Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інтелектуальних інформаційних технологій та автоматики
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ДІАГНОСТИКИ НЕСПРАВНОСТІ ПК

Виконав: студент 4 курсу, групи 3КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Шимонюк В.О.
(прізвище та ініціали)
Керівник: к.т.н., доцент кафедри КН
Колодний В.В.
(прізвище та ініціали)
«07» 06 2024 р.
Рецензент: к.т.н., доцент кафедри АПТ
Барабан М.В.
(прізвище та ініціали)
«07» 06 2024 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"07" 06 2024 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Шимонюку Владиславу Олександровичу

1. Тема роботи: Програмний модуль діагностики несправності ПК.
керівник роботи: Колодний Володимир Володимирович, к.т.н., доцент
затверджені наказом вищого навчального закладу "11" 03 2024 року №__

2. Строк подання студентом роботи_____ 27.05.2024

3. Вихідні дані до роботи :
Об'єм пам'яті для тестування – не більше 8Гб;
Кількість циклів тестування – 5шт;
Кількість тестів – 5шт;
Мова програмування – об'єктно-орієнтована.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
вступ, аналіз сучасних технологій діагностики несправності ПК, розробка алгоритму діагностики несправності ПК, програмна реалізація програмного модуля діагностики несправності ПК, висновки, перелік використаних джерел, додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
структура програмного модуля діагностики несправності ПК; схема загального алгоритму функціонування програмного забезпечення; загальна UML-діаграма класів; приклади роботи програми

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Колодний В.В., к.т.н., доцент		

6. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Обґрунтування доцільності розробки програмного модуля діагностики несправності ПК	06.05.2024	07.05.2024	
2	Розробка програмного модуля діагностики несправності ПК	08.05.2024	09.05.2024	
3	Програмна реалізація модуля діагностики несправності ПК	13.05.2024	14.05.2024	
4	Аналіз функціонування програмного модуля діагностики несправності ПК	19.05.2024	19.05.2024	
5	Оформлення додатків	23.05.2024	26.05.2024	
6	Розробка інструкції користувача	26.05.2024	27.05.2024	
7	Оформлення матеріалів до захисту БДР	04.06.2024	06.06.2024	

Студент _____ Шимонюк В.О
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ Колодний В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 621.374.415

Шимонюк В.О. Програмний модуль діагностики несправності ПК. Бакалаврська дипломна робота складається з 70 сторінок формату А4, на яких є 21 рисунок, список використаних джерел містить 21 найменування.

В даній бакалаврській дипломній роботі досліджено процес розробки програмного модуля діагностикит несправності ПК.

В роботі проведено аналіз основних несправностей ПК та методів їх виявлення. Проаналізовані програми аналоги для діагностики несправності ПК, виявлені їх переваги та недоліки. Проаналізовано основні несправності цифрових схем комірок пам’яті ПК. Розроблено алгоритми діагностики несправності комірок пам’яті ПК.

Здійснено програмну реалізацію модуля діагностики несправності ПК. Проведено тестування роботи програмного забезпечення та підтверджено його працездатність практичними результатами.

Ключові слова: діагностика, несправність ПК оперативна пам’ять C++ Visual Studio.

ABSTRACT

Shimonyuk V. PC malfunction diagnostics software module. The bachelor thesis consists of 70 pages of A4 format, on which there are 21 figures, the list of used sources contains 21 names.

In this bachelor's thesis, the process of developing a software module for PC malfunction diagnostics is investigated.

The paper analyzes the main PC malfunctions and their detection methods. Analog programs for PC malfunction diagnostics were analyzed, their advantages and disadvantages were identified. The main malfunctions of digital circuits of PC memory cells are analyzed. Algorithms for diagnosing the malfunction of PC memory cells have been developed.

The software implementation of the PC malfunction diagnosis module was carried out. The software was tested and its functionality was confirmed by practical results.

Keywords: diagnostics, PC malfunction, RAM, C++ Visual Studio.

ЗМІСТ

ВСТУП

Актуальність досліджень. Незважаючи на те, що мобільні пристрої надають споживачам можливість виконувати різні завдання, персональний комп'ютер (ПК) залишається актуальним. Користувач завжди прагне, щоб його комп’ютер працював бездоганно. Проте, повної впевненості в цьому немає, і час від часу трапляються збої в роботі такої техніки.

Коли виникають подібні проблеми, перша думка – віднести комп’ютер до сервісного центру. Однак іноді користувач може самостійно з'ясувати, що не так з його ПК. Перше, що потрібно зробити – це провести тестування працездатності свого комп'ютера.

Виявлення несправностей у роботі комп'ютера, складного цифрового пристрою, допомагає знайти способи вирішення проблеми та визначити її причини.

Однак ремонт апаратної частини обладнання вимагає не тільки відповідних знань, але й часто фінансових витрат. Звісно, деякі проблеми можна спробувати вирішити самостійно вдома, без витрат. Проте такий ремонт можливий лише для відносно простих пристроїв. Всі інші випадки – це завдання для фахівців сервісного центру.

Проблема тестової діагностики цифрових схем виникає на різних етапах їх виробництва та експлуатації, включаючи взаємопов’язані задачі. Перша з них полягає у визначенні стану досліджуваного пристрою. Прийнято розрізняти кілька видів технічного стану цифрових пристроїв. Основний стан цифрових пристроїв – справний, коли схема відповідає всім вимогам, встановленим технічною документацією. У протилежному випадку пристрій перебуває в одному з несправних станів.Якщо встановлено, що цифровий пристрій несправний, то вирішується друга задача: здійснюється пошук несправності (дефекту) його складових, ціль якого – визначення місця і виду несправності.

Несправності цифрових пристроїв виникають через використання несправних компонентів, таких як логічні елементи, що реалізують найпростіші логічні функції, елементи пам'яті тощо. Крім того, причинами несправностей можуть бути розриви

або короткі замикання у міжкомпонентних з'єднаннях, порушення умов експлуатації схеми, наявність помилок при проектуванні та виробництві, а також ряд інших факторів.

У нашому випадку пропонується проводити діагностику комірок пам’яті персонального комп’ютера. Розроблений метод діагностики має одну важливу особливість – дуже низькі витрати на придбання та встановлення системи. Для введення у виробничий цикл більшості існуючих систем діагностики необхідно закупити дуже дороге програмне забезпечення та відповідні апаратні засоби, провести установку драйверів і тестування. Систему діагностики елементів пам’яті можна впровадити на підприємстві шляхом простого розширення вже існуючої системи діагностики.

Все це пояснює доцільність розробки програмного модуля діагностування несправності пам’яті ПК.

Об’єкт дослідження – це процес діагностування несправності ПК.

Предмет дослідження – програмні засоби діагностування несправності ПК.

Мета дослідження – розширення функціональних можливостей програмного забезпечення діагностики несправності ПК.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- проаналізувати сучасні технологій діагностування несправності ПК;

- проаналізувати переваги та недоліки програм аналогів діагностування несправності ПК;

- розробити алгоритм діагностування несправності пам’яті ПК;

- здійснити програмну реалізацію модуля діагностування несправності пам’яті ПК;

- провести тестування нової розробки та проаналізувати отримані результати.

1 ОБГРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ ДІАГНОСТИКИ НЕСПРАВНОСТІ ПК

1.1 Основні характерні несправності ПК та їх діагностика

Перед тим, як виконувати ремонт комп'ютерної техніки, необхідно з'ясувати причини того, чому вона не працює. Проблеми з комп'ютером поділяються зазвичай на дві основні. Це проблеми програмного характеру, тобто пов'язані зі збоями у роботі програм чи операційної системи, і проблеми апаратні, пов'язані з виходом з ладу устаткування комп'ютера, інакше кажучи – його електронної частини.

Наведемо найпоширеніші випадки: при включенні комп'ютера спалахують індикаторні світлодіоди на передній панелі системного блоку, чути обертання вентиляторів і жорсткого диска всередині, але при цьому зображення на монітор не виводиться і відсутній характерний для початкового завантаження короткий одиночний звуковий сигнал системного динаміка (спікера). У такій ситуації очевидно, що проблема носить апаратний характер, тобто несправна якась комплектуюча, або, як часто буває, десь у ланцюгу живлення зник електричний контакт.

Тому перш за все - вимикаємо живлення комп'ютера, відгвинчуємо гвинти кріплення і знімаємо бічну кришку сиастемного блоку. Уважно дивимося на дроти і колодки, що йдуть від блоку живлення до материнської плати на предмет, чи вони щільно вставлені в роз'єми на платі (як правило, це два роз'єми: 20(24) і 4 контактні). Конструкцією цих роз'ємів передбачені засувки для запобігання випаданню колодок, проте іноді це все ж таки трапляється і колодка, якщо і не випадає повністю, то має нещільний контакт з роз'ємом на материнській платі. Перевіряємо цю справу; за потреби вийняли і щільно вставили на місце. Включаємо живлення комп'ютера та натискаємо на кнопку увімкнення.

Якщо ситуація не змінилася - знову вимикаємо живлення системного блоку, після чого знаходимо на материнській платі невелику круглу батарейку CR2032. Акуратно витягаємо цю батарейку з її посадкового місця і чимось металевим

перемикаємо два контакти в посадковому місці, звідки щойно витягли батарейку.

Вставляємо батарейку на місце. Вона підживлює незалежну пам'ять материнської плати з налаштуваннями BIOS, в яких зберігається конфігурація системи. Коли ми вийняли батарейку, ми знеструмили BIOS і всі налаштування стали заводськими. Перемикати контакти потрібно для того, щоб розрядилися конденсатори, що знаходяться в ланцюзі живлення мікросхеми EPPROM, забезпечуючи скидання налаштувань BIOS на 100%.

Крім того, для скидання BIOS можна скористатися джампером (перемичкою). Ця перемичка в більшості випадків знаходиться поряд з батарейкою, знайти її не важко. Вона зазвичай позначена написом типу CLEAR CMOS. У нормальному положенні джампер встановлений у положення 1-2, а для скидання налаштувань BIOS його необхідно переставити в положення 2-3, після чого повернути назад.

Далі вмикаємо живлення комп'ютера і натискаємо кнопку включення. Якщо комп'ютер намагається завантажитись, тобто почав виводити на монітор сигнал як завжди при включенні комп'ютера, потрібно зайти в налаштування BIOS (зазвичай це клавіша Del або F1) і виставити час, дату, порядок завантаження пристроїв (SSD, HDD) як необхідно.

Якщо маніпуляції з батареєю вам ніяк не допомогли і комп'ютер не працює, можна спробувати акуратно вийняти всі плати оперативної пам'яті з роз'ємів на материнській платі та спробувати запустити комп'ютер без них. Звичайно виймати пам'ять необхідно також при вимкненому живленні системного блоку.

Якщо після включення комп'ютер став видавати звукові сигнали у вигляді писків, то проблема, швидше за все, в тому, що оперативна пам'ять вийшла з ладу. Точніше перевірити можна встановивши у комп'ютер справний модуль оперативної пам'яті того типу, який використовується у даному ПК і спробувати знову включити комп'ютер.

Буває так, що і без модулів пам'яті комп'ютер теж не вмикається. Останній варіант, напевно, витягти відеокарту, якщо вона не вбудована на материнській платі. Діагностика в цьому випадку аналогічна до попереднього пункту, де ми вилучали модулі оперативної пам'яті. Якщо звукові сигнали без відеокарти з'явилися, то

причина у відеокарті, якщо сигналів системного динаміка, як і раніше, немає, то причина, швидше за все, в материнській платі або набагато рідше – в центральному процесорі.

Буває іноді, що винен блок живлення. Але в цьому випадку материнська плата йде в захист і примусово вимикається. Часто буває, що блок живлення з якихось причин видає недостатню напругу живлення або «просідає» під навантаженням. Це перевіряється тестером чи мультиметром. Сучасний блок живлення повинен видавати три типи напруги: 3,3В, 5В, 12В. Так, для чистоти експерименту на час перевірки краще відключити від материнської плати всі шлейфи та дроти, крім дротів живлення та кнопки включення комп'ютера.

Діагностику в цьому випадку треба починати безпосередньо з кнопки включення. Дуже часто ця кнопочка виходить з ладу або просто провалюється всередину через невдалу конструкцію корпусу. Для цього необхідно зняти передню панель системного блоку та переконатися, що кнопка на місці та працездатна. Перевірити працездатність кнопки включення дуже просто: потрібно від'єднати провід, що йде від кнопки до материнської плати, і продзвонити її мультиметром. При натисканні на кнопку ланцюг повинен замикатися, відповідно при його відпусканні контакт у ланцюзі повинен пропадати. Якщо кнопка впорядку переходимо до перевірки блока живлення.

1.2 Аналіз програм анаогів діагностики несправності ПК

Несправності цифрових схем з'являються в результаті застосування несправних складових компонентів, таких, як логічні елементи, що реалізують найпростіші логічні функції, елементи пам'яті та ін. Крім того, причиною несправностей можуть бути виникнення розривів чи коротких замикань у міжкомпонентних з'єднаннях, порушення умов експлуатації схеми, наявність помилок при проектуванні і виробництві і ряд інших факторів.

В нашому випадку пропонується проводити тестування елементів пам’яті персонального комп’ютера.

Ознаки можливого виходу з ладу оперативної пам'яті:

- «випадання» комп'ютера у так званий «синій екран смерті» з кодом помилки 0x00000050, коли робота комп'ютера раптово припиняється, на екрані з'являється синя заставка з білим англійським текстом, і в ній, в абзаці «Technical Information», відображається код 0x00000050 (рис. 1.1). Код помилки 0X00000050 свідчить саме про наявність проблем з пам'яттю;

- комп'ютер повільно працює, періодично раптово зависає або перезавантажується;

- програми або комп'ютерні ігри «самовільно» припиняють роботу (з подальшою появою повідомлення про виникнення критичної помилки або без нього);

- копіювання або збереження файлів на комп'ютері відбувається з помилками (при спробі відкрити файл з'являється повідомлення про те, що він пошкоджений, навіть якщо його копіювання або збереження завершилося без проблем);

- комп'ютер не запускається. Після увімкнення всередині системного блоку починають обертатися кулери (чути гул вентиляторів), але далі цього справа не йде (на екрані нічого не відображається). У такому разі протестувати пам'ять викладеними нижче способами не вдасться.

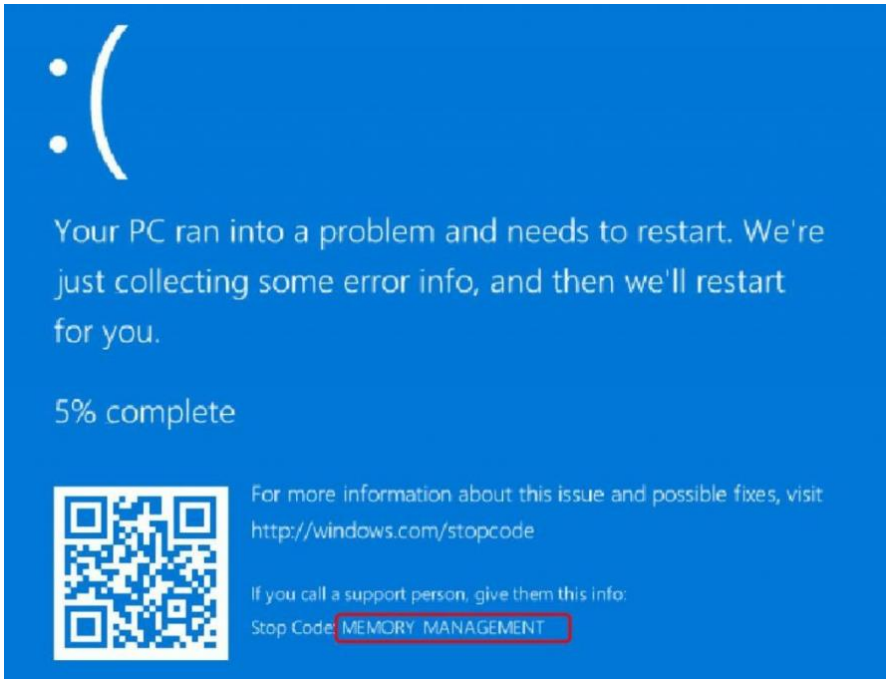


Рисунок 1.1 – Екран BSOD при несправній роботі пам’яті

Вихід з ладу оперативної пам'яті може викликати й інші наслідки, що проявляються нестабільною роботою комп'ютера.

До складу нових версій Windows, починаючи з Vista (Windows Vista, Windows 7, Windows 8, Windows 10), входить утиліта, що дозволяє протестувати оперативну пам'ять комп'ютера самостійно.

Запустити цю утиліту можна наступних способом:

1. Включити (перезавантажити) комп'ютер і під час запуску Windows де-кілька разів натискати кнопку Tab. Запуститься диспетчер завантаження Windows (рис 1.2). У ньому за допомогою клавіші Tab потрібно перенести виділення на пункт «Засіб перевірки пам'яті», після чого натиснути клавішу «Enter».

Результатом виконання буде поява вікна (рис 1.3.), в якому можна вибрати один з двох варіантів початку перевірки пам'яті. При виборі першого варіанта комп'ютер відразу ж перезавантажиться і почнеться перевірка пам'яті, при виборі другого - перевірка пам'яті розпочнеться під час наступного запуску комп'ютера.

Як виглядає екран комп'ютера під час тестування оперативної пам'яті штатної утилітою Windows показано на рисунку 1.2.

Рисунок 1.2 – Диспетчер завантаження Windows

Процес перевірки може бути досить тривалим (чим більший об'єм встановленої на комп'ютері оперативної пам'яті, тим довше доведеться чекати завершення її перевірки, рис.1.3).

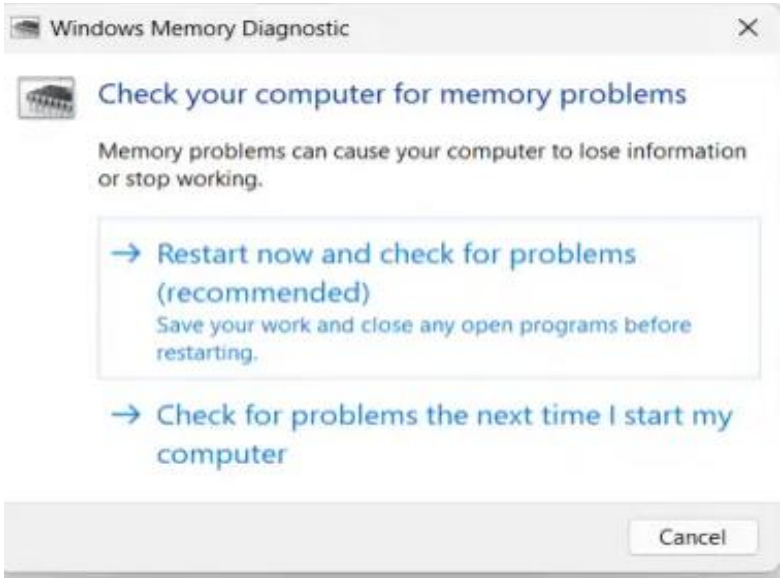


Рисунок 1.3 – Вікно програми тестової перевірки пам’яті

Виявлені під час перевірки помилки відразу будуть відображені на екрані і підсвічені червоним кольором.

Після завершення перевірки автоматично відбудеться запуск Windows. При вході в систему комп'ютер «покаже» звіт про результати перевірки (рис. 1.4).

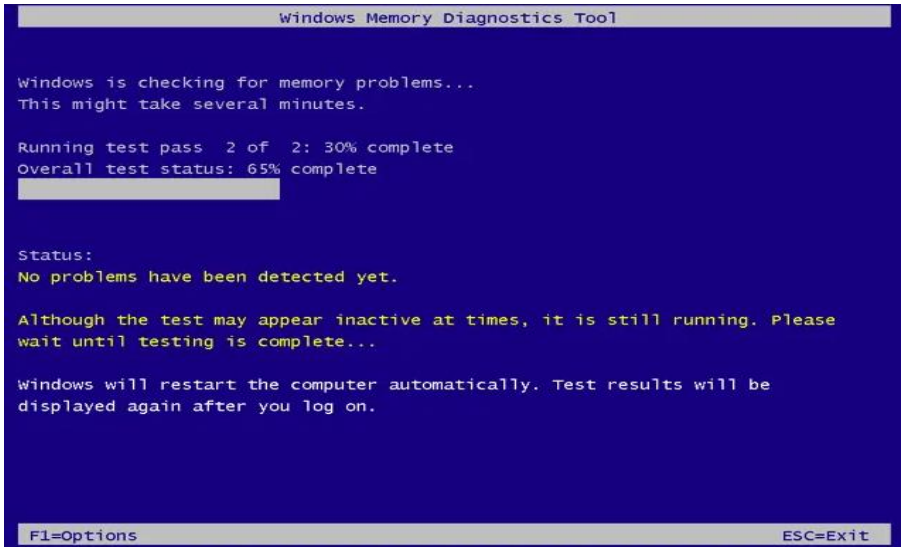


Рисунок 1.4 – Вбудовані засоби діагностування пам’яті Windows

Для тестування пам’яті утилітою Memtest86 необхідно завантажити комп'ютер з використанням створеного завантажувального диска або завантажувальної флешки Memtest86+.

Результатом завантаження комп'ютер з диска або флешки буде старт програми Memtest86+, яка відразу ж почне перевірку оперативної пам'яті. Тестування відбувається до його зупинки користувачем (тобто перевірка може тривати скільки завгодно). Для зупинки тесту достатньо натиснути кнопку Esc.

Для якісної перевірки пам'ять слід сканувати протягом від 30-40 хвилин до декількох годин (в залежності від обсягу пам'яті). Рекомендується провести не менше 3-4 проходів (кількість проходів відображається у вікні програми). Знайдені помилки відразу ж підсвічуються червоним кольором (рис. 1.5).

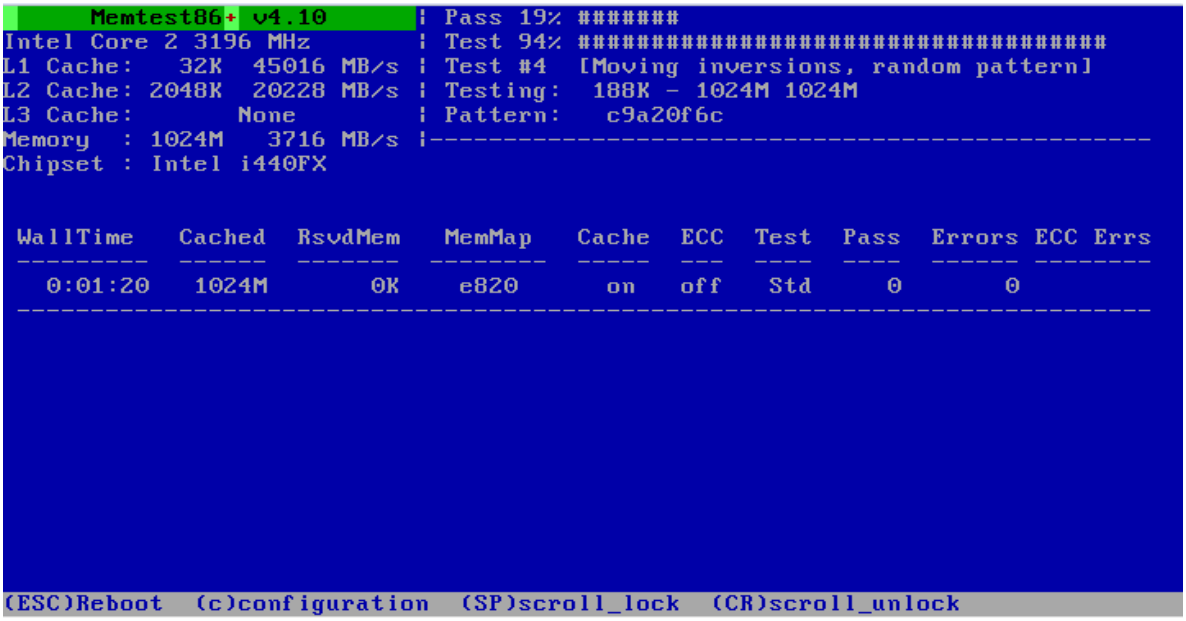


Рисунок 1.5 – Перевірка пам'яті за допомогою програми Memtest86+

В цих програмах реалізовано шість стандартних тестових перевірок ОЗП:

- запис-зчитування;
- метод одиниць, що біжать, і нулів;
- метод одиниць, що скачуть, і нулів;
- метод заповнення одиницями і нулями; метод множинної адресної вибірки; запис-зчитування з відновленням даних вперед-назад. Існує можливість

стохастичного функціонального контролю ОЗУ.

1.3 Постановка задачі

Серед всіх розглянутих систем діагностики елементів пам’яті найбільш швидкодіючими є системи з комбінованим методом діагностування. Однак існуючі системи комбінованого діагностування не можуть проводити перевірку на функціонування завершених вузлів вцілому.

Головними недоліками розглянутих аналогів є:

- відсутність українського інтерфейсу;
- поверхнева перевірка комірок пам’яті під час одного циклу.

Розроблюваний програмний модуль для тестування елементів пам’яті персональних ЕОМ має представляти собою завершену програму, яка готова до використання. В подальшому планується розробка більш удосконаленого комплексу як зі сторони технічної підтримки, так і зі сторони розширення функціональних можливостей і покращання інтерфейсу. Розробка програмного продукту передбачає можливість розширення функцій програми і використання їх для вирішення інших прикладних задач.

1.4 Висновок

В даному розділі проведено аналіз основних несправностей ПК та методів їх виявлення. Проаналізовані програми аналоги для діагностики несправності ПК, виявлені їх переваги та недоліки. Здійснено постановку задачі на розробку нового програмного рішення для діагностики оперативної пам’яті ПК.

2 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ДІАГНОСТИКИ НЕСПРАВНОСТІ ПК

2.1 Основні способи діагностики несправності ПК

Якщо комп'ютер експлуатується в несприятливих умовах або в запиленому приміщенні, то з часом він починає все повільніше і повільніше працювати, оскільки погіршується стан певних елементів у його конструкції.

Таким чином, серед найпоширеніших причин несправності сучасних комп'ютерів слід виділити такі:

- запилення різних роз'ємів або мікросхем і, як наслідок, їх перегрів;
- надмірне окиснення контактів;
- нерегрівання комплектуючих через сильне нагрівання;
- вигоряння контактів або комплектуючих через занадто сильний стрибок напруги;
- нестабільна робота встановленого блока живлення;
- неправильне заземлення.

В даний час існує величезна кількість програм, які допомагають користувачеві отримати, узагальнити та проаналізувати інформацію про комп'ютер. При подібних цілях подібні утиліти часто досить сильно відрізняються за своєю реалізацією, зручністю інтерфейсу, набору інструментів діагностики і функціональності в цілому. Серед подібних програм зустрічаються як вузькоспеціалізовані, призначені для детального розгляду однієї з підсистем комп'ютера, що дозволяють провести діагностику системи в цілому та всіх її підсистем окремо. Звичайно, значну частину інформації, що надається подібними утилітами, можна за бажання переглянути і зі штатних засобів Windows (наприклад, за допомогою утиліти Microsoft System Information), але, віддаючи належне розробникам подібних прграм, потрібно відзначити більш зручний та дружній інтерфейс таких утиліт та їх безперечно велику інформативність. Найчастіше до складу утиліт діагностики та моніторингу розробники включають тестові модулі, що дозволяють на основі нескладних, а

головне – нетривалих синтетичних тестів скласти повніше уявлення про комп'ютерну систему і прийняти продумане рішення щодо способів збільшення її продуктивності. Та й просто збір систематизованої докладної інформації про систему здатний часом відкрити користувачеві очі на причини тих чи інших проблем, що виникають під час роботи з ПК.

Проаналізуємо найпопулярніші сьогодні утиліти діагностики та моніторингу, при цьому крім таких вимог, як максимально дружній, зручний та інтуїтивно зрозумілий інтерфейс, забезпечення високого ступеня інформативності та функціональності, при виборі утиліт було висунуто умову, щоб програми були безкоштовними та доступними для вільного завантаження в Інтернеті.

Перші дві утиліти (WCPUID та CPU-Z) досить схожі програмні продукти, призначені насамперед для діагностики роботи процесорної підсистеми. Вони здатні ініціалізувати практично всі існуючі сьогодні x86-процесори (включаючи процесори з архітектурою AMD64) і більшість сучасних чіпсетів. При цьому віддамо належне творцям зазначених програм, які практично завжди вчасно встигають оновлювати версії своїх утиліт, щоб ті могли працювати з останніми моделями процесорів та наборів мікросхем системної логіки (рис. 2.1).

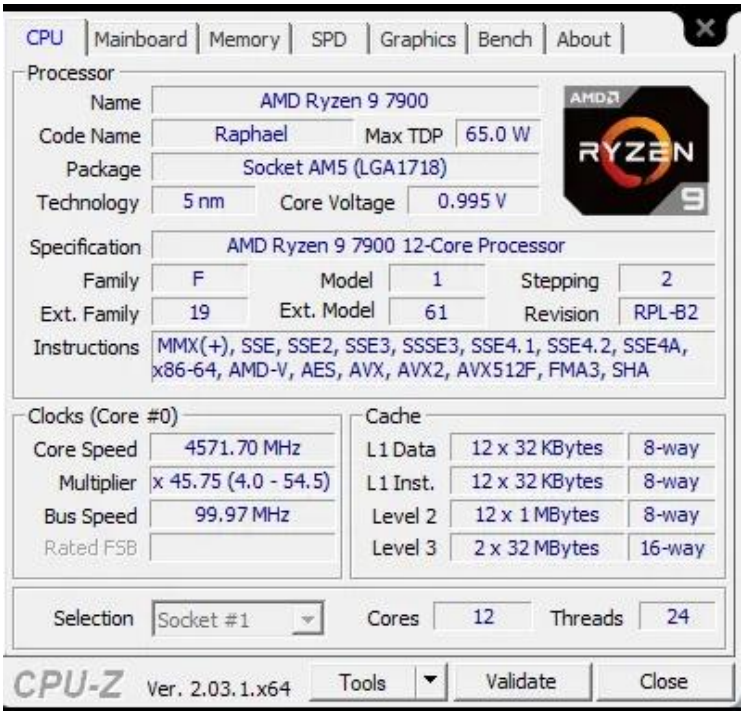


Рисунок 2.1 – Утиліта CPU-Z

Утиліта RivaTuner також є вузькопрофільною, якщо так можна сказати, утилітою, призначеною для роботи з відеопідсистемою. При цьому слід зауважити, що функції діагностики та моніторингу в даній утиліті відіграють допоміжну роль, а її призначення – тонке налаштування відеопідсистеми (насамперед на основі недокументованих можливостей драйверів NVIDIA Detonator) та розгін робочих частот графічної карти (рис. 2.2).

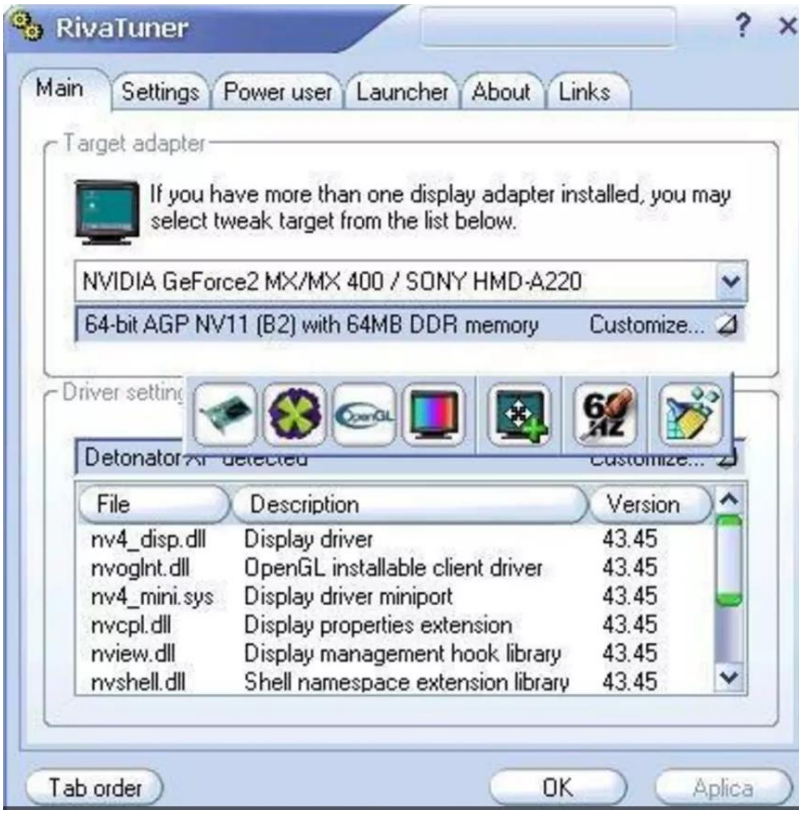


Рисунок 2.2 – Утиліта RivaTuner

Утиліта SiSoftware Sandra – це класика жанру. Вона надає величезний набір інструментів діагностики, дозволяючи збирати про систему всебічну інформацію, крім того, включає до свого складу ряд тестів, дозволяють порівняти продуктивність окремих підсистем комп'ютера та системи загалом із продуктивністю еталонних конфігурацій (рис. 2.3).

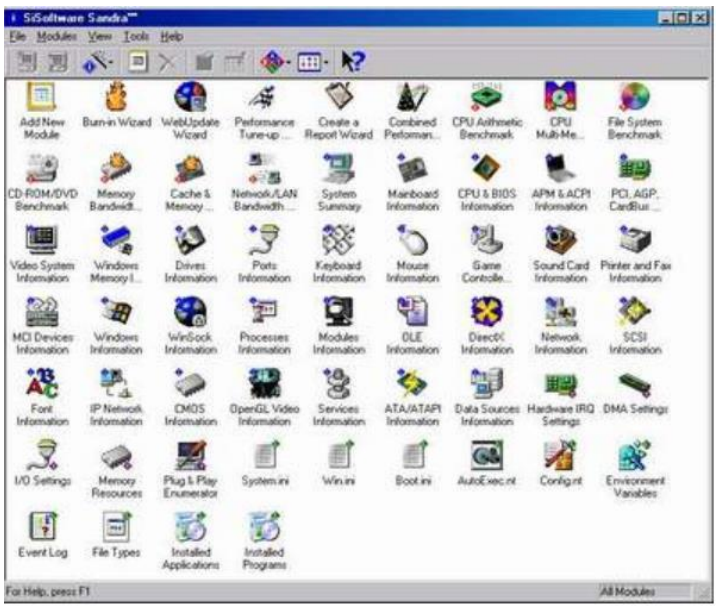


Рисунок 2.3 – Утиліта SiSoftware Sandra

Утиліта AIDA64 має багато спільного з утилітою SiSoftware Sandra, особливо щодо інструментів збору інформації про систему. Але, на відміну від неї, AIDA64 містить ряд досить цікавих тестових модулів, крім того, не вимагає установки (рис. 2.4).

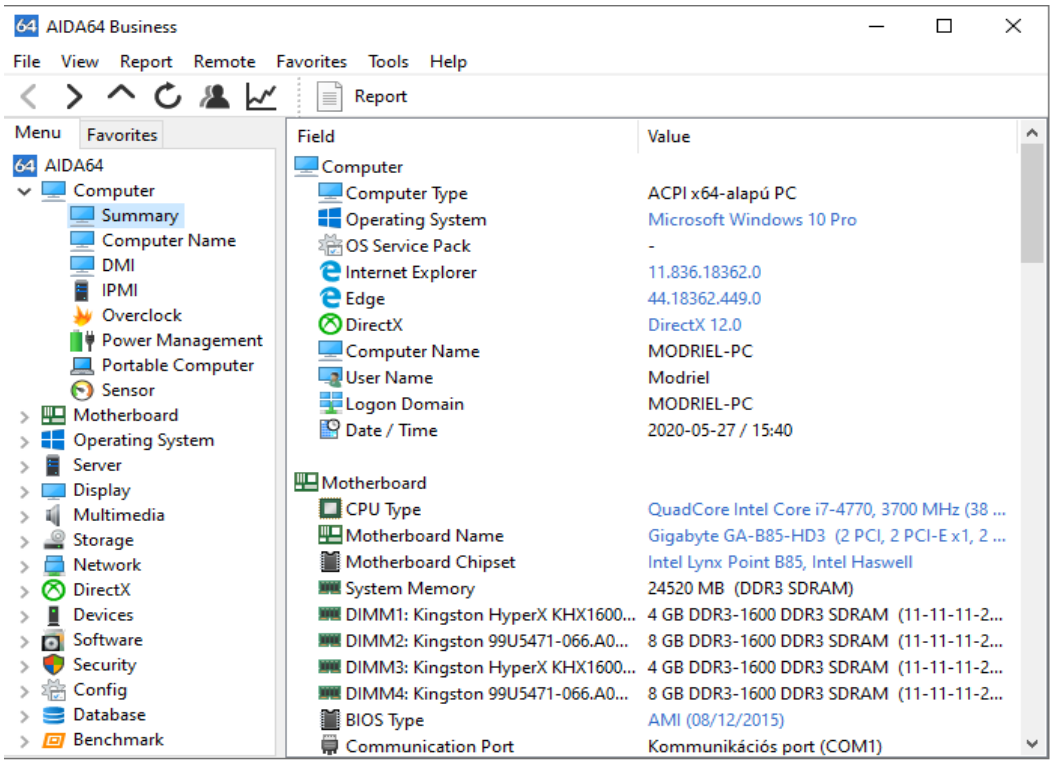


Рисунок 2.4 – Утиліта AIDA64

RivaTuner – це одна з найвідоміших сьогодні утиліт, призначених для моніторингу, діагностики та тонкого налаштування відеопідсистеми комп'ютера. Величезна популярність цієї утиліти обумовлена тим, що вона надає користувачеві доступ до безлічі недокументованих можливостей драйверів NVIDIA Detonator, а також дозволяє здійснювати розгін частот роботи графічного ядра і відеопам'яті графічних карт. Але ці функції утиліти ми цього разу залишимо за рамками статті, а основну увагу приділимо інформативним та діагностичним можливостям цієї унікальної програми.

Утиліта CPU-Z – це невелика (505 Кбайт), що не вимагає установки програма зі зручним дружнім інтерфейсом, що надає користувачеві доступ до інформації, що згруповано за п'ятьма категоріями, перехід між якими здійснюється за допомогою кнопок-закладок.

Але крім програмної діагностики несправності ПК усніють ще й апаратні методи.

Вони спрямовані на виявлення зіпсованих деталей комп'ютера. Збої в роботі системи можуть спостерігатися, якщо на техніку встановлено комплектуючу з заводським браком. Крім того, некоректне функціонування може бути спричинене зносом деталей. Ще один варіант, що впливає на роботу – перегрів.

Апаратна діагностика є методом пошуку проблем із обладнанням у системі комп'ютера. Ці діагностичні системи можуть бути запущені користувачем за допомогою внутрішньої програми, ініційованої системами керування комп'ютером або виконати тест усередині обладнання. Базова апаратна діагностика всіх систем комп'ютера, таких як процесор, чіпсет та пам'ять, перевіряється під час кожного завантаження системи. Ці апаратні діагностичні системи часто дають суттєве попереднє попередження про потенційні збої системи і мають назву POST-test.

Апаратна діагностика систем комп'ютера працює на базовому рівні. Зазвичай вона відстежує рівні потужності та час відгуку (рис. 2.5).

Рисунок 2.5 – Діагностична PCI плата для ремонту материнських плат ПК

Більшість активних апаратно-програмних комплексів сканує систему і те, що в ній відбувається під час послідовності завантаження, коли спеціалізовані програми запускаються для перевірки на різних комп'ютерних системах. Після завантаження, ця апаратна діагностика зазвичай працює у фоновому режимі, тільки попереджаючи користувача, коли відбувається щось дуже погане

2.2 Види запам’ятовуючих пристроїв ПК та їх структура

У сучасних комп'ютерах існує кілька типів пам'яті.

1. RAM (Random Access Memory, DRAM - динамічна, SRAM-статична) — оперативна пам'ять з довільним доступом розташовується на материнській платі і випускається в трьох типах чіпів: DIP (Dual in-Line Package) — двосторонні контакти, SIMM (Single in-Line Memory Module) - однобічні виводи; SIP (Single in-Line Package) - чіп має форму гребінця.

2. Стандартна пам'ять (Conventional), чи основна, рівна 640К за умови комплектації оперативною пам'яттю в 1 Мбайт, призначена для роботи програм DOS.

3. Верхня пам'ять (Upper) резервується у виді залишку між 640К і 1Мбайтом, використовується для обслуговування апаратного забезпечення й індикації завантаження окремих компонентів комп'ютера, розміщення BIOS.

4. Розширена пам'ять (Expanded) — пам'ять понад 640 К, призначена для роботи програм DOS під керуванням менеджера пам'яті HIMEM.

5. Додаткова пам'ять (Extended) понад 640 К до 64 Мбайтів і більш, використовувана під керуванням спеціального менеджера пам'яті для рішення об'ємних задач у DOS, а також для багатозадачного режиму під керуванням WIN DOWS.

6. Віртуальна пам'ять (Virtual) — частина дискового простору (Swap Disk), використовуваного системою WINDOWS для створення модуля обміну інформацією між оперативною і дисковою пам'яттю у випадку дефіциту не дискової пам'яті при виконанні завдань.

7. Постійна пам'ять (ROM — Read Only Memory), використовується через важливість поміщеної інформації тільки для читання при неможливості якої-небудь модифікації. Наприклад, у ній знаходиться BIOS, що змінюється тільки заміною мікросхем.

8. Кеш-пам'ять (Cache), чи тіньова, використовувана для збереження проміжних обчислень, що неодноразово можуть бути затребувані програмою з метою зменшення повторень обчислювальних процесів, що приводять до однакових результатів. Чіпи кеш-пам'яті встановлюються на системну плату, вони набагато швидше звичайної оперативної і тому істотно дорожче.

9. RAM-диск (віртуальний диск) пропонує оперативну пам'ять для запису часто використовуваних програм з метою прискорення роботи з файлами, що представляють як набори даних, так і програми [7].

Розглянемо більш детально деякі з вище названих видів. Характеристики і широкі можливості сучасних мікро-ЕОМ багато в чому визначаються досягненнями мікроелектроніки в створенні великих інтегральних мікросхем запам’ятовуючих пристроїв (ВІС ЗП). Мікросхеми пам’яті орієнтовані на побудову внутрішніх запам’ятовуючих пристроїв, що повинні забезпечити довгострокове збереження інформації й оперативне (порівнянне за часом з циклом роботи процесора) зчитування. Класифікація ЗП приведена на рисунку 2.6.

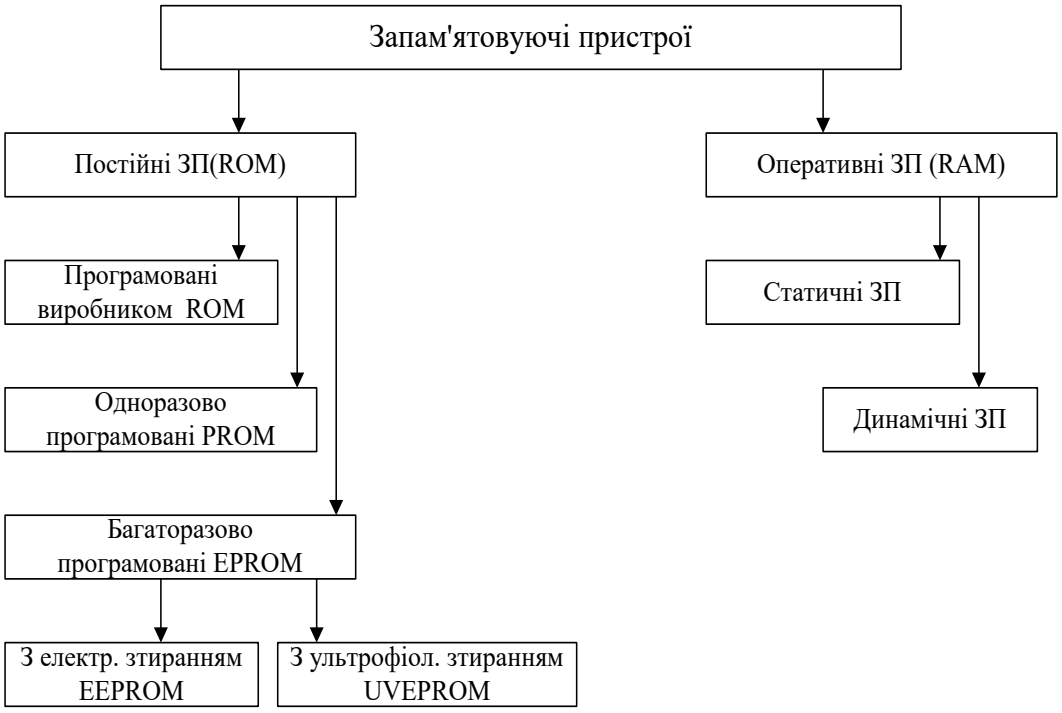


Рисунок 2.6 – Класифікація запам’ятовуючих пристроїв ПК

Можливості мікросхем пам’яті визначаються трьома групами характеристик: функціональними, схемотехнічними й експлуатаційними. до першої групи відносяться:

- функціональне призначення — основне призначення мікросхеми, наприклад, для побудови постійних ЗП, надоперативних ЗП, оперативних ЗП, відеопам’яті для графічних дисплеїв;

- інформаційна ємність (capacity) — максимальна кількість збереженої в ВІС інформації (виміряється в бітах, Кбит або Мбит);

- організація ВІС — розрядність інформаційного коду, який можна одночасно записати або зчитати з мікросхеми. Наприклад, ВІС ЗП ємністю 64 Кбит може мати організацію 16К Х 4 (4-розрядне слово) чи 8 К Х 8 (8-розрядне слово);

- час звертання (access time) — характеризує тимчасові витрати на виконання операцій читання/запису з ВІС;

- енергоспоживання — електрична потужність (чи струм), що ВІС споживає від джерел живлення. Звичайно вказується енергоспоживання в активному режимі (ВІС обрана і операції читання/запису виконуються) і в режимі збереження (ВІС не обрана, читання і запис неможливі).

Перелік схемотехнічних і експлуатаційних характеристик інтегральних схем пам'яті практично збігається з відповідними характеристиками мікропроцесорів. Усі ВІС ЗП, використовувані в мікропроцесорних обчислювачах, будуються на базі ПМОП чи КМОП технологій (чи їхніх удосконалених модифікаціях) з обов'язковою сумісністю вхідних і вихідних сигналів із ТТЛ рівнями.

Розглянемо наступні запам'ятовуючі пристрої.

Постійні запам'ятовуючі пристрої. Постійні запам'ятовуючі пристрої (ПЗП, read only memory, ROM) призначені для відтворення незмінної інформації, що заноситься в схему спеціальним чином. У ПЗП, як правило, записуються системні програми широкого призначення, таблиці, які необхідно постійно зберігати в пам'яті ЕОМ. Важливо, що при відключенні електроживлення інформація в ВІС ПЗП зберігається і стає доступною відразу після його включення. ПЗП підрозділяються на:

- програмовані виробником ПЗП. Запис інформації в такі елементи виконується виробником схеми за замовленням і специфікаціями користувача;

- одноразово програмовані ПЗП (ППЗП, programmable ROM — PROM). За допомогою спеціального пристрою (програматора) користувач має можливість самостійно записати в ВІС інформацію. Змінити занесені в ППЗП коди неможливо;

- перепрограмовані (стираємі, перепрограмовані) ПЗП (СППЗП, РПЗП, reprogrammable, erasable PROM — EPROM). Допускають стирання записаної інформації і повторне її занесення за допомогою програматорів, стирання можна здійснити електрично (ЕСППЗП, electrically erasable — EEPROM) чи за допомогою ультрафіолетових променів (УФ СППЗП, ultraviolet erasable — UVEPROM).

Слід зазначити, що розроблювачі і виробники мікропроцесорних обчислювальних систем в останні роки схиляються до кращого використання СППЗП. Це викликано неможливістю гарантувати стовідсоткову безпомилкову роботу програмного забезпечення і до можливості викинути масу замовлених ВІС ПЗП чи записаних ВІС ППЗП. Розглянемо 1 Мбіт СППЗП з електричним стиранням і програмуванням типу Am27C1024 фірми Advanced Micro Devices [2]. Пам'ять організована в 64 К шістнадцатирозрядних слів, що цілком відповідає двом сегментам МП 8086. Мікросхема виконана по КМОП технології на кристалі площею 52 мм2 і

розміщається в пластмасовому корпусі типу DIP з 40 виходами. Основою ПЗП є 16-розрядна матриця запам’ятовуючих елементів (кожен площею 20,25 мкм2), що складається з 128 стовпців і 512 рядків. Повна 16-розрядна адреса A15-A0 розділена на два поля: адресу стовпця A6-A0 і адреса рядка A15-A7. За допомогою дешифраторів рядків і стовпців код, що надходить по адресних входах, визначає одне слово з 64 К можливих. Інформаційний код із ПЗП видається по двонаправлених лініях даних D15—D0 із трьома станами. Низький активний рівень сигналу на вході вибірки кристала CS (chip select) включає буфер уведення-висновку даних і переводить схему в активний стан. У цьому режимі з затримкою 170 нс після встановлення адреси на входах A16-A0 на виходах D16-D0 формується код, записаний в обрану комірку пам’яті. Для виконання операції читання не потрібно ніякої синхронізації. В активному режимі при роботі з частотою 5 МГц ВІС споживає струм 50мА і розсіює потужність 50мВТ. У пасивному (невибраному) режимі потужність, що розсіюється, падає в 50 разів (струм живлення 1 мА).

У режим програмування ВІС переводиться високим рівнем сигналу на вході PR подачею спеціальної живильної напруги величиною 10,5 В. У цьому режимі лінії даних налагоджуються на прийом записуваного коду. Тривалість програмуючого імпульсу на вході PR складає 0,5 мс, що дозволяє заповнити весь кристал за 50 с. У СППЗП легко розміщається повний текст досить могутньої операційної системи (ОС) із усіма програмами підтримки. Включення такий ОС на кристалі до складу ЕОМ різко знижує вимоги до ємності зовнішніх запам’ятовуючих пристроїв і підвищує продуктивність системи вцілому.

Оперативні запам’ятовуючі пристрої. Великі інтегральні схеми для побудови ОЗП одержали назву схем запам’ятовуючих пристроїв з довільною вибіркою (ЗППВ, random access memory — RAM). Вони забезпечують режими запису, збереження і зчитування інформації. У роботі напівпровідникових ЗППВ є дві істотні особливості. При читанні інформації вона залишається в ВІС незмінною, тобто допускається багаторазове зчитування коду без його відновлення. Однак, якщо відключити електроживлення, збережені дані безповоротно руйнуються. При подачі живлення на

ЗППВ у його осередках з’являються випадкові коди, що не несуть ніякої інформації. Можна вказати два основних види ВІС ЗППВ:

- статичні ЗППВ (static RAM), здатні зберігати записану інформацію поки мікросхема підключена ло джерела живлення;

- динамічні ЗППВ (dynamic RAM — DRAM), час збереження інформації в який обмежено одиницями мс, в наслідок чого її потрібно періодично відновлювати — регенерувати (refresh process). Це робиться спеціальними вбудованими чи зовнішніми схемами таким чином, щоб процес регенерації був, по можливості, «непомітний» для мікропроцесора. Типові характеристики статичних і динамічних ЗППВ приведені вище, відповідно [2]. Порівняльний їх аналіз показує, що ємність динамічних схем у середньому в чотири рази вище, ніж статичних. Однак вони не витримують зіставлення по швидкодії (у 2—3 рази нижче) і по потужності, що розсіюється. Останнє особливо помітно для динамічних ЗППВ, виконаних по КМОП технології. Різке зниження цін на динамічні ВІС, викликане конку рентною боротьбою, прогрес у створенні високоякісних статичних елементів змусили ряд ведучих фірм (Intel, Mostek) відмовитися від подальшої розробки динамічних ЗППВ. Інші виробники (особливо японські) продовжують дослідження в цій області. Так, компанія Texas Instruments оголосила про випуск КМОП динамічних ЗППВ ємністю 4 Мбіт з часом доступу 150 нс (а в режимі статичної дешифрації стовпців 30 нс) і площею кристала 8,9 мм2.

Незважаючи на зазначені недоліки динамічні ЗППВ залишаються в даний час основною елементною базою для побудови ОЗП в ЕОМ. Як приклад розглянемо схему ДЗППВ фірми IBM, орієнтовану на мікропроцесорні застосування. ВІС ємністю 256 Кбіт з організацією, що апаратно набудовується, (256Кх1, 128Кх2, 64Кх4) виконана по удосконаленій КМОП технології на кристалі площею 50мм2. Мікросхема живиться напругою 5В, в активному режимі розсіює потужність 300мВт, а в режимі збереження — 25мВт, цілком сумісна по вході-виходу з ТТЛ схемами. Інформація зберігається в 16 запам’ятовуючих матрицях ємністю по 16 Кбіт, що складаються з 128 рядків і 128 стовпців. У мікросхемі мається чотири буфери введення-виведення, до яких підключене по 4 матриці запам’ятовуючих елементів.

Інформація надходить у чи схему видається з її по двонаправленим із трьома станами лініям D3—D0. Керуючі сигнали G3—G0 Дозволяють підключити буфер, і, отже, 64 Кбіт пам’яті до відповідного лінії даних. Тим самим можна встановлювати розрядність слова пам’яті. Якщо мікросхема використовується як пам’ять 256Кх1, те всі лінії даних поєднуються, а входи G3—G0 використовуються для завдання двох молодших розрядів адреса. Якщо необхідна організація 64Кх4, то входи заземлюються.

Шістнадцятирозрядна адреса розбита на два полюси. Вісім старших розрядів задають адреса рядка, а вісім молодших — стовпця. При звертанні до ВІС адреса рядка й адреса стовпця мультиплексуються на лініях адреси A7-A0, а їхній прийом стробується сигналами RAS (row adres strobe) і CAS (column adres strobe). Виконання операцій подається адреса стовпця, що фіксується по спадаючому фронті CAS. Спочатку на шину адреси подається адреса рядка, що приймається в буфер ВІС по спадаючому фронті сигналу RAS. Прийнята адреса дешифрується і визначається один з 128 рядків у всіх верхніх чи нижніх матрицях. Потім матрицях у всіх чотирьох групах. Таким чином, у кожній групі визначається унікальний біт, що лежить на перетинанні обраного рядка і стовпця. Тип операції (читання чи запис) задається сигналом W/R. При записі низький рівень сигналу W/R повинний супроводжувати поява на лініях даних коду для запису практично одночасно з адресою стовпця. При читанні сигнал W/R повинний бути високим, по сигналі CAS лінії даних переходять із третього стану б активне, а потім на них формується лічений з пам’яті код. Повний цикл чи читання запису вимагає не менш 150 нс. Можливий сторінковий режим, коли при фіксованому сигналом RAS адресі рядка послідовно чи довільно змінюється адреса стовпця. Для чи читання запису в цьому режимі доступно 256 X 4 біт, причому час звертання знижується до 100 нс. Керуючи сигналами G3—G0, можна додатково розширити розмір сторінки К 1024 біт, знизивши одночасно час звертання до 25 нс.

Динамічні елементи пам’яті в запам’ятовуючих матрицях гарантовано зберігають інформацію не більш 4 мс. Після закінчення цього інтервалу необхідно перезаписати стан усіх 256 К елементів, для чого ВІС переводиться в режим регенерації по сигналі RF. Регенерація виконується зовнішніми схемами послідовним

перебором усіх 256 адрес рядків. По спадаючому фронті сигналу RAS відбувається читання всієї поточної рядка у внутрішні підсилювачі, а потім автоматичний запис у ті ж елементи. Таким чином, за один цикл тривалістю 100 не регенерується 1024 елемента пам'яті. Повна регенерація ВІС займає не набагато більше 25 мкс, що складає 0,64 % інтервалу [4].

По оцінці фахівців у найближчі роки очікується широке впровадження в мікропроцесорній апаратурі динамічних ЗППВ, ємністю 1—4 Мбіта і статичних ЗППВ, ємністю 256 — 512 Кбіт.

2.3 Типові несправностей цифрових схем

Із множини різних видів несправностей виділяється клас логічних несправностей, котрі змінюють логічні функції елементів цифрової схеми. Вказаний клас несправностей займає одне з найперших місць серед несправностей цифрових схем. Для їх опису в більшості випадків використовуються наступні математичні моделі: константні несправності, несправності типу "коротке замикання", інверсні несправності і несправності типу "переплутування" [3].

Найбільш загальною і часто застосованою моделлю логічних несправностей являються константні несправності: константний нуль і константна одиниця, що означає наявність постійного рівня логічного нуля або логічної одиниці на входах або виходах несправного логічного елемента. Така модель несправностей часто називається класичною і широко використовується для опису інших типів несправностей.

Несправності типу «коротке замикання» (місткові несправності) з'являються при короткому замиканні входів і виходів логічних елементів. У залежності від виду застосовуваних логічних елементів можлива різна дія несправності на цифрову схему. Так, виникнення місткової несправності між полюсами x1 і x2 тривхідного елемента І (рис. 2.7а) еквівалентно фіктивному включенню двовхідного елемента І, що поєднує зазначені полюси. Подібна дія місткової несправності справедлива для елементів, що використовують позитивну логіку функціонування, і у випадку несправного елемента

І не змінює логічну функцію, реалізовану даним елементом. У той же час виникнення аналогічної несправності для трьохвхідного АБО елемента (рис. 2.7б) змінює функцію, реалізовану ним (рис. 2.7в), що приймає наступний вигляд: f = x1x2+x3. У випадку застосування елементів з негативною логікою функціонування дією містикової несправності, що виникла між вхідними полюсами логічних елементів, є фіктивне включення двовходового АБО елемента, наявність якого змінює функцію, реалізовану елементом І.

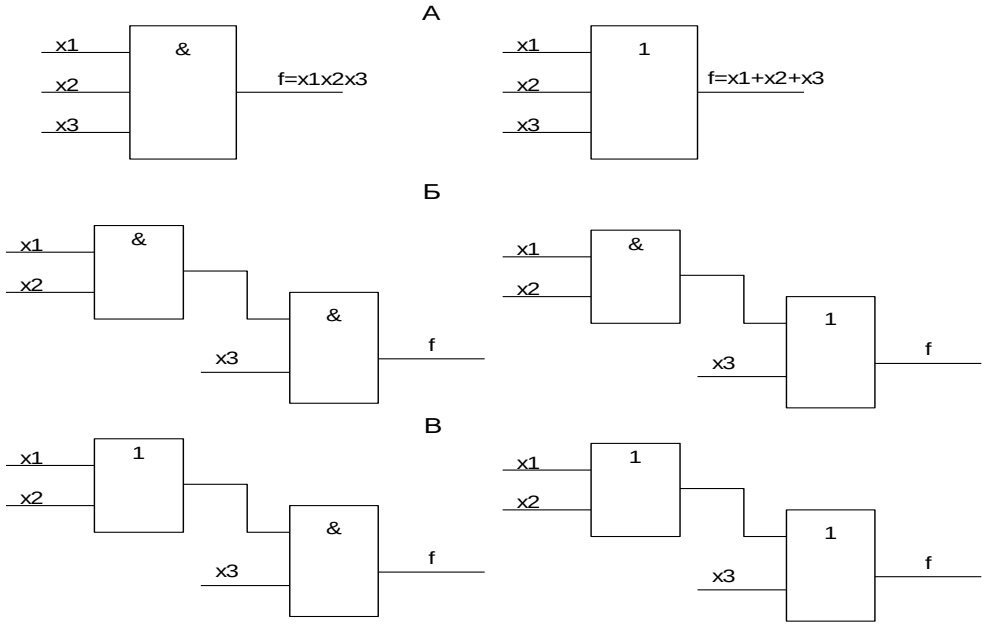


Рисунок 2.7 – Приклад несправності типу «міст» між полюсами xl і x2 елементів І і АБО (а) і результат її дії для позитивної (б) і негативної (в) логіки їхнього функціонування

Несправності типу «коротке замикання» підрозділяються на два види: несправності, викликані коротким замиканням входів логічного елемента, і несправності типу зворотного зв'язку. Для комбінаційної цифрової схеми, описуваної виразом F(x1, x2, .., xn), коротке замикання її μ, вхідних полюсів приводить до появи μ-кратної несправності типу «коротке замикання». Модель несправності, викликаної коротким замкненням μ входів схеми для позитивної логіки, приведена на рисунку 2.8а. На рисунку 2.8б показана логічна модель несправності, викликаної коротким замиканням вихідного полюса схеми з її μ входами. Існування подібного зв'язку між вихідним полюсом схеми і її входами може привести або до виникнення процесу

генерування, або до перетворення комбінаційної схеми в послідовну. Так, для схеми, приведеної на рисунку 2.4а, коротке замикання між вихідним полюсом і входами x1 і x2 для x1=x2=x3= 1 і x4 = 0 приводить до генерування високочастотного коливального процесу, у той же час коротке замикання між виходом і входами x3 і x4 для x3=x4=1— до перетворення розглянутої схеми в асинхронну послідовну.

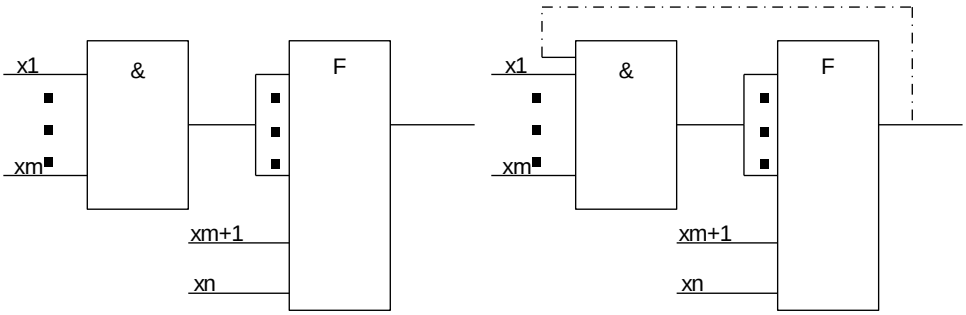


Рисунок 2.8 – Моделі несправностей, викликані коротким замиканням μ входів схеми (а) і виходу схеми з μ її входами (б)

У загальному випадку комбінаційна схема F(x1, x2, ..., xn), що містить μ-кратну несправність типу «коротке замикання» вихідного полюса на μ входів, перетвориться в послідовну, якщо для набору змінних x1, x2, ..., xn виконується рівність:

$$x1x2...x_{\mu}F^{/}(0,0,...,0,x_{\mu+1},...,x_n)F(1,1,...,1,x_{\mu+1},...,x_n)=1 \quad (2.1)$$

При умові, що комбінаційна схема перетвориться в схему генератора високочастотного коливального процесу

$$x1x2...x_{\mu}F(0,0,...,0,x_{\mu+1},...,x_n)F^{/}(1,1,...,1,x_{\mu+1},...,x_n)=1 \quad (2.2)$$

Для прикладу схеми, що містить дану несправність (рис. 2.9а), використовуючи (2.2) і аналітичний опис її функціонування F= x3(x/1 + x/2 +x4), одержуємо, що рівність x1=x2=x3=x/4=1 є умовою виникнення коливального процесу. З іншого боку, F(x1,x2,0,0)=0 і F(x1,x2,1,1)=1 і відповідно x3 =x4=1-умова перетворення вказаної схеми в послідовну (рис. 2.9б).

У результаті проведених досліджень показано, що будь-яка місткова несправність беззбиткової цифрової схеми може бути виявлена, а в роботі [4]

приводяться умови побудови універсального тесту для виявлення всіх константних несправностей на вхідних і вихідних полюсах цифрової схеми, а також можливих місткових несправностей між її виходами і входами. Складність виявлення всіляких несправностей типу «коротке замикання» пояснюється великою їхньою кількістю. Так, для цифрової схеми, що має n полюсів, число-n-кратних несправностей типу «коротке замикання» буде визначатися величиною Cn μ, що для практичних значень n і μ, є досить великою [4].

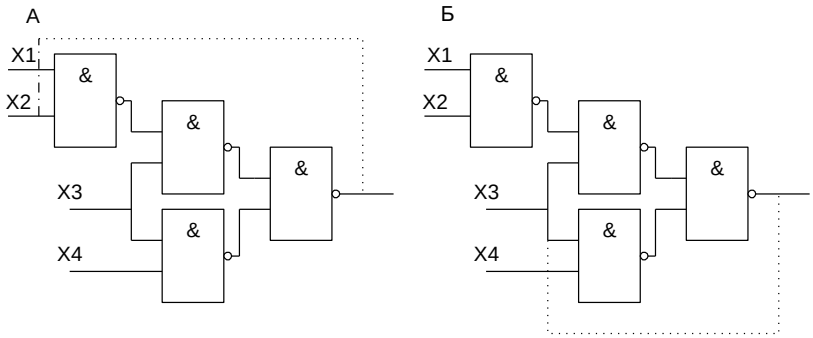


Рисунок 2.9 – Приклад виникнення несправності типу «коротке замикання» між вихідним полюсом і входами x1 і x2 для x1=x2=x3 = 1 і x4 = 0(а) і між виходом і входами x3 і x4 для x3=x4=l (б)

Інверсні несправності описують фізичні дефекти цифрових схем, що приводять до появи фіктивного інвертора по входу або виходу логічного елемента, що входить у дану схему. Інверсні несправності в сукупності з константними в ряді випадків [5] використовуються для побудови повної моделі несправної цифрової схеми. У цьому випадку всілякі несправності описуються тільки моделями константних і інверсних несправностей.

Несправності типу «переплутування» полягають у переплутуванні зв'язків цифрової схеми і викликаються помилками, що виникають при проектуванні і виробництві цифрових схем, що змінюють функції, виконувані схемою. Як правило, кількість несправностей даного класу досить висока, а процедура їхнього пошуку зводиться до процедури пошуку константних, місткових і інверсних несправностей.

Існує ряд моделей несправностей, що характеризуються визначеною технологією виробництва цифрових схем. Деякі з зазначених моделей і ефективність їхнього застосування розглядаються в [8].

2.4 Розробка алгоритмів діагностики комірок ОЗП

Для повної перевірки комірок ОЗП ємністю 64 біт потрібно 264 тактів, що навіть при гіпотетичній швидкодії 10Гц займе близько 32 роки. Астрономічним стає час повних іспитів для великих обсягів пам'яті. Однак повні перевірки є зовсім необов'язковими для того, щоб з високою вірогідністю оцінювати якість виготовлених комірок ОЗП. Для такої оцінки достатньо використати розроблені усічені тестові послідовності (кодові малюнки), що базуються на аналізі й обліку фізики відмовлень, технологічних, схемотехнічних і структурних особливостей комірки, що перевіряється. На сьогодні розроблено кілька десятків кодових малюнків. Однак вони ще й досі не стандартизовані і, мабуть, через це різні автори привласнюють тим самим малюнкам різні назви і під тією самою назвою мають на увазі різні малюнки. Наприклад, Маршалл і Гнатек відносять тест «Пінг-понг» до виду N log2 N, а Морган і фахівці фірми Minato Electronixs (Японія) — до тексту виду N2.

Оскільки таких прикладів можна привести чимало, представляється доцільним описати з достатнім ступенем подробиці найбільш уживані в практиці послідовності іспитів ВІС ОЗП.

«Марш одиниць і нулів». Усі комірки пам'яті заповнюють нулями («фон нулів»). Потім з першого осередку (вона має нульовий адрес) зчитують нуль і в неї записують одиницю. Цю процедуру повторюють для всіх інших осередків (тобто виконується «марш одиниць»).

Записавши одиницю в останній осередок, у зворотному порядку роблять зчитування одиниць і запис нулів в усі осередки, починаючи з останньої. Потім осередку заповнюють одиницями («фон одиниць») і алгоритм тестування повторюють для комплементарних даних (відбувається «марш нулів»). Тест «Марш одиниць і нулів» займає 10 N циклів тактової частоти (з урахуванням циклів

створення фону) і з достатньою вірогідністю виявляє несправності дешифратора. Він відноситься до мінімальних функціональних тестів.

Алгоритм такої тестової перевірки має вигляд, як показано на рисунку 2.10.

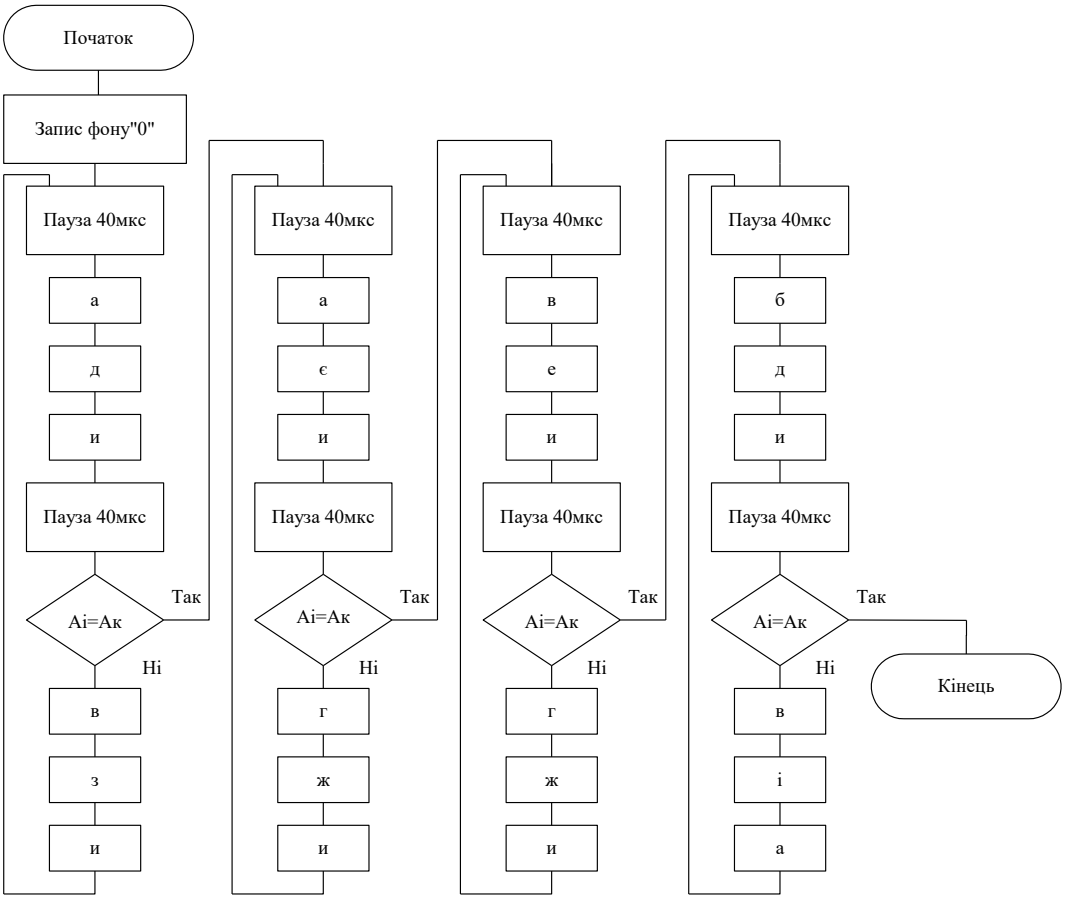


Рисунок 2.10 – Схема алгоритму тестової перевірки «Марш одиниць і нулів»

«Рухома діагональ». Усі комірки пам'яті, крім розташованих по діагоналі, заповнюють нулями (у діагональні осередки записують одиниці). Потім роблять зчитування вмісту комірок пам'яті уздовж стовпців у порядку зростання адрес. На виході утвориться довга послідовність нулів з одиницею на √n циклі і далі на кожному наступному (√M-1)-M циклі. Потім створюють новий малюнок діагоналі одиниць: кожна одиниця переміщається вправо на один стовпець, тобто в осередок з адресою, збільшеним на √N, і знову роблять зчитування умісту всіх комірок пам'яті. Цю процедуру повторюють для всіх можливих положень діагоналі, тобто √N раз. Тест « діагональ, що зрушується,» орієнтований на перевірку якості підсилювачів

зчитування (їхньої здатності до відновлення), що ставляться в тяжкі умови, приймаючи спочатку на свій вхід довгий потік нулів, а потім одну одиницю (чи навпаки — для комплементарного тесту). Тест «Рухома діагональ» забирає час, що відповідає 2N√N (тобто 2 N3/2) циклів тактової частоти.

«Парність-непарність». В осередки з адресою, що містить парне число одиниць (до них відноситься й осередок з нульовою адресою), записують одиниці, а в інші осередки — нулі. Потім роблять послідовне зчитування й ідентифікацію умісту всіх осередків, починаючи з першої. Тест повторюють для комплементарного тла, коли з осередку з парною кількістю одиниць в адресі записують нулі, а в інші — одиниці. Будучи одним з найбільш коротких мінімальних функціональних тестів, цей тест дозволяє проконтролювати правильність функціонування адресного дешифратора. Його довжина складає 4 N циклів.

Алгоритм такої тестової перевірки має вигляд, як показано на рисунку 2.11.

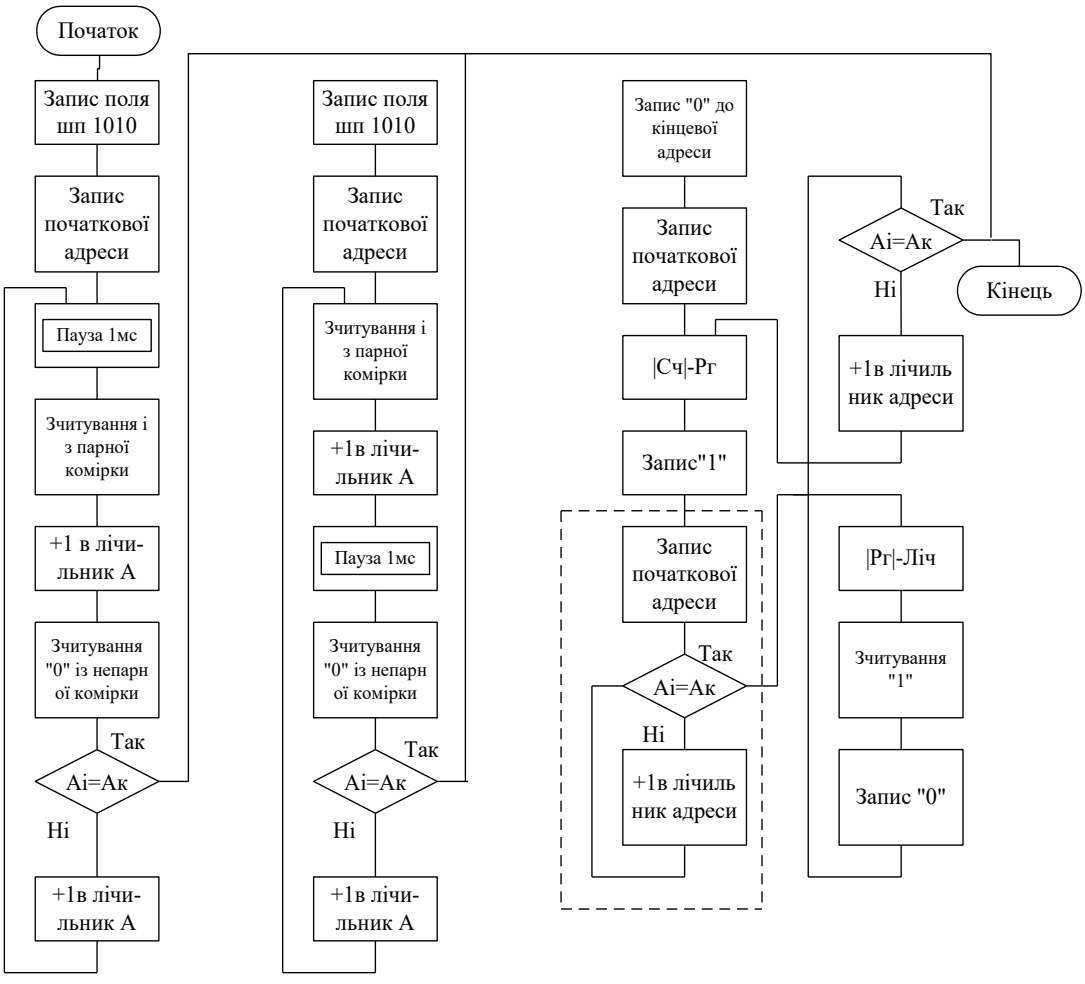


Рисунок 2.11 – Схема алгоритму тестової перевірки «Парність-непарність»

Можна модифікувати цей алгоритм на основі перетворень Калмана. Модифікований алгоритм такої тестової перевірки має вигляд, як показано на рисунку 2.12.

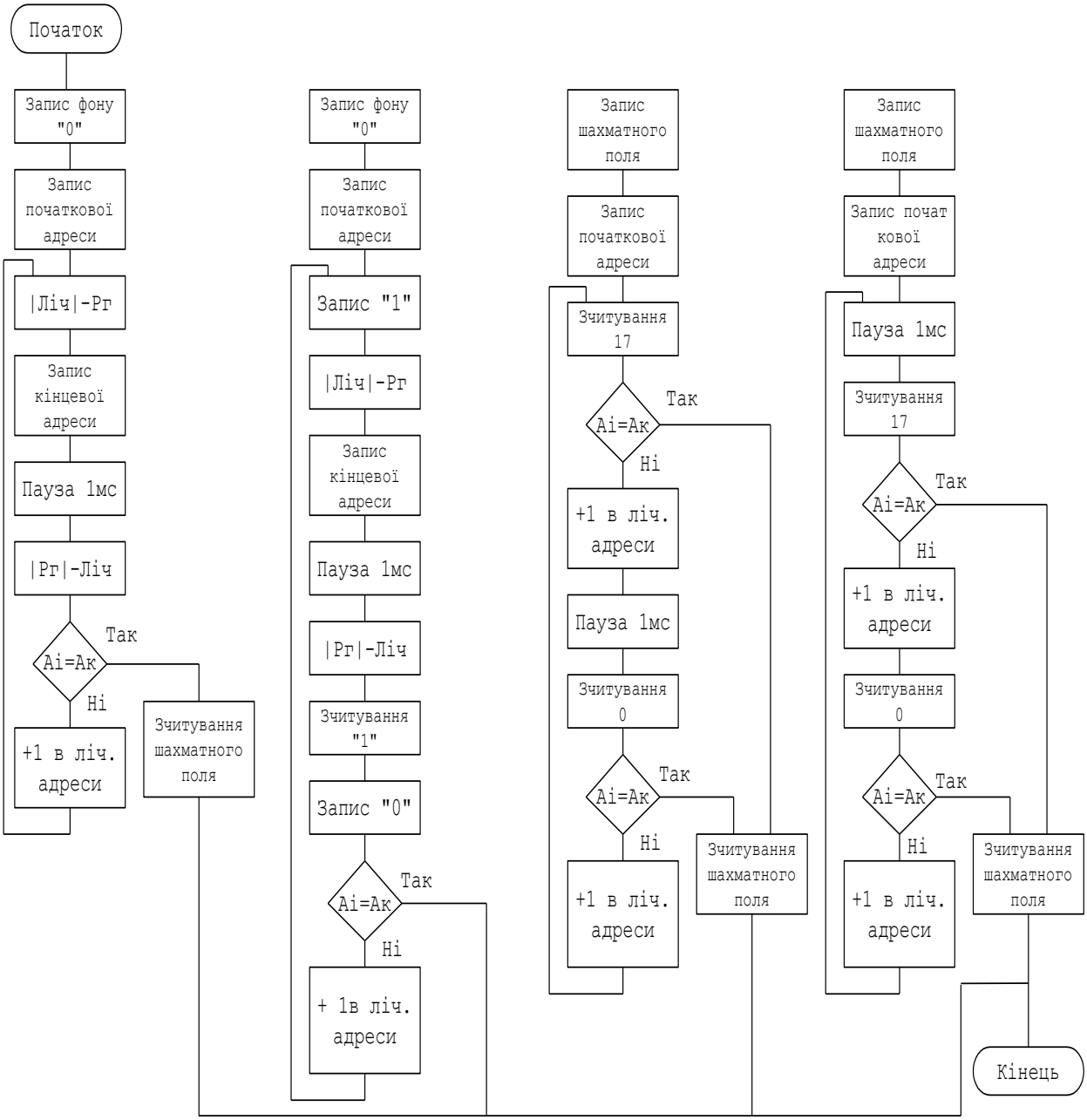


Рисунок 2.12 – Схема модифікованого алгоритму тестової перевірки

«Прогулянка одиниць і нулів». Створюють «фон нулів»; у перший осередок (нульову адресу) записують одиницю; потім всі інші осередки зчитують, перевіряючи схоронність нулів, а в першому осередку перевіряють схоронність записаної одиниці, після чого в неї знову записують нуль. Цю процедуру повторюють для кожної

наступної комірки пам'яті, а потім проводять аналогічний тест для комплементарних даних. Тривалість тесту «Прогулянка одиниць і нулів» дорівнює 2N2 + 6N циклів. Цей тест найбільше часто застосовується на практиці і є дуже ефективним при контролі ВІС ОЗП. Тут кожна комірка пам'яті перевіряється на запис і зчитування одиниць і нулів, на схоронність інформації при записі в кожний з осередків нових даних; крім того, перевіряється правильність функціонування схем адресації. Важливою перевагою тесту «Прогулянка» є перевірка підсилювачів, що зчитують, на здатність швидкого відновлення після довгого потоку нулів (чи одиниць), за яким випливає одиниця (чи один куль). Однак цей тест не дозволяє виявити схеми пам'яті з великим часом вибірки, з тенденцією до багаторазової вибірки; через рідку зміну інформації при зчитуванні цей тест має обмежені можливості виявлення впливу перехідних процесів на роботу ВІС ОЗП.

«Галоп одиниць і нулів». Створюють «фон нулів»; у перший осередок записують одиницю, перевіряють схоронність нуля в другому осередку, перевіряють схоронність одиниці в першому осередку, перевіряють схоронність нуля в третьому осередку й одиниці в першому осередку і т.д. доти, поки не буде перевірена схоронність нуля в останньому осередку й одиниці в першій. Потім у перший осередок записують нуль і переходять до аналогічної процедури щодо другої і кожен наступної осередків. Далі проводять такий же тест, але з комплементарними даними. Довжина цього тесту складає 4N2 + 2N циклів. Додатково до характеристик тесту «Прогулянка» тест «Галоп» перевіряє всі можливі адресні переключення і всі можливі переключення даних при зчитуванні.

«Пінг-понг». Заповнюють осередку «фон нулів»; у перший осередок описують одиницю, перевіряють схоронність нуля в осередку з новою адресою, що одержують інвертуванням одного адресного біта, підтверджують схоронність одиниці в першому осередку; провернеться схоронність нуля в осередку з проінвертованим тільки другим бітом і т.д. доти, поки не будуть проінвертована вага біта адрес. Оскільки кількість біт в адресі ЗУ обсягом N складає log N, повне число циклів, займаних тестом «Пінг-понг», складає 2N\og2N+ + 2N. Цей тест представляє деякий різновид «Галопу» И ефективно перевіряє роботу адресного декодера.

«Галоп запису — зчитування». Створення фону не обов'язково; у попередню комірку записують одиницю; у другий осередок записують нуль і перевіряють схоронність одиниці в першому осередку; у другий осередок записують одиницю і перевіряють схоронність одиниці в першій. Зазначені операції виконують для всіх осередків, що залишилися, при цьому в першому осередку зберігається одиниця, а на закінчення в перший осередок записують нуль. Далі приведений алгоритм повторюють для другий, третій і всіх інших осередків. З урахуванням комплементарного тесту довжина тесту «Галоп запису — зчитування» складає 8 N2— 4 N циклів. На відміну від простого «Галопу одиниць і нулів» тест «Галоп запису-зчитування» дозволяє знайти вплив циклів запису і зчитування одиниць і нулем у кожну з пар осередків.

«Галоп стовпців». У комірки пам'яті записують «фон нулів» у перший осередок поміщають одиницю. Потім здійснюють «галоп стовпців» зчитування уздовж рядка, тобто «перескоки» з одного стовпця матриці пам'яті на іншій. Після кожного послідовного зчитування нулів з осередків першого рядка (вони мають номери ПРО, √N, 2√N, 3√N,..., (√N— 1) √N)роблять зчитування одиниці з першого осередку. Наприкінці цієї частини тесту в перший осередок знову записують нуль. Потім у порядку зростання адрес зчитують уміст двох наступних осередків. В осередок з наступним адресою записують одиницю і повторюють операцію «Галопу стовпців» уздовж цього нового рядка, у яку записана одиниця. Зазначену процедуру виконують для кожної комірки пам'яті, а потім весь тест повторюють для комплементарних даних. Трохи незвичайний перехід на наступні рядки матриці пам'яті (зчитування нуля з осередків двох рядків і запис одиниці в осередок наступної рядка для пам'яті з розташуванням осередків 64Х64) розуміється необхідністю відновлення (регенерації) вмісту осередків. Час проходу тестового слова (наприклад, одиниці) по всьому стовпці, тобто послідовної адресації його до кожного рядка, складає (3√N+5)21Тц, , де Тц— час циклу. Для ВІС ОЗП ємністю 64К, що працює з тактовим циклом Гц =470 не, цей час дорівнює 1,9 мс, що задовольняє вимогам відновлення інформації, оскільки кожен рядок повинний зчитуватися кожні 2 мс. Регулюючи час циклу, можна змінювати інтервал між кожним відновленням умісту комірок пам'яті.

Приведений тест здійснює перевірку впливу переключень адрес на стан осередків, перевірку шумового зв'язку між всіма осередками стовпця. Тривалість тесту «Галоп стовпців» 6N3/2 +12 N циклів.

«Рух по діагоналі». В осередки записують «фон нулів»; у перший осередок записують одиницю. Починаючи з осередку з номером √N , роблять зчитування вмісту осередків уздовж їхньої щирої діагоналі (для цього до номера адреси попередньої осередку, що зчитується, додають цифру (√N—1). Потім зчитують тестове слово (у даному випадку — це одиниця в першому осередку) і в цей осередок записують нуль. Після цю адресу збільшують на одиницю. Тестове слово (одиниця) записується в усі осередки уздовж діагоналі, сформованою рядком і стовпцем, що граничать з цим осередком (для ВІС ОЗП 4К с матрицею 64х64 це означає запис одиниць в осередки з номерами 1 і 64). Далі роблять зчитування вмісту осередків по зрушеній діагоналі. Процедуру повторюють доти, поки тестове слово не буде записано в усі діагоналі матриці з позитивним нахилом. На закінчення весь описаний тест повторюють для комплементарних даних.

На практиці тест «Рух по діагоналі» використовується для виявлення випадків руйнування записаної інформації через внутрішню багаторазову вибірку чи зв'язку несправних підсилювачів зчитування. Під час виконання тесту перевіряються всі дешифратори адрес рядків і стовпців, що замикають адресні схеми, що зчитують підсилювачі. Відновлення даних у цьому тесті не представляє складності, оскільки зчитування осередків Уздовж щирої і зрушеної діагоналей забезпечує доступ до будь-якого рядка протягом заданого інтервалу часу. Крім того, відновлення даних відбувається і при зчитуванні діагоналей з тестовими словами. Тривалість тесту складає 4N3/2 +8N+10 1/2 циклів.

«Прогулянка рядків», Створюють «фон нулів»; записують одиниці в перший рядок; зчитують уміст для всіх комірок пам'яті. Інвертують одиниці на нулі. Повторюють запис одиниць, зчитування пам'яті й інвертування для другого і наступного рядків. Потім проводять тест для комплементарних даних. Тривалість тесті «Прогулянка рядків» складає 2N3/2 + 6N циклів.

«Прогулянка стовпців». Створюють «фон нулів»; записують одиниці в перший стовпець; зчитують уміст комірок пам'яті. Інвертують одиниці на нулі. Повторюють запис одиниць, зчитування й інвертування для другого і наступного стовпців. Потім проводять тест для комплементарних даних. Число циклів тесту 2N3/2 + 6N.

«Зчитування — інвертування — запис». Записують нулі в усі осередки; зчитують уміст всіх осередків; записують у них одиниці; зчитують дані з всіх осередків; інвертують їх, ідентифікують нулі у всіх осередках. Цей мінімальний тест має тривалість 6N циклів.

«Відновлення одиниць». Записують одиниці в усі осередки, вичікують 2 мс, потім зчитують уміст комірок пам'яті, знову очікують 2 мс і знову повторюють зчитування з всіх осередків. Тривалість тесту 3N циклів +4 мс.

«Руйнування рядків». Створюють тло нулів; записують одиниці й одну з рядків матриці. Через 2 мс зчитують уміст рядків, розташованих вище і нижче рядка з випробуваним словом. Зчитують уміст випробуваного рядка і записують у неї нулі. Далі повторюють тест для всіх рядків матриці ЗУ, а потім проводять аналогічний тест із комплементарним тестовим словом. Тривалість тесту: 10N циклів + 4 мс. Цей тест стає найбільш розповсюдженим для динамічних ВІС ОЗП великої ємності.

«Множинна вибірка адреси». Записують тло у виді одиниць, що чергуються, і нулів; зчитують уміст першого й останнього осередків; перевіряють уміст першого осередку. Дають збільшення адреси на одну одиницю, зчитують вміст осередку по цій адресі і за адресою N-1 (комплементарному стосовно попереднього); зчитують уміст попередньої осередку. Повторюють тест для наступних осередків, що зрушується послідовно (по номерах адрес) назустріч один одному. Потім зчитують уміст всіх осередків. Повторюють тест для комплементарного запису. Тривалість тесту складає 7N циклів.

«Шахи». Записують в осередки ЗП шаховий малюнок; зчитують його. Записують інверсний (комплементарний) шаховий малюнок і знову зчитують його; тривалість тесту 4N циклів. Цей Тест часто поширюють на контроль часу відновлення вмісту осередків, для чого перед зчитуванням роблять витримку тривалістю 2 мс.

У зв'язку з безупинним удосконаленням старих тестових послідовностей і створенням нових починаються спроби формалізації процесу розробки тестів для ВІС ОЗП. Це важливий крок у забезпеченні високої якості та надійності сучасних інтегральних схем. Формалізація процесу розробки тестів дозволяє систематизувати підхід до тестування, зменшити можливість помилок, а також підвищити ефективність і точність виявлення дефектів. Удосконалення старих тестових послідовностей є необхідним через постійний розвиток технологій і зміну характеристик компонентів. Нові технології виробництва інтегральних схем призводять до появи нових типів дефектів, які потребують виявлення. Тому оновлення існуючих тестів дозволяє вчасно виявляти такі дефекти та забезпечувати якість продукції.

Порівняльна характеристика описаних іспитових послідовностей по тривалості іспитів для ВІС ОЗП ємністю 4Кб і 16Кб. приведена в табл. 2.1 Варто підкреслити тенденцію повернення до тестів типу N для контролю ВІС ОЗП підвищеної ємності, наприклад 16Кб. У цих випадках, як правило, використовується не один, а комбінація різних кодових малюнків, кожний з який орієнтований на виявлення визначених видів відмовлень.

Для того, щоб тестування ВІС ОЗП здійснювалося ефективно як з погляду складності іспитів, у процесі іспитів необхідно виявити два фактори: параметричну чутливість випробуваного пристрою до виду іспитової кодової послідовності (кодовому малюнку). Перший фактор відноситься до таким 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 малюнків, подаваним у виді визначених імпульсних впливів на висновки випробуваного пристрою. Оскільки чутливість до виду іспитового кодового малюнка для ВІС ОЗП існує, виготовлювач зобов'язаний при записі в паспорт основних параметрів ВІС ОЗП, таких, наприклад, як час вибірки, домовлятися про умови іспитів своїх пристроїв, при яких ці параметри отримані.

Таблиця 2.1 - Дані по числу циклів і часу тестування ВІС ОЗП

Тип випробовувальної послідовності(кодового малюнка)	Число циклів для квадратної матриці	ВІС ОЗП з ємністю 4К		ВІС ОЗП з ємністю 16К	
		Число циклів	Час проходження одного цілого тесту (для циклу 450нс),мс	Число циклів	Час проходження одного цілого тесту (для циклу 450нс),мс
“Відновлення одиниць”	3N	12288	5.5+4	49152	22
“Шахмати”	4N	16384	7.4	65536	29
“Шахмати з витримкою”	4N	16384	7.4+4	65536	29
“Парність-непарність”	4N	16384	7.4	65536	29
“Зчитування-інвертування-запис”	6N	24576	11	98304	44
“Багаторазовий вибір адреси”	7N	28672	13	114688	52
“Марш одиниць і нулів”	10N	40960	18	163840	74
“Пошкодження срічок”	10N	40960	18+256	163840	74+512
“Хрест комірок”	12N	49152	22	196608	88
“Пінг-понг”	2Nlg2N+2N	106496	50	491520	221
“Зсувна діагональ”	$2N^{3/2}$	524288	236	4194304	1,9 с
“Прогулянка стрічок”	$2N^{3/2}+6N$	548864	247	4292608	1,9 с
“Прогулянка стовбчиків”	$2N^{3/2}+6N$	548864	247	4292608	1,9 с
“Хрест стрічок і стовбчиків”	$4N^{3/2}$	1048576	472	8388608	3,8 с
“Зсув по діагоналі”	$4N^{3/2}+8N+10N^{1/2}$	1081984	487	8520960	3,8 с
“Галоп стовбців”	$6N^{3/2}+12N$	1622016	730	12779520	5,6 с
“Прогулянка одиниць і нулів”	$2N^2+6N$	33579008	15с	536979204	4 хв
“Галоп одиниць і нулів”	$4N^2+2N$	67117056	30с	1 млрд	8 хв
“Галоп запису-зчитування”	$8N^2-4N$	134200334	1 хв	2 млрд	16 хв

Створення нових тестових послідовностей є невід'ємною частиною розвитку галузі діагностики вмісту комірок оперативної пам’яті ПК. Нові тести дозволяють

виявляти дефекти, які раніше були непомітні або важко виявлялися. Це, в свою чергу, сприяє підвищенню надійності та довговічності інтегральних схем.

Формалізація процесу розробки тестів включає декілька ключових етапів. Спочатку проводиться аналіз вимог до тестування, який визначає, які характеристики і параметри повинні бути перевірені. Далі розробляються тестові послідовності, які забезпечують перевірку всіх необхідних параметрів. Після цього проводиться моделювання та верифікація тестів, що дозволяє перевірити їх ефективність на етапі розробки. І, нарешті, тести впроваджуються в процес виробництва, де вони використовуються для перевірки готової продукції.

Таким чином, формалізація процесу розробки тестів для ВІС ОЗП є важливим кроком для забезпечення високої якості та надійності сучасних інтегральних схем. Вона дозволяє систематизувати процес тестування, підвищити його ефективність і точність, а також забезпечити своєчасне виявлення та усунення дефектів.

2.5 Висновок

В даному розділі проаналізовано основні способи діагностики несправності ПК. Проаналізовано основні несправності цифрових схем комірок пам’яті ПК. Розроблено алгоритми діагностики несправності комірок пам’яті ПК.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ДІАГНОСТИКИ НЕСПРАВНОСТІ ПК

3.1 Обґрунтування вибору мови програмування та середовища розробки

Для розробки програмного модуля діагностики несправності оперативної пам’яті ПК потрібно обрати потужне та зручне середовище програмування, яке б підтримувало всі основні вимоги об’єктно-орієнтованого напряму. До таких середовищ можна віднести NetBeans що підтримує мову програмування PHP, а також таким середовищем є Microsoft VisualStudio 2008, яка підтримує мову програмування C, C++ C#.

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи NET. Розроблена АндерсомГейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (при фірмі Microsoft) [12].

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників – мов C++, Delphi, Модула і Smalltalk – C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++) [13].

Мова програмування Java є простою і потужною мовою. Головною перевагою Java являються вбудовані класи-абстракції, що зробили його мовою, яка дійсно не залежить від платформи. Проте дане середовище має суттєві недоліки, до яких слід віднести: відсутність вбудованого відладчика, важкість компілювання exe програм, що не залежать від віртуальної Java-машини, а також низька швидкодія при роботі з великими масивами даних, або з складними обчисленнями [13].

NetBeans IDE – вільне інтегроване середовище розробки (IDE) для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, Python, Groovy. Середовище може бути встановлене і для підтримки окремих мов у повній конфігурації.

Середовище розробки NetBeans за умовчанням підтримує розробку для платформ J2SE і J2EE [15].

Мова С — універсальна, процедурна, імперативна мова програмування загального призначення, розроблена у 1972 році Денісом Рітчі у Bell Telephone Laboratories з метою написання нею операційної системи UNIX.

Хоча С і було розроблено для написання системного програмного забезпечення, наразі вона досить часто використовується для написання прикладного програмного забезпечення.

С імовірно, є найпопулярнішою у світі мовою програмування за кількістю вже написаного нею програмного забезпечення, доступного під вільними ліцензіями коду та кількості програмістів, котрі її знають. Версії компіляторів для мови С існують для багатьох операційних систем та апаратних архітектур. С здійснила великий вплив на інші мови програмування, особливо на C++, яка спочатку проектувалася, як розширення для С, а також на Java та C#, які запозичили у С синтаксис.

Наведемо основні переваги даної мови програмування:

- зручний та швидкий інтерфейс користувача;
- наслідування класів та інтерфейсів;
- перевантаження методів та операторів.

Інкапсуляція даних за допомогою властивостей із розмежованими модифікаторами доступу, різними областями видимості (public, protected, private, internal), класами із частковою реалізацією (partialclasses), тощо.

Узагальнені типи (generictypes), які дозволяють реалізувати логіку роботи класу для декількох типів даних.

Таким чином проаналізувавши переваги і недоліки мов об’єктно-орієнтованого програмування ми можемо зробити висновок що мова програмування С є більш зручнішою для нашої розробки. Присутність таких потужних можливостей і відсутність серйозних недоліків і є основною причиною вибору середовища Microsoft Visual Studio для проектування.

3.2 Створення програмного модуля діагностики несправності оперативної пам’яті ПК

Перед розробкою графічного інтерфейсу користувача визначимо базовий функціонал модулю тестування оперативної пам’яті комп’ютерів, який включає:

- вибір методу тестування;

- кнопкову форму для початку тестування;

- виведення результату тестування.

В програмі передбачена можливість введення послідовності нулів та одиниць, яка буде обрана за початковий паттерн для тестування оперативної пам’яті комп’ютера. Це здійснюється у процедурі __fastcall;

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    //Application->MessageBox("Ты ВАЛИК?", NULL, MB_OKCANCEL);

int i,j,k,q,result;
int matrix[32][32];
int matrix_[32][32];
int mas [32];

    if (RadioButton1->Checked==True)
    {
        void *p1=malloc(1024);
        result=heapfillfree(0xAAAAAAAA);
        result=heapcheckfree(0xAAAAAAAA);
        result=heapcheckfree(0);
        void *p2=malloc(1024);
        result=heapfillfree(0x55555555);
        result=heapcheckfree(0x55555555);
        result=heapcheckfree(0);
        void *p3=malloc(1024);
        result=heapfillfree(0xAAAAAAAA);
        result=heapcheckfree(0xAAAAAAAA);
        result=heapcheckfree(0);
        void *p4=malloc(1024);
        result=heapfillfree(0x55555555);
        result=heapcheckfree(0x55555555);
        result=heapcheckfree(0);

        void *p5=malloc(1024);
        result=heapfillfree(0xAAAAAAAA);
        result=heapcheckfree(0xAAAAAAAA);
        result=heapcheckfree(0);
        free(p1);
        free(p2);
```

```
            free(p3);
            free(p4);
            free(p5);

            switch (result)
            {
                  case _HEAPEMPTY:
                  Form1->Edit1->Text="Оперативна пам'ять зайнята";
                  break;
            case _HEAPOK:
                  Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
                  break;
            case _HEAPCORRUPT:
                  Form1->Edit1->Text="Збої в оперативній пам'яті";
                  break;
            case _BADVALUE:
                  Form1->Edit1->Text="Невірне значення в оперативній пам'яті";
                  break;
            }
//Report("Після перевірки вільних блоків на значення 0",result);
            free(p1);
            free(p3);
            free(p5);
      }

      if (RadioButton5->Checked==True)
      {
            for (i=0;i<32;i++)
                  for (j=0;j<32;j++)
                        matrix[i][j]=(i+j)%2;

            for (i=0;i<32;i++)
            {
                  mas[i]=(int)((random(399))/200);
            }

            for (i=0;i<32;i++)
                  for (j=0;j<32;j++)
                  {
                        k=matrix[i][j];
                        for(q=0;q<32;q++)
                        {
                              k=k*mas[q];
                        }
                        matrix_[i][j]=matrix[i][j]*k;
                  }

            switch (result)
            {
                  case _HEAPEMPTY:
                  Form1->Edit1->Text="Оперативна пам'ять зайнята";
                  break;
```

```
        case _HEAPOK:
            Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
            break;
        case _HEAPCORRUPT:
            Form1->Edit1->Text="Збої в оперативній пам'яті";
            break;
        case _BADVALUE:
            Form1->Edit1->Text="Невірне значення в оперативній пам'яті";
            break;
        }

        for (i=0;i<32;i++)
            for (j=0;j<32;j++)
            {
                k=matrix[i][j];
                for(q=0;q<32;q++)
                {
                    k=k*mas[q];
                }
                if (matrix_[i][j]==matrix[i][j]*k)
                {
                    Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
                }
                else
                {
                    Form1->Edit1->Text="Збої в оперативній пам'яті";
                };
            }
    }
}
```

Програмне забезпечення має одну форму, яка відкривається після запуску програми (рис. 3.1). Основна форма інтерфейсу користувача пропонує вибір, яким саме тестом необхідно виконати тестування оперативної пам’яті. Перед проведенням тестування необхідно вибрати тест.

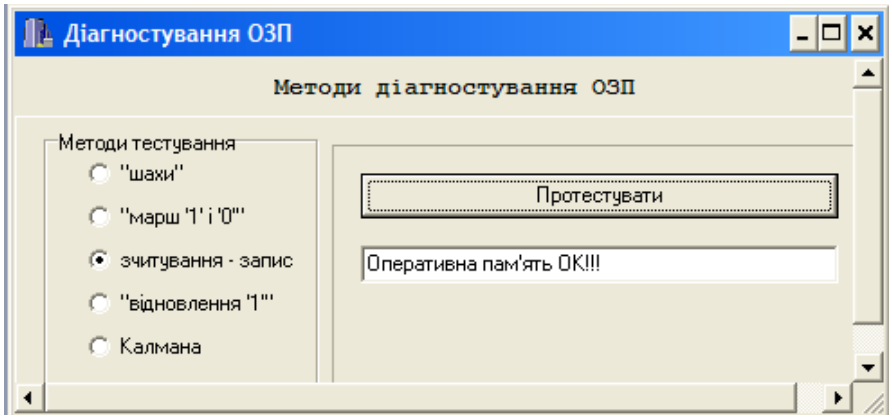


Рисунок 3.1 – Основне вікно головної програми

У розділі «методи тестування» можна задати такі параметри:

- тестування методом «шахи»;
- тестування методом «марш 1 і 0»;
- тестування методом «зчитування запис»;
- тестування методом «відновлення 1»;
- тестування з використанням перетворень Калмана.

Після вибору методу тестування, необхідно натиснути клавішу «Протестувати», попередньо у вікно введення можна записати послідовність нулів та одиниць, яка буде обрана за початковий паттерн для тестування оперативної пам’яті комп’ютера.

3.3 Тестування розробленого програмного забезпечення та аналіз результатів роботи

Для перевірки працездатності розробленого програмного забезпечення був проведений дослід, а саме: сформована початкова послідовність нулів та одиниць, яка вважається паттерном при проведенні тестування пам’яті (рис. 3.2).

Така послідовність має вигляд: 10101011101010101110101 01

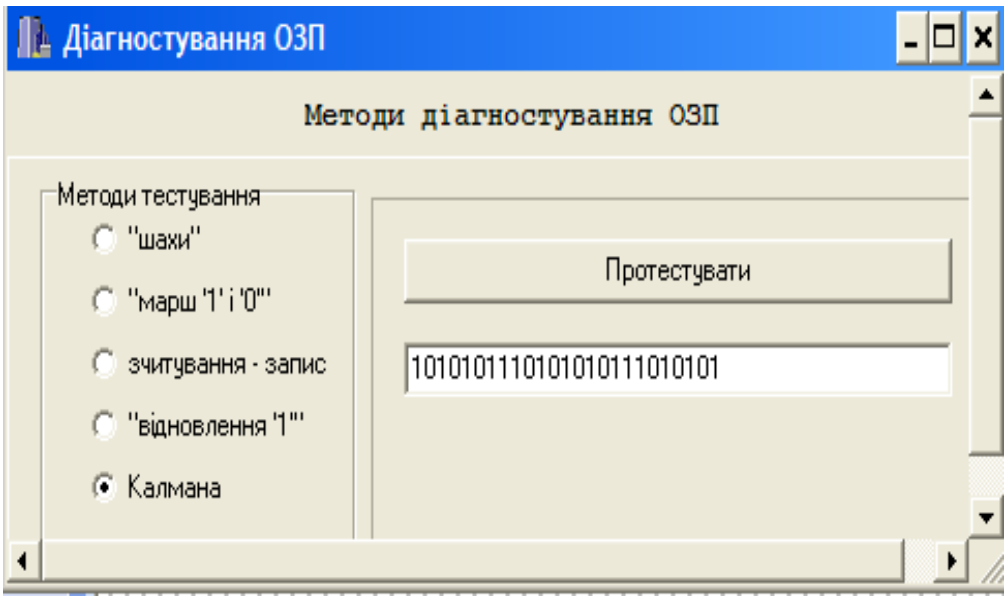


Рисунок 3.2 – Введення головного паттерну

Якщо у процесі тестування не виявлено помилок пам’яті, то буде видано повідомлення, як показано на рисунку 3.3.

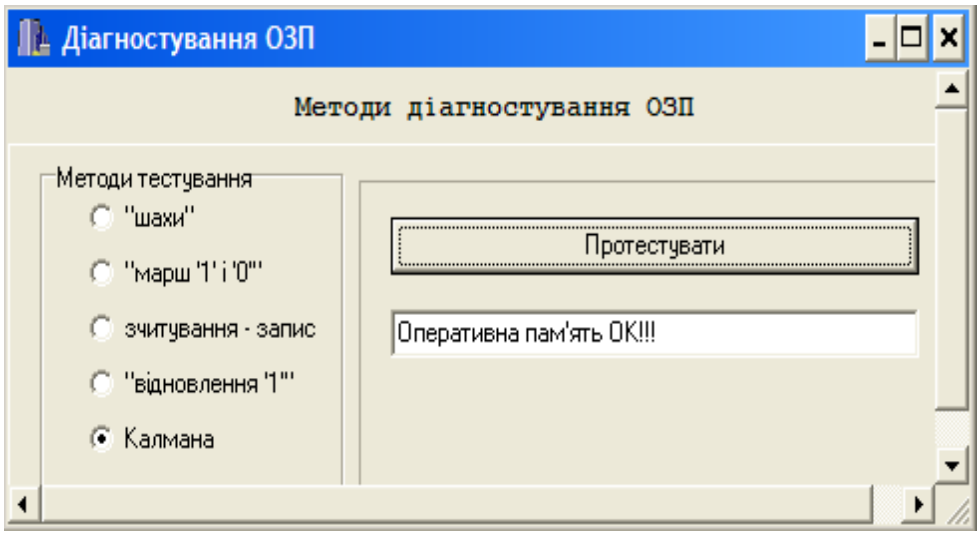


Рисунок 3.3 – Вдале завершення тесту

Якщо у процесі тестування виявлено помилки пам’яті, то буде видано повідомлення, як показано на рисунку 3.4.

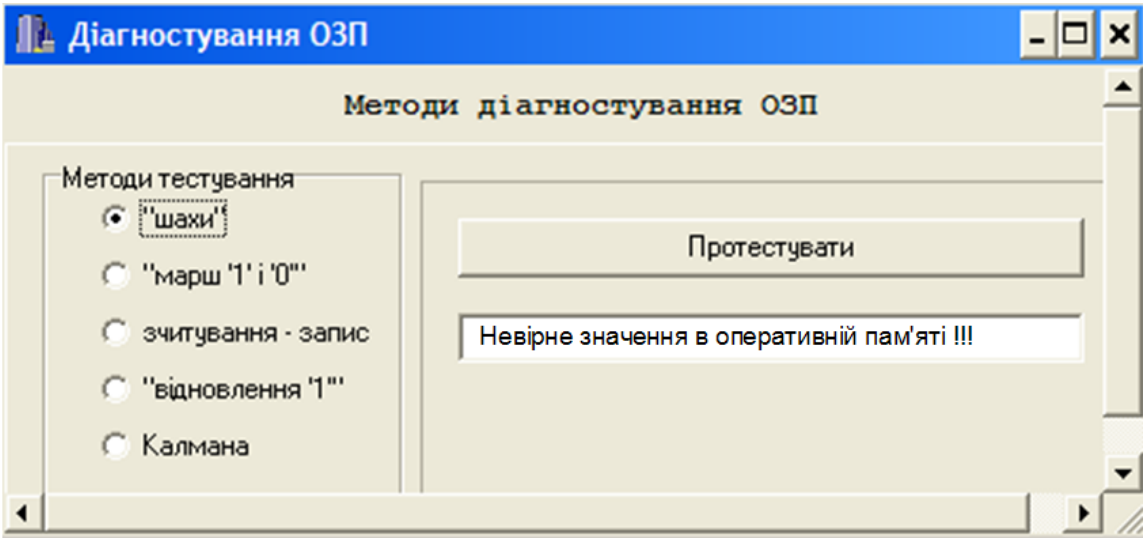


Рисунок 3.4 – Оперативна пам’ять має помилкові біти

Розроблений додаток має функціонал, наведений в таблиці 3.1, який було порівняно із функціоналом найпопулярніших аналогів для діагностики оперативної пам’яті ПК.

Таблиця 3.1 – Порівняльна характеристика розробленого ПЗ

Назва програми	Зчитування-запис	Алгоритм Калмана	Марш одиниць і нулів	Алгоритм шахової перевірки
AIDA64	+	-	-	-
Memtest+	+	+	-	-
Розроблений модуль	+	+	+	+

Отже, розроблений модуль діагностики несправності оперативної пам’яті ПК було протестовано на відповідність меті дослідження. Функціонал програмного забезпечення є розширеним по відношенню до існуючих сучасних рішень та відповідає завданню.

3.4 Висновок

В даному розділі описано процес створення програмного модуля діагностики несправності оперативної пам’яті ПК. Проаналізовано переваги і недоліки мов об’єктно-орієнтованого програмування та обрано мову програмування C++, яка є найбільш зручнішою для нашої розробки. Проведено тестування роботи програмного забезпечення та підтверджено його працездатність практичними результатами. Доведено, що розроблений програмний модуль має більший функціонал, по зрівняню з іншими програмами аналогами.

ВИСНОВКИ

Під час виконання бакалаврської дипломної роботи було розроблено програмний модуль діагностики несправності оперативної пам’яті ПК.

В роботі проведено аналіз основних несправностей ПК та методів їх виявлення. Проаналізовані програми аналоги для діагностики несправності ПК, виявлені їх переваги та недоліки. Проаналізовано основні несправності цифрових схем комірок пам’яті ПК. Розроблено алгоритми діагностики несправності комірок пам’яті ПК.

Здійснено програмну реалізацію модуля діагностики несправності ПК. Проведено тестування роботи програмного забезпечення та підтверджено його працездатність практичними результатами. Доведено, що розроблений програмний модуль має більший функціонал, по зрівняню з іншими програмами аналогами.

Отже всі поставлені задачі виконано, мету роботи досягнуто.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Іванов О.В. Історія створення комп'ютерів/ Іванов О.В.: Комп'ютер Прес - 1996 - N5.
2. Програмні засоби контролю ЕОМ. − Режим доступу: http://rdmcn.ptngu.com/lekcyi/13-lekcya-8.html.
3. Лувішіс І. Коротко про історію розвитку комп'ютерів/ Лувішіс І., Зарубін Ю., Мазо Б.: Комп'ютер Прес - 1996 - N5.
4. Тхір І.Л., Калушка В.П., Юзьків А.В. Посібник користувача ПК. Третє видання. — Тернопіль: “Підручники та посібники”, 2006. -1024с.
5. 3DMark − Режим доступу: https://uk.wikipedia.org/wiki/3DMark
6. Мельник А.О. Архітектура комп’ютера . Наукове видання. – Луцьк: Волинська обласна друкарня, 2008. – 470 с.
7. Заміховський Л.М., Калявін В.П. Основи теорії надійності і технічної діагностики систем: Навчальний посібник.–Івано-Франківськ: Вид-во “Полум’я”, 2009.– 360 с.
8. Шимон О. М. Лекція №1. Апаратне та програмне забезпечення ПК.: Конспект лекцій; Житомирський держ. унів. ім. І. Франка. – Житомир, 2006 – 17с.:Режим доступу: http://eprints.zu.edu.ua/18/1/Konspect_modul_1_Windows.pdf
9. Dynamic-link libraries [Electronic resource] – 2013. Access mode: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx. – Title from the screen.
10. How to change the color of progressbar in C# .NET 3.5? Режим доступу: https://stackoverflow.com/questions/778678/how-to-change-the-color-of-progressbar-in-c-sharp-net-3-5.
11. Unified Modeling Language – Вікіпедія: веб-сайт. - Режим доступу: URL: https://uk.wikipedia.org/wiki/ Unified_Modeling_Language.
12. Microsoft Visual Studio – Вікіпедія: веб-сайт. Режим доступу: URL: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio
13. Ситник В. Ф., Писаревська Т. А., Єрьоміна Н. В., Краєва О. С. Основи

інформаційних систем: Навчальний посібник Київ, 2005. 308 с.

14. Josuttis, Nicolai M. The C++ standard library : a tutorial and reference / Nicolai M. Josuttis.—2nd ed. с. 78-80.
15. Свароп С. Г. A Bite Of Python v2.01. 2013. 159 с. с. 20-23. PyCharm [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/pycharm/
16. Chollet Francois Deep Learning with Python. 2018. — 400 с.
17. Miguel Grinberg Flask Web Development. Published by O'Reilly Media, Inc. 2018.
18. C++ 17 STL. Стандартна бібліотека шаблонів. Яцек Галовіц. 2018. ISBN: 978-5-4461-0680-6
19. Arduino Pro Mini [Електронний ресурс] // doc.arduino.ua – 2021. – Режим доступу до ресурсу: https://doc.arduino.ua/ru/hardware/ProMini.
20. Arduino Nano [Електронний ресурс] // arduino.ua – 2021. – Режим доступу до ресурсу: https://arduino.ua/prod166-arduino-nano-v3-0-avratmega 328p-s-raspayannimi-razemami.
21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТОК А
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль діагностики несправності ПК

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 87,6% Схожість 12,4%

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 виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

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

Автор роботи Шимонюк В.О.

Керівник роботи Колодний В.В.

ДОДАТОК Б

ІНСТРУКЦІЯ КОРИСТУВАЧА

Для роботи з програмним модулем потрібно сформувати початкову послідовність нулів та одиниць, яка вважається паттерном при проведенні тестування пам’яті (рис. Б.1).

Така послідовність має вигляд: 10101011101010101111010101

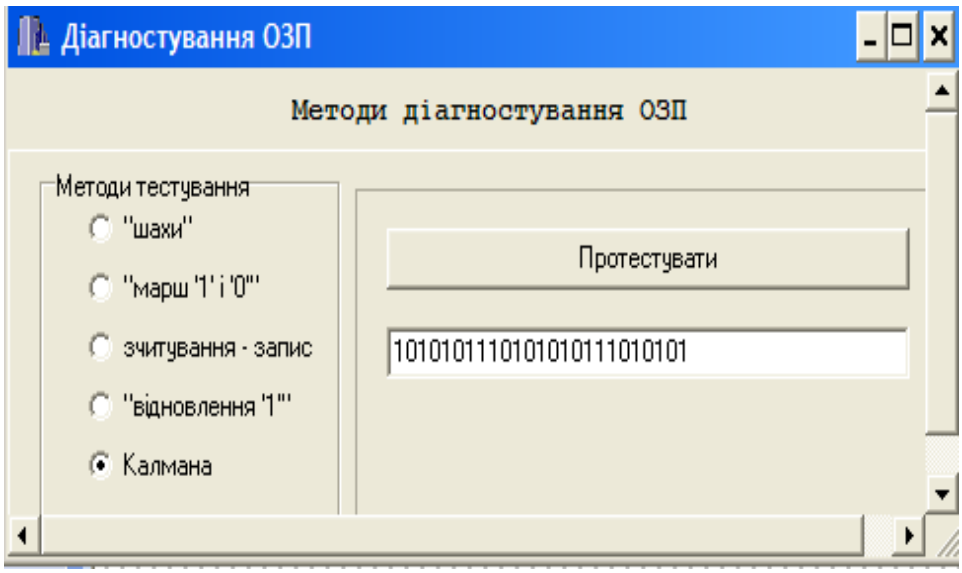


Рисунок Б.1 – Введення головного паттерну

Якщо у процесі тестування не виявлено помилок пам’яті, то буде видано повідомлення, як показано на рисунку Б.2.

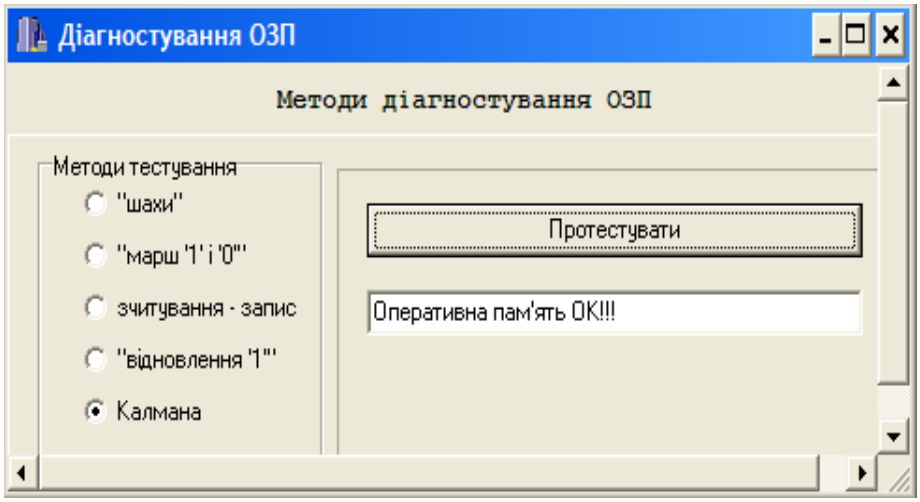


Рисунок Б.2 – Вдале завершення тесту

Якщо у процесі тестування виявлено помилки пам’яті, то буде видано повідомлення, як показано на рисунку Б.3.

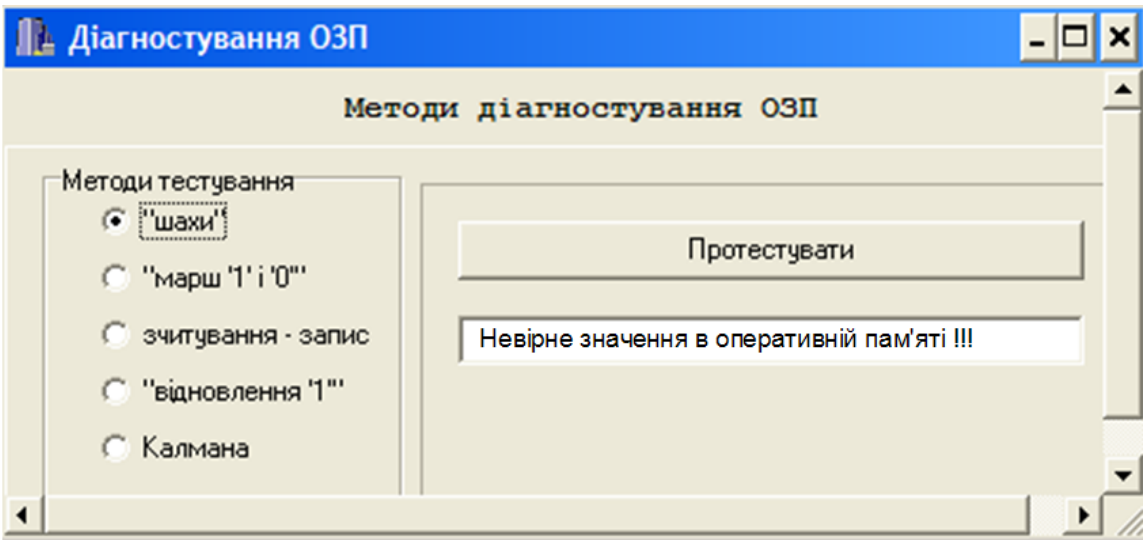


Рисунок Б.3 – Оперативна пам’ять має помилкові біти

ДОДАТОК В
ЛІСТИНГ ПРОГРАМИ

```
#include <vcl.h>
#pragma hdrstop
USERES("MemTest.res");
USEFORM("Unit1.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
	try
	{
		Application->Initialize();
		Application->CreateForm(__classid(TForm1), &Form1);
		Application->Run();
	}
	catch (Exception &exception)
	{
		Application->ShowException(&exception);
	}
	return 0;
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include <stdio.h>
//#include <process.h>
#include <alloc.h>


//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
	: TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
int i,j,k,q,result;
int matrix[32][32];
int matrix_[32][32];
int mas [32];

	if (RadioButton1->Checked==True)
	{
```

```
        void *p1=malloc(1024);
        result=heapfillfree(0xAAAAAAAA);
        result=heapcheckfree(0xAAAAAAAA);
        result=heapcheckfree(0);

        void *p2=malloc(1024);
        result=heapfillfree(0x55555555);
        result=heapcheckfree(0x55555555);
        result=heapcheckfree(0);

        void *p3=malloc(1024);
        result=heapfillfree(0xAAAAAAAA);
        result=heapcheckfree(0xAAAAAAAA);
        result=heapcheckfree(0);

        void *p4=malloc(1024);
        result=heapfillfree(0x55555555);
        result=heapcheckfree(0x55555555);
        result=heapcheckfree(0);

        void *p5=malloc(1024);
        result=heapfillfree(0xAAAAAAAA);
        result=heapcheckfree(0xAAAAAAAA);
        result=heapcheckfree(0);

        free(p1);
        free(p2);
        free(p3);
        free(p4);
        free(p5);

        switch (result)
        {
              case _HEAPEMPTY:

              Form1->Edit1->Text="Оперативна пам'ять зайнята";
              break;
        case _HEAPOK:

              Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
              break;
        case _HEAPCORRUPT:

              Form1->Edit1->Text="Збої в оперативній пам'яті";
              break;
        case _BADVALUE:

              Form1->Edit1->Text="Невірне значення в оперативній пам'яті";
              break;
        }
/*
        void *p1=malloc(1024);
        result=heapfillfree(0x55555555);
        result=heapcheckfree(0x55555555);
        result=heapcheckfree(0);

        void *p2=malloc(1024);
```

```
            result=heapfillfree(0xAAAAAAAA);
            result=heapcheckfree(0xAAAAAAAA);
            result=heapcheckfree(0);

            void *p3=malloc(1024);
            result=heapfillfree(0x55555555);
            result=heapcheckfree(0x55555555);
            result=heapcheckfree(0);

            void *p4=malloc(1024);
            result=heapfillfree(0xAAAAAAAA);
            result=heapcheckfree(0xAAAAAAAA);
            result=heapcheckfree(0);

            void *p5=malloc(1024);
            result=heapfillfree(0x55555555);
            result=heapcheckfree(0x55555555);
            result=heapcheckfree(0);

            free(p1);
            free(p2);
            free(p3);
            free(p4);
            free(p5);
*/
        }

        if (RadioButton2->Checked==True)
        {
            void *p1=malloc(1024);
//            void *p2=malloc(1024);
            void *p3=malloc(1024);
//            void *p4=malloc(1024);
            void *p5=malloc(1024);
//            free(p2);
//            free(p4);
            result=heapfillfree(0x00000000);
            result=heapfillfree(0xFFFFFFFF);
            result=heapcheckfree(0xFFFFFFFF);
            result=heapfillfree(0xFFFFFFFF);
            result=heapfillfree(0x00000000);
            result=heapcheckfree(0x00000000);
//            result=heapcheckfree(0);
            switch (result)
            {
                case _HEAPEMPTY:
                Form1->Edit1->Text="Оперативна пам'ять зайнята";
                break;
            case _HEAPOK:
                Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
                break;
            case _HEAPCORRUPT:
                Form1->Edit1->Text="Збої в оперативній пам'яті";
                break;
            case _BADVALUE:
                Form1->Edit1->Text="Невірне значення в оперативній пам'яті";
                break;
```

```
            }

//Report("Після перевірки вільних блоків на значення 0",result);
            free(p1);
            free(p3);
            free(p5);
      }
      if (RadioButton3->Checked==True)
      {
            void *p1=malloc(1024);
            void *p2=malloc(1024);
            void *p3=malloc(1024);
            void *p4=malloc(1024);
            void *p5=malloc(1024);

            free(p2);
            free(p4);
            result=heapfillfree(0x01234567);
            result=heapcheckfree(0x01234567);
            result=heapcheckfree(0);
            switch (result)
            {
                  case _HEAPEMPTY:
                  Form1->Edit1->Text="Оперативна пам'ять зайнята";
                  break;
            case _HEAPOK:
                  Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
                  break;
            case _HEAPCORRUPT:
                  Form1->Edit1->Text="Збої в оперативній пам'яті";
                  break;
            case _BADVALUE:
                  Form1->Edit1->Text="Невірне значення в оперативній пам'яті";
                  break;
            }

//Report("Після перевірки вільних блоків на значення 0",result);
            free(p1);
            free(p3);
            free(p5);
      }
      if (RadioButton4->Checked==True)
      {
            void *p1=malloc(1024);
            void *p2=malloc(1024);
            void *p3=malloc(1024);
            void *p4=malloc(1024);
            void *p5=malloc(1024);

            free(p2);
            free(p4);
            result=heapfillfree(0x00000000);
//          пауза
            result=heapcheckfree(0x00000000);
            result=heapcheckfree(0);
            switch (result)
            {
```

```
                case _HEAPEMPTY:
                Form1->Edit1->Text="Оперативна пам'ять зайнята";
                break;
            case _HEAPOK:
                Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
                break;
            case _HEAPCORRUPT:
                Form1->Edit1->Text="Збої в оперативній пам'яті";
                break;
            case _BADVALUE:
                Form1->Edit1->Text="Невірне значення в оперативній пам'яті";
                break;
            }

//Report("Після перевірки вільних блоків на значення 0",result);
            free(p1);
            free(p3);
            free(p5);
        }
        if (RadioButton5->Checked==True)
        {
            for (i=0;i<32;i++)
                for (j=0;j<32;j++)
                    matrix[i][j]=(i+j)%2;
            for (i=0;i<32;i++)
            {
                mas[i]=(int)((random(399))/200);
            }
            for (i=0;i<32;i++)
                for (j=0;j<32;j++)
                {
                    k=matrix[i][j];
                    for(q=0;q<32;q++)
                    {
                        k=k*mas[q];
                    }
                    matrix_[i][j]=matrix[i][j]*k;
                }
            void *p1=malloc(1024);
            result=heapfillfree(0xAAAAAAAA);
            result=heapcheckfree(0xAAAAAAAA);
            result=heapcheckfree(0);

            void *p2=malloc(1024);
            result=heapfillfree(0x55555555);
            result=heapcheckfree(0x55555555);
            result=heapcheckfree(0);

            void *p3=malloc(1024);
            result=heapfillfree(0xAAAAAAAA);
            result=heapcheckfree(0xAAAAAAAA);
            result=heapcheckfree(0);

            void *p4=malloc(1024);
            result=heapfillfree(0x55555555);
            result=heapcheckfree(0x55555555);
            result=heapcheckfree(0);
```

```
        void *p5=malloc(1024);
        result=heapfillfree(0xAAAAAAAA);
        result=heapcheckfree(0xAAAAAAAA);
        result=heapcheckfree(0);
        free(p1);
        free(p2);
        free(p3);
        free(p4);
        free(p5);
        switch (result)
        {
            case _HEAPEMPTY:
            Form1->Edit1->Text="Оперативна пам'ять зайнята";
            break;
        case _HEAPOK:
            Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
            break;
        case _HEAPCORRUPT:
            Form1->Edit1->Text="Збої в оперативній пам'яті";
            break;
        case _BADVALUE:
            Form1->Edit1->Text="Невірне значення в оперативній пам'яті";
            break;
        }
        for (i=0;i<32;i++)
            for (j=0;j<32;j++)
            {
                k=matrix[i][j];
                for(q=0;q<32;q++)
                {
                    k=k*mas[q];
                }
                if (matrix_[i][j]==matrix[i][j]*k)
                {
                    Form1->Edit1->Text="Оперативна пам'ять ОК!!! ";
                }
                else
                {
                    Form1->Edit1->Text="Збої в оперативній пам'яті";
                };
            }
    }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RadioButton2Click(TObject *Sender)
{
    Form1->Edit1->Text=" ... ";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RadioButton3Click(TObject *Sender)
{
    Form1->Edit1->Text=" ... ";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RadioButton4Click(TObject *Sender)
{
```

```
	Form1->Edit1->Text=" ... ";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RadioButton5Click(TObject *Sender)
{
	Form1->Edit1->Text=" ... ";
}
//---------------------------------------------------------------------------
void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
	Form1->Edit1->Text=" ... ";
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:	// IDE-managed Components
	TRadioGroup *RadioGroup1;
	TRadioButton *RadioButton1;
	TRadioButton *RadioButton2;
	TRadioButton *RadioButton3;
	TRadioButton *RadioButton4;
	TRadioButton *RadioButton5;
	TBevel *Bevel1;
	TBevel *Bevel2;
	TButton *Button1;
	TEdit *Edit1;
	TLabel *Label1;
	void __fastcall Button1Click(TObject *Sender);
private:	// User declarations
public:		// User declarations
	__fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

ДОДАТОК Г
ГРАФІЧНА ЧАСТИНА

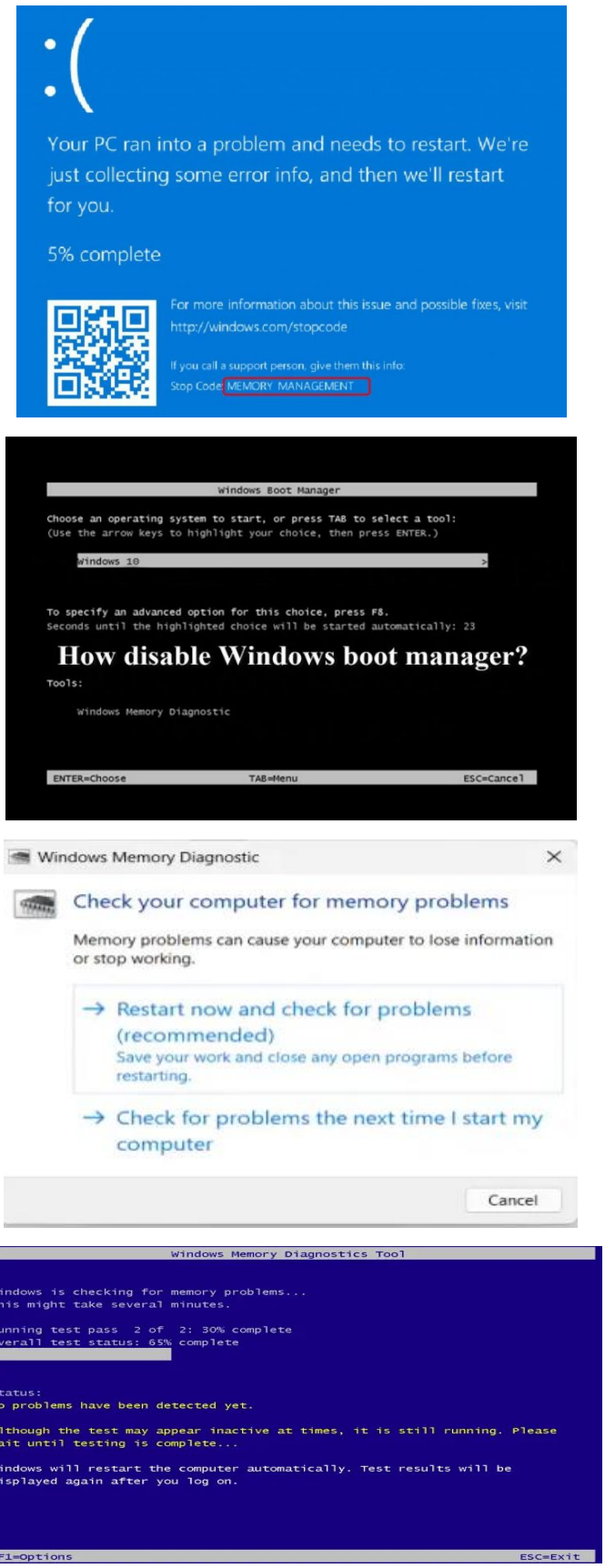
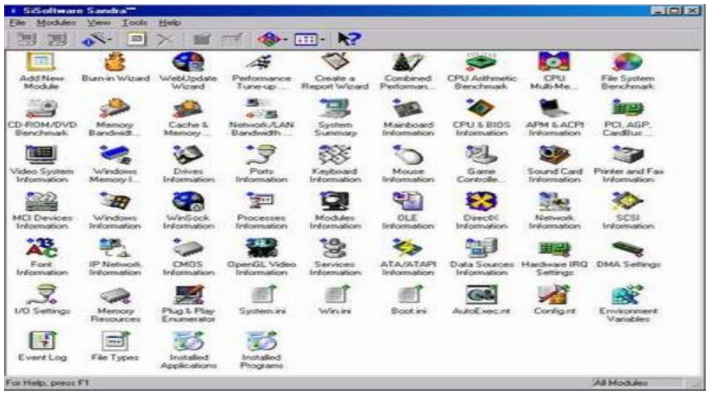
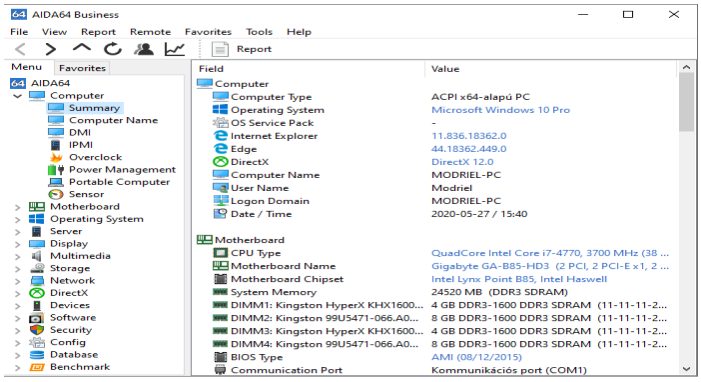


Рисунок Г.1 – Зображення на екрані при несправності оперативної пам’яті ПК





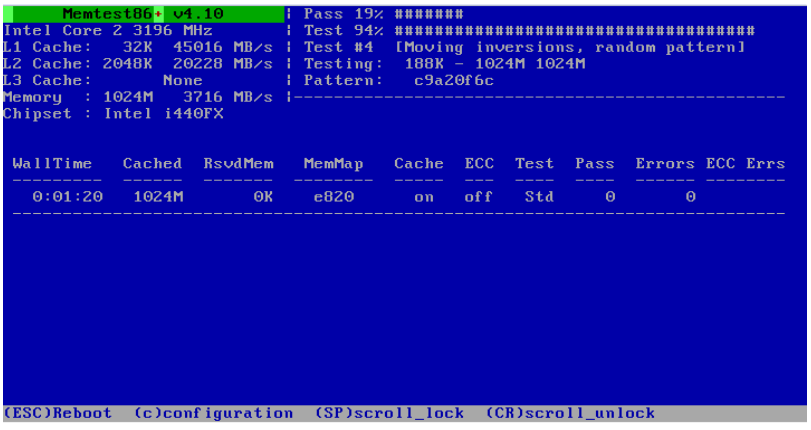


Рисунок Г.2 – Апаратні та програмні засоби діагностики несправності ПК

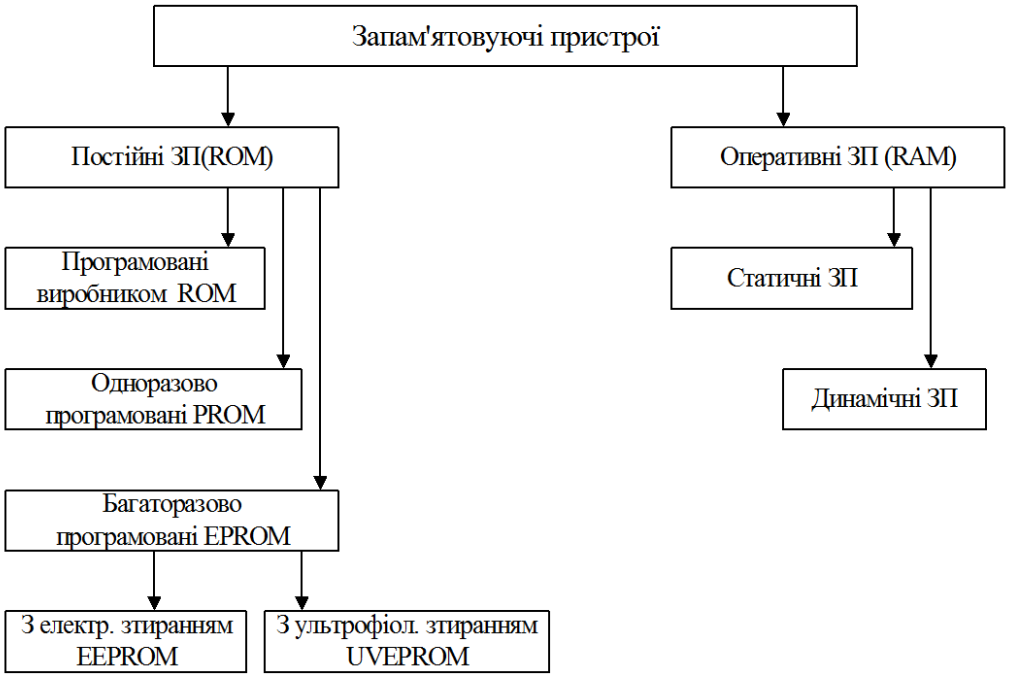


Рисунок Г.3 – Класифікація запам’ятовуючих пристроїв

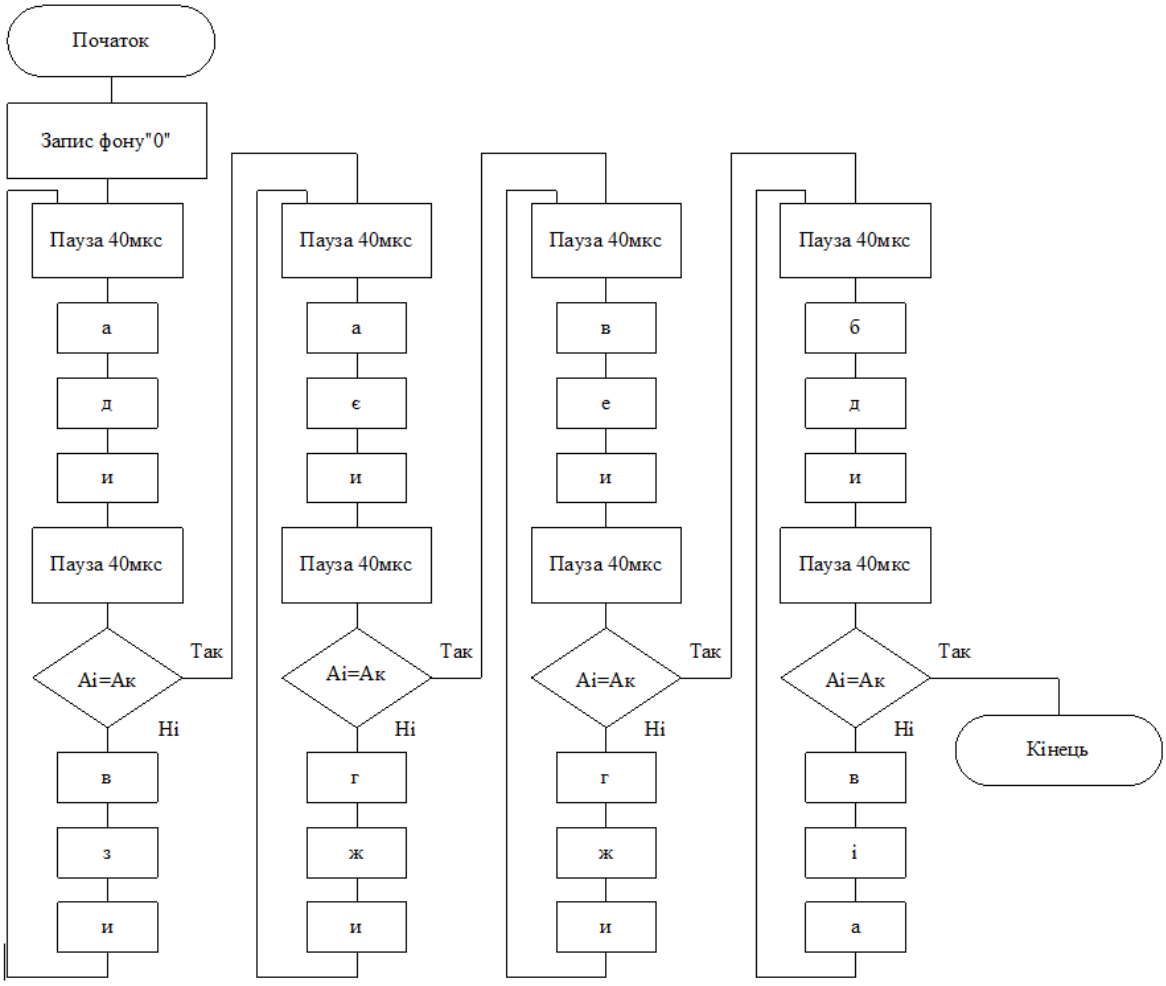


Рисунок Г.4 – Алгоритм тестової перевірки «марш одиниць і нулів»

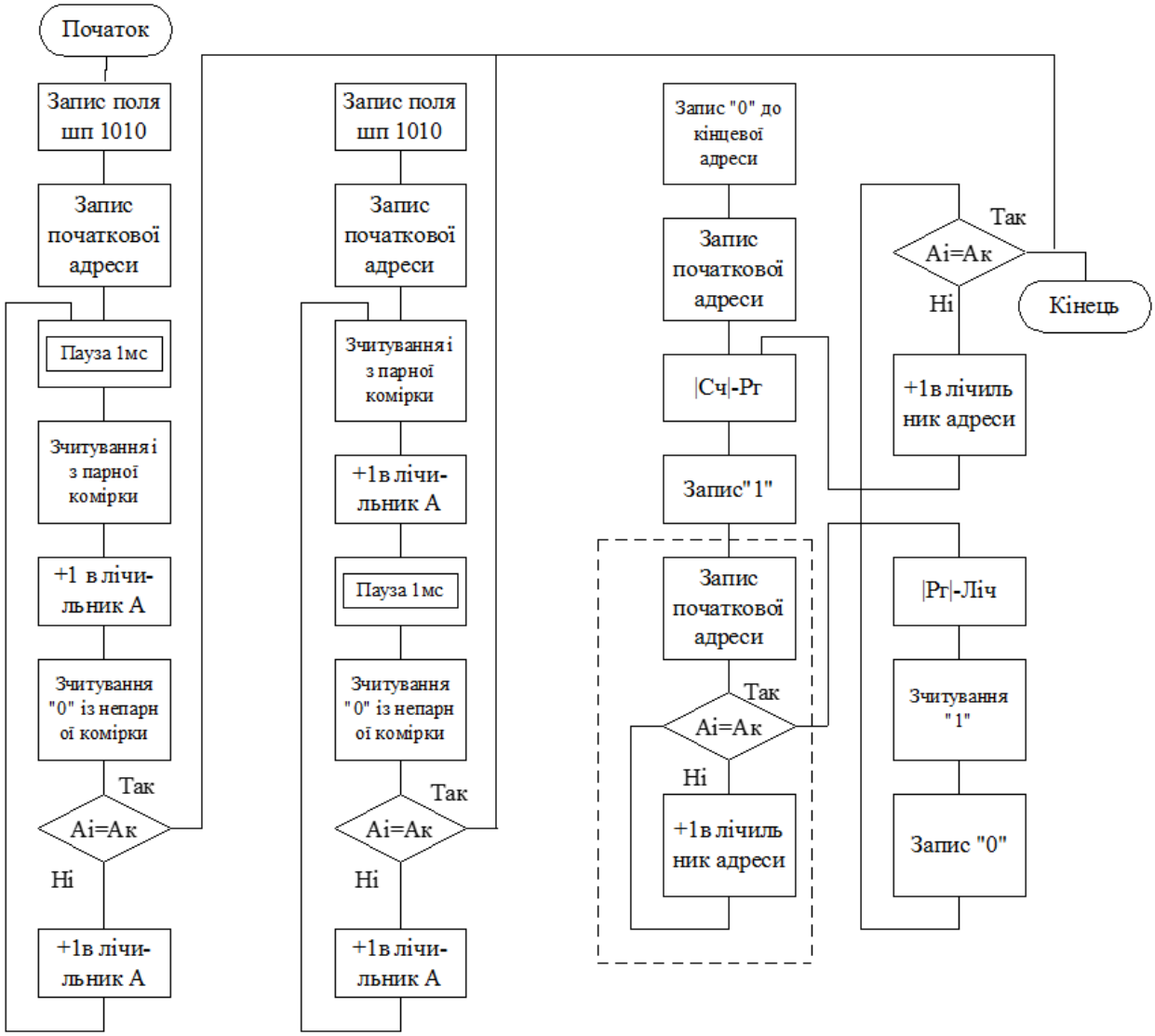


Рисунок Г.5 – Алгоритм тестової перевірки «парність-непарність»

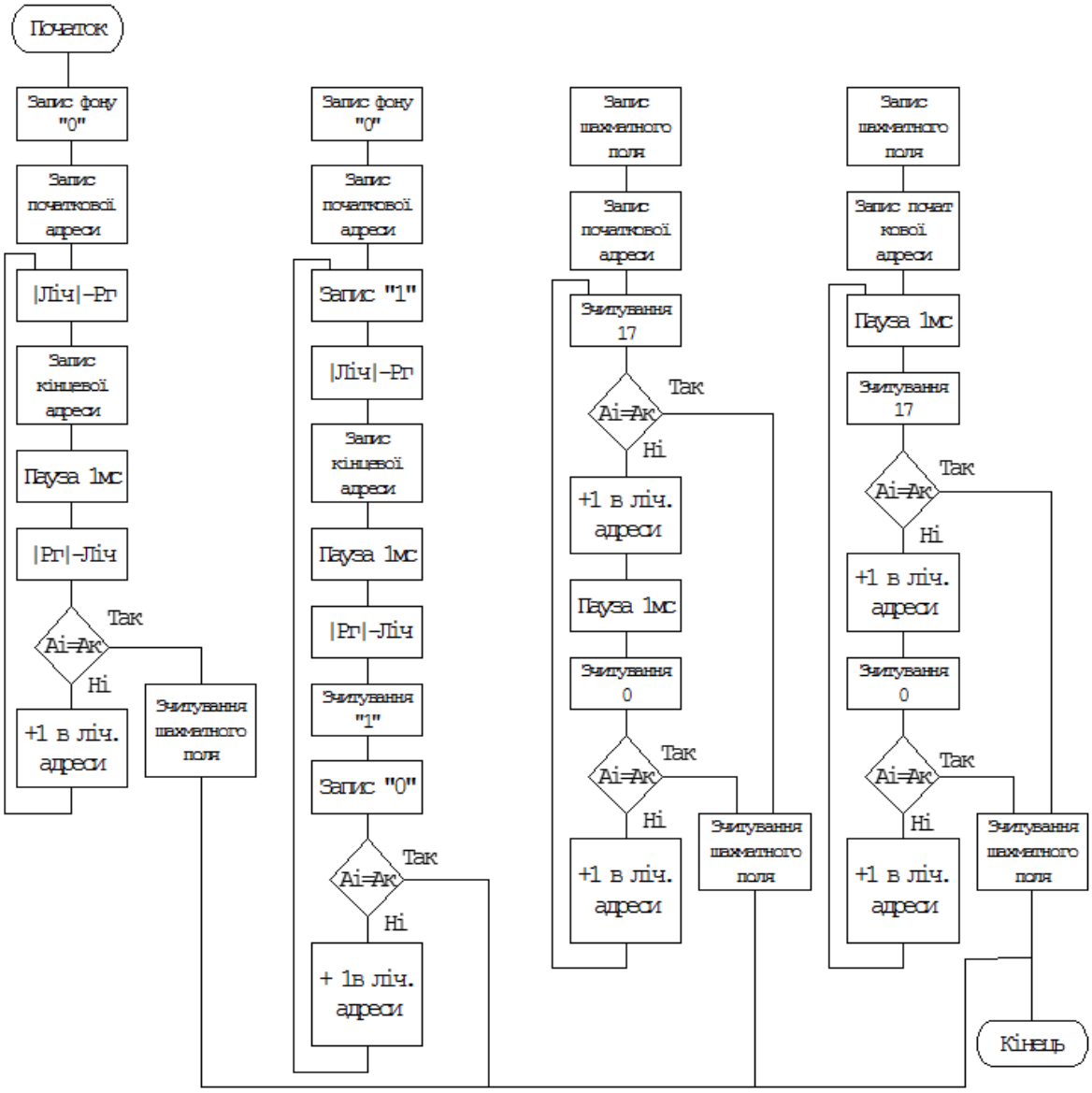


Рисунок Г.6 – Модифікований алгоритм тестової перевірки пам’яті ПК

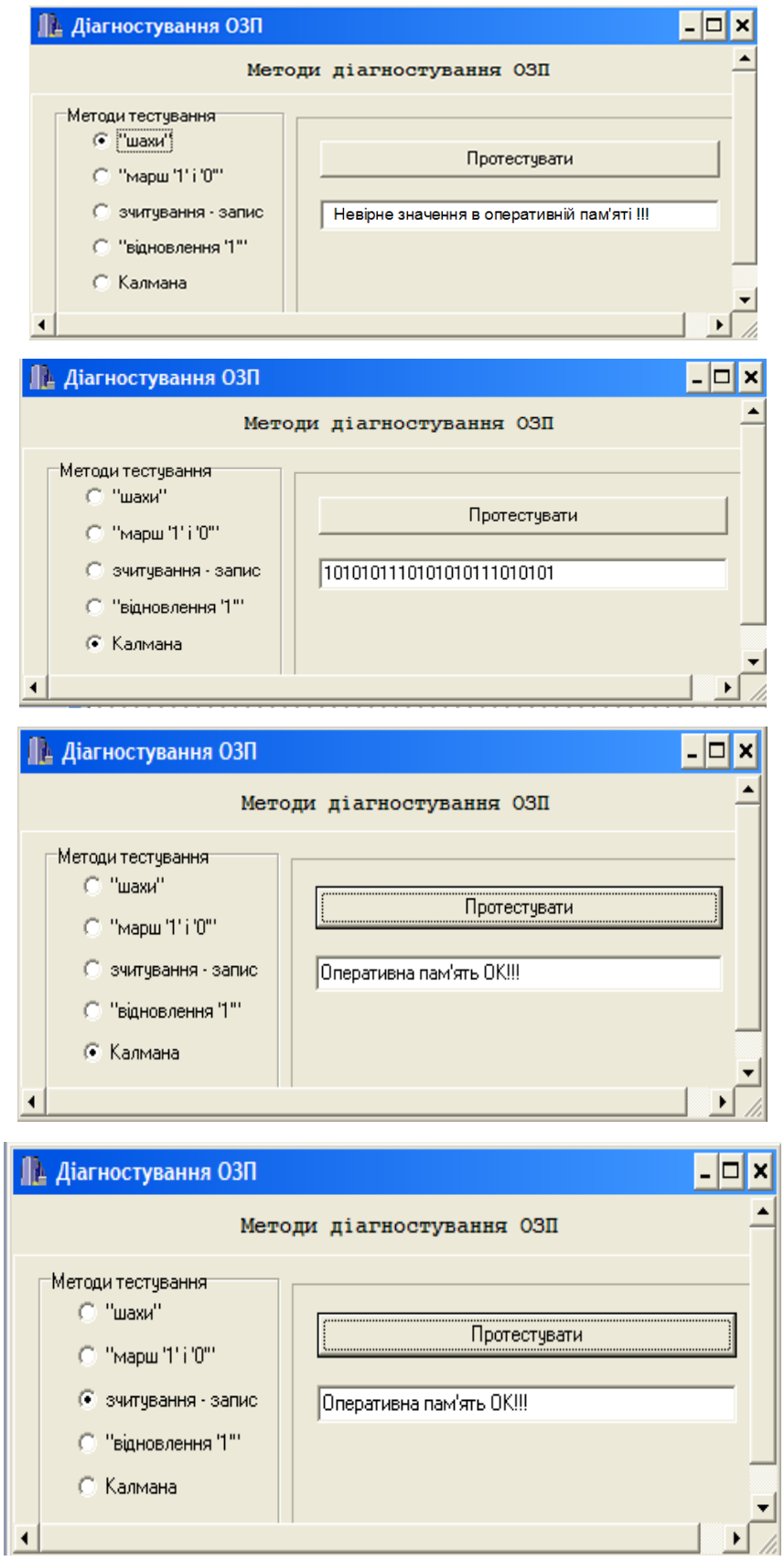


Рисунок Г.7 – Інтерфейс розробленого програмного модуля