УДК 004.02

Є.В. НІКІТЕНКО*

ОДИН З ПРИКЛАДІВ ВИКОРИСТАННЯ МЕТОДУ ПОШУКУ НЕСПРАВНОСТЕЙ У СКЛАДНИХ ЕЛЕКТРОННИХ ПРИЛАДАХ З УРАХУВАННЯМ ЗОВНІШНІХ ФАКТОРІВ

*Чернігівський національний технологічний університет, Чернігів, Україна

Анотація. *Проведено експериментальну перевірку розробленого методу пошуку несправностей з урахуванням зовнішніх факторів. Результати показали більшу ефективність поліпшеного методу в порівнянні з базовим. Особливості розробленого методу, який застосовується при діагностиці складних електронних пристроїв, дозволять швидше знайти несправний блок у схемі.*
Ключові слова: *діагностика, несправність.*

Аннотация. *Проведена экспериментальная проверка разработанного метода поиска неисправностей с учетом внешних факторов. Результаты показали большую эффективность улучшенного метода по сравнению с базовым. Особенности разработанного метода, который применяется при диагностике сложных электронных устройств, позволят быстрее найти неисправный блок в схеме.*
Ключевые слова: *диагностика, неисправность.*

Abstract. *The experimental verification of the developed fault finding method taking into account the external factors was conducted. The results showed higher efficiency of improved method compared to the basic approach. Particularities of this method which is used in the diagnosis of complex electronic devices will allow quickly identify the faulty unit in the scheme.*
Keywords: *diagnostics, fault.*

1. Вступ

Контроль, діагностику і налаштування радіоелектронної апаратури проводять програмними та апаратними методами. Підприємствами розробляються спеціальні інструкції для користувачів і діагностичні програми, що додаються до виробів у вигляді технічного опису, вбудованого програмного забезпечення або спеціальних програм на носіях інформації. Їх можна умовно поділити на дві групи: POST (Power-On Self Test – програма самоперевірки при включенні) та спеціалізовані діагностичні програми [1]. Складність програм та їх потенційні можливості на кожному наступному рівні, як правило, зростають.

Програми POST представляють собою послідовність коротких програм, які знаходяться в ПЗУ, призначені для перевірки основних компонентів системи безпосередньо після її включення і запускаються при включенні системи. Зазвичай перевіряються центральний процесор, ПЗУ, системні плати, оперативна пам'ять і основні периферійні пристрої. Ці тести виконуються швидко і не надто ретельно в порівнянні з діагностичними програмами, записаними на дисках. Якщо при виконанні програми POST виявляється несправний компонент системи, то видається повідомлення про помилку або попереджувальний сигнал. Якщо несправність досить серйозна ("фатальна помилка"), то подальше завантаження системи припиняється і видається повідомлення, за яким можна визначити причину несправності.

Спеціалізовані діагностичні програми – це набори тестів для повної перевірки всіх компонентів систем і складних приладів, які записуються на окремому діагностичному диску. Діагностичні програми передбачені двох рівнів. Перший рівень – це загальна діагностика, яка орієнтована на користувачів. Так як процедури пошуку несправностей у більшості сучасних систем досить прості, у користувачів зазвичай не виникає труднощів при роботі з програмами загальної діагностики. Другий рівень – технічний, він розрахований

на фахівців. Повідомлення про помилки виводяться у вигляді кодів, за якими можна визначити причину несправності або звузити коло її пошуків.

2. Аналіз досліджень і публікацій

Метод, описаний в роботі [2], спрямований на пошук несправностей для складних електронних пристроїв. Цей метод дає можливість скоротити час на пошук несправного блока, а також локалізувати зону його пошуку за рахунок урахування зовнішніх факторів. Розглянемо роботу методу на прикладі складного електронного пристрою. Як такий візьмемо ноутбук Sony з материнською платою m750 [3] (рис. 1).

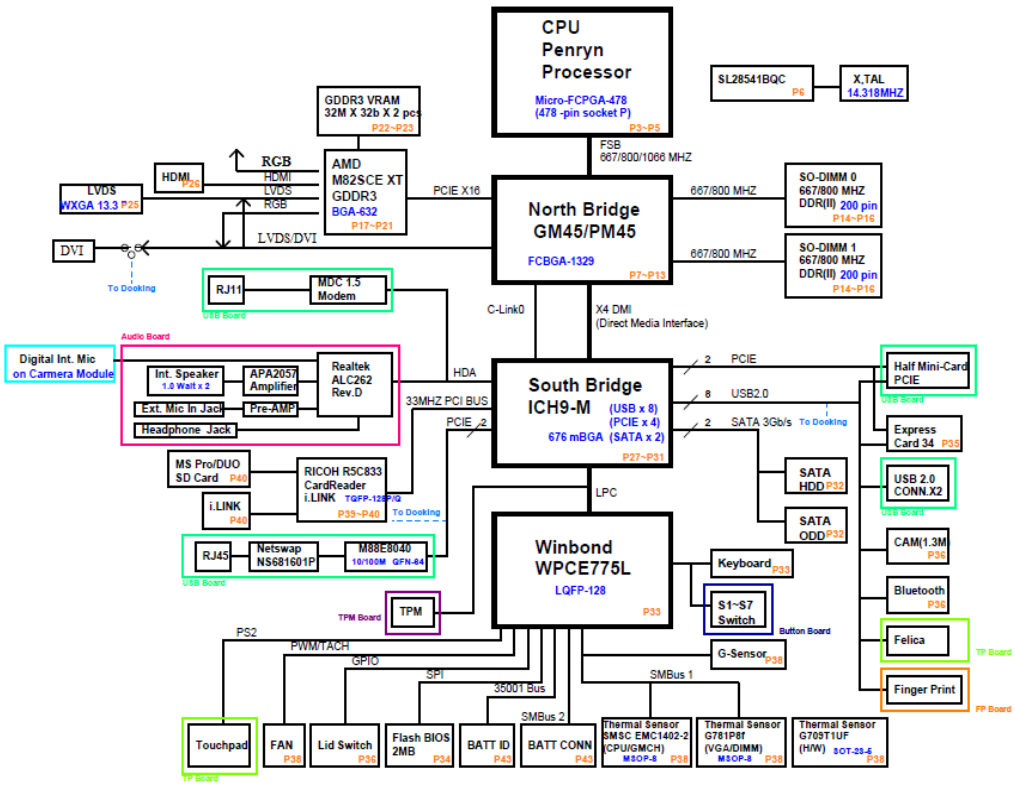


Рис. 1. Схема з'єднання блоків плати Sony m750

Схема, що зображена на рис. 1, досить просто може бути представлена у вигляді функціональних логічних елементів. Більшість із елементів плати мають один вхід та один вихід.

3. Опис схеми у вигляді функціонально-логічних блоків

Опишемо ноутбук у системі пошуку несправностей, вказуючи внутрішні зв'язки і зв'язки між блоками. Будемо вважати, що всі лінії зв'язку справні, тому не виділяємо їх в окремі функціональні елементи (рис. 2).

Розглянемо для прикладу такий сценарій: ми отримуємо дані по витій парі, які передаються в ноутбук через порт RJ45. Отримані дані повинні пройти обробку і передаватися далі на зовнішній екран через порт DVI. Вважаємо, що інформація точно приходить і що пристрій, підключений до порту DVI, справний. Ми можемо спостерігати тільки за станом входу і виходу. Про справність або несправність інших виходів ми нічого не знаємо. Якщо за якими-то причинами ми перестали отримувати сигнал на виході порту DVI, приймаємо це за несправність.

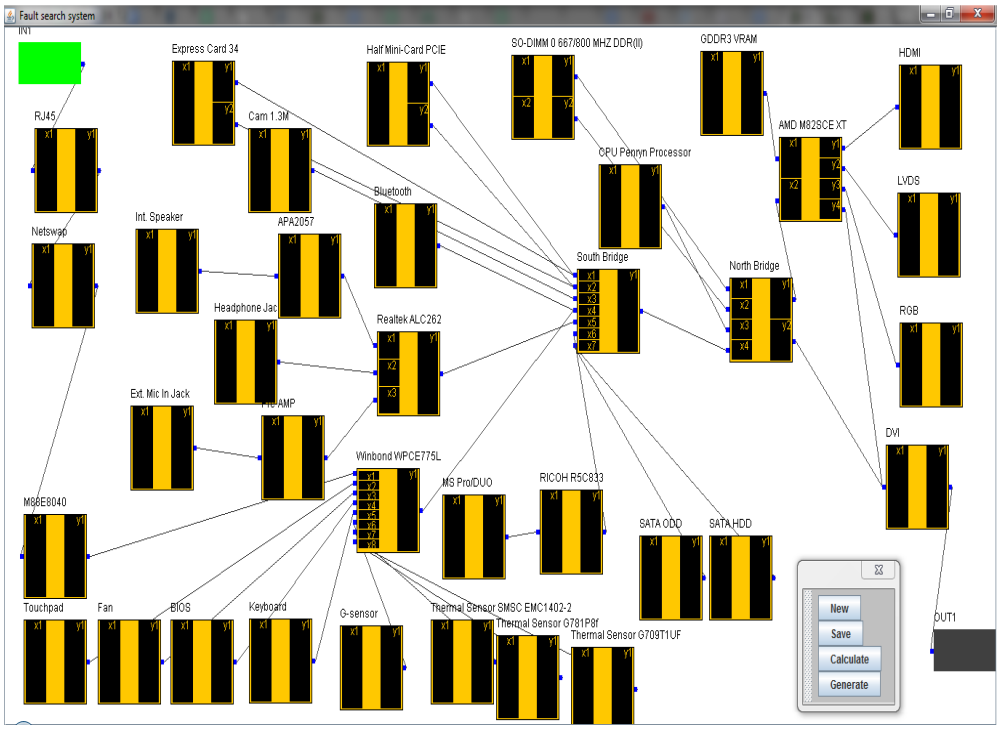


Рис. 2. Представлення материнської плати у вигляді функціонально-логічних блоків (спостерігається один вихід)

Рис. 3. Меню вказівки несправних виходів

Для того, щоб знайти список елементів, які необхідно перевірити при появі несправності на виході порту DVI, активуємо кнопку "Calculate". Нам пропонується вказати виходи, на яких спостерігається несправність. Так як ми контролюємо тільки вихід OUT1, то вказуємо тільки його (рис. 3).

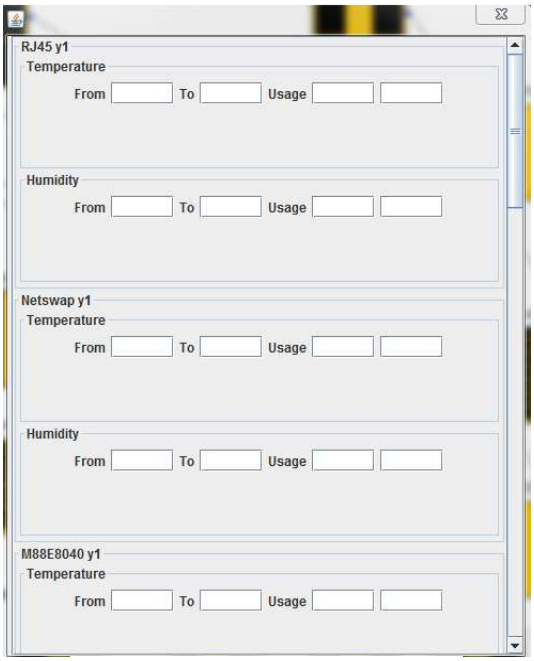


Рис. 4. Меню зазначення даних про зовнішні фактори

Вже на наступному кроці ми отримуємо набір виводів, які необхідно перевірити. Для кожного з них необхідно вказати дані про температуру і вологість, а саме: допустимий мінімум і максимум, а також середнє значення під час експлуатації (рис. 4).

На даному етапі необхідно, по можливості, якомога точніше і повною мірою вказати дані про виводи функціональних блоків. Нам потрібно знати інформацію, яка взята з технічної документації кожного блока тільки для певних виводів, а не абсолютно всіх у пристрої.

Представимо дані про зовнішні фактори у вигляді табл. 1 і 2.

Нехай ми знаємо, що ноутбук використовувався при температурі -5 °С і відносній

вологості повітря 50%.

Таблиця 1. Дані про температуру для певних виводів

№ п/п	Ім’я блока	Мінімальна границя температури, °С	Максимальна границя температури, °С	Середнє значення температури, °С
1	RJ45 y1	-25	125	-5
2	Netswap NS681601P y1	0	70	-5
3	M88E8040 y1	10	115	-5
4	Winbond WPCE775L y1	-10	100	-5
5	South Bridge ICH9-M y1	0	95	-5
6	North Bridge GM45/PM45 y1	0	100	-5
7	North Bridge GM45/PM45 y2	0	100	-5
8	AMD M82SCE XT y1	0	60	-5
9	DVI y1	-40	85	-5

Таблиця 2. Дані про вологість для певних виводів

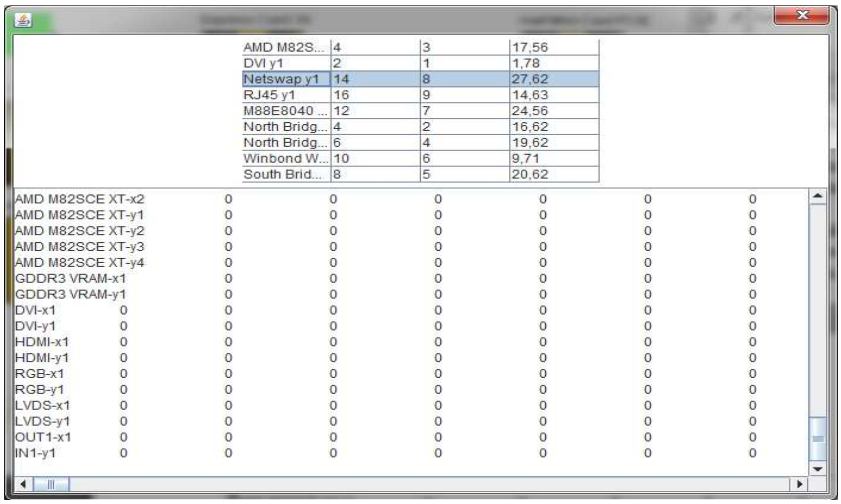
№ п/п	Ім’я блока	Мінімальна границя вологості, %	Максимальна границя вологості, %	Середнє значення вологості, %
1	RJ45 y1	0	100	50
2	Netswap NS681601P y1	0	80	50
3	M88E8040 y1	0	90	50
4	Winbond WPCE775L y1	0	75	50
5	South Bridge ICH9-M y1	0	80	50
6	North Bridge GM45/PM45 y1	0	80	50
7	North Bridge GM45/PM45 y2	0	80	50
8	AMD M82SCE XT y1	0	90	50
9	DVI y1	0	100	50

У табл. 1 з даними про температуру бачимо, що були порушені умови експлуатації шести елементів, так як -5 °С не входить у діапазон допустимих для цих компонентів температур. Значення відносної вологості для всіх компонентів знаходяться в межах норми (табл. 2).

У табл. 1 у колонці, де зазначено середнє значення температури експлуатації, у всіх рядках одне і те ж число. Материнська плата ноутбука представляє собою один пристрій: усі компоненти, що знаходяться на ній, перебувають в однакових температурних умовах. Значення в цій колонці можуть відрізнятися, якщо елементи пристрою перебували в різних умовах експлуатації. Таке може бути, наприклад, в системі вимірювання температури на підприємстві, коли різні датчики експлуатуються в різних температурних умовах.

4. Результат роботи методу

Провівши коригування ваг, ми отримаємо кінцевий результат, в якому зазначені остаточні ваги. Слідуючи значенням ваг, отримуємо порядок, в якому необхідно перевіряти функціональні блоки у пристрої (рис. 5).



AMD M82S...	4	3	17,56			
DVI y1	2	1	1,78			
Netswap y1	14	8	27,62			
RJ45 y1	16	9	14,63			
M88E8040 ...	12	7	24,56			
North Bridg...	4	2	16,62			
North Bridg...	6	4	19,62			
Winbond W...	10	6	9,71			
South Brid...	8	5	20,62			
AMD M82SCE XT-x2	0	0	0	0	0	0
AMD M82SCE XT-y1	0	0	0	0	0	0
AMD M82SCE XT-y2	0	0	0	0	0	0
AMD M82SCE XT-y3	0	0	0	0	0	0
AMD M82SCE XT-y4	0	0	0	0	0	0
GDDR3 VRAM-x1	0	0	0	0	0	0
GDDR3 VRAM-y1	0	0	0	0	0	0
DVI-x1	0	0	0	0	0	0
DVI-y1	0	0	0	0	0	0
HDMI-x1	0	0	0	0	0	0
HDMI-y1	0	0	0	0	0	0
RGB-x1	0	0	0	0	0	0
RGB-y1	0	0	0	0	0	0
LVDS-x1	0	0	0	0	0	0
LVDS-y1	0	0	0	0	0	0
OUT1-x1	0	0	0	0	0	0
IN1-y1	0	0	0	0	0	0

Рис. 5. Результат роботи методу

У результаті в самій правій колонці таблиці на рис. 5 бачимо значення ваг, які визначають порядок перевірки елементів у пристрої. У першу чергу повинен бути перевірений Netswap y1, так як значення ваги цього елемента має найбільше значення, а саме 27.62. Якби ми не враховували в розрахунках інформацію про зовнішні фактори (базовий алгоритм), то елемент Netswap y1 повинен бути перевіреним другим.

Розглянемо ще ситуацію з RJ45. У базовому алгоритмі він повинен бути перевірений першим, але так, як він експлуатувався згідно з його технічними характеристиками, то і перевіряти його варто тільки після тих елементів, чиї умови експлуатації не дотримувалися, тому що ймовірність виходу з ладу таких блоків вища.

До цього ми розглядали випадок, коли спостерігався тільки один вхід і один вихід, в результаті чого ми отримали дев’ять елементів, які слід перевірити.

Розглянемо випадок, коли ми знаємо дані про стан інших виходів пристрою. Нехай на екрані ноутбука є зображення, тобто відображаються коректні дані. Екран ноутбука представлений виходом RGB. А на виході DVI також спостерігаємо відсутність сигналу. Додамо на схему ще один вихід типу OUT, який показує стан даних на виході блока RGB (рис. 6).

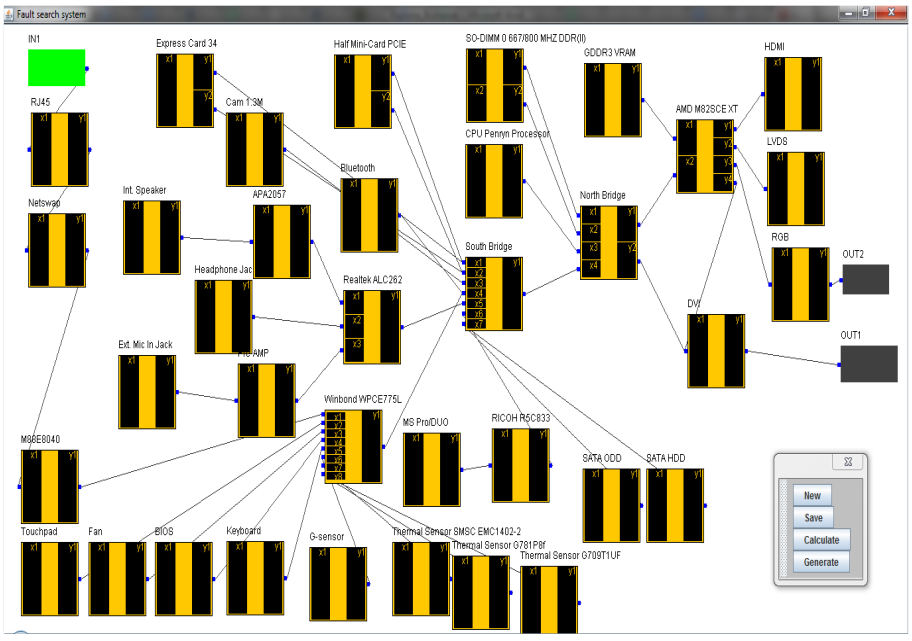


Рис. 6. Представлення пристрою у вигляді функціонально-логічних блоків (спостерігаються два виходи)

На наступному етапі, коли буде запропоновано вказати несправні блоки, ми вкажемо тільки вихід OUT1, а OUT2 відзначимо як справний. У результаті ми отримаємо значно меншу кількість виводів, які необхідно перевіряти, а саме три, які представлені в табл. 3.

Таблиця 3. Елементи, які слід перевірити

№п/п	Назва блока
1	North Bridge y2
2	AMD M82SCE XT y4
3	DVI y1

Вкажемо дані про зовнішні фактори і отримаємо кінцевий порядок перевірки для цих елементів (рис. 7).

Варто звернути увагу на те, що базовий метод виводам North Bridge y2 і AMD M82SCE XT y4 присвоїв однакову вагу, так як для нього вони виглядають як елементи одного рівня. Метод з урахуванням зовнішніх факторів (поліпшений метод) успішно вирішує цю проблему і визначає однозначний порядок перевірки елементів. Чим більше вказали даних про стан пристрою, тим ефективніше спрацював метод, тим менше елементів потрібно перевіряти, що прискорює пошук несправного блока.

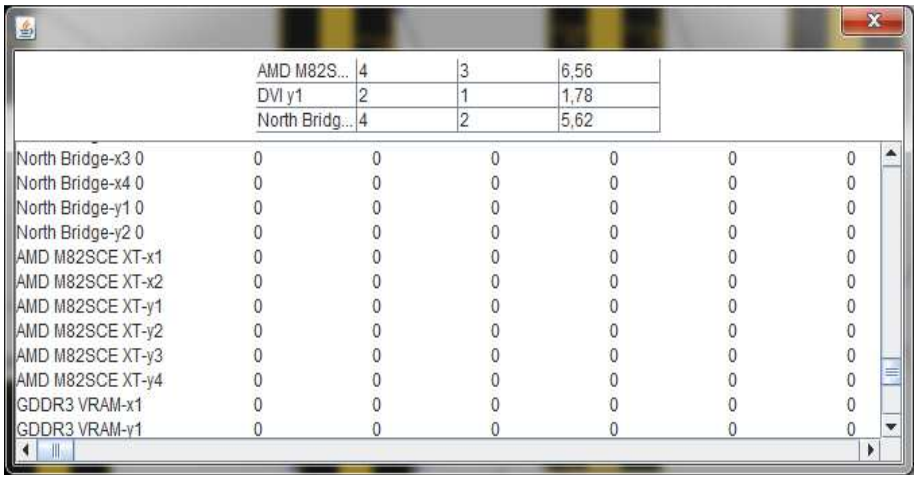


Рис. 7. Результат роботи методу (спостерігалося два виходи пристрою)

Ми розглянули весь цикл пошуку списку елементів, що підлягають перевірці при появі несправності на виході DVI ноутбука Sony. У даному випадку використовувалося два зовнішніх фактори: вологість і температура навколишнього середовища, при яких використовувався ноутбук.

5. Порівняння базового та поліпшеного методів

Проведемо порівняння базового та поліпшеного методів при різних умовах і вхідних даних. Для цього в систему, що реалізовує поліпшений метод, була додана функціональність, яка дозволяє генерувати випадкові схеми функціонально-логічних блоків. Можна було б цей етап провести і шляхом ручного завдання схем, але це вимагало дуже багато часу.

Був програмно реалізований алгоритм, що генерує схеми функціонально-логічних елементів відповідно до вказаних вхідних параметрів, а саме:

– кількість функціонально-логічних елементів у схемі;

– максимальна кількість входів функціонально-логічного елемента;

– мінімальна кількість входів функціонально-логічного елемента;

– максимальна кількість виходів функціонально-логічного елемента;

– мінімальна кількість виходів функціонально-логічного елемента;

– максимальна кількість контактів, до яких може бути приєднаний один вихідний контакт функціонально-логічного елемента;

– максимальна кількість підконтрольних вхідних елементів (тип IN);

– максимальна кількість підконтрольних вихідних елементів (тип OUT).

Далі представлений основний код класу, який генерує схеми пристроїв за вказаними параметрами.

```
	public class RandomSchemaGenerator {

	private int COUNT_OF_NODES = 40;
	private int MAX_COUNT_INPUT = 5;
	private int MAX_COUNT_OUTPUT = 4;
	private int MAX_EXTERNAL_CONNECTIONS = 1;
	private int COUNT_NODES_TO_INPUTS = 4;
	private int COUNT_NODES_TO_OUTPUTS = 4;
	private Random randomGeneretor = new Random(System.currentTimeMillis());

	public void generateNodes() {
		List<Node> nodes = new ArrayList<Node>();
		List<Node> nodesToInputs = new ArrayList<Node>();
		List<Node> nodesToOutputs = new ArrayList<Node>();
		for (int k = 0; k < COUNT_OF_NODES; k++) {
			int countInputs = randomGeneretor.nextInt(MAX_COUNT_INPUT) + 1;
			int countOutputs = randomGeneretor.nextInt(MAX_COUNT_OUTPUT) + 1;
			List<ContactDB> contacts = new ArrayList<ContactDB>();
			ContactDB tmp = null;
			for (int i = 1; i <= countInputs; i++) {
				tmp = new ContactDB("x" + i, true, i);
				contacts.add(tmp);
				DBManager.getInstance().saveContact(tmp);
			}
			for (int i = 1; i <= countOutputs; i++) {
				tmp = new ContactDB("y" + i, false, i);
				contacts.add(tmp);
				DBManager.getInstance().saveContact(tmp);
			}
			Node n = new Node(PositionCounter.getNextPosition(), 55,
countInputs, countOutputs, Kind.TYPICAL);
			n.name = "Node " + PositionCounter.getNextNumber();
			DBManager.getInstance().saveNode(n);
			n.contacts.addAll(contacts);
			DBManager.getInstance().updateNodeCotacts(n);
			List<ContactDB> inputs = n.getAllInputs();
			List<ContactDB> outputs = n.getAllOutputs();
			for (ContactDB c : inputs) {
				int contactConnections =
randomGeneretor.nextInt(outputs.size()) + 1;
				Set<Integer> connectTo =
getCountRandomNumbersInRange(contactConnections, outputs.size());
				for (Integer contNumber : connectTo) {
					ContactDB toDB = outputs.get(contNumber);
					DBManager.getInstance().saveLink(new Edge(n, n, c,
toDB));
				}
			}
			nodes.add(n);
			if (k < COUNT_NODES_TO_INPUTS) {
				nodesToInputs.add(n);
			}
			if (k > COUNT_OF_NODES - COUNT_NODES_TO_OUTPUTS) {
				nodesToOutputs.add(n);
			}
		}
		List<ContactDB> outputs;
		List<ContactDB> inputs;
		for (Node n : nodes) {
			if (nodesToOutputs.contains(n)) {
				List<ContactDB> connectOutType = n.getAllOutputs();
				for (ContactDB toOutType : connectOutType) {
					Node outNode = generateInOutNode(false);
					DBManager.getInstance().saveLink(new Edge(n, outNode,
toOutType, outNode.getAllInputs().get(0)));
				}
				continue;
			}else if(nodesToInputs.contains(n)){
```

```
                        List<ContactDB> connectInType = n.getAllInputs();
                        for (ContactDB toInContact : connectInType) {
                              Node inNode = generateInOutNode(true);
                              DBManager.getInstance().saveLink(new  Edge(inNode,  n,
inNode.getAllOutputs().get(0), toInContact));
                        }
                  }
                  outputs = n.getAllOutputs();
                  for (ContactDB out : outputs) {
                        for (int t = 0; t < MAX_EXTERNAL_CONNECTIONS; t++) {
                              boolean isFind = false;
                              Node toN = null;
                              while (!isFind) {
                                    int                    numb                    =
randomGeneretor.nextInt(nodes.size());
                                    toN = nodes.get(numb);
                                    if           (!toN.id.equals(n.id)           &&
!nodesToInputs.contains(toN)) {
                                          isFind = true;
                                    }
                              }
                              inputs = toN.getAllInputs();
                              int                    inputNumber                    =
randomGeneretor.nextInt(inputs.size());
                              DBManager.getInstance().saveLink(new Edge(n, toN, out,
inputs.get(inputNumber)));
                        }
                  }
            }

      }

      private Set<Integer> getCountRandomNumbersInRange(int count, int max) {
            Set<Integer> rez = new HashSet<Integer>();
            while (rez.size() < count) {
                  rez.add(randomGeneretor.nextInt(max));
            }
            return rez;
      }

      private Node generateInOutNode(boolean isIn) {
            Node node;
            if (isIn) {
                  node  =  new  Node(PositionCounter.getNextPosition(),  55,  1,  1,
Kind.IN);
                  node.name = "IN " + PositionCounter.getNextIn();
            } else {
                  node  =  new  Node(PositionCounter.getNextPosition(),  55,  1,  1,
Kind.OUT);
                  node.name = "OUT " + PositionCounter.getNextOut();
            }

            ContactDB in = new ContactDB("x1", true, 1);
            ContactDB out = new ContactDB("y1", false, 1);
            DBManager.getInstance().saveNode(node);
            DBManager.getInstance().saveContact(in);
            DBManager.getInstance().saveContact(out);
            node.contacts.add(in);
            node.contacts.add(out);
            DBManager.getInstance().updateNodeCotacts(node);
            return node;
      }
   }
```

Була проведена генерація схем при різній кількості блоків і різних даних про зовнішні фактори. Для кожного випадку було зроблено 20 експериментальних запусків системи. Цієї кількості цілком достатньо для визначення точного значення результату по відношен-

ню до поставлених нами умов. На графіках зображено середнє значення, отримане за цими експериментами.

Основною метою є визначення, наскільки швидше поліпшений метод дозволяє знайти несправний блок у схемі. При генерації схеми функціонально-логічних елементів кожному блоку присвоюється порядковий номер.

Будемо вважати, що у пристрої вийшов з ладу один елемент. Для всіх експериментів повинні бути рівні умови, тому за несправний блок приймаємо той, що знаходиться посередині схеми. Наприклад, якщо в генерованому пристрої 50 блоків, то несправний матиме номер 25, якщо 100, то 50-й, якщо 200, то 100-й.

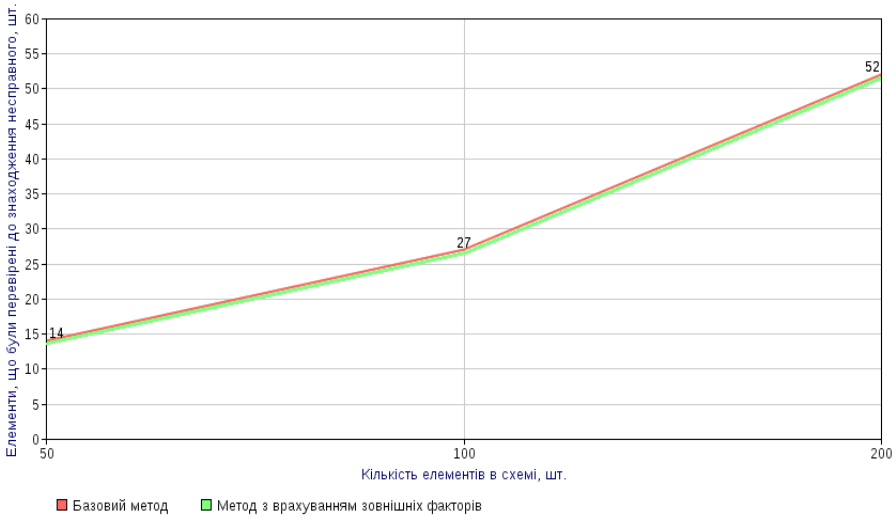


Рис. 8. Залежність кількості елементів, які необхідно перевірити для виявлення несправного (дані про зовнішні фактори відсутні)

Графік на рис. 8 показує, що кількість елементів, які необхідно перевірити до виявлення несправного, абсолютно збігається для базового методу і методу з урахуванням зовнішніх факторів, при умові, що використовується поліпшений метод, але дані про зовнішні фактори відсутні. Так як дані про температуру і вологість невідомі, то і ваги не змінюються відносно тих, що визначені базовим алгоритмом. Так само важливим результатом, який бачимо на цьому графіку, є те, що навіть за відсутності необхідної інформації поліпшений метод не погіршує результат, отриманий від базового методу.

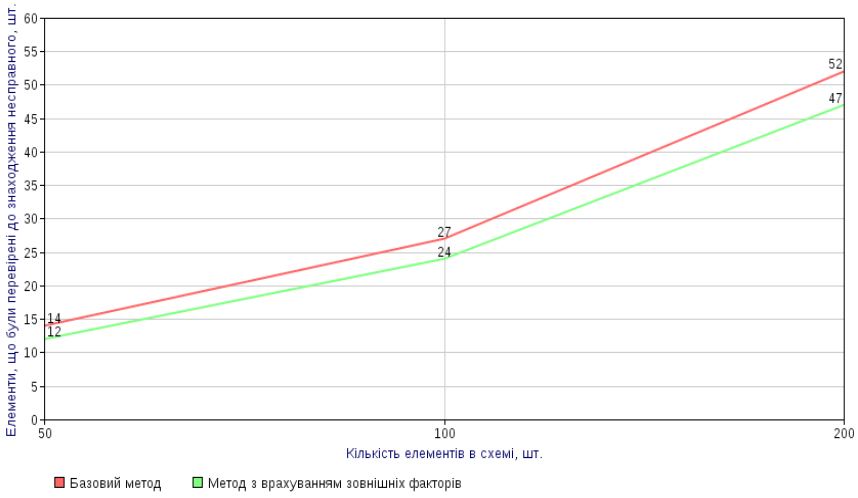


Рис. 9. Залежність кількості елементів, які необхідно перевірити для виявлення несправного (дані про зовнішні фактори відомі тільки для половини елементів)

Графік на рис. 9 показує зменшення кількості елементів у методі з урахуванням зовнішніх факторів, які необхідно перевірити до того, як дійдемо до несправного елемента. Після нескладних підрахунків відзначимо, що поліпшений метод швидше виявить несправний елемент у порівнянні з базовим на 11,3%.

На рис. 10 показаний графік залежності елементів, які необхідно перевірити до виявлення несправності, від кількості елементів в електронному пристрої. Це найбільш ефективні умови роботи поліпшеного методу, так як присутні всі необхідні дані про зовнішні

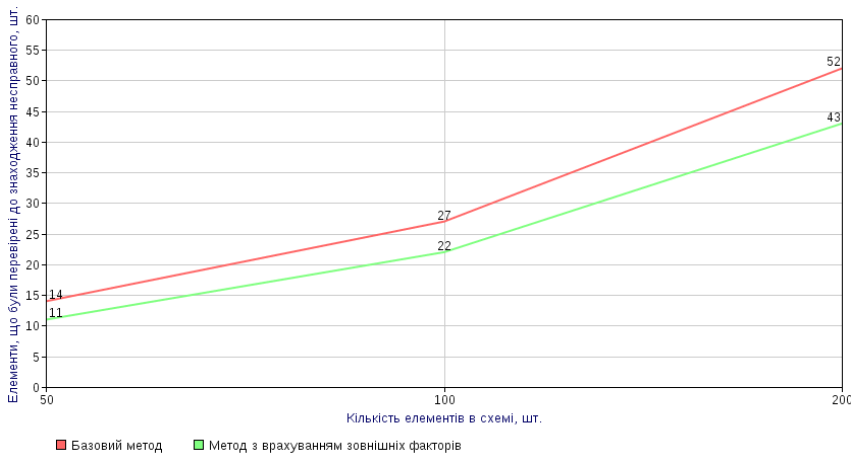


Рис. 10. Залежність кількості елементів, які необхідно перевірити для виявлення несправного (дані про зовнішні фактори відомі для всіх елементів)

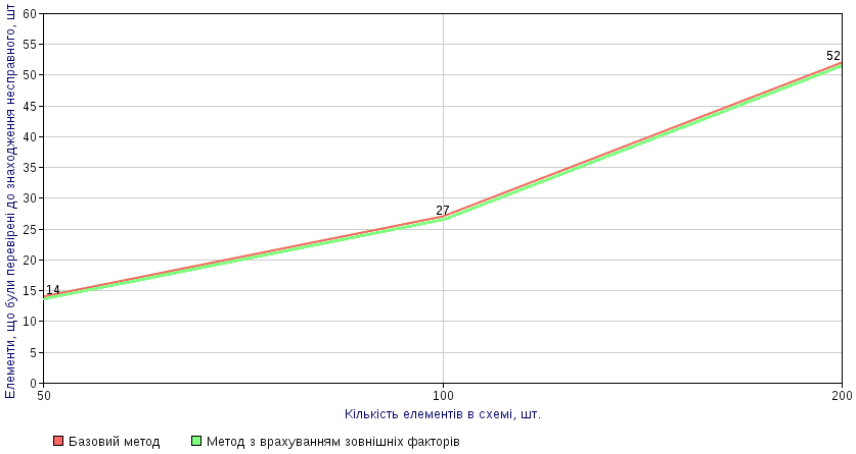


Рис. 11. Залежність кількості елементів, які необхідно перевірити для виявлення несправного (умови експлуатації всіх блоків у пристрої – порушені)

фактори. Метод показав зменшення часу пошуку несправного блока в порівнянні з базовим методом на 18%.

Розглянемо один із пригранічних випадків при пошуку несправного елемента, коли середні значення всіх факторів виходять за їхні межі (рис. 11). Таке може бути при грубому порушенні умов експлуатації електронного пристрою. У цьому випадку метод з урахуванням зовнішніх факторів змінить значення ваг всіх елементів, але всі вони збільшаться на одне і те ж число (кількість елементів, які підлягають перевірці), що ніяким чином не змінює порядок перевірки блоків при виявленні несправності на одному з виходів. Тому в цьому випадку поліпшений алгоритм показує такі ж результати, як і базовий.

6. Висновки

У роботі проведено експериментальну перевірку розробленого методу пошуку несправностей з урахуванням зовнішніх факторів. Результати показали більшу ефективність поліпшеного методу в порівнянні з базовим. Викладені особливості розроблених діагностичних методів, які застосовуються при діагностиці складних електронних пристроїв, дозволять швидше знайти несправний блок у схемі.

На закінчення слід підкреслити, що розробка нових методів для діагностики сучасної апаратури є актуальним і соціально значимим завданням сучасного приладобудування.

СПИСОК ЛІТЕРАТУРИ

1. Глушков С.В. Персональный компьютер: учебный курс / С.В. Глушков, И.В. Мельников. – М.: «АСТ», 2000. – 512 с.
2. Метод пошуку несправностей в складних електронних приладах з урахуванням зовнішніх факторів / Є.В. Нікітенко // Математичні машини і системи. – 2014. – № 1. – С. 70 – 79.
3. M750 Main Board [Електронний ресурс]. – Режим доступу: http://zremcom.ru/scheme/scheme-sony/file/754-scheme-sony-m751pvtmb0627vgn-sr55e-m754h-mbx-190.

Стаття надійшла до редакції 07.02.2014