistics for the analysis of defects in components of operational storage devices, and used computer modeling to analyze and verify the correctness of the obtained theoretical and practical results.

The experimental part of the work includes the development of an algorithm for testing computer RAM cells, on the basis of which a software tool for testing RAM cells was developedThe economic section of the research contains a commercial and technological audit of the scientific and technical development, cost forecasting, and calculation of the potential economic efficiency of the project.

Keywords: testing, RAM, computer modeling, method, algorithm.

ЗМІСТ

ВСТУП

Працездатність та висока надійність роботи комп’ютера залежить від знаходження його вузлів у робочому стані, а особливо усіх запам’ятовуючих пристроїв. Досягнення високого рівня надійності функціонування комп’ютера досягається поєднанням різних заходів по перевірці стану працездатності пам'яті комп’ютера [1-3].

Одним із видів контролю за станом вузлів комп'ютера є тестове діагностування. Його використовують для контролю стану вузлів комп'ютера, що дозволяє визначити ступінь їхньої працездатності й виконати завдання по пошуку можливих несправностей, що у результаті поліпшує основні параметри надійності апаратури (коефіцієнт готовності, ймовірність безвідмовної роботи, вірогідність функціонування) [4-6].

Найбільш поширеним способом діагностування вузлів пам'яті на теперішній час є функціональне тестове діагностування. Перевірка стану та тестування вузлів пам'яті комп'ютера здійснюється із метою пошуку місця, причини й характеру можливої несправності оперативних запам’ятовуючих пристроїв (ОЗП). Для виконання процесу перевірки стану вузлів існують такі види тестування пам'яті [7,8]: апаратне зовнішнє та внутрішнє тестування і програмне зовнішнє й внутрішнє тестування. Програмний варіант тестування пам'яті є більш простим способом перевірки її стану із-за значно більшого «покриття» зони тестування і легкості модифікації алгоритмів контролю, а також їх значного розмаїття для використання, але при цьому програє по швидкості тестування порівняно із внутрішнім апаратним способом тестуванням вузлів пам'яті.

На теперішній час існує значна кількість різного типу тестів, що створюють можливість швидкого та ефективного виконання процесу пошуку можливих несправностей вузлів ОЗП в умовах виробництва і експлуатації. Але із збільшенням ступеня інтеграції у повному обсязі задача тестування так і не вирішена. Зменшення розмірів комірок пам’яті та підвищення ступеня інтеграції схем призводить до появи нових видів дефектів, що вимагає поповнення

переліку моделей несправностей вузлів пам’яті, розробки нових методів пошуку й локалізації несправності запам’ятовуючих пристроїв із метою виявлення та розпізнавання різних типів дефектів комірок ОЗП та створення нових алгоритмів пошуку несправності ОЗП.

Задача подальшого розширення й вдосконалення методів тестування вузлів оперативної пам’яті, що мають кращі показники за критерієм ступінь покриття можливих несправностей вказують на **актуальність** даної роботи.

Метою дослідження магістерської роботи є вдосконалення методів зовнішнього програмного тестування оперативної пам’яті для виявлення та локалізації дефектів компонентів оперативної пам’яті комп’ютерів.

Задачі дослідження:

— проаналізувати існуючі методи та алгоритми тестування оперативної пам’яті комп’ютера;

— запропонувати поліпшений метод тестування оперативної пам’яті комп’ютера на основі аналізу можливостей застосування різних підходів до процесу діагностування;

— створити алгоритм та програму тестування компонентів оперативної пам’яті комп’ютерів;

— перевірити працездатність створеного програного засобу та виконати його тестування;

— обґрунтувати доцільність виконання нового наукового рішення по створенню тестів для перевірки працездатності оперативної пам’яті комп’ютера, провести розрахунок фінансових витрат по створенню запропонованої розробки та визначити економічні переваги від впровадження нового програмного продукту.

Об’єкт дослідження — процес отримання даних про стан компонентів оперативної пам’яті комп’ютера шляхом виявлення та локалізації несправності у вузлах оперативної пам’яті.

Предмет дослідження — методи тестування оперативної пам’яті для виявлення та локалізації несправності у вузлах оперативної пам’яті комп’ютера.

Методи дослідження: використовувались методи дискретної математики для формування тестових послідовностей, методи технічної діагностики, методи цифрової схемотехніки, методи алгебри Буля, методи математичної статистики для здійснення аналізу дефектів компонентів оперативних запам’ятовуючих пристроїв, комп’ютерне моделювання для аналізу і перевірки коректності отриманих теоретичних та практичних результатів. Під час програмної реалізації засобів тестування використано принципи об’єктно-орієнтованого програмування.

Наукова новизна отриманих результатів полягає у тому, що удосконалено метод тестування оперативної пам’яті комп’ютера, який відрізняється від існуючих поєднанням методів тестування із застосуванням тестів типу марш та тестів для виявлення несправності впливу сусідніх комірок пам’яті, що дозволяє із більшою точністю виявляти несправності впливу сусідніх комірок та виконувати процес тестування оперативної пам’яті комп’ютера.

Практичне значення одержаних результатів*:*

– розроблено алгоритм тестування комірок оперативної пам’яті комп’ютера;

– розроблено програмний засіб тестування комірок оперативної пам’яті.

Апробація результатів магістерської роботи: зроблено доповідь на LIV Всеукраїнській науково-технічній конференції підрозділів ВНТУ (Вінниця, 2024 р.).

Публікації [9]: Колесник І. С., Очкуров МА., Стрілковський В. С. / Програмний засіб тестування оперативної пам’яті комп’ютера. Тези доповіді. Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів ВНТУ (Вінниця, 2024 р.).

Особистий внесок здобувача. Основні положення та результати магістерської роботи автором отримані самостійно.

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ТЕСТУВАННЯ ОПЕРАТИВНОЇ ПАМ’ЯТІ КОМП’ЮТЕРА

У класичному варіанті комп'ютер складається з трьох основних частин: процесора, пам'яті та зовнішніх пристроїв. Від того, що всі ці вузли знаходяться в робочому стані, залежить надійність комп'ютера. Тому велика увага приділяється перевірці стану кожного з цих вузлів. У цьому розділі розглядаються типи і компоненти запам'ятовуючих пристроїв комп'ютера, описуються типові дефекти комірок пам'яті та методи і засоби діагностики пам'яті комп'ютера.

1.1 Запам’ятовуючі пристрої комп’ютерів і їх несправності

Для забезпечення нормальної роботи будь-якого комп'ютера необхідне спеціалізоване програмне забезпечення, яке здатне організувати введення, обробку та виведення різноманітної інформації. Після увімкнення машини та завершення ініціалізації пристроїв ці програми повинні бути доступні для взаємодії з процесором, який може забезпечити обмін даними між різними пристроями комп’ютера на основі закодованих у них алгоритмів, реагуючи на них. до дій користувача, систематизуючи результати роботи на моніторі або принтері. Усі ці програми зчитуються з носія на спеціальний апаратний модуль, званий оперативною пам'яттю (RAM). Оперативна пам'ять є одним з найважливіших компонентів будь-якого комп'ютера. Відповідає за швидкість роботи пристрою, запуск і виконання програм, а також тимчасове зберігання даних. Пам'яттю комп'ютера керує операційна система, яка зберігає дані в пам'яті, захищає від несанкціонованого доступу і виконує ряд інших функцій.

Процес доступу до оперативної пам'яті розділений в часі на окремі операції — операції запису і читання. Цими операціями зазвичай керує спеціальний пристрій, який називається контролер пам'яті.

Якщо є операції з даними, розділіть пам’ять для читання/запису (RAM, оперативна пам’ять) і постійну пам’ять (ROM, постійна пам’ять). Енергонезалежне сховище та енергозалежна пам’ять також розділені. (Volatile

storage) — це види пам'яті, записи в яких не зберігаються під час відключення електроживлення, що стосується енергозалежної пам'яті, то до неї відносяться ОЗП і кеш-пам'ять. ОЗП поділяється на динамічну і статичну пам'ять [4].

Динамічна пам'ять (dynamic storage, DRAM, будують на конденсаторах) — це тип енергозалежної пам'яті, у ній інформація з через певний період часу руйнується, тому необхідно її періодично відновлювати, тобто здійснювати регенерацію, цей процес відбувається при постійній подачі електроживлення. Статична пам'ять (static storage, SRAM, її будують на тригерах), вона належить до енергозалежних типів пам'яті, якій для зберігання інформації потрібно постійно подавати напругу живлення.

Залежно від типу доступу до масивів даних запам'ятовуючі пристрої поділяються на пристрої з послідовним, довільним, прямим і асоціативним доступом. [3-5]. Послідовний доступ (SAM, Sequential access memory) відбувається послідовно, він передбачає звертання до комірок пам'яті в черговості їх розташування одна за одною. Варіантом такої пам'яті є стекова пам'ять. Як правило, оперативна пам'ять організована з довільним доступом (RAM, Random access memory), даному випадку звертання може бути виконано за будь-якою вказаною адресою до будь-якої комірки цієї пам'яті.

Від сторінкової пам'яті (page mode DRAM) комп'ютерна оперативна пам'ять пройшла довгий шлях до пам'яті типу DDR (Double Data Rate) різного виду DDR1, DDR2, DDR3, DDR4, DDR5 [5-8].

Оперативна пам'ять сучасних комп'ютерів є синхронною динамічною пам'яттю з довільним доступом (SDRAM, Synchronous Dynamic Random Access Memory). Основу мікросхеми SDRAM-пам'яті можна розглядати як якийсь двовимірний масив у вигляді матриці комірок пам'яті, які знаходяться на перетині стовпців (Column) (також іноді стовпці матриці пам'яті називають сторінками (Page)) та рядків (Row). Фактично комірки пам'яті під'єднані до ліній стовпців та ліній рядків.

У порівнянні із іншими динамічний спосіб побудови пам'яті комп'ютерів є економічним типом організації пам'яті. Для зберігання одного розряду (біта)

застосовується схема, що складається з одного транзистора та одного конденсатора (у деяких варіантах схем конденсаторів два або три). Суттєвий недолік такого типу пам'яті — заряд на конденсаторі схильний до «стікання», тобто з часом конденсатори розряджаються.

Для боротьби з втрати пам'яті вдаються до її регенерації — періодичному зчитування осередків з подальшою перезаписом. Залежно від конструктивних особливостей регенератор може перебувати як в контролері, так і в самій мікросхемі пам'яті. У сучасних модулях пам'яті регенератор найчастіше вбудовується всередину самої мікросхеми, причому перед регенерацією вміст поновлюваного рядка копіюється в спеціальний буфер, що запобігає блокування доступу до інформації.

На теперішній час існує декілька більш економічних варіантів процесу регенерації — пакетний, розподілений, розширений; найбільш економічним є прихована або тіньова регенерація. Регенерація комірок пам'яті виконується в період очікування тільки в тих модулях пам'яті, в яких є занесені дані. Із цією технологією паралельно реалізується метод TCSR (Temperature Compensated Self Refresh), який призначений в залежності від робочої температури для регулювання швидкості процесу регенерації [16, 17].

При зверненні до комірки пам'яті контролер пам'яті задає номер банку модуля пам'яті, номер сторінки в цьому банку, номер стовпчика та номер рядка. Для роботи з пам'яттю комп’ютера використовується чотири типи сигналів: CS#, WE#, RAS# та CAS# [16].

Сигнал вибору чіпа — це сигнал CS# (Chip Select). Сигнал RAS# — це сигнал вибору рядка матриці пам'яті (Row Access Strobe), а CAS# — сигнал вибору стовпця матриці пам'яті (Column Access Strobe). Для розрізнення операцій читання і запису використовується спеціальний сигнал WE# (Write Enable). Операція читання виконується тоді, коли сигнал WE# підтримується на високому рівні, а операція запису реалізується тоді, коли сигнал WE# переводиться в низький рівень. У сукупності сигнали CS#, WE#, RAS # та CAS# дозволяють сформувати всі команди, що необхідні для роботи з пам'яттю, тобто

команду запису (Write), команду читання (Read), команди активізації лінії і чіпа пам'яті (Active) і команду деактивізації (закриття) рядка (Precharge).

Пам'ять крім максимальної пропускної здатності характеризується ще латентністю. Під терміном латентність прийнято розуміти затримку між моментом надходженням команди і моментом її реалізації. Латентність пам'яті вимірюється її таймінгами, тобто значеннями затримки, що вимірюються кількістю тактів між окремими командами.

Основними таймінгами пам'яті DRAM є:

— затримка між подачею даних про номер рядка і номер стовпця, що називається часом повного доступу (RAS to CAS delay);

— затримка між подачею даних про номер стовпця і отриманням вмісту комірки, що називається часом робочого циклу (CAS delay);

— затримка між читанням інформації із останньої комірки і подачею даних про номер нового рядка (RAS precharge).

Найбільш вагомими по їх впливу на продуктивність роботи пам'яті є таймінги tCL, tRAS, tRP і tRCD, що іноді називаються основними — саме вони приводяться на модулях пам'яті. Прийнято записувати основні таймінги пам'яті у вигляді послідовності tCL-tRCD-tRP-tRAS [16]. Інші таймінги пам'яті, які є неосновними, як правило не приводяться на модулях пам'яті.

Основним блоком оперативної пам'яті є матриця запам'ятовуючих елементів у вигляді множини комірок, у кожній із яких зберігається один біт даних. Кожна комірка пам'яті складається із одного конденсатора і трьох транзисторів. Для побудови комірки пам'яті використовують транзистори, виготовлені по МОН-технології (метал-окисел-напівпровідник).

При виготовлені комірок пам'яті виникають технологічні дефекти схем. Найбільш поширеними типовими технологічними дефектами електронних схем, що виконані по МОН-технології, є дефекти з'єднань, літографії, окислу, обриви і короткі замикання виводів транзисторів та інші [8].

Для опису поведінку електричної схеми при наявності дефекту формується модель дефекту, яку називають несправністю. Для комплементарних КМОН-

схем найбільш поширеними є три класи несправностей: постійні або константні несправності (SA, постійне значення логічного 0 або 1), обриви виводів (SOP) і короткі замикання між виводами (SON). У польових транзисторах при виготовлені електричних схем товщина оксидного шару досягає значень порядку 2нм та менше, що приводить до виникнення сильних електричних полів із напруженістю близько 10…12 МВ/см, які збільшують ймовірність електричного пробою цього шару. Такий пробій оксидного шару транзистора, а також іонне забруднення кристалу призводять до такого ефекту, що струм між витоком і стоком МОН-транзистора може вийти із-під контролю його затвором [9]. Це у свою чергу призводить до появи різних типів дефектів.

Найчастіше виникає дефект у вигляді замикання між шарами металізації у схемі, який становить 38% від можливих типів дефектів. Іншим типом дефектів є поява обривів між шарами металізації (15%), наступним є замикання між шарами дифузії (14%) та обриви між шарами дифузії (до 7%). Також існують замикання між шарами металізації та підкладкою електронної схеми, а також інші типи технологічних дефектів виготовлення схем [9]. Ці та подібні їм дефекти описуються у вигляді несправності. Найбільш поширеними типами несправності при виготовленні інтегральних схем є константні несправності (SA), несправності взаємного впливу та так звані перехідні несправності [10]. Виникаючі дефекти полікремнію, окислу та літографічні дефекти призводять до появи замикань, що тоді описуються як несправність короткого замикання (SON). Обрив у шарі металізації, недостатній контакт або ж недостатній імплантат формують несправність типу SON або SOP, яка потім описується константна несправність. Це основні типи несправностей, що виникають при виготовленні інтегральних схем.

В оперативних запам’ятовуючих пристроях найчастіше виникають дефекти транзисторів, із яких побудована оперативна пам'ять. Це дефект стійкого замикання між виводами КМОН-транзистора (транзистор постійно знаходиться у відкритому стані) або ж обрив одного із виводів МОН-транзистора (тоді транзистор постійно знаходиться в закритому стані). Цей тип дефекту

відповідає одиночній константній несправності SA, при якій значення комірки пам'яті відповідає стану логічної 1(stack at faults 1) або стану логічного 0 (stack at faults 0).

Коли є дефекти декількох транзисторів, то тоді з'являються кратні константні несправності. Також у електронних схемах існує й інший тип дефекту, коли для двох або більше ліній схеми сигнали становляться залежними в схемі несправній, тоді як у справній схемі значення цих сигналів не залежать один від одного. Такий тип несправності класифікується як місткова несправність (англійською bridging faults) і є несправністю впливу. Крім того, існують також ще нестійкі дефекти інтегральних схем, що можуть проявитися при деяких певних значеннях стану схеми: дія електромагнітних полів на інтегральну схему, високі або ж низькі значення температури навколишнього середовища, індуктивна або ж ємнісна наводки на схему або інші види впливів. Такі нестійкі дефекти із часом можуть перейти у стан стійких дефектів та потім проявити себе уже як стійкі несправності [7, 10].

Перераховані типи несправності виявляються на підприємствах, що виготовляють інтегральні схеми. Але під час експлуатації електричних схем можуть проявитися і інші типи несправності, моделі яких не були виявлені під час виготовлення та тестування інтегральних схем для оперативної пам'яті. Розгляду інших типів несправності, що можуть проявитися при експлуатації інтегральних схем оперативної пам'яті, присвячено наступні розділи даної роботи.

1.2 Способи пошуку несправності оперативної пам’яті комп’ютерів

В даний час розроблена теорія діагностування цифрових схем, орієнтована на виявлення класу константних несправностей (stuck-at, SA), тобто передбачається, що поява дефекту еквівалентно присвоюванню певній точці схеми постійного значення 0 або 1. Цей клас несправностей добре відображає реальні технологічні і експлуатаційні дефекти схем, виконаних по ЕЗЛ- і ТТЛ-технології. Однак в схемах, виконаних за технологією КМОН, реальні дефекти

слабо покриваються моделлю несправностей stuck-at і тести орієнтовані на ці несправності, не можуть ефективно виявляти дефекти КМОН-схем. Аналіз причин дефектів показує, що більша їх частина призводить до утворення несправностей «стійкий обрив транзистора» (stuck-open, SOP) і «стійке замикання транзистора» (stuck-on, SON). Несправності stuck-open переводять схему з класу комбінаційних в клас послідовностних, несправності stuck-on призводять до появи на виходах схеми нестійких сигналів, що мають проміжне значення між 0 і 1. Рішення завдання тестування КМОН-схем пов'язано перш за все з розробкою ефективної моделі несправності [11 , 12].

Частота появи несправності у кристалах оперативної пам'яті у 5-8 разів вище, ніж у компонентах постійної пам'яті, якщо їх порівнювати до кількості знайдених дефектів по відношенню до одиниці значення обсягу пам'яті накопичувача. Подібні оцінки були дані в ряді багатьох досліджень [4, 5]: відмови ОЗП досягають до 70% від загального числа несправностей електронних компонентів цифрових систем. Тому для забезпечення надійної роботи цифрових систем значна увага повинна бути приділена ефективному контролю та тестовому діагностування пам'яті комп’ютерів, особливо оперативної, як на етапі виготовлення та налагодження, так і в ході експлуатації цифрових пристроїв.

Для електричних схем SRAM розглядаються моделі класичних функціональних дефектів, таких як: статичних — обриву лінії SOF, замикання SAF, взаємного впливу CF, переходу TF, роботи дешифратора адреси ADF та динамічних дефектів — відновлення підсилювачів запису/читання, утримання даних DRF і перемикання адреси. При цьому враховується можлива фізична причина дефекту, а також виконання операцій, необхідних для його виявлення.

При визначенні покриваючої здатності різних діагностуючих тестів до уваги беруться не реальні дефекти в структурі ОЗП (обриви виводів транзисторів або запам'ятовуючого конденсатора, замикання між виводами ліній передачі сигналів і т. п.), а створені математичні моделі, що описують функціональну дію цих дефектів у процесі аналізу роботи пам'яті комп’ютерів. Моделі несправності

роботи комірок запам’ятовуючих пристроїв, що найчастіше використовуються, є несправності постійного стану, перехідні або нестійкі несправності та несправності взаємного впливу [3, 9].

Несправності постійного стану — константні несправності (SAF, Stuck At Fault): у комірці пам'яті логічне значення завжди дорівнює 1 (SA1) або 0 (SA0) незалежно від типу операцій, які здійснюються з цією або іншими комірками пам'яті комп’ютера.

Перехідна або нестійка несправність (TF, Transition Fault): комірка пам'яті не може виконувати перехід зі стану логічної 1 в стан логічного 0 або навпаки.

Несправності взаємного впливу (CF, Coupling Fault): при зміні логічного стану однієї (впливаючої) комірки відбивається зміна стану другої (залежної) комірки. Комірка зі старшою адресою може впливати на комірку із меншою адресою (позначимо як ↓) і навпаки — впливати на старшу адресу (позначимо як ↑). При виконанні тестування з усіма комірками пам'яті у визначеному порядку (↕ — в будь-якому напрямку, ⇓ — від старших адрес до молодших, ⇑ — навпаки,) виконуються ті операції, що визначаються кількома елементами тесту.

У свою чергу несправності взаємного впливу існують трьох типів [3, 9].

Неінверсна несправність CF (CFid, Idempotent CF). У цьому випадку зміна значення стану впливаючої комірки переводить залежну комірку в певний визначений стан. Тут можливими є вісім видів несправностей CFid.

Інверсна несправність CF (CFin, Inversion CF). У цьому випадку зміна значення стану впливаючої комірки викликає інвертування значення стану залежної комірки. Тут можливі чотири види несправностей CFin.

Константні несправність CF (CFst, State CF). У цьому випадку перехід залежної комірки в який-небудь стан можливий тільки при певному значенні стану впливаючої комірки. Тут розрізняють вісім видів таких несправностей CFst.

Розглянуті типи несправностей покладемо в основу побудови діагностичних моделей запам’ятовуючих пристроїв комп’ютера.

Наявність великої кількості структурних елементів пам'яті комп’ютерів (запам'ятовуючих комірок) створює астрономічне число комбінацій двійкових чисел для опису всіх їх можливих станів. Для масиву пам'яті комп’ютера із N комірок таких можливих станів може бути 2^N. Хоча для виявлення реально існуючих дефектів у пам'яті комп’ютера достатньо генерувати тільки невелику їх частину, вона все ж залишається чисельно величезною. Тому це висуває на перший план завдання побудови ефективних методик пошуку та локалізації несправностей, що забезпечують значну покриваючу здатність для виявлення різних типів дефектів і при цьому задовольняють встановленим часовим обмеженням. У міру збільшення ємності пам'яті комп’ютера важливість цього завдання зростає. Відомі алгоритми з досить високою покриваючою здатністю – – GALPAT, Sliding Diagonal та WALPAT, що мають складність порядку N^2 і $N^{3/2}$ (кількість звернень до комірок пам'яті як функція числа комірок N є складністю тесту), вимагають для перевірки пам'яті навіть незначної ємності занадто велику кількість часу. Інші тести, такі як шахівниця, Zero-One, MSCAN, мають складність порядку N, але забезпечують низьку покриваючу здатність [15].

Після створення маршових тестів проблема знаходження так званих класичних типів несправностей ОЗП була практично вирішена. Доведено, що саме маршові тести дозволяють знаходити так звані класичні типи несправностей ОЗП, є неперевершеними за простотою реалізації, повнотою покриття і часу виконання на пошук несправностей (мають складність порядку N) [16,17]. Але ще є проблема виявлення більш рідкісних дефектів — кодочутливих несправностей (PSF, pattern sensitive faults). Основною проблемою при пошуку несправностей даного класу є генерація наборів тестів для можливої активації дефекту у різних конфігураціях задіяних для цього комірках. Для пошуку таких несправностей можна використовувати видозмінені маршові тести, але для етапу перевірки працездатності ОЗП існуючі методи знаходження кодочутливих дефектів на даний час не можна визнати повністю задовільними через їх недостатній рівень покриття.

У залежності від виду впливів технічне діагностування поділяється на робоче і тестове. При робочому діагностуванні контроль і пошук дефектів відбувається у процесі функціонування об'єкта на вхідних впливах та робочих частотах при використанні пристрою за призначенням. В другому випадку використання об'єкта служить тільки для можливості визначення його технічного стану із застосуванням тестів із множини дозволених.

Функціональне тестове діагностування на теперішній час є найбільш поширеним способом діагностування ОЗП. Тестування з подальшим діагностуванням пам'яті здійснюється із метою виявлення причини, характеру та місця несправності ОЗП. Відповідно до технічної реалізації існують такі види тестування, як зовнішнє апаратне тестування, внутрішнє апаратне тестування та зовнішнє програмне тестування [6-9].

У свій час зовнішнє апаратне тестування було основним способом тестування оперативної пам’яті, вимагало досить дорогого та спеціалізованого зовнішнього обладнання, і мало досить значний недолік у вигляді неможливості проведення такого тестування умовах експлуатації.

На теперішній час внутрішнє апаратне тестування досить широко поширене та реалізується вмонтованою апаратурою для внутрішнього діагностування (BISD, built-in self diagnosis). Для діагностування оперативної пам’яті маємо MBIST — Memory BIST. Для зовнішнього користувача у випадку внутрішнього апаратного тестування видається тільки підсумковий результат тестування. Цей вид тестування порівняно із зовнішнім тестуванням є значно дешевшим й може виконуватися як при виробництві блоків пам'яті, так і при їх експлуатації.

Зовнішнє програмне тестування здійснюється шляхом виконання звертань до оперативної пам'яті комп’ютера за сформованими алгоритмами із програми тестування. Має значне «покриття» для пошуку можливих несправностей, а легкість модифікації алгоритмів перевірки дозволяє формувати різні програмні продукти для тестування. При порівнянні із внутрішнім апаратним тестуванням операції тестування мають значно меншу швидкість виконання.

Внутрішнє апаратне тестування є подальшим розвитком зовнішнього тестування, у якому компоненти тестування внесені всередину модулів пам'яті. На теперішній час практично всі сучасні модулі пам'яті мають вбудовану BIST. При програмному тестуванні найпростіший та найнадійніший спосіб — це звернутися до всієї пам'яті і по черзі записувати і зразу ж зчитувати та перевіряти 0 і 1 в комірки пам'яті, але по-перше такий алгоритм є «руйнівним» (тобто він руйнує попередні значення даних), а з іншої сторони — самий довготривалий. Для розробки алгоритмів тестування пам'яті проводиться аналіз можливих місць появи дефектів у компонентах пам'яті та створюються так звані моделі несправностей.

По відношенню до даних, що зберігаються в пам'яті комп'ютера, на теперішній час існують руйнівні та неруйнівні типи алгоритмів.

Руйнівні алгоритми діагностування не передбачають збереження раніше записаної інформації в ОЗП. Якщо необхідно збереження попередньої інформації, то використовується резервне копіювання. Це означає наявність в апаратній частині додаткового резервного блоку пам'яті. До цього блоку першим питанням є його надійність, а також на копіювання інформації із блоку пам'яті в обидві сторони затрачується додатковий час. У апаратурі MBIST такий підхід застосовуються вкрай рідко, а у деяких програмах тестування такого роду алгоритми часто мають місце.

Неруйнівні алгоритми діагностування, в підсумку роботи яких значення даних у комірках пам'яті, що перевіряються, залишаються початковими. Такі алгоритми є трохи більш повільними, ніж руйнівні алгоритми, але при цьому відпадає необхідність мати резервний обсяг пам'яті. Це дозволяє збільшити точність та зону покриття використовуваних алгоритмів, у тому числі і у MBIST. Основною ідеєю використання усіх неруйнівних алгоритмів є застосування логічної операції XOR (виключне АБО) [6, 16].

Значна кількість неруйнівних тестів були сформовані модифікацією руйнівних тестів (найбільше симетричних маршових) із заміною руйнівних

операцій на неруйнівні, що у свою чергу призвело до збільшення складності тестів.

Існуючі програми контролю та тестування пам'яті є досить надійним засобом пошуку та виявлення несправностей, так як спеціально формують такі умови, в яких ці несправності виявляються. Ряд несправностей у роботі пам'яті можуть залежати від стану комірок пам'яті (це такі ситуації, коли в мікросхемі деякі сусідні комірки починають впливати одна на одну). Для виявлення таких несправностей алгоритм тестування пам'яті повинен перевіряти один і той же фрагмент пам'яті декілька раз, записуючи і зчитуючи різні, спеціальним чином підібрані, патерни. Вдале поєднання патернів дасть можливість обійти ті перешкоди і досягти однакової ефективності знаходження несправностей у різних типах чіпів, що мають різне фізичне розташування елементів пам'яті.

1.3 Аналіз тестів для пошуку несправності пам’яті комп’ютера

Для перевірки працездатності і пошуку ймовірних дефектів оперативних запам’ятовуючих пристроїв необхідно в пам'ять записати і зчитати всі можливі комбінації чисел для бінарного коду. Повна і ґрунтовна перевірка пам’яті на всі можливі комбінації бінарного коду вимагає досить великих затрат часу, що навіть з урахуванням великих швидкостей обміну даними стають дуже значними. Для великих обсягів пам'яті комп’ютера час здійснення повних перевірок досягає значних аж до астрономічних значень. Однак, щоб з високою ймовірністю оцінювати якість виготовлення інтегральних схем ОЗП, повні перевірки не є зовсім обов'язковими. Для такого оцінювання досить часто можна використовувати розроблені усічені тестові коди, що базуються на врахуванні фізики відмовлень компонентів кристалу, схемотехнічних, структурних і технологічних особливостей інтегральних схем, що перевіряються. На теперішній час розроблено декілька десятків тестів, що досить широко використовуються при перевірці працездатності ОЗП [8-12]. Розглянемо найбільш використовувані на практиці тестові послідовності для контролю та діагностування запам'ятовуючих пристроїв комп’ютерів.

Створені тестові послідовності мають різне число операцій (тестових наборів) по запису і зчитування даних із пам’яті і тому характеризуються різною обчислювальною складністю. Існуючі тести у залежності від складності їх виконання можна згрупувати в три великі класи: тести, що пропорційні N, $N^{3/2}$ та N^2, де N — число комірок пам’яті. Тести, що пропорційні N («Парність — непарність», «Марш одиниць і нулів», «Зчитування — інвертування — запис», «Шахи», «Руйнування рядків», «Відновлення одиниць»), використовуються також в якості вбудованих тестів при перевірці працездатності комірок пам’яті обчислювальних засобів і застосовуються найбільш широко. Тести, що пропорційні N^2 та $N^{3/2}$ («Галоп», «Діагональ, що зрушується», «Пінг-понг» та інші), застосовуються в умовах промислового виробництва пам’яті, при проведенні експериментальних та дослідних випробувань, так як для свого виконання вимагають досить значних часових ресурсів.

«Шахи» записують в комірки запам’ятовуючого пристрою шаховий малюнок та зчитують його. Потім записують інверсний шаховий малюнок і знову виконують операцію читання; тривалість цього тесту 4N циклів. Цей тест досить часто виконують як контроль часу відновлення вмісту комірок, для чого перед виконанням операції зчитування роблять витримку тривалістю 2мс.

«Множинна вибірка адреси» — у пам'ять записують фон у вигляді одиниць та нулів, що чергуються, потім зчитують уміст першої та останньої комірок; перевіряють уміст першої комірки. Збільшують адресу на одну одиницю, зчитують вміст комірки по цій адресі і за адресою N-1 (комплементарної стосовно попередньої) зчитують уміст попередньої комірки. Тест повторюють для наступних комірок, що зміщаються послідовно (по номерах адрес) назустріч одна одній. Потім зчитують уміст всіх комірок. Тест повторюють для комплементарної форми запису. Тривалість цього тесту складає 7N циклів.

«Прогулянка одиниць і нулів» — створюють «фон нулів»; у першу комірку (за нульовою адресою) записують одиницю; потім із усіх інших комірок зчитують дані та перевіряють на збереження нулів, а в першій комірці перевіряють збереження записаної одиниці, після чого в неї записують нуль. Цю

послідовність дій повторно здійснюють для кожної наступної комірки пам'яті, а потім виконують аналогічний тест для запису та зчитування комплементарних даних. Тривалість цього тесту дорівнює $2N^2 + 6N$ циклів тактової частоти.

«Галоп запису-зчитування» — створення фону не обов'язково; у попередню комірку записують одиницю; у другу комірку записують нуль і перевіряють наявність одиниці в першій комірку; у другу комірку записують одиницю і перевіряють наявність одиниці в першій. Перераховані операції виконують для всіх комірок, що залишилися, при цьому в першій комірці зберігається одиниця, а на закінчення в першу комірку записують нуль. Надалі приведену послідовність дій повторюють для другої, третьої і всіх інших комірок. Довжина цього тесту з урахуванням комплементарного тесту складає $8N^2 + 4N$ циклів. Цей тест дозволяє виявити вплив у кожній із пар комірок дію циклів запису і зчитування одиниць і нулів.

«Галоп стовпців» — у комірки пам'яті записують «фон нулів», у першу комірку записують одиницю. Потім здійснюють «галоп стовпців» зчитування уздовж стовпця, тобто переходять та зчитують з одного стовпця матриці пам'яті на інший. Після кожного послідовного зчитування нулів з комірок першого стовпця (вони мають номери 0, N, 2N, 3N,..., (N-1)N) виконують зчитування одиниці з першої комірки. По завершені цієї частини тесту в першу комірку знову записують нуль. Потім у порядку зростання адрес зчитують вміст двох наступних комірок. В комірку з наступною адресою записують одиницю і повторюють операцію «Галопу стовпців» уздовж цього нового стовпця, у яку записана одиниця. Вказаний порядок дій виконують для кожної комірки пам'яті, а потім весь тест повторюють для комплементарних даних. Тривалість цього тесту $6N^{3/2} + 12N$ циклів тактової частоти.

«Парність-непарність» — у комірки із адресою, що містять парне число одиниць, записують одиниці, а в інші комірки (до них відноситься й комірка із нульовою адресою) заносять нулі. Потім, починаючи з першої, здійснюють послідовне зчитування й перевірку умісту всіх комірок. Тест повторюють для комплементарного фону, коли в комірки з парною кількістю одиниць в адресі

записують значення нуля, а в інші комірки — одиниці. Цей тест дозволяє перевірити правильність роботи адресного дешифратора і є одним з найбільш коротких мінімальних функціональних тестів. Довжина тесту складає 4N циклів.

«Зміщення по діагоналі» — в комірки записують «фон нулів»; у першу комірку записують одиницю. Починаючи з комірки з номером N, виконують зчитування вмісту комірок уздовж їхньої головної діагоналі (до номера адреси попередньої комірки, що зчитується, додають цифру (N-1). Потім зчитують тестове слово (у даному випадку — це одиниця в першій комірці) і в цю комірку записують нуль. Адресу цієї комірки збільшують на одиницю. Тестове слово (одиниця) записується в усі комірки уздовж діагоналі, що утворена рядком і стовпцем, що межують з цією коміркою. Далі виконують операцію зчитування стану комірок по зміщеній діагоналі. Цю послідовність дій повторюють до тих пір, поки тестове слово не буде записано в усі діагоналі матриці пам'яті з позитивним нахилом. По завершенню весь описаний тест виконують для комплементарних даних. Тривалість цього тесту складає $4N^{3/2}$ +8N циклів.

«Марш одиниць і нулів» — попередньо усі комірки пам'яті заповнюють нулями («фон нулів»). Потім з першої комірки (за нульовою адресою) зчитують значення «нуль» і в неї записують «одиницю». Цю послідовність дій повторюють для всіх інших комірок (тобто виконується фактично «марш одиниць»). Записавши одиницю в останню комірку, роблять зчитування одиниць і запис нулів в усі комірки у зворотному порядку, починаючи з останньої. Потім всі комірки заповнюють одиницями («фон одиниць») і процес тестування пам’яті повторюють для комплементарних даних (здійснюється «марш нулів»). Із урахуванням циклів створення фону тест «Марш одиниць і нулів» триває 10N циклів тактової частоти. Даний тест виявляє константні несправності та несправності дешифратора з достатньою ймовірністю. Цей тип тесту є мінімальним функціональним тестом.

Перераховані типи тестів дають можливість виконати перевірку працездатності запам’ятовуючих пристроїв комп’ютерів. Вони є основою для вибору та формування нових типів тестів та створення програми перевірки стану

оперативної пам’яті комп’ютерів на виявлення можливої появи деяких нових специфічних типів дефектів.

1.4 Аналіз засобів контролю працездатності пам’яті комп’ютерів

На теперішній час існує значна кількість різних програмних продуктів, які допомагають користувачеві перевірити працездатність комп'ютера, а також отримати, узагальнити та проаналізувати інформацію про технічні характеристики системи. При схожих призначеннях подібні програмні продукти досить часто значно розрізняються за своєю побудовою та реалізацією, зручністю інтерфейсу користувача, набору інструментів діагностики та перевірки функціональності роботи комп'ютера в цілому. Серед подібних програмних продуктів трапляються як вузькоспеціалізовані, які призначені для детального розгляду однієї з підсистем або частини комп'ютера, так і такі, які дають можливість виконати тестування та діагностику системи в цілому і всіх її підсистем окремо. Найчастіше до складу утиліт моніторингу і діагностики комп'ютера розробники включають різні тестові модулі, що дозволяють на основі нескладних, а головне — нетривалих випробувальних тестів скласти більш повне уявлення про стан комп'ютерної системи і прийняти виважене рішення, що стосується способів збільшення продуктивності її роботи.

Існуючі засоби діагностування комп’ютерів можна розділити на дві великих групи — загального призначення та для контролю та діагностування окремих вузлів та блоків комп’ютера [1, 11].

До засобів загального призначення відносяться утиліти тестування та діагностики, які здійснюють перевірку всіх або більшості блоків та модулів комп’ютера на працездатність. Головною вимогою до них в першу чергу висувають умову, як максимально зручний, дружній і інтуїтивно зрозумілий користувачу інтерфейс, забезпечення високого ступеня функціональності та інформативності, а також умову, щоб програми були бажано безкоштовними і доступними для вільного отримування та користування через мережу Інтернет

[20-23]. На даний час це такі програми, як AIDA64 Extreme Edition, Dacris Benchmarks, SiSoftware Sandra, 3DMark, CPU-ZCPU-Z та інші.

Однією з найвідоміших програм є AIDA64. Дана програма ідеально підходить для моніторингу всіх комплектуючих комп'ютера і проведення різних тестів. Головною перевагою AIDA64 перед конкурентами є наявність максимально повної інформації про комп'ютер.

AIDA64 Extreme Edition — це потужний інструмент для діагностики та тестування персонального комп'ютера, створений командою розробників відомого продукту EVEREST, який є наступним кроком у його розвитку. Програма пропонує користувачеві широкий спектр можливостей для діагностики комп'ютерного обладнання, стрес-тестування і моніторингу температури датчиків. Програма має унікальний набір тестів для процесора, оперативної пам'яті та жорстких дисків (включаючи вінчестери). Крім того, за допомогою детального аналізу комп’ютера користувач може отримати детальну інформацію про апаратне забезпечення (процесор, материнська плата, інтегрована моніторна та відеопідсистема, диски тощо), а також про програмне забезпечення, операційну систему, драйвери, процеси тощо. d.

Dacris Benchmarks — проста, але дуже корисна програма для тестування продуктивності будь-якого комп’ютера. Можливості цієї програми включають різні тести процесора, оперативної пам'яті, жорсткого диска і відеокарти. Результати тесту виводяться на екран. Для того, щоб результати були максимально достовірними, кожен пристрій буде тестуватися в кілька етапів, з акцентом на широке тестування. Dacris Benchmarks є платним дистрибутивом, але ви можете завантажити пробну версію з офіційного сайту розробника.

В SiSoftware Sandra входить безліч утиліт, допомогою яких і проводиться діагностика комп'ютерних комплектуючих. Тут присутній набір еталонних тестів, кожен з них потрібно запускати окремо. Під час тестування користувач буде отримувати різні результати, оскільки, наприклад, процесор швидко працює із арифметичними операціями, але йому досить повільно дається обробка

та відтворення мультимедійних даних. Такий підхід допомагає більш якісно здійснити перевірку комп'ютера, виявити слабкі та сильні його сторони.

Програма 3DMark від Futuremark є найбільш популярним софтом для проведення тестування комп'ютерів серед геймерів. Відносно перевірки функціональності, то у програмі присутня велика кількість різних бенчмарків, що дозволяють тестувати ОЗУ, процесор і відеокарту.

CPU-ZCPU-Z — це утиліта, яка надасть вам найдетальнішу інформацію про процесор, пам'ять, кеш і материнську плату, встановлену в системі. Програма компактна, надає зручну інформацію про компоненти, підтримує всі типи процесорів і материнських плат. Відображає таку інформацію: Процесор Назва процесора. Сходи Ernadro і технічна експлуатація. Корпус. Базова напруга. Внутрішня і зовнішня частоти, множник процесора. Набір допоміжних інструкцій. Кешувати дані. Виробник, модель і версія материнської плати. Виробник BIOS, дата та модель. Чіпсет (північний і південний міст) і датчики. Графічний інтерфейс. Періодичність і час запису. Кількість каналів пам'яті. Технічні характеристики модуля записуються в SPD (схема серійної специфікації): виробник, серійний номер, розклад. Відеокарта Назва відеокарти Ім'я графічного процесора Пряма підтримка технології. Підтримка піксельних шейдерів X Тип пам’яті Розмір пам’яті Пропускна здатність пам’яті Тип і ширина шини Частота пам’яті, графічний процесор (стандартний/прихований) Драйвер і версія BIOS Датчики Частота ядра графічного процесора Частота пам’яті графічного процесора Температура графічного процесора Швидкість графічного процесора Кулер Завантаження графічного процесора в реальному часі.

Перераховані програмні продукти дозволяють проводити перевірку модулів комп’ютера на виконання заданих функцій та працездатність, а також визначити швидкодію і продуктивність їх роботи. Для пошуку та локалізації несправностей вони не мають значних можливостей, тому для перевірки працездатності ОЗП комп’ютера зупинимося на існуючих вузькоспеціалізованих програмних продуктах, таких як Quick Memory Test OK, RAM Benchmark,

GoldMemory, Memtest86, Memtest86+, SuperRam, які в першу чергу дозволяють здійснити контроль та пошук несправностей в оперативних запам’ятовуючих пристроях комп’ютера [22-25].

Quick Memory Test OK — невелика, але ефективна утиліта для швидкої перевірки оперативної пам’яті на наявність помилок. Є можливість проводити окремі тести і контролювати поведінку при великому навантаженні на оперативну пам'ять. Основні можливості програми: Швидка перевірка пам'яті. Можливість призупинити тест для моніторингу комп'ютерів з високим навантаженням на оперативну пам'ять. Виконується тест, який можна редагувати. Очистити екран стану пам'яті. Контроль завантаження ЦП. Огляд основних специфікацій пам'яті та системи. Мінімальне споживання системних ресурсів.

RAM Benchmark — це утиліта, призначена для оцінки продуктивності та швидкості оперативної пам’яті. Це програмне забезпечення дозволяє оцінити продуктивність пам’яті, виконавши базовий тест для визначення швидкості вашої оперативної пам’яті. Програма має мінімальний інтерфейс, який дозволяє виконати тест одним натисканням кнопки. Він підтримує типи пам'яті DDR5, DDR4, DDR3, DDR2, DDR1 і SDRAM. Зазвичай результат тесту з’являється на екрані недовго. Швидкість відображається в МБ/с. На більшості материнських плат ви можете ввімкнути внутрішній профіль «розгону», який буде XMP (профіль екстремального розгону пам’яті) або DOCP (профіль розгону DRAM). Збільшивши або зменшивши значення МГц, ви можете використати вплив цієї зміни на продуктивність пам’яті. Бенчмарки зазвичай вимірюють максимальну теоретичну швидкість пристроїв. Додаток RAM Benchmark вимірює реальну швидкість, дозволяючи користувачеві змінювати налаштування розміру блоку. Ця програма може читати і записувати величезні блоки пам'яті. Завантаживши цю програму, ви можете розпочати тест одним клацанням миші. Кнопка «Run Benchmark» запустить порівняльний аналіз доступної оперативної пам’яті та відобразить результати протягом кількох секунд. Результати: мінімальна швидкість, максимальна швидкість і середня швидкість.

Утиліта перевірки працездатності оперативної пам'яті GoldMemory вважається однією з кращих у своєму класі. Вона побудована на нестандартних алгоритмах, здатних виловлювати помилки, які пропускають додатки-конкуренти. У ній реалізована повна підтримка 64-бітної архітектури і сумісність з усіма типами застарілих і сучасних ОЗУ, включаючи покоління DDR4. Максимальний обсяг тестованої пам'яті становить 64 ГБ / 1 ТБ.

GoldMemory працює в трьох основних режимах перевірки — швидкому, нормальному і поглибленому, а також в призначеному для користувача, де тривалість тестування та набори тестових груп задаються вручну. Крім того, програма зберігає історію, веде звіти і підтримує управління за допомогою командних файлів. Програма платна.

Ще одна програма для перевірки правильності роботи пам'яті комп’ютера — Memtest86 [25]. Програма може виконуватися на комп'ютері окремо сама по собі, без будь-якої іншої операційної системи. Для того, щоб це стало можливим, також необхідно створити спеціальну завантажувальну дискету з програмою. Перелік алгоритмів, що використовуються в Memtest86, відрізняється від набору алгоритмів, що застосовуються у програмі GoldMemory. Виконання програма почнеться автоматично відразу після команди завантаження. Виводити результати тестування програма може як на монітор комп'ютера, так і на додатковий термінал, який може бути підключений через послідовний порт. Користувач отримує докладну інформацію про виконувані тести і місцезнаходження знайдених несправностей. Програма Memtest86 чудово виконує свої обов'язки на комп'ютерах із 8 гігабайтами оперативної пам'яті, що для поширених 32-бітових комп'ютерів є цілком достатнім.

Memtest 86+ — одна з кращих утиліт, призначених для тестування оперативної пам'яті. Утиліта може запускатися з допомогою власного завантажувача, тому для неї операційна система, в принципі, не потрібна.

Memtest86+ — це програма, створена незалежними розробниками на основі Memtest86. У програмі заявлена підтримка новітніх платформ і технологій. Принцип роботи аналогічний Memtest86 — до завантаження також

пропонується ISO-образ для створення завантажувального діагностичного диска. Програма підтримує сучасні багатоядерні процесори, більшість чіпсетів материнських плат. Memtest 86+ доступна для скачування на офіційному сайті у вигляді кількох версій. Ця програма дає найбільш точні дані і може працювати в самих різних версіях операційних систем, в тому числі Windows10, і так далі.

Висновок

У даному розділі роботи розглянуті типи запам'ятовуючих пристроїв комп’ютерів та особливості їх функціонування у сучасних комп’ютерах, описані типові несправності комірок ОЗП комп’ютерів та відомі способи пошуку несправності залежно від можливих типів дефектів та їх моделей. Показано, що зовнішнє програмне тестування для фірм та окремих користувачів є більш ефективним.

2 РОЗРОБКА ТЕСТІВ ТА ПОСЛІДОВНОСТІ ПЕРЕВІРКИ СТАНУ ОПЕРАТИВНОЇ ПАМ’ЯТІ КОМП’ЮТЕРА

Здатність комп’ютера виконувати поставлені завдання залежить від знаходження у працездатному стані всіх його вузлів, а особливо пристроїв пам’яті. Швидкий тест пам’яті, що виконується при включені комп’ютера системою BIOS, може виявити тільки основні типи (так звані важкі) несправності оперативного запам’ятовуючого пристрою (ОЗП). Формування моделей несправності та нових алгоритмів їх пошуку у складі ОЗП є важливою задачею, один із підходів вирішення якої розглянуто в даному розділі.

2.1 Моделі для пошуку несправності компонентів пам’яті

Складність електронних пристроїв зростає з кожним роком, і перевірка їх придатності для певних умов використання є важливою технічною проблемою. У технологічних процесах ще більше загострюється процес зменшення розмірів компонентних інтегральних схем, підвищення якості бракування функціональних кристалів і збільшення відсотка виходу придатних виробів. Зі зменшенням розміру осередків пам'яті зростає ймовірність виникнення різних дефектів при їх виготовленні, внаслідок чого збільшується кількість різноманітних помилок і збоїв у роботі пристрою. Тому все більшого значення набувають типи дефектів, які рідко зустрічаються в попередніх технологічних процесах.

Виробництво та обробка чіпів пам'яті вимагає відповідної підтримки тестування. Більшість виробників мікросхем пам'яті проводять діагностику за допомогою різних тестів. Щоб довести ефективність тестів, потрібно створити моделі пристроїв пам'яті та їх можливі несправності.

Серед моделей і методів логічного аналізу цифрових об’єктів можна виділити найбільш поширене автоматичне представлення синхронної функціональної моделі як основної частини кубового покриття цифрової структури.

Аналіз моделей цифрових пристроїв у діагностичній підтримці будівель включає моделювання несправностей і роботу об’єкта, випробування конструкції, аналіз конструкції та інтерпретацію результатів діагностики.

Симуляція хорошої роботи виконується в двійковому або багатозначному алфавіті, в синхронному або асинхронному режимі. Перевага надається більш простому та швидкому синхронному моделюванню. Реалізація асинхронних алгоритмів потребує вагомих аргументів, які компенсують значні часові та матеріальні витрати [18].

Набір операцій, пов'язаних з вимірюванням j-го параметра об'єкта моніторингу, що отримує вхідний тест або робочий ефект, і з визначенням інтервалу $[y^{H}_{ij}\ ,\ y^{B}_{ij}]$, в який попадає вимірюване значення $y_j(t)$, назвемо *j*-ю перевіркою. Для її позначення використовується символ π_j. Усі перевірки, виконувані на даному об'єкті, утворюють множину

$$\Pi = \{\pi_j\}^{n}_{i=1} \qquad (2.1)$$

де *n* — число вимірюваних параметрів об'єкта.

Результатом перевірки π_j назвемо подію, яка полягає у попаданні обмірюваного значення *j*-го параметра в *i*-й інтервал $[y^{H}_{ij}\ ,\ y^{B}_{ij}]$, якому відповідає ознака

$$e_{ij} \in \Sigma\ (i = \overline{1,m}\ ;\ j = \overline{1,n}\ ). \qquad (2.2)$$

Число можливих результатів перевірки π_j позначимо ω_j. Для загального позначення результату перевірки π_j використовуватимо символ π_j, а її конкретний *r*-й результат позначатимемо π^{r}_{j} $(r = 1, \omega_j)$. Вважатимемо, що $\pi^{r}_{j} = e_{ij}$, якщо $y_j(t) \in [y^{H}_{ij}\ ,\ y^{B}_{ij}]\ \Lambda\ t \in [t_i\ ,\ t_{i+1}]$. Таким чином, кожну ознаку $e_{ij} \in \Sigma$ можна розглядати як «модельний» (очікуваний) результат *j*-ї перевірки в *i*-му технічному стані об'єкта. Надалі для стислості результатом перевірки π_j будемо називати ознаку e_{ij} *, що* позначає *i*-й інтервал, у котрий попадає обмірюване значення *j*-го контрольованого параметра. Множина Σ у цілому дає множину

очікуваних результатів усіх перевірок у всіх розглянутих технічних станах $e_i \in E$, в одному з яких може знаходитися об'єкт контролю.

При виконанні кожної перевірки $\pi_j \in \Pi$ можливі помилки першого та другого роду, обумовлені похибками контрольно-вимірювальних пристроїв, обмеженою надійністю системи контролю, перешкодами в трактах передачі вимірювальної інформації й інших факторів. Імовірності цих помилок позначимо через α_j та β_j- відповідно. Якщо відомі середньоквадратичні відхилення контрольованих параметрів, їхні пуски і середньоквадратичні похибки вимірів (ці відомості звичайно входять у паспортні дані на систему вимірів), то імовірності помилок першого й другого роду досить просто визначаються по спеціальних номограмах. Отримані тим чи іншим способом значення α_j та β_j для усіх виконуваних перевірок утворять множини $A = \{ \alpha_j \}^{n}_{i=1}$ і $B = \{ \beta_j \}^{n}_{i=1}$, що включаються в створювану модель.

При складанні моделі контрольованого об'єкта повинна враховується також можливість появи в ньому випадкових відмов. З цієї причини перехід об'єкта з одного технічного стану в інший навіть при нормальному його функціонуванні не може заздалегідь розглядатися як достовірна подія. Отже, кожен очікуваний результат e_{ij} перевірки π_j є випадковою подією. Технічний стан e_i обумовлений комбінацією випадкових подій e_{ij}, також є випадковою подією, що характеризується відповідною вірогідною мірою $P(e_i)$. Множина вірогідних мір $P = \{P(e_i) \mid i = \overline{1,m}\}$ також повинна бути одною з необхідних елементів моделі [23].

Уперше модель оперативного запам'ятовувального пристрою запропонована Хейсом С. П. (Hayes S. P.) [19] і представлена у вигляді цифрового автомата, що містить множину запам'ятовуючих комірок O_i, які піддаються впливу операторів:

$$O_i = \{ V_i, W_i, R_i \} \tag{2.3}$$

де V_t — оператор запису 0 в комірку C_i;

W_i — оператор запису одиницю в комірку C_i;

R_i — оператор зчитування стану комірки C_i.

Звичайно ємність пам'яті в такій моделі обмежують двома-трьома комірками, що істотно скорочує тривалість досліджень.

Кількість операцій, за допомогою яких можна змінити стани запам'ятовуючих комірок, визначається за формулою: $O_{sm}=n*2^n$, де n - ємність мікросхеми пам'яті. Для трьох запам'ятовуючих комірок число можливих операцій, що змінюють стани комірок пам'яті, дорівнює 24.

Доповнимо модель пам'яті Хейса для одержання моделей, які описують процес функціонування мікросхем із можливими групами різних типів несправності. Найбільш поширеними є константні несправності мікросхеми пам'яті - константний нуль і константна одиниця запам'ятовуючих комірок. Дані несправності можна представити у вигляді графа станів, у якому одна або кілька комірок не змінюють свої стани.

На рисунку 2.1 представлені моделі запам'ятовуючого пристрою для двох комірок пам'яті для двох випадків, коли обидві комірки справні, та випадок, коли комірка з нульовою адресою містить несправність константа нуль комірки.

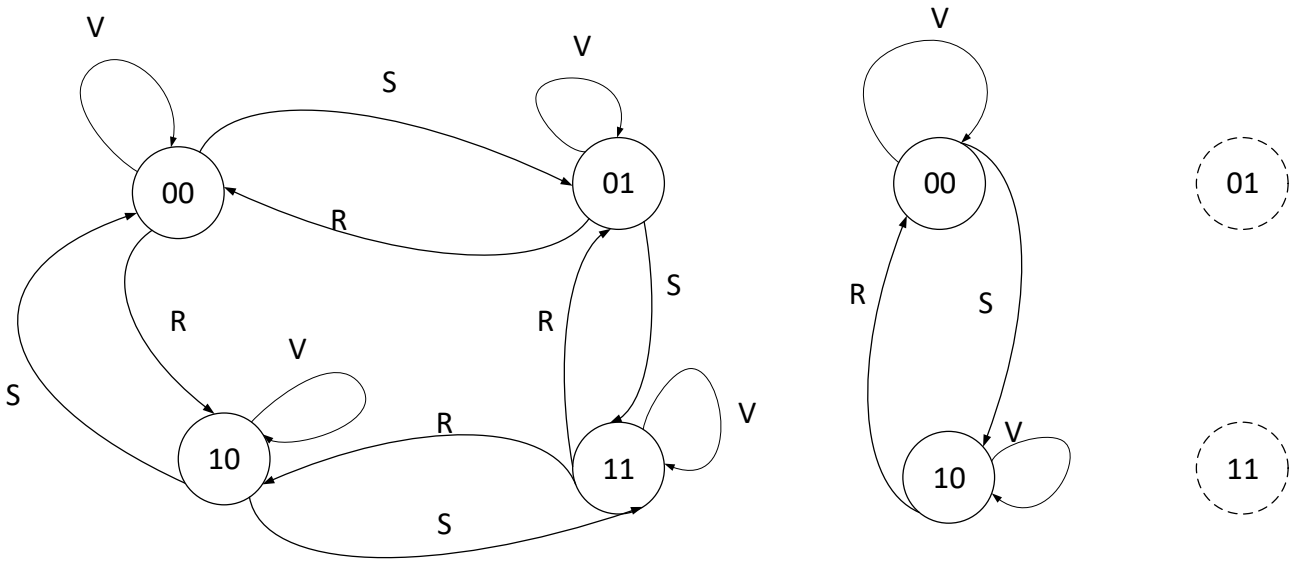

Рисунок 2.1 — Модель запам'ятовувального пристрою з несправністю константа нуль

Така комірка не може перемкнутися в одиничний стан, тому неприпустимі стани показані рисунку 2.1 пунктиром. Дану діаграму станів мікросхеми можна перетворити в матриці з'єднань, елементи яких містять операції, необхідні для переходу мікросхеми з одного стану в інший. Сигнал S встановлення комірки пам’яті в одиницю, R - в нуль, а сигнал V – операція зчитування з комірки.

Елемент матриці на перетинанні і-того рядка й j-того стовпця містять значення операцій при наявності переходів зі стану S_i у стан S_j. Якщо можливі кілька переходів S_i в S_j, то M_{ij} містить диз'юнкцію виконуваних операцій. Якщо перехід не виконується, то такий елемент матриці буде порожнім.

Небажаним наслідком зростання складності електронних пристроїв та апаратури є збільшення імовірності появи різних типів несправностей. Тому для забезпечення надійного функціонування апаратури слід значну увагу приділити своєчасній перевірці, пошуку та знаходженню можливих несправностей всіх вузлів комп’ютера, у тому числі і оперативної пам'яті, оскільки результати проведених досліджень різних організацій свідчать про значну кількість відмов при функціонуванні ОЗП, що досягають до 70% від загальної кількості відмов у роботі систем у цілому [3, 15].

В умовах експлуатації електронної апаратури широко використовуються програмні засоби зовнішнього та внутрішнього контролю [3, 11]. Такий підхід дозволяє періодично здійснювати перевірку працездатності обладнання на внутрішній частоті функціонування комп’ютера без додаткового застосування зовнішнього обладнання. Більшість відомих програмних засобів для виконання такої перевірки реалізують тести складності O(N), де N - ємність оперативного запам'ятовуючого пристрою комп’ютера. Серед таких тестів значне місце займає група маршових тестів [8, 12].

У зв'язку з безупинним удосконалюванням відомих тестових послідовностей і створенням нових починаються спроби формалізації процесу розробки тестів для ОЗП. В роботі [18] пропонується використовувати спеціальну розбивку множини комірок пам'яті на підмножини й організувати

тестові послідовності на основі такої розбивки. Це дозволяє зменшити розмірність моделі при обчисленні тестів, оскільки тестова послідовність формується з підтестів, що призначені для перевірки деяких областей ОЗП. Запропонований спосіб дає можливість формалізувати процес побудови тестів і при інтуїтивному підході. При розбивці комірок пам'яті необхідно враховувати реальну структуру ЗП, його схему, топологію кристала, а також несправності, характерні для кожної структури і технології ОЗП.

Для того, щоб тестування ОЗП здійснювалося ефективно як з погляду складності випробувань, у процесі тестування необхідно виявити два фактори: параметричну чутливість випробуваного пристрою та чутливість до виду іспитової кодової послідовності. Перший фактор відноситься до таких найбільш критичним інформаційним параметрам, як час доступу до чіпа, час відновлення запису, час встановлення адреси, час установлення правильних даних. Другий фактор відноситься до іспитових кодових послідовностей, що подаються у виді визначених імпульсних впливів на виводи випробуваного пристрою. Оскільки чутливість до виду іспитової кодової послідовності для ОЗП існує, виготовлювач зобов'язаний при записі в паспорт основних параметрів ВІС ОЗП, таких, наприклад, як час вибірки, домовлятися про умови випробувань своїх пристроїв, при яких ці параметри отримані. У іншому випадку буде мати місце неоднозначне тлумачення споживачем технічних характеристик придбаних їм ОЗП. У свою чергу споживач ОЗП повинний перевірити ці характеристики й уважно оцінити тестові умови, при яких вони були отримані для того, щоб гарантувати відповідність обраних їм ЗУ вимогам розроблювальної апаратури. Найбільший інтерес представляла залежність часу вибірки ОЗП від зміни напруги живлення.

2.2 Маршові тести для контролю ОЗП

Для тестування напівпровідникової пам'яті розроблено багато алгоритмів, серед яких найбільш популярними і ефективними є маршові тести. Маршовий тест містить послідовність маршових елементів, яка складається з операцій читання/запису, що виконуються в кожну комірку пам'яті. Марш-тести здатні

виявити декілька моделей несправностей, таких як застрягання в комірці (Stuck-at Faults, SAF), адресні несправності (Address Faults, AF) і деякі помилки з'єднання (Coupling Faults, CF). Операції, які можуть бути виконані в комірках, можуть бути наступними: запис нуля (w0), запис одиниці (w1), зчитування нуля (r0) і зчитування одиниці (r1). Операція читання перевіряє, чи значення у комірці є очікуваним. Порядок, у якому розглядаються комірки, може бути зростаючим або спадаючим. Типовим марш-тестом для тестування оперативної пам'яті є MATS++. Він має три маршових елементи M0: ↕ (w0); Ml: ↑ (r0; w1); і M2: ↓ (r1; w0; r0); вони записуються через кому або крапку з комою, а вся послідовність маршу береться у фігурні дужки.

Крім традиційного заводського контролю на етапі виробництва в даний час широко застосовується вбудована апаратура самотестування BACT (built-an self diagnosis) [11]. Такий підхід дозволяє періодично виконувати тестування на внутрішній частоті без застосування зовнішнього обладнання. Більшість BACT реалізують тести складності O (N) (N - ємність запам'ятовуючого пристрою), які називають маршовими тестами [12]. При виконанні стандартного маршового тестування з усіма осередками пам'яті в певному порядку (⇑ - від молодших адрес до старших, ⇓ - навпаки, ↕ - в будь-якому напрямку) виконуються операції, що задаються кількома маршовими елементами. Використовуються операції: w0, w1 - запис в елемент пам'яті значень 0 або 1; r0, r1 - читання поточного значення з елемента пам'яті і порівняння його зі значенням 0 або 1. Перевагами маршових тестів є відносно високі швидкість виконання і покриваюча здатність, а також простота реалізації BACT. З недоліків відзначимо повну втрату інформації, що зберігається в пам'яті, після виконання тестування, що обмежує застосування класичних маршових тестів тільки моментами початкового тестування пристрою. Для надійної роботи постійно включених систем, крім методів з використанням надлишкових даних (коди Хеммінга, контроль на парність), застосовується методи періодичного неруйнівного тестування [8-12], заснованого на класичних маршових алгоритмах.

Для оцінки можливості виявлення несправностей за допомогою тесту марш розглянемо стани комірок пам'яті, які виникають при виконанні даного тесту. При виконанні даного тесту виконується Ost=4n операцій звертання до мікросхеми пам'яті, які забезпечують одержання станів комірок, наведених на рисунку 2.2.

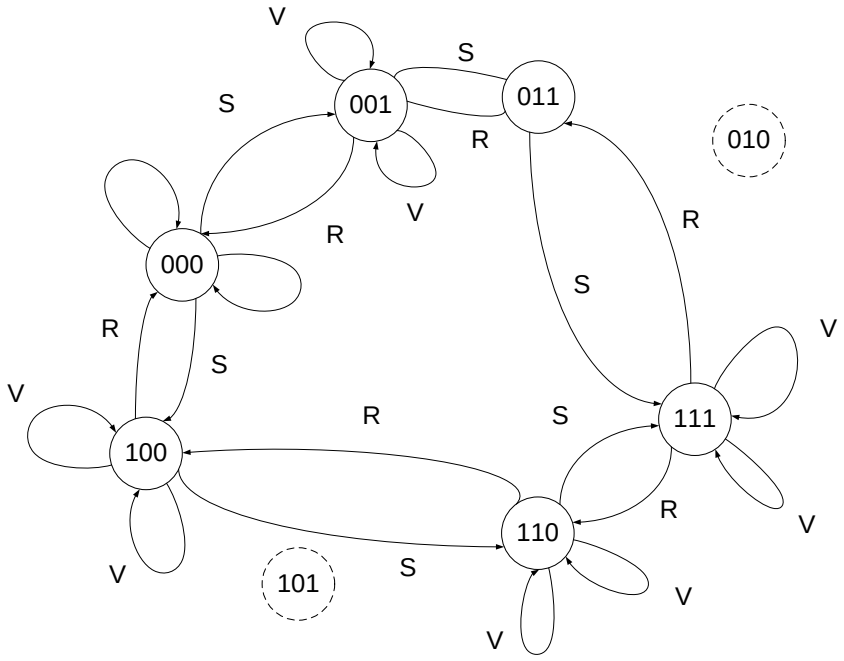

Рисунок 2.2 — Послідовність зміни станів трьох комірок пам'яті при виконанні тесту марш

Усього в мікросхемі пам'яті може бути $S_m=2^n$ можливих станів, однак при виконанні тесту марш не всі вони формуються.

Для оцінки можливості виявлення несправності константа одиниця тестом марш визначимо як різницю між елементами, що мають однакові індекси, матриці станів тесту марш M_{tm} та матриці з'єднань мікросхеми, що має несправність:

$$(m)^{no} = (m)^{tm} - (m)^{k1} \qquad (2.5)$$

При виконанні тесту марш для трьох комірок пам'яті забезпечується перевірка можливості формування станів: 000, 001, 011, 111, 110 й 100. У моделі з несправністю константа одиниця для нульового осередку відсутні стани 110 й 100, а в моделі з несправністю константа нуль відсутні стани 001, 011, 111, які

формуються тестом марш, отже, константні несправності осередків пам'яті тестом марш виявляються. Однак при діагностиці перемежованих відмов діагностичні властивості даного тесту знижуються, що не забезпечує ефективну перевірку наявності даних видів відмов у мікросхемах пам'яті великої ємності. Для виконання ефективних процедур діагностування мікросхем пам'яті рекомендується застосовувати інші тести [15].

Основною вимогою, що пред'являються до неруйнуючих тестів, є необхідність відновлення початкового стану об'єкта тестування після виконання процедури тестування. Тому всі тестові дії, спрямовані на активізацію несправностей і прояв їх у вигляді помилок на виходах схеми, повинні носити відновлюваний характер. Найпростішим способом реалізації цієї вимоги для випадку неруйнівного тестування пам'яті є збереження всього її вмісту в резервному буфері з подальшим виконанням одного з класичних руйнуючих алгоритмів. Даний спосіб потребує дублювання масиву даних із запам'ятовуючих елементів, а також затрат додаткового часу для зберігання й подальшого відновлення початкового вмісту пам'яті. Тому він непридатний для більшості випадків тестування пам'яті.

Модифікація значень пам'яті в процесі виконання базового неруйнівного алгоритму виконується шляхом інвертування вмісту. Інвертування справної комірки означає запис в неї протилежного значення. Несправність може проявити себе збереженням попереднього стану або зміною іншої клітинки. Наведення в результаті помилкове значення зчитується і модифікується подальшими операціями до тих пір, поки повторна активація несправності не встановить в залежній комірці таке значення, яке б вона мала при відсутності несправності, тобто відбудеться приховування помилки [25].

Умовою виявляючого прояву константної несправності SAF неруйнуючим тестом є інверсія несправної комірки - наступна операція читання поверне помилкове значення. Прочитане після повторного інвертування значення співпаде зі значенням, яке було б прочитано зі справної пам'яті. У загальному

випадку, будь-яке парна кількість інвертування наводить помилку в несправній комірці, непарне — приховує помилку.

Після виділення всіх умов прояву несправностей необхідно виділити пари виявляючих і приховуючих помилку умов для кожної з розглянутих несправностей. Це дозволяє визначити, які операції оперують з помилковими даними в процесі тестування. Початок кожній лінії розташоване під операцією тесту, яка відповідає символу R з умови, виявляє прояв несправності. Закінчується кожна лінія перед операцією, яка відповідає символу R з найближчої умови, що приховує прояви даної несправності.

Одним з найважливіших параметрів створюваних алгоритмів є їх складність, що оцінюється по сумарній кількості операцій в алгоритмі [4].

Ще одним критерієм оцінки ефективності алгоритмів тестування є їх покриваюча здатність. У разі неруйнівного тестування її можна розглядати з двох сторін. З одного боку, слід розглянути здатність алгоритму активізувати деякий набір несправностей - його генеруючу здатність, з іншого, створений алгоритм повинен однозначно визначати помилковість прочитаного значення, тобто мати виявлявляючі здібності.

Оскільки при тестуванні за методом неруйнуючих тестів активізація несправностей проводиться тільки в процесі роботи базового неруйнівного алгоритму, то тільки він володіє генеруючою здатністю. А так як він був отриманий з руйнівного маршового алгоритму шляхом виконання перетворень, які не впливають на покриваючу здатність, то їх покриваючі здібності співпадають [12].

Виявлення згенерованих помилок при тестуванні за алгоритмом неруйнуючих тестів проводиться при порівнянні еталонної і робочої сигнатур. Для розглянутих несправностей помилки гарантовано спотворять тільки другу з них, і, отже, всі можуть бути виявлені. На виявляючу здатність алгоритмів з глобальною симетрією впливає вибір позицій, у які додаються коригувальні симетрію операції читання. Додаткова операція на самому початку алгоритму більш краща, оскільки вона виконується перед активізацією більшості

несправностей. Однак у деяких випадках приведення до симетричного вигляду вимагає великої кількості додаткових операцій. Цього можна уникнути тільки за рахунок зниження виявляючої здатності. Після додавання операцій виконується перевірка, чи всі цільові несправності виявляються створеним алгоритмом. До появи методики аналізу поведінки несправностей ця перевірка являла собою важке завдання, особливо для тестів великої складності. Здатність локально симетричних алгоритмів виявляти помилки перевіряється в процесі вибору симетричних стадій, що також може бути автоматизовано.

Ще один тип дефекту є дефект транзистора доступу bitline (показника вибору заданого стовпчика, Bit Line Imbalance Fault BLIF). При наявності дефекту даного типу, струм витоку транзистора доступу bitline має підвищене значення, таким чином значення, записане в осередку, «витікає» з неї за кінцевий час. Найбільша вірогідність для прояву дефекту виникає в разі, коли всі осередки у відповідній колонці містять значення, інверсні міститься у перевіреній осередку. На цьому і ґрунтується принцип виявлення дефекту типу BLIF - створення найгірших (тобто найбільш ймовірних) умов для прояву дефекту [12].

Реалізація звернення до пам'яті за рядку дозволяє перевіряти дефекти транзистора доступу для всіх осередків рядка одночасно. У пам'яті, ініціалізованої нулями, в перевіряється рядок записується значення логічної одиниці в усі осередки, потім відбувається зчитування значення і порівняння його з одиницею. У разі успішного проходження перевірки, значення всіх осередків рядка перетворюється на стан логічного нуля, і управління переходить до перевірки наступного рядка. Для цього необхідно додавання марш-елементів, що становлять марш-тест BLIF. Однак з огляду на кінцевий час зміни струму витоку транзистора доступу, зчитування одиниці відразу ж після її запису може не виявити наявність дефекту навіть при створенні найбільш ймовірних для цього умов.

Для повного виявлення даного типу дефектів потрібна перевірка кожної пари осередків, чиї адреси wordline відрізняються тільки на 1 біт (далі будемо називати такі осередки «сусідніми»). При цьому один з осередків повинна

містити значення, інверсне значенням іншого осередку, і перемикання між ними має проводитися в обидві сторони. У такому випадку при роботі з кожною клітинкою блоку пам'яті повинні перевірятися осередків пам'яті (де n - кількість всіх елементів пам'яті), тобто тільки осередки пам'яті, які є «сусідніми» для перевіряється [13]. Таким чином, тестовий алгоритм вже не може бути названий марш-тестом, але може бути представлений схожим чином.

Порядок проходження адрес для перевірки комірок пам'яті може бути сформований двома способами. Справді, якщо необхідно перевірити правильність виконання переходів, наприклад, між адресами 101 і 100, то це може бути зроблено двома способами (100-101-100 і 101-100-101), в залежності від того, з якої адреси починати, але самі переходи при цьому абсолютно однакові, тільки переставлені місцями.

Дане зауваження дозволяє істотно знизити складність тесту і спростити його реалізацію. Найбільш зручно, щоб не порушувати симетрію тестового алгоритму, залишити і в другому, і в четвертому марш-елементах по одній операції читання з кожного осередку, що є «сусідній» для перевіряється, Наприклад, перевірка всіх переходів в ланцюжку адрес 000-001- 000 при всіх можливих значеннях в осередках буде відбуватися по частинах в зазначених марш-елементах: в першому перевіряються переходи 000-001 і 001-000 при наявності в першій клітинці одиниці, а в другій нуля; у другому перевіряються переходи 000-001 і 001-000 при наявності в першій клітинці нуля, а в другій одиниці.

2.3 Пошук кодочутливих несправностей ОЗП

Швидкий прогрес у галузі напівпровідникових технологій призвів до створення різноманітних типів запам'ятовувальних пристроїв великої ємності, які широко використовуються в різних додатках [2]. Так, запам'ятовуючі пристрої сучасних обчислювальних систем займають до 94 % площі кристала системи, що призводить до зростання вимог до їх надійності [5]. Непрямим результатом такого зростання ємності запам'ятовуючих пристроїв є не тільки збільшення ймовірності відмов пам'яті, а й збільшення складності несправностей

запам'ятовуючих пристроїв, що зумовлена високою щільністю розміщення запам'ятовуючих комірок, а також різноманітними механізмами виникнення несправностей [6].

Дефекти взаємного впливу відносяться до прояву несправностей у роботі комірок пам’яті, коли одні комірки впливають на інші та відносяться до кодочутливих несправностей. Кодочутливими несправностями (PSF — pattern sensitive faults) називають такий тип дефектів пам’яті ОЗП, які досить важко відшукати та локалізувати. Даний тип дефектів залучає дві та більше комірок масиву пам’яті (і відноситься до так званих некласичних дефектів). Для даного типу прояву дефектів поведінка однієї комірки залежить від наявного стану або логічних переходів у сусідніх комірках. У цьому процесі розрізняють базову комірку (base cell), що виступає в ролі потерпілого для моделі класичних дефектів взаємного впливу CF, а також сусідні комірки (neighborhood cells), які виконують у цьому процесі роль агресора. Якщо масив комірок пам’яті являє собою деяку прямокутну матрицю, черговість розташування комірок в якій відповідає їхнім адресам, а дефекти створюються тільки фізично суміжно розміщеними комірками, то такий тип моделі називають редукційною. Коли ж розміщення комірок пам’яті може бути довільною, то така модель є нередукційною PSF. Редукційні моделі коректно можуть описати певну обмежену кількість випадків побудови пам’яті з відомою топологією кристала. Нередукційні моделі розміщення комірок є універсальними.

Одним із типів несправності запам'ятовувальних пристроїв є несправності, що чутливі до коду (pattern sensitive faults, PSF) [7]. Несправності PSF спричинені аномаліями запам'ятовуючого пристрою та паразитними ефектами, які проявляються залежно як від збережених у пам'яті даних, так і від їхніх змін і послідовностей звернень (операцій читання та запису) до пам'яті [8]. У запам'ятовуючому пристрої ємністю N біт може зберігатися 2N різних наборів двійкових даних, надійність запису і зчитування яких забезпечує справну поведінку пам'яті. Показано, що для визначення несправності PSF запам'ятовуючого пристрою мінімальна складність тестової послідовності

визначається виразом $2N (3N^2 + 2N)$ [9]. Це призводить до того, що для великих об'ємів сучасних запам'ятовуючих пристроїв часова складність реалізації алгоритмів тестування пам'яті надзвичайно велика [10]. Тому завдання тестування запам'ятовувальних пристроїв з метою виявлення кодочутливих несправностей упродовж останніх десятиліть і до теперішнього часу було і є вельми актуальним.

Кодочутливі несправності PSF описують поведінку кількох комірок пам'яті аж до N комірок, де N - ємність пам'яті в бітах [6]. Для подібних несправностей логічний стан одного осередку пам'яті, званого базовим (base cell), може залежати від вмісту (0 або 1) або від логічних переходів з 1 у 0 або з 0 у 1 у сусідніх осередках (neighborhood cells) запам'ятовуючого пристрою. Під логічним переходом з 1 в 0, що позначається символом ↓, розуміють зміну одиничного стану комірки пам'яті на нульовий стан, а під переходом з 0 в 1 - зміну нульового стану на одиничний, що позначається символом ↑ [6, 7]. Розрізняють два види кодочутливих несправностей: необмежені (unrestricted) і обмежені (restricted), або граничні (neighborhood, NPSF). Під NPSF розуміють такі різновиди несправностей PSF, для яких вводять обмеження як на кількість осередків пам'яті, що беруть участь у несправності, так і на їхнє фізичне розташування в матриці запам'ятовувальних осередків. При тестуванні сучасних запам'ятовувальних пристроїв зазвичай дотримуються останньої, реальнішої моделі кодочутливих несправностей, для якої розглядають невелику кількість $k \leq 9$ осередків пам'яті, які входять до несправності NPSFk, а їхнє місце розташування може бути довільним [16].

Існують три класичні моделі несправностей NPSFk, а саме активні (ANPSFk), пасивні (PNPSFk) і статичні (SNPSFk) [6]. Активними, або динамічними, є такі NPSFk, для яких базова комірка змінює свій вміст через зміни в наборі, що зберігається в сусідніх k-1 комірках. Пасивними називаються такі NPSFk, для яких вміст базової комірки не може бути змінений через певний набір даних у сусідніх k-1 комірках. Статичні SNPSFk - це несправності, за яких вміст базового осередку при нудитивно переводиться в один із двох станів 0 або

1 через певний набір у сусідніх k-1 комірках. Зазначимо, що всі перераховані несправності виникають при зміні вмісту пам'яті, тобто при виконанні операції запису в комірку пам'яті [5, 6].

Існують різні види кодочутливих несправностей NPSFk залежно від тих обмежень, які накладаються на узагальнюючу модель PSFk. Першим необхідним обмеженням є кількість запам'ятовуючих елементів, що беруть участь у несправності NPSFk [5, 6]. Як правило, k не перевищує 9. Це випливає з того факту, що для тестування подібних несправностей необхідний час, пропорційний величині 2k. Другим обмеженням є фізичне сусідство комірок пам'яті. На практиці зазвичай використовують моделі кодочутливих несправностей NPSFk із числом k, що дорівнює 3, 5 або 9, конфігурації та позначення яких показано на рис. 2.3 [11].

Як випливає з рис. 2.3, у кожній із наведених несправностей у явному вигляді виокремлюють базовий запам'ятовувальний елемент b і сусідні запам'ятовувальні елементи n. Сусідні комірки запам'ятовуючого пристрою є фізичними сусідами щодо базового елемента. Дослідження, присвячені побудові тестів для виявлення NPSF3 за відомої фізичної топології, досить повно представлені в роботах [15]. Найпоширенішими моделями NPSF є NPSF5 і NPSF9, які отримали назву несправностей типу 1 (type 1) і типу 2 (type 2) [10]. Існують й інші конфігурації NPSFk як за взаємним розташуванням запам'ятовувальних елементів, так і за їхньою кількістю. Одним із таких поширених різновидів несправностей є NPSFk, для якої всі k запам'ятовувальних елементів належать одному стовпчику або одному рядку матриці комірок пам'яті [6], а максимальне значення k досягає значення 25 [17]. Серед зазначеної множини несправностей виокремлюють neighborhood word-line sensitive faults (NWSF), тобто несправності, пов'язані з шиною даних [20]. У роботі [22] було запропоновано варіант сусідства з чотирма комірками (T-type), яке забезпечувало виявлення несправностей, чутливих до двійкового набору оточення бітової шини пам'яті (bit-line neighborhood pattern sensitive faults, NBLSFs). Досить повно моделі кодочутливих несправностей (row (column)

pattern sensitive faults), пов'язані з топологією рядків і стовпчиків матриці запам'ятовувальних елементів, досліджено в роботі [18].

Розглянемо тест для виявлення взаємного впливу комірок пам'яті в режимі запису. Ця несправність приводить до руйнування інформації в осередку, що перевіряється, при багаторазовому записі в інші осередки, причому найбільший вплив роблять сусідні осередки. Тест має вид [18]:

$$P = \prod_{d=0}^{N-1} V_a W(E_a) R_a \qquad (2.4)$$

де V_a— запис нуля в осередок E_a;

R_a — зчитування стану осередку E_a;

$W(E_a)$ — послідовність операцій запису одиниці в усі осередки підмножиниE_a.

Для повної перевірки розглянутої несправності тест повторюють для комплементарних даних. Загальна довжина тесту дорівнює 2N(n + 2), n — число сусідніх осередків. При відомій топології кристала n приймають рівним 4, оскільки як сусідні осередки вибирають хрестоподібно розташовані (стосовно що перевіряється) осередку. Якщо топологія кристала невідома, то в множину сусідніх осередків рекомендують включати всі осередки рядка і стовпця (крім тих, що перевіряється), на перетинанні яких знаходиться осередок, що перевіряється. У такому випадку загальна довжина сформованого тесту стає рівною $4N^{3/2}$. Такого типу тести гарантують перевірку функціонування комірок пам'яті при визначеній моделі несправностей і дешифратора адреси пам'яті.

На рисунку 2.4 показаний граф стану комірок пам’яті запам’ятовуючого пристрою, в якому присутня кодочутлива несправність. Несправність полягає у тім, що друга комірка пам’яті діє на третю комірку пам’яті при певних умовах. Коли в другу комірку пам’яті записується значення логічної 1, то вона діє на третю комірку і записує в третю комірку автоматично значення логічної 1 і тим самим спотворює код, що мав бути записаний у ці три комірки пам’яті.

В таблиці 2.1 наведені стани вірно і не вірно функціонуючих комірок пам’яті для розглянутої кодочутливої несправності (несправності впливу).

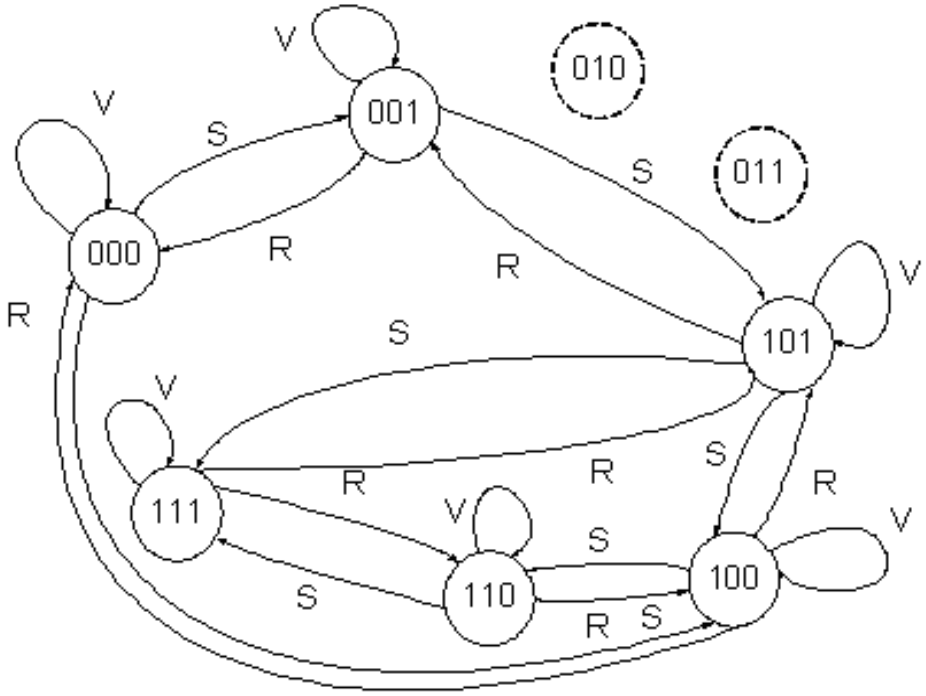

Рисунок 2.4 — Граф-схема пристрою пам’яті з кодочутливою несправністю

Таблиця 2.1 —Вірний і спотворений коди

№ стану	Вірний код x2 x1 x0	Спотворений код y2 y1 y0
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	1 1 0
3	0 1 1	1 1 1
4	1 0 0	1 0 0
5	1 0 1	1 0 1
6	1 1 0	1 1 0
7	1 1 1	1 1 1

Послідовність тестування фігуруватиме такою. В комірку за певною адресою записуємо значення 0. Після цього у дві сусідніх комірки з адресами більше на 1 та менше на 1 записуємо значення 0. Зчитуємо ці дані і перевіряємо

із попередніми. На слідую чому кроці у сусідні старшу та молодшу комірки по відношенню до вибраної комірки записуємо 1. Зчитуємо ці дані та звіряємо їх із початковими. Після цього у базову вибрану комірку заносимо значення 1, послідовно у старшу та молодшу сусідні записуємо спочатку по 0, зчитуємо та звіряємо, після цього у ці ж комірки записуємо по 1, зчитуємо та звіряємо. Якщо виявлена несправність, то фіксуємо, і переходимо до перевірки наступної за адресою комірки. Дана послідовність дій може бути описана таким чином.

w000, r000, xx000xx

w1x1, r101, xx101xx

w010, r010, xx010xx

w1x1, r111, xx111xx

w000, r000, xxx000x

w1x1, r101, xxx101x

w010, r010, xxx010x

w1x1, r111, xxx111x

де w0 — запис логічного нуля в комірку; w1 — запис логічної одиниці; r0, r1 — зчитування відповідно логічних 0 та 1 із комірки; стан аналізованих (000) та оточуючих xx із двох сторін комірок xx000xx, із яких для перевірки відбираються три підряд комірки.

2.4 Формування послідовності перевірки працездатності ОЗП

У загальному випадку вважається, що помилка в роботі кристала відбувається через будь-які механічні або електричні пошкодження транзисторів і прилеглих елементів при виготовленні або роботі пристрою, що відбивається на функціонуванні більших частин схеми, таких як елементи пам'яті, окремі логічні елементи або великі логічні блоки. Для виявлення помилок в SRAM-пам'яті широко поширене сімейство так званих марш-тестів (March-тестів), які

характеризуються не тільки хорошим покриттям різних типів дефектів, але і порівняно простою реалізацією, а також лінійно залежать від кількості елементів пам'яті часом тестування. Марш-тест являє собою послідовність марш-елементів, кожен з яких складається з послідовності операцій запису (w0 - запис логічного нуля в клітинку, w1 - запис логічної одиниці) і читання (r0 - читання з комірки і порівняння результату з "0", r1 - читання з комірки і порівняння з "1"; в разі розбіжності при порівнянні реєструється помилка) і індикатора напрямку (⇑ - інкрементування адреси, ⇓ - декрементування адреси, ↕ - порядок адресації не важливий). Операції виконуються послідовно з кожною клітинкою пам'яті, адреса змінюється відповідно до індикатором напрямку. За підсумками порівняння результатів читання з очікуваними (еталонними) формується сигнатура помилок, яка може бути використана в подальшому для отримання статистики або для компенсації несправностей пам'яті.

Алгоритм діагностування комірок пам’яті запам’ятовуючого пристрою у загальному випадку включає в себе такі етапи:

— установка комірок пам’яті і лічильника адрес в заданий стан;

— підрахунок мікрокоманд;

— опитування стану комірок пам’яті;

— порівняння результату опитування з еталоном;

— виведення результатів тесту (індикація несправності).

Для перевірки правильності роботи запам’ятовуючого пристрою пропонується такий алгоритм:

— визначається вільний об’єм оперативної пам’яті, який можна перевіряти;

— формується масив даних розміром n x n, який заноситься до вибраної ділянки пам’яті;

— в першу комірку першого рядка вибраного масиву записуємо значення 1;

— зчитуємо записану інформацію і звіряємо із початковим значенням до занесення в пам’ять;

— якщо є відмінність між записаною та зчитаною інформацією, то фіксуємо несправність.

— в наступну комірку пам’яті записуємо значення 1;

— перевіряємо, чи весь масив пам’яті заповнений значеннями 1. Якщо ні, то на 4, інакше на 8;

— в першу комірку першого рядка вибраного масиву записуємо значення 0;

— зчитуємо записану інформацію і звіряємо із початковим значенням до занесення в пам’ять;

— якщо є відмінність між записаною та зчитаною інформацією, то фіксуємо несправність;

— в наступну комірку пам’яті записуємо значення 0;

— перевіряємо, чи весь масив пам’яті заповнений значеннями 0, якщо ні, то на 9, інакше на 12;

— виводимо інформацію про наявні несправності та завершення тесту;

— кінець алгоритму.

Запропонований алгоритм служить основою для побудови програми тестування персонального комп’ютера.

В даному розділі були розроблені моделі і алгоритм перевірки працездатності вузлів оперативних запам’ятовуючих пристроїв комп’ютерів. Вони ґрунтуються на способі послідовного функціонального аналізу. Для цього була розроблена модель дефектів комірок оперативного запам’ятовуючого пристрою. Розроблена модель і проведений її аналіз зробили можливим виконання оптимізації отриманих засобів тестування, що підвищило контролездатність і ефективність роботи засобів діагностування в цілому.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ТЕСТУВАННЯ ПАМ'ЯТІ КОМП'ЮТЕРА

У даному розділі роботи розглянуті стратегія діагностування, виконана розробка програм перевірки стану ОЗП в умовах експлуатації із використанням різних типів тестів для виявлення кодочутливих несправностей пам'яті комп'ютера.

3.1 Формування стратегії діагностування стану електронних вузлів

Вузли реалізуються у вигляді різних стратегій, потім у вигляді програм моніторингу ефективності та виявлення несправностей електронного обладнання. Розглянемо основні методи організації контролю працездатності та виявлення несправностей електронного обладнання.

Метод послідовного функціонального аналізу є одним із варіантів побудови програми моніторингу роботи та несправностей ЕА [15], суть якого полягає в тому, що виходячи з призначення ЕА визначаються основні функції Fi, що дає нам можливість. Завдання. Значення основних функцій в будь-який момент часу залежать від значень ряду аргументів. Для ЕА такими аргументами основних функцій будуть його фізичні параметри. Інші ключові функції ЕА будуть залежати від низки інших фізичних параметрів. Як правило, для всіх фізичних параметрів ЕА відомі їх допустимі межі варіації. Таким чином, контроль продуктивності ЕА буде складатися з контролю параметрів, від яких залежать основні функції. Якщо якась основна дія не виконується за одним із своїх аргументів (параметрів), то виникає помилка виявлення завдання. У цьому випадку аргумент (параметр), значення якого перевищує встановлену межу допуску, слід, у свою чергу, розглядати як функцію деяких інших аргументів. Останні також, очевидно, є фізичними розмірами ЕА малих пристроїв або елементів схеми.

Запропонована методика побудови програми контролю працездатності та пошуку несправностей вимагає дуже спрощеного логічного аналізу об'єкта контролю (ОК), наочності отриманих результатів, при цьому потрібна

мінімальна вихідна інформація про ОК. Однак, незважаючи на очевидні переваги, цей підхід має недоліки, основний з яких полягає в тому, що отриманий алгоритм пошуку несправностей не є оптимальним з точки зору часу або вартості залучених засобів перевірки.

Якщо в ЕА є послідовно сполучені елементи, тоді при розробці програми пошуку несправностей досить часто використовують спосіб половинної розбивки [8]. При цьому вихідна інформація для побудови програми пошуку утримується в межах функціональної моделі. ЕА з послідовно сполученими елементами та має певні особливості. Контроль працездатності цих пристроїв полягає у перевірці вихідного параметра Z_n. Для пошуку несправності вз заданою глибиною слід контролювати всі виходи Z_i функціональних елементів, окрім останнього.

ому для побудови пошукової програми немає необхідності вибирати контрольовані параметри, оскільки вони задані множиною Z1, Z2,…, ZN. Залишається лише визначити послідовність контролю цього набору параметрів, щоб середня вартість пошуку будь-якого несправного елемента була мінімізована.

Відомий також інший механізм керування – пошук несправностей за допомогою відгалужень і меж [13]. Такий математичний підхід дозволяє визначити порядок отримання найкращого рішення з безлічі можливих рішень. Для цього область можливих розв’язків послідовно розбивається на все менші підмножини, і для кожної з них обчислюється нижня межа функції, що зводиться. Підмножини, значення нижньої межі яких перевищує деякий заданий розмір, виключаються з подальшого розгляду. Процес розбиття триває до тих пір, поки не буде знайдено будь-яке рішення, в якому значення функції вибору не перевищує значення нижньої межі для будь-якої підмножини. Джерелом інформації є функціональна модель ОК для побудови пошукової програми та таблиця несправностей з ймовірностями різних станів.

Також широко використовується метод пошуку несправностей за ієрархічним принципом [8]. В цьому випадку N первинних функціональних

елементів розбивають на групи по декілька елементів в кожному. При відмові якогось первинного функціонального елементу послідовно перевіряються індикатори всіх сходинок, дотримуючи порядок та тривалість кожного перегляду відповідно до прийнятого циклу. Після виявлення індикатора, який зафіксував несправність ОК, подальший перегляд індикаторів першого кроку припиняється. Після цього слід переглянути індикатори другого кроку, які сполучені з індикатором першого кроку, де зафіксована несправність і далі повторювати послідовність виконання поставленої задачі.

Процедура триває до тих пір, поки не буде виявлено несправний перший функціональний елемент. Така ієрархічна схема дозволяє суттєво скоротити час, необхідний для пошуку несправного елемента ОК, порівняно з безпосереднім пошуком околиці з N елементів.

Описані процедури усунення несправностей дозволяють налаштувати електронні пристрої керування з різними конфігураціями.

В умовах експлуатації існують різні стратегії контролю продукту для перевірки точності електронних пристроїв. Щодо контролю електронних блоків можна виділити два підходи: контрольне випробування та перевірка, коли на об’єкт діагностування (ОД) подаються спеціально сформовані тестові сигнали, і функціональна перевірка, коли ОД перевіряється при подачі робочих сигналів [4].

При використанні тестового контролю існує набір тестових послідовностей $\overline{T_k}=\{t_i \mid i = 1, n\}$, при цьому кожному з них відповідає певна реакція $\overline{R_k}=\{r_i \mid i=1,n\}$. При розбіжності такої реакції R_k вузла відбувається діагностування з використанням відповідних тестів $\overline{T_Д} = \{t_j \mid j=1, m\}$.

Мета і сутність функціонального контролю полягає у спостереженні реакцій $U_в$ вузла, який перевіряється, при подачі впливів $U_{вх}$ на вхідні точки вузлів і подальшого порівняння вихідних сигналів із еталонними значеними $U_{во}$. Вузол, який перевіряється, визнається справним, якщо для всіх робочих впливів отримуємо:

$$\Delta U = U_в - U_{во} \le \Delta U_{эт}, \qquad (3.1)$$

де $\Delta U_{эт}$ — допустиме відхилення реакції вузла, яуий перевіряється, від еталонного значення.

У плані організації процесу діагностування для електронних блоків найбільшими можливостями володіє тестове діагностування. У випадку тестового діагностуванні виконується тестування об'єкта в цілому. На входи ОД з боку роз'єму надходять певні тестові впливи, а на виходах ОД відбувається перевірка справності роботи функції, яка реалізовується контрольованою схемою. Після такого завершення процесу тестування ОД визнається придатним, коли для заданого набору вхідних сигналів отримуємо такий вихідний сигнал, який з заданою точністю відповідає еталонному наборові сигналів, які подаються на вхід . Якщо ця умова не виконується в якомусь вихідному наборі, то ОД вважається несправним. Потім, щоб визначити тип дефекту, представлені додаткові підказки, які дозволять визначити тип дефекту (тобто представлений тест пошуку дефекту).

У цьому випадку для організації перевірок блоків електронної оперативної пам’яті необхідно використовувати конструкційний контроль. Особливості конструкційної тестової установки наведено в наступних параграфах даної роботи.

3.2 Розробка структури програми тестування ОЗП комп’ютера

Задача, яку має вирішити дана програма, є пошук та виявлення зон у ОЗП, де є несправності взаємного впливу (кодочутливі несправності). Повинні бути виявлені такі несправності взаємного впливу, де певна комірка пам’яті впливає на наступну або попередню сусідні комірки. Так як при такій перевірці відбувається почерговий запис і зчитування логічних 0 та 1 в усі комірки пам’яті, то таке тестування можна виконати тільки до тієї частини пам’яті комп’ютера, яка у даний момент часу не задіяна під інші ресурси. Тому обов’язковою умовою початку процесу тестування є першочергове визначення обсягу тієї частини пам’яті комп’ютера, яка у даний час вільна. Ця частина програми має бути виконана на першому етапі роботи програми. Інша частина програми, це набір

модулів, що дозволяють запускати та виконувати тести для виявлення тих моделей несправності, що описують дію несправності взаємного впливу. Якщо будуть виявлені несправності пам’яті комп’ютера на цьому етапі перевірки, то про це також має бути надана інформація. Отже, структура програми має складатися із трьох частин: підготовка до тестування, власне сам процес тестування і виведення результатів тестування. Кожна із цих частин буде мати свій набір модулів.

Перша частина програми буде складатися із модуля визначення вільного обсягу ОЗП комп’ютера для виконання операцій подальшого тестування. У цій частини програми бажано також отримати основні дані про тип запам’ятовуючого пристрою та його характеристики. Це бажано виконати з використанням окремої дії, по якій запускається процес для визначення вільного обсягу пам’яті комп’ютера та аналіз її характеристик.

Наступна частина програми, це набір тестів для виявлення несправності взаємного впливу. Тут також бажано надати дані про час проведення операцій тестування. Виконання кожного із підготовлених тестів програми вимагає деякого інтервалу часу, про який можна доводити до користувача програми.

Завершальною частиною програми тестування є формування звіту про сам процес тестування та його результати. Тут мають бути вказані несправності, якщо вони є у складі пам’яті комп’ютера, що перевіряється, та вказати про кількість виявлених несправностей.

Пошук можливих типів несправності ОЗП комп’ютера будемо здійснювати із використанням тестів, які спроможні виявити найбільш ймовірні несправності пам’яті комп’ютера. До них у першу чергу відносяться кодочутливі несправності (несправності взаємного впливу). Для написання програми тестування ОЗП комп’ютера використовуємо мову програмування C# [20-22].

Мова програмування C# є такою мовою програмування, яка є об'єктно-орієнтованою й дозволяє зменшити час на розробкуу програмного продукту. Швидкість створених програм, написаних на мові C#, досить висока. Додатковою перевагою цієї мови є те, що розмір програмного забезпечення для

виконання роботи є досить невеликим і не вимагає залучення додаткових бібліотек. Для виконання програм, написаних мовою C#, обов’язковою умовою є установка на комп'ютері лише додатку .Net Framework. Ще однією перевагою програм на мові C# є можливість їх виконання на різних платформах, що використовують .Net Framework. Тому програми, написана на мові C#, не залежать ні від апаратної конфігурації комп'ютера, ні від платформи.

Мова програмування C#, будучи однією з широко використовуваних мов програмування, увібрала в себе значний досвід і найкращі аспекти попередніх існуючих мов програмування, і при цьому вона була створена спеціально для архітектури .NET. Використання архітектури .NET диктує їй (так як і іншим мовам програмування, на яких можна писати і працювати під .NET) об'єктно-орієнтованого підхід при створенні програм.

Синтаксис мова C# успадкувала від попередніх мов C++ і Java. Для розробників програм, які мають досвід створення програм на цих мовах, легко адаптуватися і відшукають в C# багато знайомих ознак. Проте в ній є багато інновацій, таких як делегати та події, атрибути, вони чудово вмонтовані в ідеологію цієї мови та зайняла стабільну позицію серед розробників програм із використанням .NET. Їх введення створили умови для використання принципово нових методик програмування [32]. Підтримуючи основні риси своїх великих попередників, мова програмування C# стала значно простішою та більш захищеною.

Поєднання таких кращих ідей сучасних мов програмування робить мову C# не тільки надбанням їх переваг, але й мовою програмування нового покоління. Враховуючи усі перераховані переваги цієї мови для створення програмного засобу, що розробляється, була вибрана саме мова програмування C#.

Сформована структурна схема програми складається із трьох частин: перша частина надає дані про вільний обсяг пам’яті та її основні характеристики, друга частина здійснює тестування пам’яті та третя частина програми формує звіти про виконані тести та виявлені можливі несправності пам’яті. Виконання

кожної із частин можливе тільки після виконання попередньої частини. Перша частина програми формує дані про основні характеристики пам’яті, скільки є всього пам’яті та яка частина пам’яті доступна, вказує тип пам’яті, слот пам’яті, форм-фактор чіпу та вид використовуваної пам’яті. Друга частина складається із групи незалежних один від одного тестів та визначає час для їх виконання. По завершенню процесу тестування виводиться інформація про задіяні тести та час їх виконання. Графічне зображення структурної схеми створеної програми для тестування оперативної пам’яті наведено у Додатку Б до магістерської роботи.

Перша частина програми складається із групи модулів, виконання яких відбувається після задіяння процесу під назвою «Отримати інформацію». Для обробки цієї дії використовуємо метод private void_button1_Click (object sender, EventArgs e). Він запускає окремий потік для визначення кожного із таких параметрів, як форм-фактор чіпу Thread(ddrFormFactor), тип пам’яті Thread(memoryType), слот пам’яті Thread(socketDDR) та інші характеристики пам’яті. Фрагмент коду програми приведено у лістингу 3.1 програми.

Лістинг 3.1 — Фрагмент коду визначення характеристик пам’яті

```
info++;
 Thread tRealMemory = new Thread(realMemory);
 tRealMemory.Start();
 Thread tMemoryType = new Thread(memoryType);
 tMemoryType.Start();
```

Для отримання даних про тип представлення пам’яті комп’ютеру використовується метод private void TypeDetail(). У цьому методі використано цикл foreach, який дозволяє виконати опитування запам’ятовуючого пристрою. Визначаються такі можливі показники пам’яті комп’ютера, як Reserved, Fast-paged, Other, Unknown, Static column та інші, сформований перелік яких приведено у Додатку В до магістерської роботи. Фрагмент коду програми по визначенню типу представлення пам’яті приведено у лістингу 3.2 програми.

Лістинг 3.2 — Фрагмент коду визначення типу представлення пам’яті

```
TypeDetail = Convert.ToInt32(typedetReturn["TypeDetail"]);
```

```
switch (TypeDetail)
{case 1:
      textBox6.Text = ("Other");
      break;
```

Деталізація даних про тип пам’яті комп’ютеру відбувається із використанням методу private void memoryType (). Тут уточнюються такі можливі типи пам’яті комп’ютеру, як DRAM, RAM, EDRAM та інші, сформований перелік яких приведено у Додатку В до магістерської роботи. Для визначення типу пам’яті комп’ютеру виконується такий порядок дій (лістинг 3.3).

Лістинг 3.3 — Фрагмент коду уточнення типу пам’яті

```
memoryType = Convert.ToInt32(typeReturn["MemoryType"]);
switch (memoryType)
```

Надалі послідовно визначаються можливі типи пам'яті, наприклад визначається динамічна оперативна пам’ять (лістинг 3.4).

Лістинг 3.4 — Фрагмент коду визначення конкретного типу пам’яті

```
case 4:
   textBox4.Text = ("Cache DRAM");
   break;
```

Лістинг розробленої програми для визначення характеристик оперативної пам'яті наведено у додатку В до магістерської роботи.

3.3 Контроль стану пам’яті за допомогою маршових тестів

Перевірку стану оперативної пам'яті комп’ютера будемо здійснювати із застосуванням декількох типів тестів. Пошук найбільш поширених констатних несправностей пам'яті будемо виконувати із використанням стандартного маршового тесту. Для пошуку кодочутливих несправностей використаємо інші типи тестів. Контроль із використанням маршового тесту будемо здійснювати таким чином. Попередньо у виділений вільний об’єм пам'яті занесемо значення нулів. Наступним етапом послідовно у комірки пам'яті будемо записувати

значення логічних одиниць та перевіряти стан комірок шляхом виконання операції зчитування. По досягненню занесення логічних одиниць у виділений обсяг оперативної пам'яті приступаємо до послідовного занесення логічних нулів та перевірки виконання цієї операції шляхом зчитування стану комірок пам'яті. Використання цього тесту дозволяє перевірити стан запам’ятовуючого пристрою на відсутність константних несправностей.

На початку виконання кожного маршового елементу, що починається із операції запису, обов’язково виконуємо операцію зчитування. Це необхідно для запобігання стиранню наступним маршовим елементом поточного стану комірки пам'яті, що перевіряється.

Виявлення згенерованих типів несправності при тестуванні за цим алгоритмом виконується шляхом порівняння еталонних і робочих даних. Для попередньо розглянутих типів несправності такі помилки не спотворять дані й тому при виконанні такого тесту всі можуть бути виявлені. Вибір конкретних позицій для комірок пам'яті, у які додатково вносяться такі коригувальні операції зчитування, впливають на виявляючу здатність несправностей алгоритмів. На початку виконання алгоритму введення додаткової операції зчитування є кращим підходом, оскільки вона виконується перед активізацією більшості типів несправності. Після додавання операцій зчитування виконується перевірка, чи усі можливі типи несправності можуть бути виявлені створеним алгоритмом.

Для виконання маршового тесту використовуємо підтвердження, що маркер установлено на кнопці button2. Для оброблення цієї події використовуємо метод private void button2_Click. Для виконання даного алгоритму використовуємо такі типи даних, як DateTime для встановлення часу початку виконання тесту та TimeSpan, який фіксує час закінчення тесту.

Модуль програми для тестування пам'яті за допомогою маршового тесту матиме такий вигляд. Сам процес тестування пам'яті будемо виконувати для визначеного вільного обсягу пам’яті побайтно. Спочатку у визначений обсяг пам'яті заносимо значення логічних нулів.

Лістинг 3.5 — Фрагмент коду підготовки до тестування пам’яті

```
DateTime dold = DateTime.Now;
        int errorPercent = 0;
        bool wtf = true;
        byte[] marsh = new byte[N8];
        byte value;
        int p = 0;
```

Надалі виконання процесу тестування складається із таких кроків. У доступній області пам’яті створюємо масив розміром 8х8 комірок. Потім заповнюємо цю частину пам’яті послідовно одиницями і зразу перевіряємо на наявність цих занесених одиниць, тим самим виконуємо пошук можливих помилок.

Наступним етапом є заповнення цієї виділеної частину пам’яті логічними нулями і перевіряємо на наявність запису цих значень послідовно у кожну комірку пам’яті, Якщо буде виявлена невідповідність, то формуємо повідомлення на наявність помилок (див. Додаток В до магістерської роботи).

Лістинг 3.6 — Фрагмент коду занесення і перевірки одиниць у пам’ять

```
 for (int i = 0; i < N8; i++)
    { marsh[i]=255;
    if (marsh[i] != 255) errorPercent++;
    for (int j = 0; j < 8; j++)
      {  value = RightStep0((j + 1));
        marsh[i] = value;
        if (marsh[i] != value) errorPercent++;
        p++; }
```

По завершені цієї процедури перевірки виділеної частини пам’яті виводимо інформацію про помилки.

Лістинг 3.7 — Фрагмент коду виведення повідомлення про несправності

```
textBox11.Text = Convert.ToString(errorPercent).
```

Повний лістинг програми наведений у Додатку В до магістерської роботи.

3.4 Розробка програмних засобів пошуку несправностей компонентів оперативних запам’ятовуючих пристроїв комп’ютерів

Ще одним тестом буде перевірка комірок оперативної пам'яті на відсутність несправності взаємного впливу (кодочутливих помилок). Для цього послідовно в кожну наступну комірку виділеного обсягу пам'яті будемо записувати значення логічної одиниці та здійснювати її зчитування із цієї комірки. Потім у розміщені рядом сусідні справа й зліва комірки пам'яті записуємо значення логічної одиниці для виконання операції перевірки про можливий вплив цієї комірки пам'яті на розміщені рядом сусідні комірки. Таким способом буде виконуватися перевірка на можливий вплив кожної із комірок пам’яті комп’ютера та виявлення можливої несправності впливу сусідніх комірок. Перед початком тестування кожної комірки пам’яті таким типом тесту, що починається із виконання операції запису, обов’язковим етапом є виконання операції зчитування даних. Виконання цієї операції необхідно для запобіганню стиранню цим тестом поточного стану комірок пам'яті, що перевіряються.

Для виконання тесту «Тест №3» ми використовуємо підтвердження що маркер встановлено на кнопці button10. Обробником цієї події є метод private void button10_Click(object sender, EventArgs e).

В даному алгоритмі використовуємо такі типи даних: DateTime – встановлює час початку тесту, TimeSpan – фіксує час закінчення тесту, int та bool.

Виконання алгоритму наведено нижче.

По замовчуванню установлюємо, що на початку виконання даного тесту нема помилок. Встановлюємо мітку bool wtf = true. У доступній для тестування області пам’яті створюємо масив розміром 4092x4092. У цей масив записуємо значення логічних нулів в усі комірки. Потім у другу комірку записуємо значення одиниці та перевіряємо дані у попередній та наступній за цією коміркою. Якщо вони дорівнюють нулю, то у комірках вірна інформація. Тепер у попередню та наступну за цією комірки заносимо значення логічних одиниць із використанням дії byte value101. Виконуємо процес зчитування даних та перевіряємо на наявність у них одиниць із використанням маршового тесту marsh[i] | value101. Якщо це так, то переходимо до адреси наступної комірки. Повторюємо усі попередньо описані операції. Таку перевірку здійснюємо для

всього виділеного обсягу оперативної пам’яті. Якщо під час перевірки будуть виявлені помилки, то їх фіксуємо, наприклад value101 >>= 1 та виводимо повідомлення про несправність errorPercent++. Також перевіряємо на наявність послідовності занесених даних у вигляді value010. Фрагмент коду , що формує описану послідовність дій, приведено у лістингу 3.8, повний код наведено у Додатку В до магістерської роботи.

Лістинг 3.8 — Фрагмент коду пошуку несправності впливу.

```
for (int j = 0; j < N4096; j++)
{    marsh[i] = 0;
   marsh[i] = (byte)(marsh[i] | value101);
   if (marsh[i] != value101)
      errorPercent++;
   marsh[i] = value010;
   marsh[i] = (byte)(marsh[i] | value101);
   if (marsh[i] != value111)
      errorPercent++;
   value101 >>= 1;
   value010 >>= 1;
   value111 >>= 1;
```

Ще один тип тесту, що дозволяє знайти несправності подвійного впливу однієї комірки на дві сусідніх попередніх або на дві сусідніх наступних комірок. Він подібний до попереднього, тільки у маршовому режимі послідовно заносяться дані типу 1100 та 0011 та послідовно перевіряються у маршовому режимі із використанням дій if(marsh[i, j] ==1 && marsh[i, j + 1]== 1) або if (marsh[i, j] ==0 && marsh[i, j + 1] ==0). Для перевірки конкретного стану використовуємо мітки wtf = false та wtf = true. Із урахуванням цих міток та стану комірок формується повідомлення про справний або несправний стан комірок. Фрагмент коду , що формує описану послідовність дій, приведено у лістингу 3.9, повний код наведено у Додатку В до магістерської роботи.

Наприклад, при опрацюванні масиву даними типу 0011і якщо мітка wtf = true та при перевірці дві послідовних комірки дорівнюють нулям, то їх переводимо на одиниці і мітку тоді установлюємо як wtf =false. Якщо ж у ході

такої перевірки значення двох послідовних комірок дорівнюють одиницям, то у такому випадку формуємо повідомлення про помилку.

Лістинг 3.9 — Фрагмент коду пошуку подвійної несправності

```
for (int i = 0; i < 4096; i++)
        {for (int j = 0; j < 4096; j++)
        wtf = true,
         marsh[i, j] = 1; j++;
        marsh[i, j] = 1; і мітка встановиться в стан wtf = false
         for (int i = 0; i < 1024; i++)
          {
         for (int j = 0; j < 1024; j++)
```

У іншому випадку, якщо мітка установлена як wtf = false, і якщо значення двох послідовних комірок дорівнюють одиницям, то після перевірки переводимо їх значення на нулі. Якщо ж ці дві послідовних комірки дорівнюють нулям, то формуємо повідомлення про помилку. У підсумку виводимо повідомлення про загальну кількість помилок. Розроблена програма тестування оперативної пам’яті комп’ютера наведена у Додатку В до магістерської роботи.

У даному розділі магістерської роботи для запропонованого підходу була створена програма тестування компонентів оперативної пам’яті, що дозволяє визначити основні характеристики оперативної пам’яті комп’ютера, виконувати пошук можливих несправностей комірок оперативної пам’яті комп’ютера, що у підсумку покращує надійність роботи ОЗП комп’ютера в умовах експлуатації.

4 ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ КОНТРОЛЮ ОПЕРАТИВНОЇ ПАМ'ЯТІ

У даному розділі роботи виконано перевірку працездатності програмних засобів контролю оперативної пам'яті та здійснено перевірку якості роботи програми.

4.1 Перевірка працездатності програми

Перевірка працездатності програми починається із визначення обсягу вільної пам'яті на даний момент часу та перевірки основних характеристик пам'яті комп'ютера. Це виконуємо при активації кнопки «Отримати інформацію». Запускається процес про визначення вільного обсягу пам'яті та її характеристик Інформація про виконання цієї часини програми виводиться у вигляді стрічки у нижній частині вікна програми, що поступово заповнюється зеленим кольором. По завершенню дії опитування пам'яті, що візуально відтворюється під написом «Процес виконання поточної задачі», виводяться дані про загальний обсяг пам'яті комп'ютера та про обсяг доступної у даний момент часу доступної пам'яті, про форм-фактор чіпу, тип пам'яті комп'ютера та тип слоту материнської плати (рисунок 4.1). Це здійснюється шляхом створення зв'язку користувача із персональним комп'ютером за допомогою використання форми спілкування Form1. При виведені інформації із використанням форми Form1 (рис.4.1) виводяться дані про основні характеристики пам'яті комп'ютера. Після завершення цього етапу можна приступати до виконання тестування вільного обсягу пам'яті комп'ютера.

У цій формі також ще є й інші кнопки, при активації яких запускаються окремі алгоритми тестування пам'яті комп'ютера у вигляді сформованих тестів перевірки. Також у цій формі спілкування із користувачем виводиться інформація про тривалість виконання тесту в умовних одиницях та кількість знайдених несправностей при виконанні цього тесту. При активізації кнопки «Опис тесту» можна отримати коротку інформацію про алгоритмічне наповнення самого тесту. Активація кнопки «Розробник» виводить інформацію

про автора розробника цієї програми тестування. Повідомлення виводиться із використанням методу MessageBox. Show.

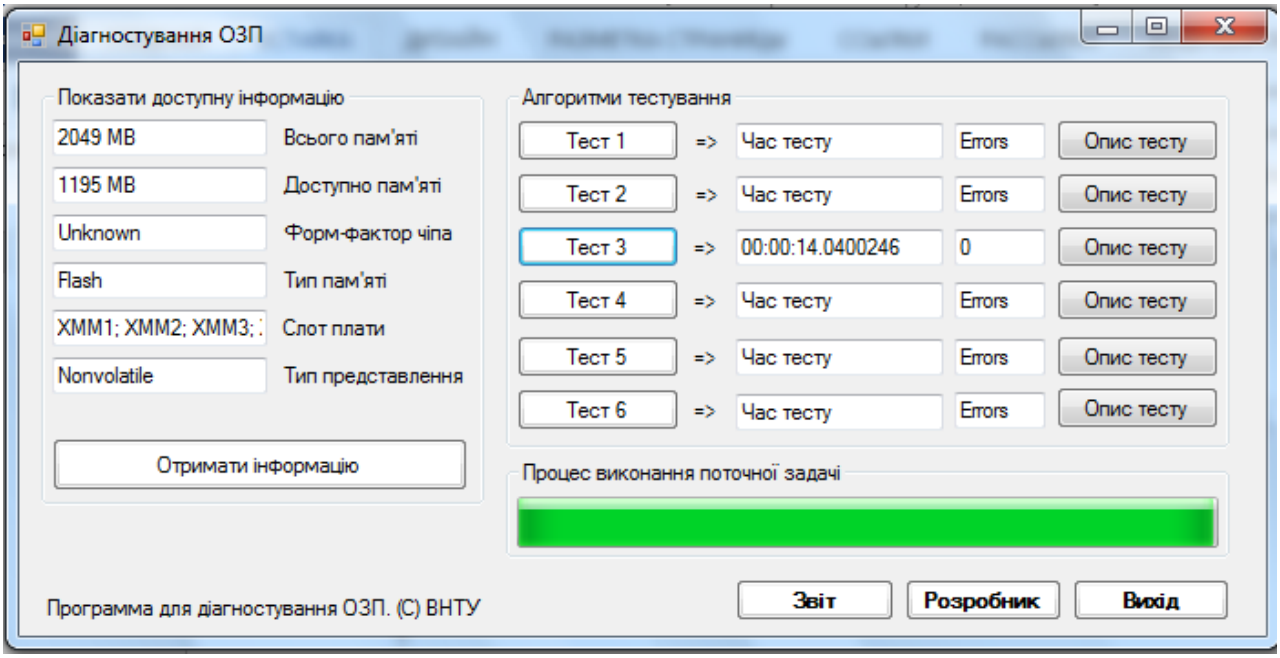

Рисунок 4.1 — Інформація про виконання програми тестування

У цій формі також ще є й інші кнопки, при активації яких запускаються окремі алгоритми тестування пам’яті комп’ютера у вигляді сформованих тестів перевірки. Також у цій формі спілкування із користувачем виводиться інформація про тривалість виконання тесту в умовних одиницях та кількість знайдених несправностей при виконанні цього тесту. При активізації кнопки «Опис тесту» можна отримати коротку інформацію про алгоритмічне наповнення самого тесту. Активація кнопки «Розробник» виводить інформацію про автора розробника цієї програми тестування. Повідомлення виводиться із використанням методу MessageBox. Show.

За допомогою форми спілкування Form1 виконується процес тестування оперативної пам’яті. У цьому випадку можна запустити виконання будь-якого із наявних тестів. Наприклад, при виконанні тесту «Тест2» відбувається перевірка вільного обсягу пам’яті комп’ютера на пошук несправності впливу між сусідніми комірками. Послідовно у вільному обсязі пам’яті комп’ютера формуються тестові посилки на пошук таких несправностей. Якщо є такі

несправності, то фіксується їх наявність. Також виводяться дані про затрачений час на тестування. По завершенню процесу виконання конкретного тесту виводиться повідомлення про факт наявності або відсутності несправності.

Після виконання хоча б одного тесту можна отримати звіт про процес тестування. Це здійснюється при активізації кнопки «Звіт». У цьому випадку виводиться вікно програми із результатами тестування.

Із використанням іншої форми Form2 формується повідомлення «Звіт роботи програми» (рис.4.2) і на екран монітору виводяться усі ті параметри, які були визначені при виконанні Form1. Формування параметрів про цей етап роботи програми відбувається при використанні таких змінних, як stringall Memory, stringform Factor, stringslot Memory, stringtype Predstavl та інші і даних про час виконання тесту та виявлені несправності, таких як public stringtimeTest 1 та public stringerrorTest 1 та усі інші про наявні тести.

При активізації кнопки «Друк звіту» можна роздрукувати паперовий звіт, що формується після запуску на виконання конкретних тестів (рисунок 4.2). Дана кнопка виконує дію, яка реалізована за допомогою програмного методу private void button2_Click, та перевіряє вірність функціонування програми. Перевіряється процес друкування звіту, що виконується дією printDialog1.PrinterSettings.

Також є варіант збереження даних про результати тестування із використанням кнопки «Зберегти до файлу», яка реалізована із застосуванням методу private void button3_Click.

Активація кнопки «Повернутися до тесту» дозволяє закрити вікно звіту про роботу програми і повторно відкрити вікно програми для проведення тестування із використанням інших тестів. Вона реалізована за допомогою методу private void button1_Click, який дозволяє перевірити вільну частини пам'яті комп'ютера за допомогою будь-якого із тестів. При цьому перевіряється вірність функціонування програми для реалізації процедур Form1() та Form2().

Завершення роботи програми тестування виконуємо за допомогою використання кнопки «Вихід», яка реалізована із використанням методу private

voidbutton 7_Click. При її активація пам'ять комп'ютера звільняється для виконання інших програм.

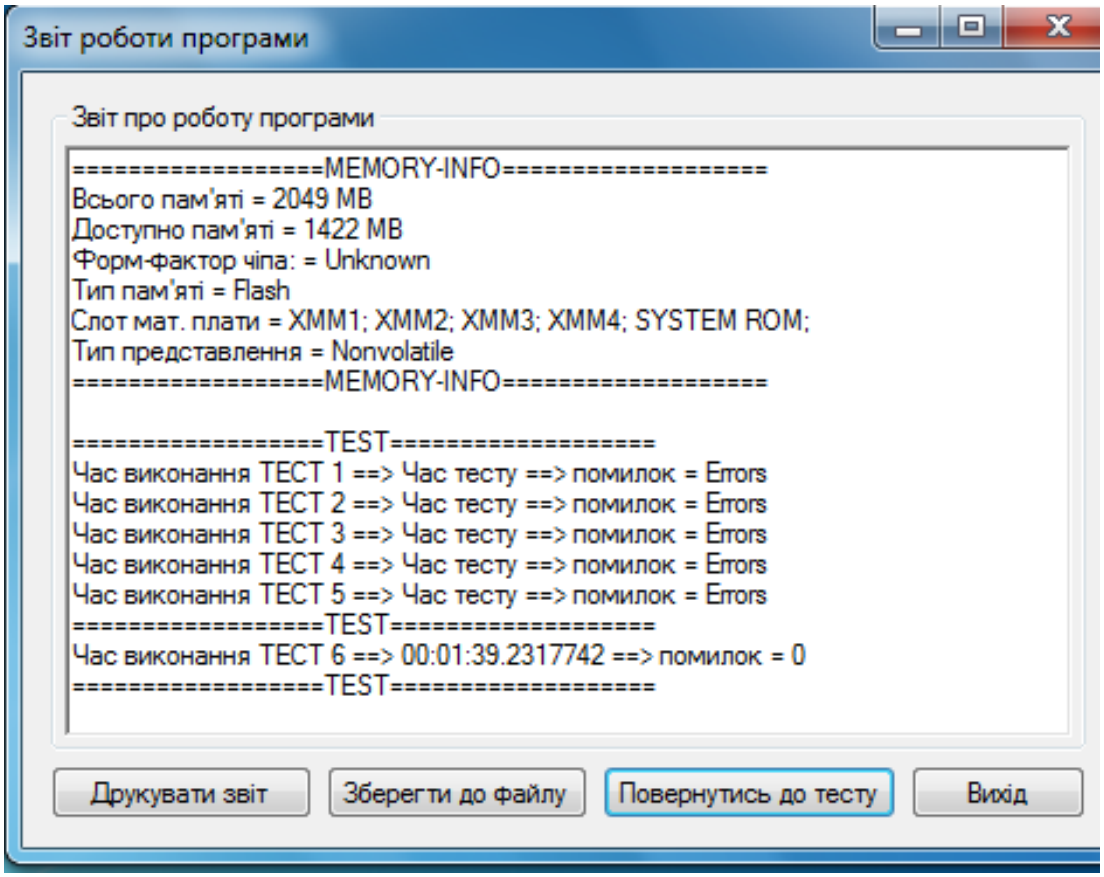

Рисунок 4.2 — Звіт про виконання програми тестування

Сформована програма тестування із значною достовірністю дозволяє здійснювати перевірку оперативної пам’яті комп’ютера із інформаційною ємністю ОЗП до декількох одиниць Гбайт на частоті роботи центрального процесора персонального комп’ютера. Запропонований підхід контролю оперативної пам’яті комп’ютера і його реалізація відрізняються від відомих методів функціонального контролю відсутністю еталонного зразку.

4.2 Тестування функціональності програми

Для тестування і перевірки функціональності створеної програми був обраний персональний комп'ютер із середнім значенням продуктивності, що включає процесор з мінімальним периферійним обладнанням, вбудовані мультимедійні засоби і засоби зв'язку з ним. Комп'ютер має такі технічні

характеристики: центральний процесор IntelCore i5-12600F із архітектурою AlderLak, що має 4 ядер та 8 потоків, робоча частота 2,8 ГГц, максимальна частота 3,4 ГГц, оперативна пам'ять 2 GB типу DIMM DDR4, твердотільний накопичувач даних SSD на 0.5 ТБ, відеокарта типу NVIDIA RTX 4070 із об'ємом відео-пам'яті 6 ГБ. Для відображення зображення використовується монітор із розміром екрану 25" на матриці IPS, що має роздільну здатність 1920×1080 типу Full HD та із частотою оновлення кадрів 155 Гц.

Перерахованих технічних параметрів комп'ютера є достатнім для перевірки якості розробленої програми і дозволило отримати результати перевірки функціонування створеного програмного продукту.

При тестуванні роботи програми були перевірені такі аспекти: виконано тестування інтерфейсу користувача, проведено інтеграційне тестування програми у складі комп'ютерної системи по перевірці працездатності окремих модулів створеної програми.

Як відомо, вимоги до інтерфейсу користувача розділяються на дві групи. Перша група вимог відноситься до зовнішнього вигляду інтерфейсу та форми взаємодії із користувачем. Друга група вимог перевіряє можливості доступу до внутрішньої функціональності створеного програмного засобу за допомогою інтерфейсу користувача. Тобто перевіряється взаємодія підсистеми інтерфейсу із користувачем, а друга із внутрішньою логікою роботи програмного засобу. У рамках першої групи вимог були перевірені такі особливості створеної програми.

Були перевірені вимоги до розміщення елементів управління на створених екранних формах. Тут перевірялися загальні принципи розміщення екранних елементів інтерфейсу користувача. Було провірено загальну вимогу щодо розміщення елементів екранної форми, які мають загальний вигляд такого порядку «Кожне вікно виведення інтерфейсу програми повинно бути розділене на три частини: рядок меню, робоча область екрану та статусний рядок».

Наступним перевірялись вимоги до змісту та оформлення виведених повідомлень про результати роботи програми. Було звернуто увагу на шрифтове та кольорове оформлення та перевірені умови, в яких випадках виводиться

конкретні повідомлення. Дані показники задовільняють процес функціонування програмного засобу.

Перевірялися вимоги до форматів введення даних, тобто в якому вигляді інформація від користувача надходить у систему та реакція програми на некоректне введення. Також перевірялися вимоги на реакцію програми на поведінку користувача. Всі дії користувача при активації кнопок приводили до адекватних результатів.

Перевірялися вимоги на час відгуку на команди користувача. Тут найбільш критичним є час перевірки програмою характеристик оперативної пам’яті комп’ютера. Це окремий тип вимог, який перевіряє вимоги до часу відгуку програми на різні операції користувача. Тривалість часу на виконання функції опитування даних про характеристики не перевищувала 2 секунди та підтверджувалась поступовим заповненням стрічки зеленим кольором. Це дозволяє користувачу спостерігати за процесом підготовки даних.

Функціональне тестування інтерфейсу користувача проводилося вручну при безпосередній участі користувача без допомоги інструментарію, що може бути використаний для автоматизованого тестування. Це пов'язано із тим, що підготовка автоматизованих тестів для такого типу програм вимагатиме значних затрат часу та ресурсів.

Був сформований сценарій із перерахування послідовності дій, які повинен виконувати користувач, і опис важливих для перевірки результатів тестування програми та відповідних реакцій системи, що поступають та відображаються у вікні інтерфейсу. Використовувалася типова форма опису сценарію для проведення ручного тестування у вигляді таблиці, в якій у одній колонці були описані дії як окремі кроки сценарію, а в іншій колонці очікувана реакція програмного продукту у складі персонального комп'ютера, а у третій колонці фіксувались результати запису того, чи збігалися очікувані реакції системи із реальною або перераховувалися розбіжності.

Такий сценарій тестування дозволив здійснити інтеграційне тестування, яке призначене для перевірки зв'язків між компонентами програми, а також взаємодії із операційною системою та складовими вузлами комп'ютера.

У даному розділі розглянуто процес перевірки працездатності програми та описано процес тестування функціональності створеного програмного продукту.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку програмного засобу тестування оперативної пам’яті комп’ютерів. Особливістю розробки є те, що удосконалено метод тестування оперативної пам’яті комп’ютерів, який відрізняється від існуючих поєднанням методів тестування з використанням маршових тестів та тестів для пошуку несправностей впливу сусідніх комірок, що дозволяє більш ефективно виконувати процес діагностування пам’яті комп’ютерів.

Аналогом може бути програма «Random number sequence» – засіб пошуку несправності пам’яті комп’ютерів, вартістю 16 200 грн.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 5.1

3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х

Продовження табл. 5.1

12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	3	3
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	4	3	3
Експлуатаційні витрати	4	3	3
Ринок збуту	4	4	4
Конкурентоспроможність	3	3	3
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	3	3	4
Сума	43	41	42
Середньоарифметична сума балів	(43+41+42) / 3 = 42		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок вдосконалення методів зовнішнього програмного тестування оперативної пам’яті для пошуку та локалізації дефектів компонентів оперативної пам’яті комп’ютерів.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t, \quad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ — число робочих днів за місяць, 23 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Таблиця 5.4 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	36500	1586,96	34	53956,522
Програміст	33000	1434,78	34	48782,609
Всього				102739,13

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Додаткова заробітна плата розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 13,8 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 13,8\ \% / 100\ \% \quad (5.2)$$

$$З_д = (102739,13 \cdot 13,8\ \% / 100\ \%) = 14178,00\ \text{(грн.)}$$

5.2.3 Нарахування на заробітну плату розробників.

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \quad (5.3)$$

$$Н_з = (102739,13 + 14178,00) \cdot 22\ \% / 100\ \% = 25721,77\ \text{(грн.)}$$

5.2.4. Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12}\ \text{[грн.]}. \quad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років;

$t_{вик}$ — термін використання під час розробки, місяців.

Розрахуємо, для прикладу, амортизаційні витрати на комп’ютер балансова

вартість якого становить 28000 грн., термін його корисного використання згідно податкового законодавства – 2 роки, а термін його фактичного використання – 1,48 міс.

$$A_{обл} = \frac{280000}{2} \times \frac{1{,}48}{12} = 1724{,}34\ грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.5. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $B_{нем.ак.}$ = 6300 грн.

Таблиця 4.5 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	28000	2	1,48	1724,638
Офісне обладнання (меблі)	25000	4	1,48	769,928
Приміщення	1500000	20	1,48	9239,130
Всього				11733,70

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \qquad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,3 кВт;

Ф — фактична кількість годин роботи обладнання, годин.

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,3 \cdot 8 \cdot 34 \cdot 10,42 = 765,2448 \text{ (грн.)}$$

5.2.7 Інші витрати та загальновиробничі витрати.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$I_в = 102739,13 * 58\% / 100\% = 59588,7 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$H_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $H_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$H_{нзв} = 102739{,}13 * 135\ \% / 100\ \% = 138698 \text{ (грн.)}$$

5.2.9 Витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = 102739{,}13+14178{,}00+25721{,}77+11733{,}70+6300+765{,}24+59588{,}7+$$
$$+138698 = 359724{,}36 \text{ грн.}$$

5.2.11 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7;

впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 359724,36 / 0,5 = 719449 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до

отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

4.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.9)$$

де $\pm\Delta Ц_o$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

p — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 7500 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на *500* грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року – на 3000 шт., протягом другого року – на 2500 шт., протягом третього року на 2000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$\Delta П_1$ = (0*500 + (7500 + 500)*3000)* 0,8333* 0,27) * (1 - 0,18) = 4151249,834 грн.

$\Delta П_2$ = (0*500 + (7500 + 500)*(3000+2500)* 0,8333* 0,27) * (1 - 0,18) = 8117999,675 грн.

$\Delta П_3$ = (0*500 + (7500 + 500)*(3000+2500+2000)* 0,8333* 0,27) * (1 - 0,18) = 1106999,557 грн.

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 23339249,07 грн.

4.3.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності.

Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.10)$$

де $\Delta П_i$ - — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (4151249,834/(1+0,1)^1)+(8117999,675/(1+0,1)^2)+(11069999,557/$$
$$/(1+0,1)^3) = 3773863,49 + 6709090,641 + 8317054,513 = 18800008,64 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 719449 = 1438897,45 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{5.12}$$

$$E_{абс} = 18800008{,}64 - 1438897{,}45 = 17361111{,}19 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.13}$$

$T_ж$ — життєвий цикл наукової розробки, роки.

$$E_в = \sqrt{\text{з}\ (1 + 17361111{,}19/1438897{,}45} - 1 = 1{,}355$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \tag{5.15}$$

$$T_{ок} = 1 / 1{,}355 = 0{,}74 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,74 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 719449 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,74 роки.

ВИСНОВКИ

Надійне та ефективне вирішення питання стабільного функціонування усіх вузлів комп'ютера практично неможливе без застосування потужних засобів діагностування, які виконують перевірку працездатності та пошук можливих несправностей у вузлах комп'ютера, у тому числі і в оперативній пам'яті комп'ютера. Дана магістерська робота присвячена створенню одного із способів тестування оперативної пам'яті комп'ютера. Тема є актуальною та вимагає подальшого вдосконалення засобів та методів для її вирішення.

У першому розділі магістерської роботи виконано огляд існуючих типів пам'яті комп'ютерів, розглянуто особливості процесу функціонування оперативної пам'яті комп'ютера, здійснено аналіз можливості появи дефектів та пов'язаних із ними відповідних типів несправності, що можуть проявити себе при функціонуванні компонентів пам'яті комп'ютера, а також відомих на даний час методів і засобів пошуку несправності та показано, що метод тестування оперативної пам'яті комп'ютера із використанням зовнішніх програмних засобів на основі спеціалізованих тестів, які формують необхідні умови для виявлення несправності взаємного впливу комірок у запам'ятовуючому пристрою, є досить ефективним.

У другому розділі магістерської роботи для вирішення поставлених задач по тестуванню компонентів ОЗП комп'ютера запропоновано використати метод перевірки стану оперативної пам'яті із використанням маршових тестів та пошуку несправностей можливого впливу сусідніх по адресі комірок оперативної пам'яті для виявлення таких суміжних комірок пам'яті. У роботі сформовані моделі для аналізу несправності впливу суміжних комірок оперативної пам'яті на комірку, що перевіряється. Розроблено послідовність тестування запам'ятовуючих пристроїв комп'ютера.

У третьому розділі магістерської роботи для запропонованого підходу створена програма тестування компонентів оперативної пам'яті, що дозволяє визначити основні характеристики оперативної пам'яті комп'ютера, виконувати

пошук можливих несправностей комірок оперативної пам’яті комп’ютера, що у підсумку покращує надійність роботи ОЗП комп’ютера в умовах експлуатації.

У четвертому розділі магістерської роботи розглянуто процес перевірки працездатності програми та описано процес виконання тестування функціональності створеного програмного продукту

У п’ятому розділі магістерської роботи обґрунтована доцільність виконання нового наукового рішення по підготовці тестів для контролю компонентів оперативної пам’яті комп’ютерів на працездатність, проведені розрахунки фінансових витрат по створенню запропонованої розробки та визначено економічні переваги від впровадження нового програмного продукту.

Сформована програма тестування оперативної пам’яті комп’ютера створює умови для ефективної та надійної її перевірки в умовах експлуатації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Матвієнко М. П. Архітектура комп’ютера. Навчальний посібник. / М. П. Матвієнко, В. П. Розен, О. М. Закладний — К: Видавництво Ліра-К, 2016. — 264 с.

2. Тхір І.Л., Калушка В.П., Юзьків А.В. Посібник користувача ПК. Третє видання. — Тернопіль: «Підручники та посібники», 2006. -1024с.

3. Азаров О. Д. Діагностування цифрових пристроїв: навчальний посібник. / О. Д. Азаров, С. І. Перевозніков, Н. О. Біліченко, В. С. Озеранський.: НП з грифом МОН. - Вінниця, Універсум-Вінниця. – 2009.- 136 с.

4. Sarah L. Harris, David Harris. Digital Design and Computer Architecture / Elsevier Science & Technology, 2022. – 720 p.

5. Hamdioui S. Memory test experiment: industrial results and data / S. Hamdioui, A.J. Van de Goor, D.J. Reyes and others // IEE Proc. Computer. Digit. Tech., Jan. 2006. – Vol. 153. – № 1. – P. 1–8.

6. Mueller S.M. Upgrading and Repairing PCs. — 22 th Edition. 2021. — 1102 p.

7. Алгоритми тестового діагностування напівпровідникових запам'ятовуючих пристроїв. – К.: Корнійчук, 2008. – 220 с.

8. Оліх О. Я. Дефекти у напівпровідникових та діелектричних кристалах. – Вінниця : ФОП Корзун Д.Ю., 2016. – 152 с.

9. Стрілковський В. С. Програмний засіб тестування оперативної пам’яті.// В. С. Стрілковський, І. С. Колесник, М. А. Очкуров //. Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів ВНТУ. [Електронний ресурс]. Режим доступу: https://conferences.vntu. edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/22893/.

10.Yarmolik S. V., Yarmolik V. N. Multiple non-destructive marching tests with variable address sequences. Automation and Remote, 2007, no. 2, pp. 21–30.

11.O'Regan G. Concise Guide to Software Testing. / G. O'Regan. — New York: Springer, 2019. — 309 p.

12.Yarmolik V. N., Mrozek I., Yarmolik S. V. Pseudo-exhaustive memory testing based on March A tests. Informatics, 2020, no. 2(17), pp. 54–70.

13.Sharma, A. K. Advanced Semiconductor Memories: Architectures, Designs, and Applications / A. K. Sharma. – London : John Wiley & Sons, 2003. – 652 p.

14.Nicolaidis M. Theory of Transparent BIST for RAMs. // IEEE Trans. on Computers. 1996. Vol. 45, No. 10. P. 1141–1155.

15. Mrozek, I. Multi-run Memory Tests for Pattern Sensitive Faults / I. Mrozek. – Cham : Springer International Publishing AG, 2019. – 135 p.

16.Бабич М.П., Жуков І.А. Комп'ютерна схемотехніка: навчальний посібник. –К.: МК-Прес, 2004.- 412с.

17.Prince B. High-performance memories: new architecture DRAM's and SRAM's, evolution and function. Gr.Britain, John Wiley & Sons, 2006.

18. Neuberger G., de Lima F., Carro L., Reis R. A multiple bit upset tolerant SRAM memory // ACM Trans. Des. Autom. Electron. Syst. 2003. V. 8. № 4. P. 577–590.

19.Umanesan G., Fujiwara E. A class of random multiple bits in a byte error correcting (S t/b EC) codes for semiconductor memory system // IEEE Transaction on Computers. 2003. V. 52. № 7. P. 835–847.

20.Memtest86. [Електронний ресурс] – Режим доступу: https://www.memtest86.com/

21.Docmemory. [Електронний ресурс] – Режим доступу: www.docmemory.com

22.Memtest86+. [Електронний ресурс] – Режим доступу: https://soft-windows.info/12610-memtest-70-pro-x86-x64-2021-eng.html.

23.Goldmemory. [Електронний ресурс] – Режим доступу: https://soft-landia.ru/goldmemory_7.html

24.Memtest-pro-deluxe [Електронний ресурс] – Режим доступу: https://www.fullversiondl.com/memtest-pro-deluxe-v6-2/.

25.http://biblprog.org.ua/everest_home_edition/.

26.Програми для перевірки оперативної пам’яті. Електронний ресурс https://daad.org.ua/4888-programi-dlya-perevirki-operativnoyi-pamyati.html#google_vignette

27.Nicolaidis M. Soft Errors in Modern Electronic Systems / M. Nicolaidis. – Springer, 2010. – 336 p.

28. Діагностичні моделі трансляторів n-МОН та КМОН структур виготовлення / М.К. Жердєв, В.В. Вишнівський, Г.Б. Жиров, С.І. Глухов //Збірник наукових праць ВІТІ НТУУ ―КПІ – К., 2006. – №3 – С.9-12.

29.Кравчук Р.В. Функціональний підхід в діагностуванні цифрових процесорів і елементів пам’яті / Р.В. Кравчук, О.І. Стецюк, В.М. Чешун // Міжнародний науково-технічний журнал «Вимірювальна та обчислювальна техніка в технологічних процесах». – Хмельницький: ХНУ, 2018. – Вип. No2 (62) –С.106-109.

30.Коноваленко І. В. Програмування мовою C# 6.0 : навчальний посібник / І. В. Коноваленко. — Тернопіль : ТНТУ, 2017. — 300 с.

31.Troelsen Andrew. Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming: Paperback 2021. 1640 p.

32.Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. — Вінниця : ВНТУ, 2021. — 42 с.

33.Тарифи на електроенергію [Електронний ресурс]. Режим доступу: http://index.minfin.com.ua/tarif/electric.php

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Інститут інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н. Азаров О.Д.

"29" вересня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Програмний засіб тестування оперативної пам’яті комп’ютерів

08-54.МКР.018.00.000.ПЗ

Науковий керівник: к.т.н., доцент

_____ Колесник І. С.

Студент групи 1КІ-23м

_____ Стрілковський В. С.

Вінниця 2024

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність дослідження пов’язана з необхідністю вирішення основних проблем існуючих засобів контролю оперативного запам’ятовуючого пристрою комп’ютера.

1.2 Наказ про затвердження теми МКР.

2 Мета МКР і призначення розробки

2.1 Мета роботи — аналіз сучасного стану і тенденцій розвитку зарубіжних і вітчизняних розробок у сфері контролю стану оперативного запам’ятовуючого пристрою комп’ютера і вдосконалення технології тестування оперативного запам’ятовуючого пристрою комп’ютера.

2.2 Призначення розробки — створення програмного продукту для виконання процесу тестування оперативного запам’ятовуючого пристрою комп’ютера.

3 Вихідні дані для виконання МКР

3.1 Технічний опис сучасних методів тестування оперативного запам’ятовуючого пристрою комп’ютера.

3.2 Середовище розробки ПЗ Visual Studio Code.

4 Вимоги до виконання МКР

4.1 Провести аналіз різних методів тестування оперативного запам’ятовуючого пристрою комп’ютера.

4.2 Вивчити принцип роботи та особливості функціонування систем тестування оперативного запам’ятовуючого пристрою комп’ютера.

4.3 Запропонувати підхід до тестування оперативного запам’ятовуючого пристрою комп’ютера.

4.4 Вдосконалити технологію тестування оперативного запам’ятовуючого пристрою комп’ютера.

Для нормальної роботи програми необхідно комп’ютер з такою мінімальною конфігурацією:

- наявність кольорового монітору SVGA, з діагоналлю 27”, роздільна здатність не менше 1920×1080 точок, при частоті 85 hz оновлення екрану та глибині кольору 32 bit;
- процесор рівня IntelCorei5 з частотою 2,66 Ghz;
- розмір оперативної пам’яті (RAM) не менше 2 Gb;
- розмір жорсткого диску (HDD) 20 Gb;
- операційна система Microsoft Windows 10x.

5 Етапи МКР та очікувані результати

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

Таблиця А.1 — Етапи МКР

№ з/п	Назва етапів МКР	Строк виконання	Підпис
1	Постановка задачі	19.09.24	
2	Аналіз предметної області	20.09.24	
3	Розробка структурної схеми	25.09.24	
5	Розрахунок аналогової частини	1.10.24	
6	Вибір ПЗ для розробки	19.10.24	
7	Розробка роботи нейромережі	20.10.24	
8	Розрахунок економічної частини	10.10.24	
9	Виконання МКР	19.10.24	
10	Оформлення пояснювальної записки та ілюстративного матеріалу	26.11.24	
11	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	2.12.24	
12	Підписи супроводжувальних документів у керівника, опонента, нормоконтролера	09.12.24	

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відзив

наукового керівника, відзив опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, нормоконтроль про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

8.1 При оформлюванні МКР використовуються:

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

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

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

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

Несправності комірок пам'яті

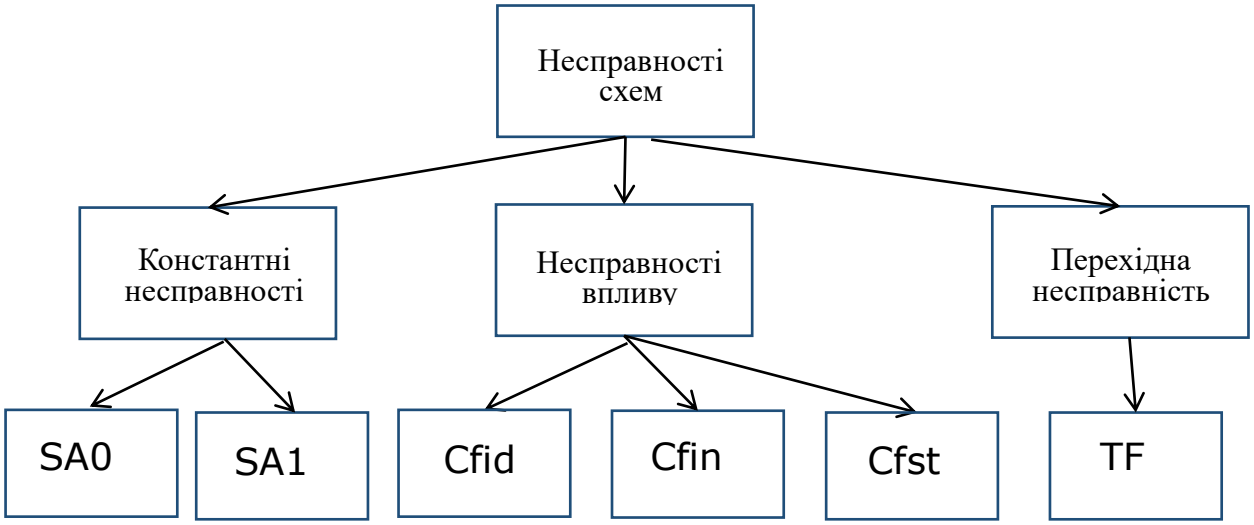

Рисунок Б.1 — Несправності комірок пам'яті

ДОДАТОК В

Структура програми тестування ОЗП

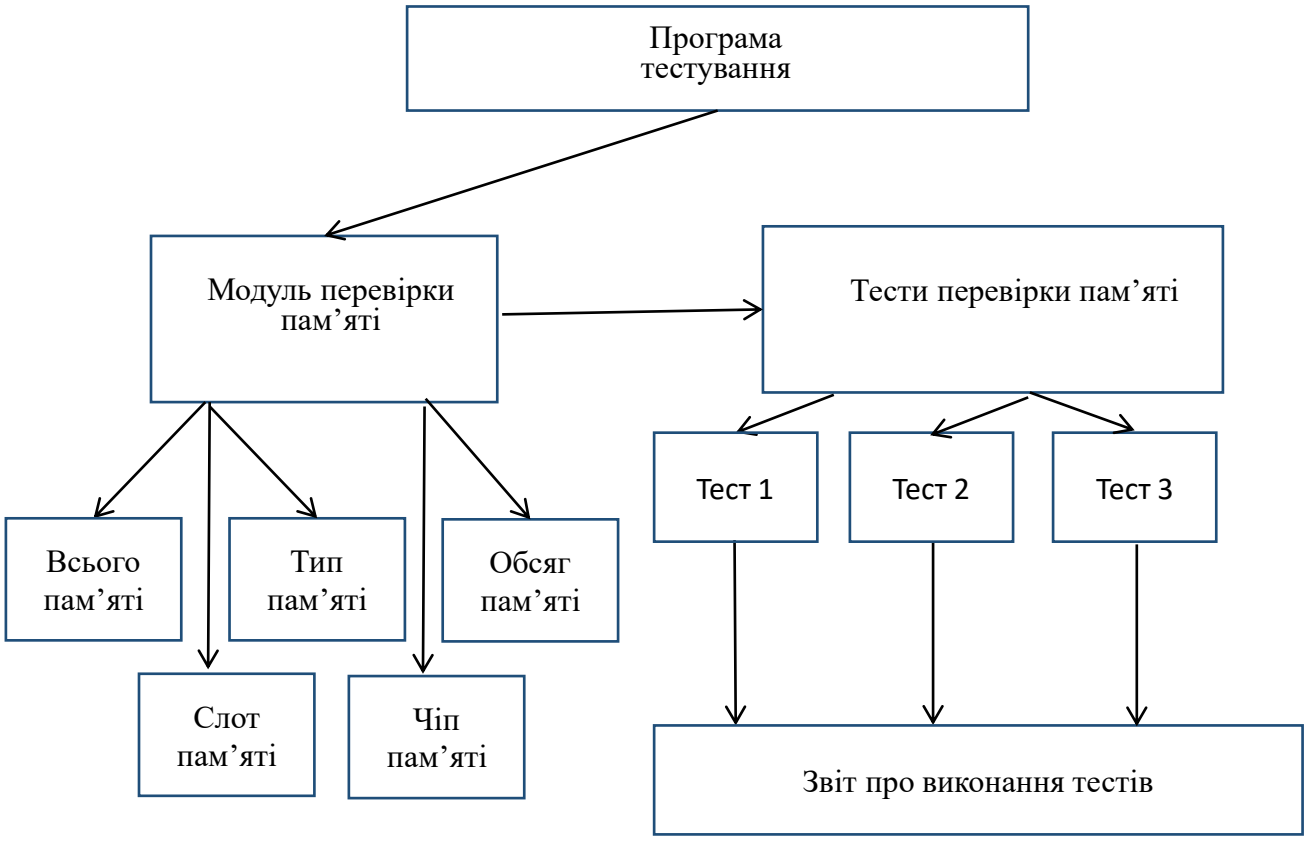

Рисунок В.1 — Структура програми тестування ОЗП

ДОДАТОК Г

Модель несправності впливу

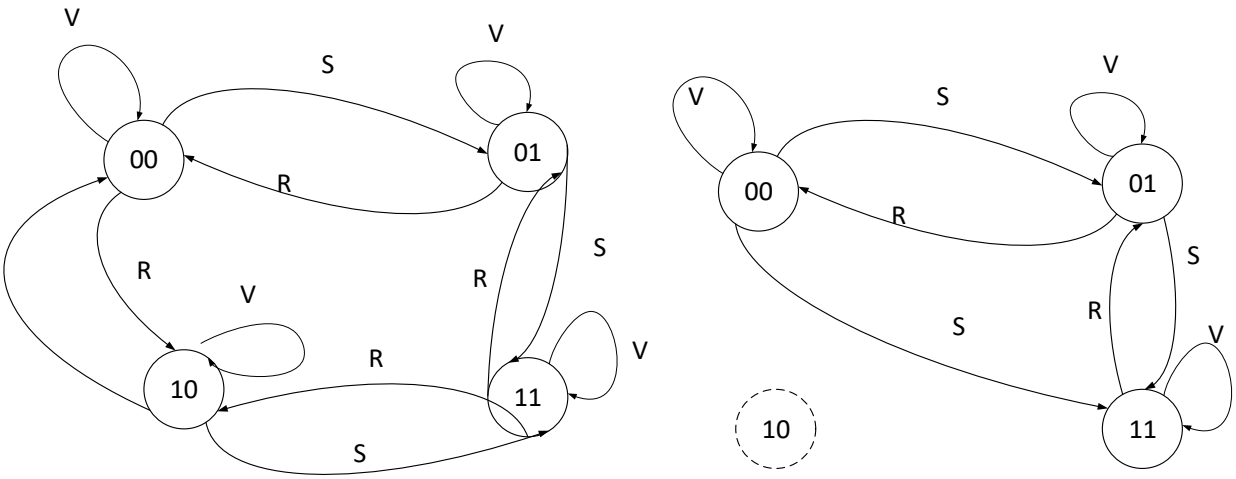

Рисунок Г.1 — Модель несправності впливу

ДОДАТОК Д

Лістинг програми визначення характеристик пам'яті

```
using System;
using System.Threading;
using System.Management;
using System.Windows.Forms;
using System.ComponentModel;
using System.Text;

namespace WindowsFormsApplication1
{    public partial class Form1 : Form
   {
      int info = 0;
      int test1 = 0;
      int test2 = 0;
      int test3 = 0;
      int test4 = 0;
      public Form1()
      {
         InitializeComponent();
      }
      private void button1_Click(object sender, EventArgs e)
      {
         info++;
         Thread tRealMemory = new Thread(realMemory);
         tRealMemory.Start();
         Thread tAviableMemory = new Thread(aviableMemory);
         tAviableMemory.Start();
         Thread tDdrFormFactor = new Thread(ddrFormFactor);
         tDdrFormFactor.Start();
         Thread tMemoryType = new Thread(memoryType);
         tMemoryType.Start();
         Thread tSocketDDR = new Thread(socketDDR);
         tSocketDDR.Start();
         Thread tTypeDetail = new Thread(TypeDetail);
         tTypeDetail.Start();

         for (int i = progressBar1.Minimum; i <= progressBar1.Maximum; i++)
         {
            progressBar1.Value = i;
            System.Threading.Thread.Sleep(10);
         }
         System.Threading.Thread.Sleep(100);
```

```
        }
        private void TypeDetail()
        {
          if (InvokeRequired)
           {
           BeginInvoke(new MethodInvoker(() =>
           {
           Int32 TypeDetail;
           ManagementScope typedetMs = new ManagementScope();
           ObjectQuery typedetQuery = new ObjectQuery("SELECT TypeDetail FROM
Win32_PhysicalMemory");
           ManagementObjectSearcher typedetSearcher = new
ManagementObjectSearcher(typedetMs, typedetQuery);
           ManagementObjectCollection typedetReturnCollection =
typedetSearcher.Get();
           foreach (ManagementObject typedetReturn in typedetReturnCollection)
           {
              TypeDetail = Convert.ToInt32(typedetReturn["TypeDetail"]);
              switch (TypeDetail)
              {
                 case 1:
                    textBox6.Text = ("Reserved");
                    break;
                 case 2:
                    textBox6.Text = ("Other");
                    break;
                 case 4:
                    textBox6.Text = ("Unknown");
                    break;
                 case 8:
                    textBox6.Text = ("Fast-paged");
                    break;
                 case 16:
                    textBox6.Text = ("Static column");
                    break;
                 case 32:
                    textBox6.Text = ("Pseudo-static");
                    break;
                 case 64:
                    textBox6.Text = ("RAMBUS");
                    break;
                 case 128:
                    textBox6.Text = ("Synchronous");
                    break;
                 case 256:
```

```
                textBox6.Text = ("CMOS");
                break;
            case 512:
                textBox6.Text = ("EDO");
                break;
            case 1024:
                textBox6.Text = ("Window DRAM");
                break;
            case 2048:
                textBox6.Text = ("Cache DRAM");
                break;
            case 4096:
                textBox6.Text = ("Nonvolatile");
                break;
            default:
                textBox6.Text = ("ERROR");
                break;
        }
      }
      }));
      }
    }
    private void socketDDR()
    {
      if (InvokeRequired)
      {
      BeginInvoke(new MethodInvoker(() =>
      {
        textBox5.Text = "";
      ManagementScope socketMs = new ManagementScope();
      ObjectQuery socketQuery = new ObjectQuery("SELECT DeviceLocator
FROM Win32_PhysicalMemory");
      ManagementObjectSearcher socketSearcher = new
ManagementObjectSearcher(socketMs, socketQuery);
      ManagementObjectCollection socketReturnCollection =
socketSearcher.Get();
      foreach (ManagementObject socketReturn in socketReturnCollection)
        {
        textBox5.Text += string.Format(socketReturn["DeviceLocator"] + "; ");
        }
      }));
      }
    }
    private void memoryType()
    {
```

```
        if (InvokeRequired)
        {
        BeginInvoke(new MethodInvoker(() =>
        {
        Int32 memoryType;
        ManagementScope type = new ManagementScope();
        ObjectQuery typeQuery = new ObjectQuery("SELECT MemoryType FROM
Win32_PhysicalMemory");
        ManagementObjectSearcher typeSearcher = new
ManagementObjectSearcher(type, typeQuery);
        ManagementObjectCollection typeReturnCollection = typeSearcher.Get();
        foreach (ManagementObject typeReturn in typeReturnCollection)
        {
           memoryType = Convert.ToInt32(typeReturn["MemoryType"]);
           switch (memoryType)
           {
              case 0:
                 textBox4.Text = ("Unknown");
                 break;
              case 1:
                 textBox4.Text = ("Other");
                 break;
              case 2:
                 textBox4.Text = ("DRAM");
                 break;
              case 3:
                 textBox4.Text = ("Synchronous DRAM");
                 break;
              case 4:
                 textBox4.Text = ("Cache DRAM");
                 break;
              case 5:
                 textBox4.Text = ("EDO");
                 break;
              case 6:
                 textBox4.Text = ("EDRAM");
                 break;
              case 7:
                 textBox4.Text = ("VRAM");
                 break;
              case 8:
                 textBox4.Text = ("SRAM");
                 break;
              case 9:
                 textBox4.Text = ("RAM");
```

```
            break;
        case 10:
            textBox4.Text = ("ROM");
            break;
        case 11:
            textBox4.Text = ("Flash");
            break;
        case 12:
            textBox4.Text = ("EEPROM");
            break;
        case 13:
            textBox4.Text = ("FEPROM");
            break;
        case 14:
            textBox4.Text = ("EPROM");
            break;
        case 15:
            textBox4.Text = ("CDRAM");
            break;
        case 16:
            textBox4.Text = ("3DRAM");
            break;
        case 17:
            textBox4.Text = ("SDRAM");
            break;
        case 18:
            textBox4.Text = ("SGRAM");
            break;
        case 19:
            textBox4.Text = ("RDRAM");
            break;
        case 20:
            textBox4.Text = ("DDR");
            break;
        case 21:
            textBox4.Text = ("DDR-2");
            break;
        default:
            textBox4.Text = ("Unknown Type");
            break;
      }
    }
   }));
 }
}
```

```
        private void ddrFormFactor()
        {
            if (InvokeRequired)
            {
                BeginInvoke(new MethodInvoker(() =>
                {
                    Int32 formFactor;
                    ManagementScope ddr = new ManagementScope();
                    ObjectQuery ddrQuery = new ObjectQuery("SELECT FormFactor
FROM Win32_PhysicalMemory");
                    ManagementObjectSearcher ddrSearcher = new
ManagementObjectSearcher(ddr, ddrQuery);
                    ManagementObjectCollection ddrReturnCollection = ddrSearcher.Get();
                    foreach (ManagementObject ddrReturn in ddrReturnCollection)
                    {
                        formFactor = Convert.ToInt32(ddrReturn["FormFactor"]);
                        switch (formFactor)
                        {
                            case 0:
                                textBox3.Text = ("Unknown");
                                break;
                            case 1:
                                textBox3.Text = ("Other");
                                break;
                            case 2:
                                textBox3.Text = ("SIP");
                                break;
                            case 3:
                                textBox3.Text = ("DIP");
                                break;
                            case 4:
                                textBox3.Text = ("ZIP");
                                break;
                            case 5:
                                textBox3.Text = ("SOJ");
                                break;
                            case 6:
                                textBox3.Text = ("Proprietary");
                                break;
                            case 7:
                                textBox3.Text = ("SIMM");
                                break;
                            case 8:
                                textBox3.Text = ("DIMM");
                                break;
```

```
case 9:
    textBox3.Text = ("TSOP");
    break;
case 10:
    textBox3.Text = ("PGA");
    break;
case 11:
    textBox3.Text = ("RIMM");
    break;
case 12:
    textBox3.Text = ("SODIMM");
    break;
case 13:
    textBox3.Text = ("SRIMM");
    break;
case 14:
    textBox3.Text = ("SMD");
    break;
case 15:
    textBox3.Text = ("SSMP");
    break;
case 16:
    textBox3.Text = ("QFP");
    break;
case 17:
    textBox3.Text = ("TQFP");
    break;
case 18:
    textBox3.Text = ("SOIC");
    break;
case 19:
    textBox3.Text = ("LCC");
    break;
case 20:
    textBox3.Text = ("PLCC");
    break;
case 21:
    textBox3.Text = ("BGA");
    break;
case 22:
    textBox3.Text = ("FPBGA");
    break;
case 23:
    textBox3.Text = ("LGA");
    break;
```

```
                    default:
                        textBox3.Text = ("Unknown Type");
                        break;
                }
            }
        }));
    }
}
private void realMemory()
{
    if (InvokeRequired)
    {
        BeginInvoke(new MethodInvoker(() =>
        {
            int i = 0;
            int capacity=0;
            ManagementScope oMs = new ManagementScope();
            ObjectQuery oQuery = new ObjectQuery("SELECT Capacity FROM
Win32_PhysicalMemory");
            ManagementObjectSearcher oSearcher = new
ManagementObjectSearcher(oMs, oQuery);
            ManagementObjectCollection oReturnCollection = oSearcher.Get();
            foreach (ManagementObject oReturn in oReturnCollection)
            {
                i++;
                capacity += (Convert.ToInt32(oReturn["Capacity"]) / 1048576);
            }
            textBox2.Text = string.Format(capacity + " MB");
        }));
    }
}
private void aviableMemory()
{
    if (InvokeRequired)
    {
        BeginInvoke(new MethodInvoker(() =>
        {
            performanceCounter1.CategoryName = "Memory";
            performanceCounter1.CounterName = "Available Mbytes";
            performanceCounter1.InstanceName = null;
            long aviMB = performanceCounter1.RawValue;
            textBox1.Text = string.Format("{0} MB \n", aviMB);
        }));
    }
}
```

```
        private void button11_Click(object sender, EventArgs e)
        {
            if ((info > 0) && (test4>0) && (test3>0) && (test2>0) && (test1>0))
            {
                Form2 Form2 = new Form2();
                Form2.allMemory = textBox2.Text;
                Form2.aviableMemory = textBox1.Text;
                Form2.formFactor = textBox3.Text;
                Form2.typeMemory = textBox4.Text;
                Form2.slotMemory = textBox5.Text;
                Form2.typePredstavl = textBox6.Text;
                Form2.timeTest1 = textBox7.Text;
                Form2.timeTest2 = textBox8.Text;
                Form2.timeTest3 = textBox9.Text;
                Form2.timeTest4 = textBox10.Text;
                Form2.errorTest1 = textBox11.Text;
                Form2.errorTest2 = textBox12.Text;
                Form2.errorTest3 = textBox13.Text;
                Form2.errorTest4 = textBox14.Text;
                this.Hide();
                Form2.Show();
            }
            else
            {
                MessageBox.Show("Ви не виконали жодного тесту", "Помилка!",
MessageBoxButtons.OK, MessageBoxIcon.Stop);

            }
        }
    }
}
```

ДОДАТОК Е

Лістинг програми виконання тестів

```
using System;
using System.Threading;
using System.Management;
using System.Windows.Forms;
using System.ComponentModel;
using System.Text;
        private void button2_Click(object sender, EventArgs e)
        {
            test1++;
            DateTime dold = DateTime.Now;
            for (int p = progressBar1.Minimum; p <= progressBar1.Maximum; p++)
            {
                progressBar1.Value = p;
            }
            int errorPercent = 0;
            int [,]marsh = new int[10,10];
            for (int i = 0; i < 10; i++)
                {
                    for (int j = 0; j < 10; j++)
                    {
                        marsh[i, j] = 1;
                    }
                }
            for (int i = 0; i < 10; i++)
            {
                for (int j = 0; j < 10; j++)
                {
                    if (marsh[i, j] != 1)
                    {
                        errorPercent++;
                    }
                    marsh[i, j] = 0;
                }
            }
            for (int i = 0; i < 10; i++)
            {
                for (int j = 0; j < 10; j++)
                {
                    if (marsh[i, j] != 0)
                    {
                        errorPercent++;
```

```
            }
        }
    }
    TimeSpan sp = DateTime.Now - dold;
    textBox7.Text = Convert.ToString(sp);
    textBox11.Text = Convert.ToString(errorPercent);
}

private void button3_Click(object sender, EventArgs e)
{
    MessageBox.Show("В доступній області пам'яті створюється масив\nдовільного розміру. "Інформація про тест", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
private void button4_Click(object sender, EventArgs e)
{
    test2++;
    DateTime dold = DateTime.Now;
    int errorPercent = 0;
    bool wtf = true;
    int[,] marsh = new int[10, 10];
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            if (wtf == true)
            {
                marsh[i, j] = 1;
                wtf = false;
            }
            else
            {
                marsh[i, j] = 0;
                wtf = true;
            }
        }
    }
    wtf = true;
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            if (wtf == true)
            {
                if (marsh[i, j] == 0)
```

```
            {
                errorPercent++;
            }
        marsh[i, j] = 0;
        wtf = false;
        }
        else
        {
            if (marsh[i, j] == 1)
            {
                errorPercent++;
            }
         marsh[i, j] = 1;
         wtf = true;
        }
    }
}
//////////////////////////////////////////////////////
for (int p = progressBar1.Minimum; p <= progressBar1.Maximum; p++)
{
   progressBar1.Value = p;
}
wtf = true;
for (int i = 0; i < 10; i++)
{
   for (int j = 0; j < 10; j++)
   {
      if (wtf == true)
      {
         if (marsh[i, j] == 1)
         {
            errorPercent++;
         }
         wtf = false;
      }
      else
      {
      if (marsh[i,j]==0)
        {
          errorPercent++;
        }
      wtf = true;
      }
   }
}
```

```
            TimeSpan sp = DateTime.Now - dold;
            textBox8.Text = Convert.ToString(sp);
            textBox12.Text = Convert.ToString(errorPercent);
            }
        private void button5_Click(object sender, EventArgs e)
        {
            MessageBox.Show("В доступній області пам'яті створюється матриця\nтипу 1010. Потім іде перевірка чи правильно розмістились 1 та 0 в матриці, "Інформація про тест", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        private void button7_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
        private void button6_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Программа ТЕСТУВАННЯ ОЗП ", "Розробник", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        private void button8_Click(object sender, EventArgs e)
        {
            test3++;
            DateTime dold = DateTime.Now;
            for (int p = progressBar1.Minimum; p <= progressBar1.Maximum; p++)
            {
                progressBar1.Value = p;
            }
            int errorPercent = 0;
            bool wtf = true;
            int[,] marsh = new int[100, 100];
            for (int i = 0; i < 100; i++)
            {
                for (int j = 0; j < 100; j++)
                {
                    if (wtf == true)
                    {
                        marsh[i, j] = 1;
                        j++;
                        marsh[i, j] = 1;
                        wtf = false;
                    }
                    else
                    {
                        marsh[i, j] = 0;
```

```
            j++;
            marsh[i, j] = 0;
            wtf = true;
        }
    }
}
//////////////////////
wtf = true;
for (int i = 0; i < 100; i++)
{
    for (int j = 0; j < 100; j++)
    {
        if (wtf == true)
        {
            if (marsh[i, j] == 1 && marsh[i, j + 1] == 1)
            {
                marsh[i, j] = 0;
                j++;
                marsh[i, j] = 0;
                wtf = false;
            }
            else
            {
                errorPercent++;
                wtf = false;
            }
        }
        else if (wtf == false)
        {
            if (marsh[i, j] == 0 && marsh[i, j + 1] == 0)
            {
                marsh[i, j] = 1;
                j++;
                marsh[i, j] = 1;
                wtf = true;
            }
            else
            {
                errorPercent++;
                wtf = true;
            }
        }
    }
}
//////////////////
```

```
    wtf = true;
    for (int i = 0; i < 100; i++)
    {
        for (int j = 0; j < 100; j++)
        {
            if (wtf == true)
            {
                if (marsh[i, j] == 0 && marsh[i, j + 1] == 0)
                {
                    marsh[i, j] = 1;
                    j++;
                    marsh[i, j] = 1;
                    wtf = false;
                }
                else
                {
                    errorPercent++;
                    wtf = false;
                }
            }
            else if (wtf == false)
            {
                if (marsh[i, j] == 1 && marsh[i, j + 1] == 1)
                {
                    marsh[i, j] = 0;
                    j++;
                    marsh[i, j] = 0;
                    wtf = true;
                }
                else
                {
                    errorPercent++;
                    wtf = true;
                }
            }
        }
    }
    ////////////////////////////////////////////////
    TimeSpan sp = DateTime.Now - dold;
    textBox9.Text = Convert.ToString(sp);
    textBox13.Text = Convert.ToString(errorPercent);
}
private void button9_Click(object sender, EventArgs e)
{
```

```
        MessageBox.Show("В вільній області пам'яті створюється масив послідовностей 11 та 00", "Інформація про тест", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    private void button10_Click(object sender, EventArgs e)
    {
        test4++;
        DateTime dold = DateTime.Now;
        for (int p = progressBar1.Minimum; p <= progressBar1.Maximum; p++)
        {
            progressBar1.Value = p;
        }
        int errorPercent = 0;
        bool wtf = true;
        int[,] marsh = new int[1000, 1000];
        for (int i = 0; i < 1000; i++)
        {
            for (int j = 0; j < 1000; j++)
            {
                if (wtf == true)
                {
                    marsh[i, j] = 1;
                    j++;
                    marsh[i, j] = 1;
                    wtf = false;
                }
                else
                {
                    marsh[i, j] = 0;
                    j++;
                    marsh[i, j] = 0;
                    wtf = true;
                }
            }
        }
        //////////////////////
        wtf = true;
        for (int i = 0; i < 1000; i++)
        {
            for (int j = 0; j < 1000; j++)
            {
                if (wtf == true)
                {
                    if (marsh[i, j] == 1 && marsh[i, j + 1] == 1)
                    {
```

```
                marsh[i, j] = 0;
                j++;
                marsh[i, j] = 0;
                wtf = false;
            }
            else
            {
                errorPercent++;
                wtf = false;
            }
        }
        else if (wtf == false)
        {
            if (marsh[i, j] == 0 && marsh[i, j + 1] == 0)
            {
                marsh[i, j] = 1;
                j++;
                marsh[i, j] = 1;
                wtf = true;
            }
            else
            {
                errorPercent++;
                wtf = true;
            }
        }
    }
}
//////////////////
wtf = true;
for (int i = 0; i < 1000; i++)
{
    for (int j = 0; j < 1000; j++)
    {
        if (wtf == true)
        {
            if (marsh[i, j] == 0 && marsh[i, j + 1] == 0)
            {
                marsh[i, j] = 1;
                j++;
                marsh[i, j] = 1;
                wtf = false;
            }
            else
            {
```

```
                    errorPercent++;
                    wtf = false;
                }
            }
            else if (wtf == false)
            {
                if (marsh[i, j] == 1 && marsh[i, j + 1] == 1)
                {
                    marsh[i, j] = 0;
                    j++;
                    marsh[i, j] = 0;
                    wtf = true;
                }
                else
                {
                    errorPercent++;
                    wtf = true;
                }
            }
        }
    }
    ////////////////////////////////////////////
    TimeSpan sp = DateTime.Now - dold;
    textBox10.Text = Convert.ToString(sp);
    textBox14.Text = Convert.ToString(errorPercent);
}
private void button12_Click(object sender, EventArgs e)
{
    MessageBox.Show("Тест аналогічний до попереднього, але розмірність матриці взята у 100 разів більша.", "Інформація про тест", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

ДОДАТОК И

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:_____ Програмний засіб тестування оперативної пам'яті комп'ютерів_____

Тип роботи: _____Магістерська кваліфікаційна робота_____

(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ _____ кафедра обчислювальної техніки_____

(кафедра, факультет (інститут), навчальна група)

Керівник _____Колкесник І.С., доц. каф. ОТ_____

(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	91%
Загальна схожість	9%

Аналіз звіту подібності (відмітити потрібне)

- ✓ Запозичення, виявлені 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 автора. Роботу направити на доопрацювання.
- ☐ 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.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ _____ Стрілковський В.С._____

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____

_____

_____

_____

Особа, відповідальна за перевірку_____ _Захарченко С.М__

(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали, посада)