МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ОТЧЁТ

по лабораторной работе №2

по дисциплине

«Контроль и диагностика вычислительных систем»

Тема: «D-алгоритм (метод активизации многомерного пути)»

Выполнил:
магистрант 1 курса магистратуры
факультета компьютерных систем и сетей
группы 556301
Лебедевич Артём Владимирович

Проверил:
доктор технических наук, профессор
Ярмолик Вячеслав Николаевич

Минск, 2025 г.

1. Цель работы

Целью лабораторной работы является изучение D-алгоритма (алгоритма Рота), предназначенного для автоматической генерации тестов комбинационных цифровых схем. В ходе работы необходимо разобрать кубическое представление логических функций, освоить правила распространения неисправностей по многомерным путям и сравнить D-алгоритм с методом активизации одномерного пути по объёму тестовых наборов и трудоёмкости применения.

2. Теоретические сведения

2.1. Кубическое представление

В D-алгоритме логические состояния сигналов описываются расширенным алфавитом:

- 0 — логический ноль;
- 1 — логическая единица;
- x — безразличное значение (*don't care*);
- d — переход $0 \to 1$ (в исправной схеме значение 0, в неисправной 1);
- D — переход $1 \to 0$ (в исправной схеме значение 1, в неисправной 0).

Состояния логических элементов и всей схемы в целом описываются при помощи *кубов* — наборов значений для всех входных и выходных переменных. D-кубы позволяют явно отслеживать путь распространения влияния неисправности от места её возникновения к выходам схемы.

2.2. Типы кубов

В рамках D-алгоритма выделяют несколько типов кубов:

- **сингулярные кубы** описывают нормальное (исправное) функционирование логического элемента;
- **D-кубы элементов** задают поведение элемента при распространении символов d и D;
- **примитивные D-кубы неисправностей** описывают условия активации конкретной константной неисправности.

2.3. Основные этапы D-алгоритма

Алгоритм генерации теста для заданной неисправности включает следующие фазы:

1. **Выбор примитивного D-куба неисправности.** Для рассматриваемой неисправности на полюсе схемы подбирается D-куб, обеспечивающий её активацию.

2. **D-проход (D-drive).** Выполняется распространение символов d или D от места неисправности к одному из выходов схемы с использованием D-кубов логических элементов.

3. **Обеспечение консистентности.** Оставшимся переменным назначаются значения, согласованные с уже выбранными кубами, после чего формируется полный входной вектор.

2.4. Операция D-пересечения

При объединении информации из разных кубов используется операция D-пересечения, позволяющая проверить совместимость назначений. Некоторые базовые правила D-пересечения можно записать в виде:

```
0  0 = 0      0  x = 0      d  d = d
1  1 = 1      1  x = 1      D  D = D
x  x = x      d  1 = d      d  D =  (конфликт)
0  1 =       D  0 = D      d  0 =  (конфликт)
```

Появление пустого результата (конфликта) свидетельствует о некорректности либо выбранных кубов, либо промежуточного пути распространения неисправности.

3. Структура исследуемой схемы

В лабораторной работе рассматривается та же комбинационная схема варианта 3, что и в лабораторной работе №1. Структура схемы описывается соотношениями:

```
F1 = NAND(x1, x2)
F2 = NOT(x4)
F3 = NOR(F2, x6)
F4 = AND(x3, F3)
F5 = OR(F1, F4)
```

Схема имеет шесть входов, пять логических элементов и один выход $F5$. Для данной структуры рассматривается 22 характеристические константные неисправности типа stuck-at.

4. Реализация D-алгоритма

Программная реализация D-алгоритма выполнена на языке Python. Используется представление схемы в виде ориентированного графа и набор процедур, реализующих построение и пересечение D-кубов. Ниже приведён обобщённый псевдокод:

```
def d_algorithm(circuit, fault):
    # Фаза 1: выбор примитивного D-куба неисправности
    if fault.pole in circuit.inputs:
        return d_algorithm_for_input_fault(circuit, fault)

    primitive_cubes = build_primitive_d_cubes(gate, fault.stuck_at)

    for prim_cube in primitive_cubes:
        # Фаза 2: D-проход
        result = d_drive(circuit, prim_cube.copy())
        if result:
            return result
    return None

def d_drive(circuit, cube):
    # Пропагация d/D к выходам схемы
    while not cube.has_output_d(circuit.outputs):
        for gate in circuit.gates:
            if gate has d/D on input:
                d_cubes = build_d_cubes(gate)
                cube = d_intersection(cube, d_cube)

    # Фаза 3: обеспечение консистентности
    return consistency_phase(circuit, cube)
```

Реализация включает также процедуры логического моделирования, перебора вариантов и проверки найденного тестового вектора.

5. Результаты экспериментов

Для исследуемой схемы варианта 3 с использованием D-алгоритма были получены тесты для всех 22 характеристических константных неисправностей. Полученные тесты совпадают по покрытию с результатами, полученными методом активизации одномерного пути, но обладают меньшей мощностью тестового набора.

5.1. Тесты для входных неисправностей

```
x1/0: 110100    x1/1: 010100
x2/0: 110100    x2/1: 100100
x3/0: 001100    x3/1: 000100
x4/0: 001000    x4/1: 001100
x6/0: 001100    x6/1: 001000
```

5.2. Тесты для внутренних неисправностей

```
F1/0: 110100    F1/1: 010100
F2/0: 001100    F2/1: 001000
F3/0: 001100    F3/1: 001000
F4/0: 001100    F4/1: 000100
F5/0: 110100    F5/1: 000000
```

```
Уникальные тесты: 8 тестов для 22 неисправностей
```

Таким образом, D-алгоритм обеспечивает покрытие всех характеристических неисправностей восемью уникальными тестовыми векторами, тогда как при использовании метода активизации одномерного пути было получено 22 теста.

6. Сравнительный анализ методов

Для наглядности сравним основные характеристики двух методов генерации тестов, использованных в лабораторных работах №1 и №2.

Характеристика	**Одномерный путь**	**D-алгоритм**
Характер метода	Детерминированный, по путям	Детерминированный, по кубам
Выбор пути	Вручную, один путь	Автоматически, многомерные пути
Число тестов (для варианта 3)	22 теста	8 уникальных тестов
Сложность реализации	Относительно простая	Существенно более сложная
Степень автоматизации	Ограниченная	Высокая
Минимальность набора	Не обеспечивается	Частично достигается

Из таблицы видно, что D-алгоритм позволяет значительно сократить число тестов при сохранении полного покрытия неисправностей, однако требует более сложной программной реализации и использования кубического представления.

7. Выводы

В ходе выполнения лабораторной работы был изучен D-алгоритм (алгоритм Рота) для автоматической генерации тестов комбинационных цифровых схем. На примере схемы варианта 3 показано, что применение D-алгоритма позволяет сократить тестовый набор с 22 до 8 уникальных тестов при сохранении полного покрытия характеристических константных неисправностей.

D-алгоритм является фундаментальным методом автоматизированного построения тестов и лежит в основе многих промышленных ATPG-инструментов, предназначенных для тестирования цифровых устройств большой степени интеграции. Результаты работы подтверждают целесообразность его использования в задачах тестирования и диагностирования вычислительных систем.