В.Н. Ярмолик

КОНТРОЛЬ И ДИАГНОСТИКА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

В. Н. Ярмолик

КОНТРОЛЬ И ДИАГНОСТИКА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Минск
«Бестпринт»
2019

УДК 681.325.36

Ярмолик, В. Н. Контроль и диагностика вычислительных систем / В. Н. Ярмолик. – Минск : Бестпринт, 2019. – 387 с. : ил. – ISBN 978-985-90509-5-4.

Монография посвящена проблеме тестового диагностирования современных вычислительных систем. Проанализированы основные тенденции автоматизированного построения тестовых последовательностей для программного и аппаратного обеспечения вычислительных систем. Показана ограниченность применения классических методов генерирования тестов и обоснована необходимость развития и использования методов вероятностного и псевдо-вероятностного тестирования, а также различных их модификаций. Излагаются вопросы управляемого и многократного вероятностного тестирования, квазивероятностного и псевдослучайного тестирования. Большое внимание уделяется вопросам исчерпывающего и псевдо-исчерпывающего тестирования, основанных на регулярности тестовых последовательностей.

Предназначена для научных и инженерно-технических работников, занимающихся проектированием и эксплуатацией современных вычислительных систем, а также магистрантов и аспирантов соответствующих специальностей.

Табл. 128 Ил. 75. Библиогр.: 353 назв.

Рекомендована Советом БГУИР, протокол № 12 от 13.03. 2019 г.

Рецензенты:
доктор технических наук, профессор *Бибило П. Н.;*
доктор технических наук, профессор *Татур М. М.*

ISBN 978-985-90509-5-4

Содержание

Введение

Современные вычислительные системы, такие как встроенные системы, системы на кристалле и сети на кристалле, характеризуются большим разнообразием функциональных возможностей, степенью сложности и спецификой применяемых технологий для их разработки и изготовления. Приведенные факторы имеют большое влияние на три составляющие вычислительных систем, такие как программную, аппаратную и запоминающую части системы. Каждая из указанных составляющих системы имеет свои особенности и характерные свойства, которые обусловливают существование достаточно специфичных разновидностей неисправных состояний вычислительных систем в целом. Это вызвало значительные изменения как в процессе проектирования вычислительных систем, так и в разработки методов и средств их контроля и диагностики.

В настоящей монографии основное внимание уделено вопросам контроля и диагностики программного и аппаратного обеспечения современных вычислительных систем, а также их запоминающих устройств с использованием последних достижений в области теории и практики тестового диагностирования.

Монография написана на основании результатов научных исследований, проводимых в рамках НИЛ 3.3 и кафедры ПОИТ Белорусского государственного университета информатики и радиоэлектроники. В предлагаемой книге использованы материалы лекций, практических и лабораторных занятий, а также научных семинаров, проводимых на кафедре ПОИТ под руководством автора настоящей монографии и с непосредственным его участием. Большая часть излагаемого материала была получена автором в результате научных исследований совместно с его коллегами и учениками при выполнении различных отечественных и зарубежных научных грантов.

В 1 и 2 главах представлены основные понятия и определения, изложены классические вопросы автоматизированного построения тестов как для программного и аппаратного обеспечения вычислительных систем, так и для их запоминающих устройств. Подробно рассматривается многообразие первопричин возникновения неисправных состояний вычислительных систем. Анализируются различные подходы к классификации неисправностей систем и их математические модели, которые используются для построения тестовых последовательностей.

Глава 3 посвящена определению классического вероятностного тестирования систем и рассмотрению его основных свойств где показывается его доминирующее положение по отношению к другим методологиям как для построения тестов, так и для реализации тестовых процедур вычислительных

систем. Подробно излагаются достоинства и недостатки вероятностного тестирования, и показывается принципиальная возможность достижения максимально возможной эффективности вероятностных тестов. Последующие главы в той или иной степени являются дальнейшим развитием классического вероятностного тестирования либо представляют методы, которые близки по своей сути к вероятностным тестам, или являются его аппроксимациями.

В последующих 4 и 5 главах подробно проанализированы современные методы вероятностного тестирования, такие как управляемое вероятностное тестирование и его многократное применение в виде многократных вероятностных тестов с изменяемыми параметрами. Материал четвертой главы посвящен исследованию оптимальных управляемых вероятностных тестов, а также управляемым тестам малой длины. Пятая глава целиком посвящена исследованию многократных вероятностных тестов и оценке их эффективности. Содержание указанных глав и всех последующих разделов монографии в большей своей части представляет авторский материал, опубликованный в различных периодических изданиях.

Новому и многообещающему методу тестирования, основанному на применении так называемых квази-случайных последовательностей, посвящена 6 глава. Предложенные в данной главе решения позволяют генерировать широкий спектр тестовых последовательностей с различными их свойствами, такими как равномерность, случайность, детерминированность, а также многообразными значениями численных характеристик квазислучайных последовательностей. Широкие возможности для решения задач тестирования современных вычислительных систем обеспечивают, так называемые, последовательности Соболя, а также модифицированные последовательности Соболя.

В главе 7 рассматриваются основы псевдослучайного тестирования, важной особенностью которого является повторяемость тестового эксперимента при сохранении всех основных достоинств вероятностного тестирования. Показано, что доминирующее место в части используемых псевдослучайных тестовых последовательностей занимают *M*-последовательности. Приводятся методики синтеза генераторов псевдослучайных тестовых последовательностей и генераторов комбинированных тестовых последовательностей. Анализируются свойства и численные характеристики псевдослучайных последовательностей, показана их близость к истинно случайным последовательностям.

Главы 8, 9 посвящены исследованию как свойств, так и методик применения исчерпывающих и псевдо-исчерпывающих тестовых последовательностей для тестирования современных вычислительных систем. Даны определения и граничные оценки исчерпывающих тестов, показана их связь с

почти исчерпывающими и вероятностными тестовыми последовательностями. Обоснована относительная простота их построения и реализации, а также высокая эффективность обнаружения неисправностей систем. Большое внимание в главе 9 уделено использованию *M*-последовательностей для реализации псевдо-исчерпывающего тестирования систем.

В главе 10 кратко изложены основы методов компактного тестирования, приведены основные их свойства и методика оценки их эффективности. Показано, что основу методов компактного тестирования составляют алгоритмы и методики генерирования тестовых последовательностей, рассмотренные в предыдущих разделах монографии. В качестве метода сжатия выходных реакций тестируемых систем выделен метод сигнатурного анализа, который широко применяется в практике реализации процедуры самотестирования вычислительных систем. Детально анализируется достоверность сигнатурного анализа и методология проектирования сигнатурных анализаторов. Изложены основы адаптивного сигнатурного анализа и показаны его широкие возможности при тестировании современных вычислительных систем и, в особенности, их запоминающих устройств.

Последующие четыре главы посвящены анализу и развитию существующих методов тестирования запоминающих устройств современных встроенных вычислительных систем. Показывается, что безальтернативным решением является применение неразрушающих маршевых тестов, реализуемых в рамках процедур самотестирования запоминающих устройств, которые характеризуются свойством сохранения исходного содержимого памяти после процедуры тестирования. Глава 11 целиком посвящена анализу неразрушающего тестирования и особенностям его применения, в том числе, в рамках адаптивного сигнатурного анализа. Следующая 12 глава сконцентрирована на использовании свойства симметрии маршевых тестов для синтеза неразрушающих тестов запоминающих устройств. Глава 13 представляет решения по реализации многократного тестирования памяти, а глава 14 описывает методологию псевдо-исчерпывающего тестирования современных запоминающих устройств. В качестве модели многократного вероятностного тестирования запоминающих устройств в данной главе предлагается математическая модель собирателя купонов, которая позволяет установить зависимость близости тестовой процедуры к псевдо-исчерпывающему тесту в зависимости от кратности теста.

В главе 15 представлены решения по генерированию исчерпывающих детерминированных тестовых последовательностей и приведены оценки их эффективности. Большое внимание уделяется различным приемам модификации счетчиковых последовательностей. Интересными являются результаты по инвертированию значений разрядов счетчиковых последовательностей, а

также по применению последовательностей, полученных как результат децимации исходной последовательности. Систематически изложен материал по вопросам генерирования тестовых последовательностей с высокой переключательной активностью, к которым относятся последовательности анти-Грея и последовательности с максимальной переключательной активностью.

Содержательная часть книги включает классические вопросы тестового диагностирования современных вычислительных систем, однако основу данной монографии составляют оригинальные результаты, полученные автором, и в большей части опубликованные им совместно с его учениками и коллегами в периодических отечественных и зарубежных научно-технических изданиях. Главы 4, 5, 6, 13 и 14 написаны автором совместно с к.т.н. С. В. Ярмолик, а в главе 5 и 14 использованы результаты, полученные аспирантом кафедры программное обеспечение информационных технологий БГУИР В. А. Леванцевичем.

Автор искренне благодарен рецензентам – доктору технических наук, профессору П. Н. Бибило и доктору технических наук, профессору М. М. Татуру за ценные замечания и своевременные рекомендации, которые способствовали улучшению содержания книги, а также выражает свою признательность заведующей кафедрой Н. В. Лапицкой и всем сотрудникам кафедры программного обеспечения информационных технологий Белорусского государственного университета информатики и радиоэлектроники за помощь и поддержку при подготовке рукописи к изданию.

Глава 1. Основные понятия и определения

1.1. Тестирование, верификация и валидация

Для современных вычислительных систем весьма важными являются различные уровни проверки корректности и правильности их функционирования, реализуемые как в процессе разработки, так и в применении по назначению. Для этих целей служат процедуры *тестирования* (*testing*), *верификации* (*verification*) и *валидации* (*validation*) [61]. Несмотря на кажущуюся схожесть приведенных терминов, тестирование, верификация и валидация означают разные процессы проверки функционирования вычислительных систем.

Под *тестированием* вычислительных систем понимают вид деятельности в процессе их разработки и использования, связанный с выполнением процедур, направленных на обнаружение ошибок, несоответствий, неполноты, двусмысленностей и других некорректных ситуаций при функционировании текущей версии вычислительной системы [143, 161, 184]. Процесс тестирования относится, в первую очередь, к проверке корректности реализации системы, который выполняется путем управляемого функционирования системы с целью обнаружения несоответствий между ее поведением и предъявляемым к ней требованиям.

Понятие верификация вычислительных систем является более общим понятием, чем тестирование. Под *верификацией* понимают процедуру проверки факта того, что вычислительная система соответствует предъявляемым к ней требованиям и реализована без непредусмотренных функций и удовлетворяет проектным спецификациям и стандартам. Процесс верификации включает в себя тестирование функциональности системы, анализ результатов тестирования, формирование и анализ отчетов о выявленных несоответствиях и проблемах. Таким образом, можно считать, что процесс тестирования является составной частью процесса верификации.

Валидация вычислительных систем представляет собой процедуру, целью которой является доказательство того, что в результате разработки системы были достигнуты те цели, которые планировалось достичь благодаря применению системы по ее прямому назначению. Валидация, по сути, является процедурой проверки соответствия системы ожиданиям потребителя, чаще всего заказчика вычислительной системы.

Таким образом, основные задачи процессов верификации и валидации состоят в том, чтобы проверить и подтвердить, что конечный продукт, представляющий собой вычислительную систему, отвечает назначению и удовлетворяет требованиям заказчика и потребителей. Эти два понятия тесно связаны с процессами тестирования и обеспечения качества вычислительных систем [13, 15, 57, 207, 208, 302].

Верификация и валидация основаны на планировании их как процессов для систем в целом, так и для наиболее критичных элементов вычислительных систем, таких как компоненты, интерфейсы (программные, технические

и информационные) с использованием различных тестов и тестовых процедур. После проверки отдельных компонентов системы проводятся их интеграция и повторная верификация и валидация интегрированной вычислительной системы. Схематично, в упрощенном виде, взаимосвязь между процедурами тестирования, верификации и валидации можно представить следующей диаграммой, приведенной на рис. 1.1. На более низком уровне выполняется процедура тестирования, затем следует верификация и завершающей является валидация. Получение отрицательного результата при тестировании требует изменений в реализации системы. Негативный результат верификации требует коррекции функциональности системы, а валидация, по сути, отвечает за правильность формулировки требований к системе.

Понятия верификации и валидации тесно связаны с процессами тестирования и обеспечения качества современных вычислительных систем. К сожалению, интерпретация понятий верификация и валидация в разных источниках существенно отличается, их часто путают, хотя отличия между ними достаточно значимы. Очень часто эти два понятия отождествляются, а иногда вообще игнорируются и заменяются понятием тестирование.

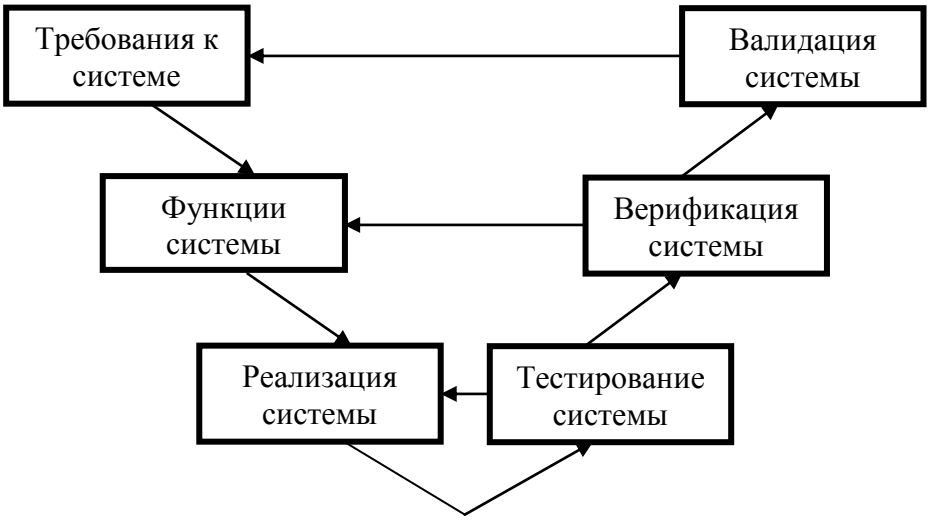


Рисунок 1.1 – *Тестирование, верификация и валидация*

В тоже время верификация и валидация отличаются тем, что они позволяют получить ответы на разные вопросы, как для разработчиков системы, так и для ее потребителей. Кроме того, они проводятся на различных этапах жизненного цикла системы, а также отличаются методологией своей реализации.

Ключевые отличия между верификацией и валидацией приведены в таблице 1.1.

Если рассматривать приведенные процедуры с точки зрения вопроса, на который необходимо получить ответ, то тестирование отвечает на вопрос «*Как это сделано*?» или «*Соответствует ли поведение разработанной вычислительной системы требованиям*?».

Таблица 1.1 – *Соотношения между верификацией и валидацией*

№	Верификация	Валидация
1	Делаем ли мы правильно вычислительную систему?	Делаем ли мы правильную вычислительную систему?
2	Реализована ли вся функциональность системы?	Правильно ли реализована функциональность системы?
3	Верификация проводиться раньше и включает проверку правильности написания документации, кода и т. д.	Валидация происходит после верификации и, как правило, отвечает за оценку системы в целом
4	Выполняется на уровне разработчика	Выполняется, в большей части, на уровне заказчика (потребителя)
5	Включает статический анализ, заключающийся в тестировании программных и аппаратных средств, сравнении требований и т.д.	Включает динамический анализ функционирования системы в реальных условиях по ее прямому назначению
6	Основывается на объективной оценке реализуемых вычислительной системой функций	Субъективный процесс, включающий экспертную оценку качества работы системы

В свою очередь верификация отвечает на вопрос «*Что сделано*?» или «*Соответствует ли разработанная вычислительная система требованиям*?». Процедура валидации позволяет получить ответ на вопрос «*Сделано ли то, что нужно*?» или «*Соответствует ли разработанная система ожиданиям заказчика* (*потребителя*)?».

1.2. Неисправность, ошибка и несправное поведение системы

Большинство терминов и понятий в области тестирования и диагностики вычислительных систем имеют достаточно общий смысл и характеризуются отсутствием точности и однозначности. В основном это связано с разнообразием типов современных вычислительных систем, большим количеством различных методов их тестирования, различными уровнями абстракции их описания, а также формулировкой целей и задач тестирования. Рассматривая современные вычислительные системы, такие как *встроенные системы* (*embedded systems*), *системы на кристалле* (*systems-on-a-chip*) и *сети на кристалле* (*nets-on-a-chip*), выделим их основные три составляющие, такие как *программную* (*software*), *аппаратную* (*hardware*) и *запоминающую* (*memory*) части системы [163, 262, 348]. Каждая из указанных составляющих системы имеет свои особенности и характерные свойства, которые обусловливают использование достаточно специфичных разновидностей неисправных состояний системы. Поэтому для последующего обсуждения очень важно однозначно определить следующие термины: *физический дефект* (*physical defect*), *ошибка проектирования* (*mistake*), *неисправность* (*fault*), *ошибка системы* (*error*) и *неисправное поведение* системы (*failure*) [57].

Обычно на самом низком уровне абстракции используется термин *неисправность* как математическая модель, описывающая физические дефекты

системы и ошибки при ее проектировании, изготовлении и использовании. Основными источниками неисправностей являются: *ошибки спецификации* (*specification mistakes*) из-за неправильных или неправильно интерпретированных проектных требований к системе и/или ее спецификации; *ошибки проектирования* (*implementation mistakes*) в основном из-за *ошибок кодирования* (*bugs*); физические дефекты аппаратных компонентов системы (*component physical defects*); *внешние факторы* (*external factors*), такие как условия окружающей среды (температура, электромагнитные и радиационные излучения и др.) или *человеческий фактор* (*human factor*) [163, 57, 82, 87]. Тогда неисправность как математическую модель неисправного состояния вычислительной системы, вызванного вышеперечисленными факторами, определяется следующим образом [57].

Неисправность (*Fault*) – это физический дефект, несовершенство или недостаток, который происходит в аппаратном и программном обеспечении вычислительной системы, а также в ее запоминающих устройствах.

Более полное определение неисправности вычислительной системы формулируется в следующем определении [143, 161, 184, 348].

Определение 1.1. Неисправностью вычислительной системы является такая ее функциональность, которая может приводить к новым выходным значениям (наблюдаемому поведению) и/или к новому состоянию системы, не соответствующим спецификации системы и требованиям, предъявляемым к ней.

Согласно определению 1.1 наличие неисправности вычислительной системы может приводить к ее неверным наблюдаемым выходным значениям и ошибочным внутренним состояниям или только к ошибочным внутренним состояниям. Отметим, что не для всего множества входных данных системы и ее состояний наличие неисправности приводит к ошибочным состояниям системы и ошибочным выходным значениям. Однако, в большинстве случаев, наличие неисправности, как правило, приводит к ошибкам вычислительной систем.

Наиболее компактное и однозначное определение ошибки приведено в [57].

Ошибка (*Error*) – это отклонение полученных результатов от правильных значений или их точности.

Для вычислительных систем определение ошибки приведено в [143, 161, 184, 348].

Определение 1.2. Ошибкой вычислительной системы из-за наличия в ней неисправности является формирование для некоторых подмножеств входных данных и состояний системы одного или более неверных промежуточных результатов, или состояний системы, в том числе и наблюдаемых выходных значений системы, отличающихся от ожидаемых.

Ошибки вычислительной системы, по сути, являются результатом активизации ее неисправностей. Они могут привести к неисправному поведению системы (системному сбою), которое, в свою очередь, может быть

наблюдаемым признаком неисправного поведения (состояния) вычислительной системы. Под неисправным состоянием системы часто понимают отказ системы, который определяемый следующим образом [57].

Отказ (*Failure*) – это невыполнение системой какой-либо предопределенной функциональности, или получение ею результата отличного от ожидаемого значения.

Определение 1.3. Неисправным поведением вычислительной системы (отказом, либо сбоем вычислительной системы) называется формирование наблюдаемых выходных значений системы, отличных от ожидаемых, для некоторого множества ее состояний и входных значений.

Неисправное поведение вычислительной системы возникает как результат активизации и транспортировки ошибочного значения (ошибки) системы на ее наблюдаемые выходы.

Приведенные определения в равной степени применимы как к программному, так и аппаратному обеспечению вычислительных систем, а также к различным видам их запоминающих устройств. Обоснованность и применимость этих определений первоначально проиллюстрируем примером из программного обеспечения [7].

Рассмотрим программу, которая вычисляет остаток от деления на три, возведенного в квадрат (*data***2), входного значения *data* (рис. 1.2).

```
begin
        read (data);
                data: = data*2;   (← неисправность)
                data: = data mod 3;
        write (data)
 end.
```

Рисунок 1.2 – *Пример программы, содержащей неисправность*

В связи с наличием неисправности, из-за ошибки (*mistake*) программиста, в третьей строке кода программы эта программа фактически вычисляет остаток от деления на три удвоенного значения *data*, так как при кодировании ошибочно вместо операции возведения в степень использовалась операция умножения. Рассмотрим поведение приведенного кода программы для трех значений *data*, а именно 2, 3 и 4.

При рассмотрении программы, представленной выше, для исходного значения *data* = 2 выходной результат оператора *data*: = *data**2, содержащего неисправность, составляет 4, что соответствует правильному результату. Таким образом, наличие неисправности для случая входных данных *data* = 2 не приводит к ошибочному промежуточному значению (ошибке) и тем более к неисправному наблюдаемому поведению. То есть, программа функционирует корректно с точки зрения как промежуточных значений, так и выходного результата, равного 1.

Для исходного значения *data* = 3 выходной результат оператора *data*: = *data**2, содержащего неисправность, составляет 6, тогда как ожидаемое значение, при отсутствии неисправности, должно быть 9. Таким образом, возникает вычислительная ошибка. Однако для *data* = 3 программа вычисляет ожидаемо правильный результат, а именно 0, что свидетельствует об отсутствии ее наблюдаемого неисправного состояния. Это означает, что значения *data*, равные 2 и 3, не вызывают неисправного состояния (сбоя программы), так как значение *data* = 2 даже не вызывает ошибку, а значение *data* = 3 инициирует ошибочный промежуточный результат, равный 6, вместо 9. Но в обоих случаях формируется правильное выходное значение программы, равное 0.

Для *data* = 4 программа, содержащая неисправность в третьем операторе (рис. 1.2), формирует ошибочное значение (вычислительную ошибку). Действительно, в этом случае выходное значение равняется 2 вместо ожидаемой величины 1, что свидетельствует о неисправном состоянии программы и приводит к ее неисправному поведению.

В качестве второго примера, иллюстрирующего обоснованность ранее приведенных определений для неисправности ошибки и неисправного поведения вычислительной системы, рассмотрим комбинационную схему, приведенную на рис. 1.3.

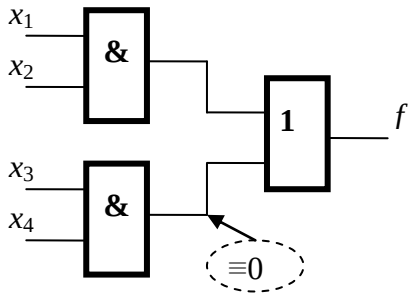


Рисунок 1.3 – *Пример комбинационной схемы, содержащей неисправность*

В приведенной схеме в силу физического дефекта соединения выхода второго двухвходового элемента 2И со вторым входом элемента 2ИЛИ возникла константная неисправность тождественный 0 (≡0). Наличие подобной неисправности характеризуется формированием на данном полюсе постоянного уровня логического 0, независимо от выходного значения второго элемента 2И. Таким образом, приведенная схема неисправна и содержит неисправность тождественный 0 [207, 208, 302].

Формирование, например, входных значений $x_1\,x_2\,x_3\,x_4$ = 0 1 0 1 не приводит к появлению ошибки в данной схеме, так как выходное значение второго элемента 2И в исправном состоянии схемы и в случае наличия неисправности равняется 0. Цифровая схема для $x_1\,x_2\,x_3\,x_4$ = 0 1 0 1 функционирует согласно своей спецификации, описываемой булевой функцией $f = x_1\,x_2 + x_3\,x_4$. В исправном и неисправном состояниях схемы $f = 0$, что свидетельствует о ее исправном поведении.

При входных значениях схемы $x_1\,x_2\,x_3\,x_4$ = 1 1 1 1 на ее промежуточном полюсе возникает ошибка, так как выходное значение второго элемента 2И равное 0, в силу наличия неисправности, отличается от значения 1, которое должно формироваться при отсутствии данной неисправности. В тоже время выходное значение схемы f = 1 для $x_1\,x_2\,x_3\,x_4$ = 1 1 1 1 соответствует исправному состоянию схемы. В данном случае наличие неисправности повлекло возникновение ошибки, однако не привело к неисправному поведению схемы, приведенной на рис. 1.3.

Применение, например, входных значений $x_1\,x_2\,x_3\,x_4$ = 0 1 1 1 приводит к появлению как ошибки, так и неисправного поведения приведенной схемы. В этом случае выходное значение схемы будет принимать значение 0, которое отличается от случая неисправного поведения схемы, для которого f = 1.

В следующем примере рассмотрим аппаратную составляющую вычислительной системы, представленную запоминающим устройством [331, 348]. Предположим, что память содержит четырехразрядные запоминающие ячейки, которые используются для хранения данных, состоящих из четырех бит. В одной из ячеек памяти возникла константная неисправность ≡0, которая приводит к тому, что, например, во втором разряде ячейки постоянно находится значение 0 [331, 348]. Что касается остальных разрядов ячейки, то их содержимое может быть 0 или 1. Рассмотрим случай шестнадцатеричных, данных равных 4, 3 и 2, которые должны быть записаны в ячейку памяти, содержащую указанную неисправность, и затем считаны. Для значения данных 4 наличие неисправности во втором бите ячейки не приводит к ошибке, несмотря на наличие неисправности ≡0, так как $4_{(16)}$ = 0 1 0 $0_{(2)}$. Оба значения данных 3 и 2 в двоичном представлении (0 0 1 1, 0 0 1 0) содержат 1 во втором бите, что является причиной возникновения ошибки выборки после выполнения операции чтения. Это, в свою очередь, может привести к неисправному состоянию вычислительной системы в целом. Взаимосвязь неисправного поведения системы и его первопричин приведена на рис. 1.4.

Из приведенного выше рисунка видно, что первопричиной неисправностей являются: ошибки спецификации, ошибки реализации, физические дефекты аппаратных компонент системы и внешние факторы [57, 13, 87, 271, 207, 208, 301]. Наличие неисправностей может вызывать ошибки как программных, и аппаратных средств вычислительных систем, так и их запоминающих устройств. Ошибки возникают в результате активизации неисправностей вычислительной системы, как это было показано в приведенных выше примерах. И наконец, активизированная ошибка вычислительной системы может привести к наблюдаемому неисправному поведению вычислительной системы.

Существующие методологии построения тестовых последовательностей решают задачу нахождения такого подмножества входных *тестовых воздействий* (*failure patterns*), которые приводили бы к наблюдаемому неисправному поведению вычислительной системы в случае возникновения в ней неисправностей на всех стадиях жизненного цикла системы [57, 82, 87].

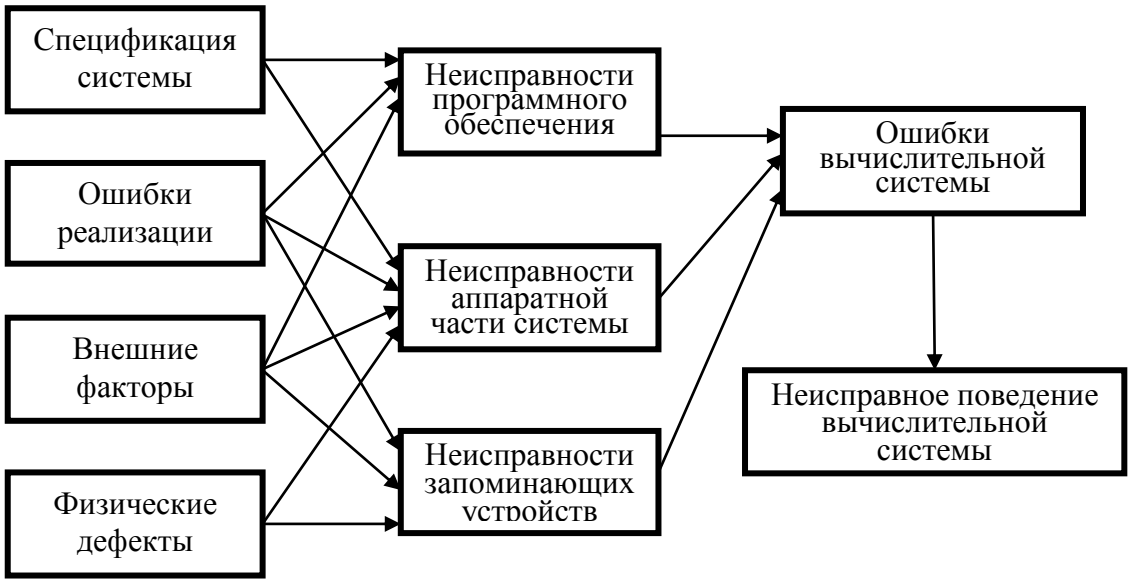


Рисунок 1.4 – *Причинно-следственная связь между неисправностями, ошибками и неисправным поведением вычислительной системы*

Как уже отмечалось ранее, каждая из составляющих частей вычислительной системы характеризуется специфическим множеством моделей неисправностей. Последовательно рассмотрим наиболее часто применяемые модели неисправностей для автоматизированного построения тестов.

1.3. Неисправности программного обеспечения

В русскоязычной и англоязычной современной литературе можно встретить целый ряд синонимов для категории *неисправность* программного обеспечения [11, 40, 57, 75, 80, 115]. Помимо термина *неисправность* (*fault*) довольно часто встречаются такие, по сути, эквивалентные понятия, как: *баг* (*bug*), *ошибка* (*mistake, blunder*), *ошибочное значение* (*error*), *проблема* (*problem*), *дефект* (*defect*), *недостаток* (*flaw*), *дефицит* (*deficiency*), *опечатка* (*erratum*), *небрежность* (*botch*) [11, 40, 57, 75, 80, 115]. Иногда все эти термины обобщенно представляются жаргонным термином – *глюк* [57, 271]. Однако, какой бы термин не был использован суть его остается неизменной, программное обеспечение во все случаях, идентифицированных определенными выше терминами, не соответствует предъявляемым к нему требованиям.

Существует достаточно много классификаций неисправностей программного обеспечения. Прежде всего это связано с тем, что в основе этих классификаций лежат различные критерии и различные уровни их рассмотрения. Часть классификаций носит больше теоретический характер, часть является полезной с практической точки зрения. К сожалению, охватить все возможные классификации – задача довольно сложная, да и вряд ли целесообразная. Остановимся на некоторых, наиболее значимых из них.

В статье *Д. Кнута* (*D. Knuth*) [115], об эволюции системы верстки *TEX*, рассматривается одна из наиболее ранних и часто упоминаемых схем клас-

сификации неисправностей программного обеспечения. Приведенная им категоризация неисправностей охватывает 867 изменений, внесенных в *TEX* при реализации данной системы верстки на языках *SAIL* и *Pascal*. Д. Кнут сам вносил изменения и, насколько известно, являлся единственным пользователем созданной им схемы классификации неисправностей. Основная заявленная им цель схемы категоризации заключалась в том, чтобы помочь собственному пониманию автора об обнаруженных неисправностях и обеспечить удобную основу для выявления их особенностей, свойств и причин возникновения.

Схема Д. Кнута используется непосредственно программистом и применяется сразу после обнаружения сбоя программного обеспечения. Выявленная неисправность протоколируется и соотносится с одним из 9 классов неисправностей, помеченных прописными буквами алфавита:

A – неверный, некорректный алгоритм (*algorithm awry*);

B – ошибка или небрежность (*blunder or botch*);

D – искажение (*повреждение*) *структуры данных* (*data structure debacle*);

F – забытая функция (*forgotten function*);

L – причины относящиеся к языку программирования (*language liability*);

M – несоответствие между модулями системы (*mismatch between modules*);

R – надежность, сбой при плохих входных данных (*robustness*);

S – удивительный, необычный сценарий функционирования (*surprising scenario*);

T – типичная опечатка (*trivial typo*).

Данные, собранные *Б. Бейзером* (*B. Beizer*), в результате анализа коммерческих проектов, написанных на нескольких языках программирования (*Assembler, Fortran, Ada, COBOL* и *C*), включают в общей сложности 982 неисправности [11]. Основываясь на их природе и особенностях, Б. Бейзером была предложена иная система классификации. Некоторые примеры его системы включают:

- *время появления неисправности* (*time of fault introduction*), например, на уровне спецификации, реализации, или процедуры тестирования;

- *эффекты активации неисправности* (*effects of fault activation*), например, нежелательный поток управления или искажение данных;

- *местоположение* (*location*), например, объект или модуль приложения, который содержит неисправность;

- *тип требуемого корректирующего действия* (*type of required corrective action*), например, требуется, не требуется или неоднозначно.

Д. Грэй (*J. Gray*) в своей работе [75] впервые представил классификацию неисправностей с точки зрения их влияния на надежность программного обеспечения. По сути, им были предложены два множества неисправностей, а именно:

- *сбои* (*transient faults called Heisenbugs*), активизация которых может быть замаскирована, другими словами, они не могут легко воспроизводиться путем повторного исполнения программного приложения;

- *неисправности* (*non-transient faults called Bohrbugs*), активация которых не может быть замаскирована при повторном воспроизведении программного приложения, то есть это легко воспроизводимые неисправности.

Разработчиками фирмы IBM была предложена так называемая *ортогональная классификация неисправностей* (*orthogonal defect classification*) [40]. Предложенная классификация предназначалась для обеспечения обратной связи для программистов на разных этапах разработки программного обеспечения с целью уменьшения количества неисправностей программного обеспечения. Авторами данной системы классификации предусмотрен следующий набор (исключительных) типов неисправностей:

- *функция* (*function*), которая характеризует неправильную функциональность или ее отсутствие, что может потребовать формального изменения проекта;

- *интерфейс* (*interface*) идентифицирует ошибки в общении между пользователем и приложением, между его модулями или драйверами приложения;

- *проверка* (*checking*) характеризуется ошибкой данных в исходном коде;

- *назначение* (*assignment*) включает в себя неисправности адресации, в том числе ошибочную инициализацию;

- *временные/повторяющиеся* (*timing/serialization*) содержат множества неисправностей, которые легко исправляются при улучшенном управлении разработкой проекта;

- *проблемы в повторяемых модулях* (*build/package/merge*) идентифицируют проблемы из-за ошибок в библиотечных системах и при управлении изменениями и контроле версий;

- *документация* (*documentation*) характеризует ошибки в документации и, в первую очередь, в руководстве для пользователей программного продукта;

- *алгоритм* (*algorithm*) включает проблемы связанные с эффективностью или корректностью программного продукта, которые могут быть исправлены путем повторной реализации без необходимости изменения дизайна.

Более практичной является классификация, используемая в ряде *программ-коллекторов* (*bug collector*), или как их принято называть, – системах отслеживания проблем (системах трекинга багов). К сожалению, далеко не все системы трекинга используют одну и ту же классификацию. Однако, благодаря гибкости настроек, в большинстве из них можно с легкостью менять используемые понятия классификации на те, которые более привычны или приняты в компании.

Весьма важной системой классификации неисправностей программного обеспечения является попытка обеспечить однозначность во взаимопонимании между разработчиками программного обеспечения и специалистами по его тестированию.

Классификация, приведенная ниже [271], носит исключительно практический характер, она далека от совершенства но, тем не менее, позволяет обеспечить достаточную степень однозначности как для разработчиков, так и для специалистов по использованию, анализу и тестированию программного обеспечения:

- *аварийные причины* (*causes crash*) объединяющие все те неисправности в программе, которые могут вызвать крах или зависание всей вычислительной системы, существенно нарушить стабильность ее работы;
- *критические* (*critical*) неисправности представляют собой все то, что ведет к зависанию или краху самой программы, не затрагивая вычислительной системы в целом;
- *функциональные* (*functional*) неисправности характеризуются отклонениями в функциональном поведении программного обеспечения по сравнению с требуемымой функциональностью;
- *инсталляционные* (*setup*) – неисправности, внесенные на этапе инсталляции программного приложения;
- *косметические* (*cosmetic*) неисправности объединяют ошибки дизайна (например, не тот цвет линии или шрифт), пользовательского интерфейса и т. д.
- *незначительные* (*minor*) неисправности теоретически малозначимые, либо неисправности не уточнённого генезиса;
- *неисправности желательного типа* (*suggestion*), которые заключаются в предложении об улучшении программного обеспечения в части каких-либо его характеристик, либо свойств. Иногда подобные неисправности обозначаются как особенность (*feature*).

Приведенная классификация является усредненной интерпретацией большого множества классификаций, в том числе и рассмотренных выше.

1.4. Неисправности аппаратной части систем

Проблема тестового диагностирования аппаратной части вычислительных систем в большей мере связана с *цифровыми устройствами* (*digital device*), составляющими, наряду с памятью, их основу [21, 120, 163, 262, 290, 302, 348]. Принято различать несколько видов технического состояния цифровых устройств. Основным состоянием цифрового устройства является исправное состояние, при котором оно удовлетворяет всем требованиям, установленным технической документацией. В противном случае устройство находится в одном из неисправных состояний. Если установлено, что цифровое устройство неисправно, то решается задача поиска неисправности (дефекта) устройства, целью которого является определение места и вида неисправности [302].

Неисправности цифровых устройств появляются в результате применения неисправных составных компонентов, таких как логические элементы, реализующие простейшие логические функции. Кроме того, причиной неисправностей могут быть возникновение разрывов или коротких замыканий в межкомпонентных соединениях, нарушение условий эксплуатации схемы, наличие ошибок при проектировании и производстве и ряд других факторов [302].

Общепринятым подходом в области диагностики аппаратных средств вычислительных систем является представление их неисправных состояний в виде обобщающих моделей неисправностей на логическом и функциональном уровне. Следует отметить, что отображение из физической области на логический и функциональный уровни облегчает как процедуру построения тестов, так и процесс обнаружения неисправных состояний цифровых устройств. Заметим, что сложность такого анализа существенно уменьшается, так как многие физические дефекты можно моделировать одной и той же неисправностью на логическом или функциональном уровне. Кроме этого некоторые неисправности логического или функционального уровня не зависят от технологии изготовления цифровых устройств [207, 290, 302].

Из множества различных видов неисправностей выделяется класс логических неисправностей, которые изменяют логические функции элементов *цифровой схемы* (*digital circuit*). Указанный класс неисправностей занимает доминирующее место среди неисправностей цифровых схем. Для их описания в большинстве случаев используются следующие математические модели: *константные неисправности* (*stuck-at fault*); *неисправности* типа *короткое замыкание* (*short fault*); *инверсные неисправности* (*inverse fault*); *неисправности* типа *задержка сигнала* (*delay fault*); *неисправности* типа *обрыв*, либо *отсутствие контакта* (*open fault*); и *неисправности* типа *перепутывание* [2, 21, 50, 163, 207, 208, 250, 290, 301, 302, 348].

Наиболее общей и часто применяемой моделью логических неисправностей являются одиночные *константные неисправности* (*single stuck-at fault*). Различают два вида таких неисправностей, а именно константный ноль (тождественный ноль, или ≡0) и константная единица (тождественная единица, или ≡1) [120, 290, 302]. В англоязычной литературе эти неисправности обозначаются, соответственно, как *stuck-at* 0 (*s-a*-0) и *stuck-at* 1 (*s-a*-1), что означает наличие постоянного уровня логического нуля или логической единицы на соединении между логическими элементами [302]. По определению, одиночная константная неисправность действует только на соединение между элементами цифровой схемы, при этом считается, что логические элементы функционируют правильно [120]. Каждое соединение логических элементов схемы может иметь одну из двух указанных неисправностей: константу 0 и константу 1 (*s-a*-0, *s-a*-1), которые часто обозначаются как *SF*0 и *SF*1. Наличие константной неисправности фиксирует на данном полюсе постоянное значение сигнала 0 или 1, независимо от значения, подаваемого на

нее сигнала. Часто такие неисправности моделируют замыкания соединений схемы на землю (*s-a*-0) или источник питания (*s-a*-1).

Физический дефект цифрового устройства может моделироваться и *кратной константной неисправностью* (*multiple stuck-at fault*), при которой несколько соединений логических элементов принимают постоянное значение 0 или 1 одновременно. Отметим, что число кратных константных неисправностей растет экспоненциально с числом неисправных соединений (полюсов), составляющих данную неисправность. Для схемы с N полюсами имеется $2N$ одиночных константных неисправностей и 3^N-1 всевозможных кратных константных неисправностей, включая и одиночные. Каждый из N полюсов может быть в исправном состоянии, либо иметь неисправность *s-a*-0 или *s-a*-1. Очевидно, что для подмножества из m фиксированных полюсов имеется 3^m-1 кратных константных неисправностей. Число таких подмножеств определяется соотношением:

$$\binom{N}{m} = \frac{N!}{m!(N-m)!}. \qquad (1.1)$$

Неисправности *типа короткое замыкание* (*мостиковые неисправности*) возникают при коротком замыкании входов и выходов логических элементов [120, 290, 302]. Такие неисправности в англоязычной литературе обозначаются как *short fault* либо *simple bridging fault* [16, 120, 163]. В зависимости от вида применяемых логических элементов возможно различное действие неисправности на цифровую схему. Так, возникновение мостиковой неисправности между полюсами x_1 и x_2 трехвходового элемента 3И (рис. 1.5) эквивалентно фиктивному включению двухвходового элемента 2И, объединяющего указанные полюса [302].

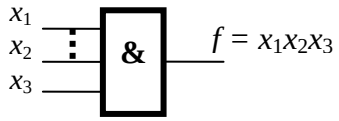


Рисунок 1.5 – *Пример схемы, содержащей мостиковую неисправность*

Подобное действие мостиковой неисправности справедливо для элементов, использующих положительную логику функционирования, и в случае неисправного элемента 3И не изменяет логическую функцию $f = (x_1\, x_2)\, x_3$, реализуемую данным элементом. В случае применения элементов с отрицательной логикой функционирования действием мостиковой неисправности, возникшей между входными полюсами логических элементов, является фиктивное включение двухвходового элемента 2ИЛИ, наличие которого изменяет функцию $f = x_1\, x_2\, x_3$, реализуемую элементом 3И, на функцию $f = (x_1 + x_2)\, x_3$ [207, 302].

Неисправности типа *короткое замыкание* подразделяются на два вида: неисправности, вызванные коротким замыканием входов логического элемента (*simple bridging fault*), и неисправности типа *обратная связь* (*feedback bridging fault*) [302]. Для *комбинационной цифровой схемы* (*combinational digital circuit*), описываемой выражением $f(x_1, x_2, ..., x_n)$, короткое замыкание ее μ входных полюсов приводит к появлению μ-кратной неисправности типа короткое замыкание. Случай двукратной неисправности, вызванной коротким замыканием входов схемы для положительной логики, приведен на рис. 1.5. На рис. 1.6 и 1.7 показаны два примера логической неисправности, вызванной коротким замыканием выходного полюса схемы с ее двумя входами [302]. Существование подобной связи между выходным полюсом схемы и ее входами может привести или к возникновению процесса генерирования высокочастотного сигнала, или к преобразованию комбинационной схемы в последовательностную схему (*sequential circuit*) [120, 302]. Так, для схемы, приведенной на рис. 1.6, короткое замыкание между выходным полюсом f и входами x_3 и x_4, для $x_3 = x_4 = 1$, приводит к преобразованию рассматриваемой комбинационной схемы в последовательностную схему. Независимо от входных значений x_1 и x_2 на выходе схемы будет генерироваться значение единицы.

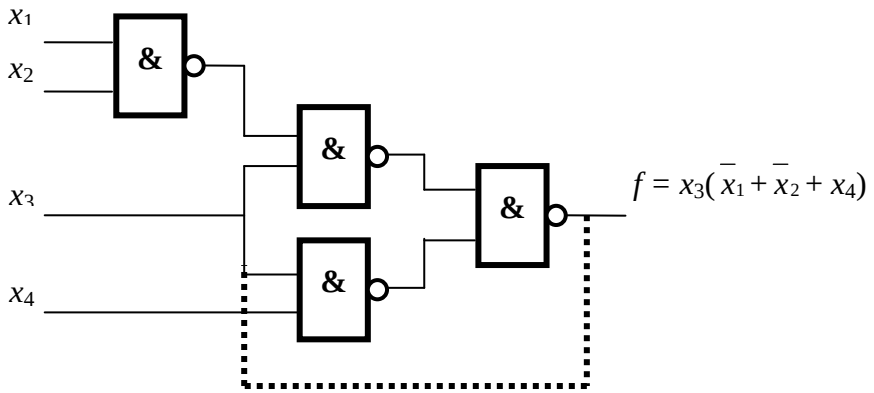


Рисунок 1.6 – *Мостиковая неисправность, преобразующая комбинационную схему в последовательностную схему*

В то же время короткое замыкание между выходом f и входами x_1 и x_2 при $x_1 = x_2 = x_3 = 1$ для схемы (рис. 1.7) приводит к генерированию высокочастотного колебательного сигнала.

В результате проведенных исследований показано, что любая мостиковая неисправность *неизбыточной цифровой схемы* (*non-redundant combination circuit*) может быть обнаружена. В работе [108] приводятся условия построения универсального теста для обнаружения всех константных неисправностей на входных и выходных полюсах цифровой схема, а также обнаружения возможных мостиковых неисправностей между входами и выходами. Понятие неизбыточности является весьма существенным при проектиро-

вании как вычислительных систем в целом, так и комбинационных устройств. В случае комбинационных схем определение неизбыточности формулируется следующим определением [120]:

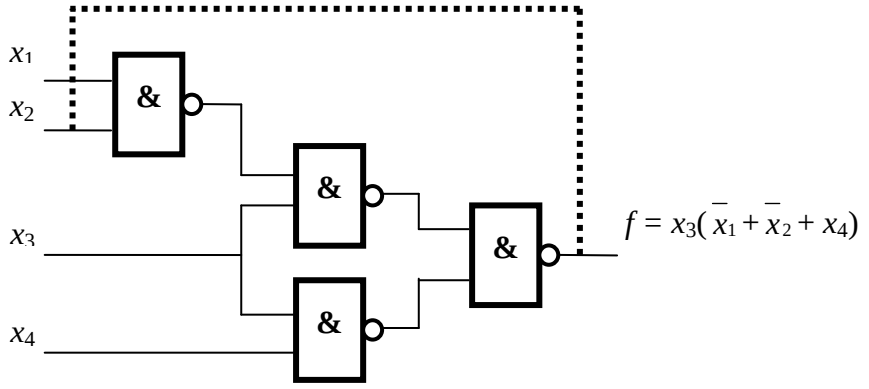


Рисунок 1.7 – *Мостиковая неисправность, приводящая к генерированию высокочастотных колебаний*

Определение 1.4. Под неизбыточной комбинационной схемой понимают схему, которая в неисправном состоянии описывается булевой функцией, отличной от функции, соответствующей исправному ее состоянию.

Инверсные неисправности описывают физические дефекты цифровых комбинационных схем, приводящих к появлению фиктивного инвертора по входу либо выходу логического элемента, входящего в данную схему. Инверсные неисправности в совокупности с константными неисправностями в ряде случаев используются для построения полной модели неисправных модификаций цифровой схемы [286]. В этом случае всевозможные неисправности описываются только моделями константных и инверсных неисправностей.

Правильное функционирование цифрового устройства возможно только в том случае, когда временные интервалы распространения сигналов вдоль путей логической схемы лежат в заданных пределах. Когда время распространения сигнала выходит за эти пределы, говорят, что имеет место *неисправность типа задержки сигнала*. В подавляющем большинстве эти неисправности приводят к увеличению времени распространения сигналов [99]. Наиболее распространенными и рассматриваемыми в литературе моделями неисправности задержки являются: *неисправность задержки перехода* (*transition delay fault*) и *неисправность задержки пути* (*path delay fault*) [36, 118, 291]. Очевидно, что неисправность задержки пути более точная, так как предполагает исследование задержек всех путей распространения сигналов от всех входов комбинационной цифровой схемы ко всем ее выходам. По причине сложности данной задачи при работе с современными цифровыми устройствами, когда число путей экспоненциально зависит от сложности схем, тестирование всех путей практически невозможно. На практике для генерации тестов рассматривается небольшое число путей [99, 291].

Неисправности типа отсутствие контакта (*open faults*) чаще всего описывают действительно отсутствие контакта либо соединения между элементами цифрового устройства. Основная проблема с данным типом неисправностей заключается в том, что их наличие может преобразовать комбинационную схему в последовательностную, что существенно усложняет процедуру построения тестов для исходной комбинационной схемы. Как правило, подобные неисправности предполагают построение так называемых *двух векторных тестов* (*two-pattern tests*) [140]. Изучение подобного типа неисправностей связано с активным использованием *КМОП* (*CMOS*) технологии для изготовления современных цифровых устройств. Некоторые физические дефекты в *КМОП* технологии не могут быть представлены константными неисправностями [2, 21, 50, 163, 290]. Основная причина заключается в том, что *МОП* комбинационные схемы не всегда остаются комбинационными при некоторых физических дефектах. Наиболее распространенными являются следующие виды дефектов в *МОП* технологии: обрыв и замыкание транзисторов; обрывы между стоком, истоком и затвором; короткие замыкания: исток - сток, затвор – сток, затвор – исток [33, 104, 290].

Неисправности типа перепутывание заключаются в перепутывании связей логических элементов схемы и вызываются ошибками, возникающими при проектировании и производстве цифровых устройств [302]. Такие неисправности изменяют функции, которые должно выполнять цифровое устройство. Как правило, количество неисправностей данного типа достаточно велико, а процедура их поиска сводится к процедуре поиска константных, мостиковых и инверсных неисправностей [302].

Существует ряд моделей неисправностей, которые характеризуются определенной технологией производства цифровых устройств и их архитектурных особенностей. Особенности физических дефектов современных вычислительных систем и их математические модели даны в работах [2, 21, 50, 57, 163, 290, 304, 305], причем результаты, приведенные в [84, 110, 111], относятся к одному классу систем, а именно к системам, построенным на *программируемых логических матрицах* (*field programmable gate arrays – FPGA*).

1.5. Модели неисправностей запоминающих устройств

Причиной неисправного состояния *запоминающих устройств* (ЗУ) является наличие физического или механического дефекта либо множества подобных дефектов, количество и многообразие которых практически неограниченно. В зависимости от технологических особенностей при производстве памяти и внешних факторов при ее эксплуатации могут появляться новые типы и разновидности дефектов [162, 303, 331]. Определение факта возникновения дефекта и его классификация представляются весьма трудоемкой и зачастую неразрешимой задачей. Это прежде всего объясняется тем, что возникновение дефекта чаще всего можно определить лишь по косвенным при-

знакам, как правило, по факту неправильного функционирования запоминающего устройства [70, 73, 74, 162, 303].

Для аналитического описания неисправных состояний памяти вычислительных систем, и в первую очередь *оперативных запоминающих устройств* – ОЗУ (*random access memory – RAM*), используются математические модели неисправностей [70, 73, 74, 162, 303], так или иначе отражающие реальные физические дефекты запоминающих устройств.

Первые модели неисправностей запоминающих устройств были предложены в начале 1980-х годов, которые включали *константные неисправности*, *неисправности дешифратора адреса*, *неисправности взаимного влияния* и *кодочувствительные неисправности* [70, 73, 74, 162, 303, 331]. Функциональные неисправности ЗУ подразделяются на два подмножества: неисправности матрицы *запоминающих элементов* (ЗЭ) и неисправности электронного обрамления. Второе подмножество включает неисправности дешифраторов адреса и неисправности логики чтения/записи. Доминирующее значение имеют неисправности матрицы запоминающих элементов памяти, которые часто называют *ячейками* (*cells*) памяти [73, 303, 331].

К неисправностям матрицы ячеек ЗУ в первую очередь относят неисправности, в которых участвуют: одна ячейка; две ячейки; несколько ячеек, в общем случае более чем две, без ограничений на их количество [73].

Данная классификация характерна для всего множества разновидностей запоминающих устройств. Рассмотрим более подробно неисправности матрицы запоминающих элементов, используя их общепринятую классификацию [70, 73, 74, 162, 303, 331].

К неисправностям, затрагивающим одну ячейку ЗУ, относят [73]:

1. *Константные неисправности* (*stuck-at faults – SAF*). Неисправный запоминающий элемент памяти постоянно находится в состоянии логического нуля (*s-a*-0) или логической единицы (*s-a*-1), независимо от операций, выполняемых с неисправным элементом и другими элементами ЗУ. Аналогично, как и для случая произвольной логики, различают однократные и многократные константные неисправности.

2. *Переходные неисправности* (*transition faults – TF*). Подобные неисправности характеризуются невозможностью перехода состояния неисправного запоминающего элемента из 0 в 1 (*TF*↑) или из 1 в 0 (*TF*↓) при выполнении соответствующих операций записи [48]. Данный тип неисправности достаточно близок по своей сути к константным неисправностям. Действительно, если ячейка, имеющая переходную неисправность, оказывается в состоянии, из которого она не может перейти в другое, ее поведение повторяет поведение ячейки, содержащей константную неисправность [73].

Среди неисправностей, в которых участвуют две ячейки ЗУ, выделяют следующие неисправности [73, 162, 303, 331].

3. *Неисправности взаимного влияния* (*coupling fault – CF*). При описании данной неисправности выделяют *влияющую ячейку* (*aggressor cell*), определяемую ее адресом *i*, изменение логического состояния которой воздей-

ствует на состояние *зависимой ячейки* (*victim cell*) с адресом *j*. Различают три типа неисправностей взаимного влияния:

а) *инверсные неисправности взаимного влияния* (*inverse coupling faults – CFin*). При наличии данной неисправности изменение значения b_i влияющей ячейки вызывает инвертирование значения b_j зависимой ячейки. Возможны следующие виды неисправностей *CFin*: $\wedge(\uparrow,\overline{b_j})$, $\wedge(\downarrow,\overline{b_j})$, $\vee(\uparrow,\overline{b_j})$, $\vee(\downarrow,\overline{b_j})$. Символ $\wedge$ и символ $\vee$ задают взаимное расположение влияющего и зависимого запоминающих элементов памяти. Первый символ $\wedge$ означает, что ЗЭ с меньшим адресом влияет на ЗЭ с большим адресом ($i < j$), а символ $\vee$ используется в случае, когда адрес влияющего ЗЭ больше адреса зависимого ЗЭ ($i > j$);

б) *неисправности взаимного влияния прямого действия* (*idempotent coupling faults – CFid*). При изменении значения b_i влияющего ЗЭ происходит принудительная установка определенного логического значения 0 или 1 в зависимом ЗЭ. Различают восемь неисправностей прямого действия: $\wedge(\uparrow,0)$, $\wedge(\uparrow,1)$, $\wedge(\downarrow,0)$, $\wedge(\downarrow,1)$, $\vee(\uparrow,0)$, $\vee(\uparrow,1)$, $\vee(\downarrow,0)$ и $\vee(\downarrow,1)$. При исследовании эффективности тестов запоминающих устройств анализируется их покрывающая способность для всех 12 неисправностей взаимного влияния, в которых участвуют две ячейки (2-*coupling faults*);

в) *статические неисправности взаимного влияния* (*state coupling faults – CFst*). Переход зависимой ячейки в какое либо состояние b_j возможен при определенном значении b_i влияющей ячейки. Возможно восемь неисправностей *CFst*: $\wedge(0,0)$, $\wedge(0,1)$, $\wedge(1,0)$, $\wedge(1,1)$, $\vee(0,0)$, $\vee(0,1)$, $\vee(1,0)$ и $\vee(1,1)$.

4. *Кодочувствительные неисправности* (*pattern sensitive faults – PSF*) рассматриваются как обобщение моделей неисправностей взаимного влияния. Для подобных неисправностей логическое состояние или изменение логического состояния одного ЗЭ ЗУ может зависеть от содержимого (0 или 1) или от логических переходов из 1 в 0 или из 0 в 1 влияющих ЗЭ ЗУ. В случае кодочувствительной неисправности *PSFk*, в которой участвует *k* запоминающих элементов ЗУ, подразумевается, что влияющими ЗЭ в предельном случае могут быть любые $k - 1$ из N запоминающих элементов ЗУ, а зависимым один из оставшихся $N - k + 1$ ЗЭ. Такие неисправности называются *неограниченными* (*unrestricted*) кодочувствительными неисправностями. Очевидно, что практическая значимость подобной модели невысока, так как сложность теста памяти и время его реализации при такой интерпретации кодочувствительной неисправности превышает всякие возможные пределы для современных значений емкости ЗУ. Поэтому при рассмотрении кодочувствительных неисправностей вводятся ограничения как на количество ЗЭ *k*, так и на их местоположение.

При тестировании современных вычислительных устройств, как правило, придерживаются модели *граничных кодочувствительных неисправностей* (*neighborhood pattern sensitive faults – NPSF*) как более реальной модели кодочувствительных неисправностей. Для таких моделей первым и необходи-

мым ограничением является количество ЗЭ, участвующих в неисправности. Как правило, k не превышает 10, это следует из того факта, что для тестирования подобных неисправностей необходимо время, пропорциональное величине 2^k.

Вторым ограничением является физическое соседство ЗЭ. При этом зависимый запоминающий элемент называется *базовым* ЗЭ (*base cell*), или базовой ячейкой, которая в данном случае является аналогом жертвы, а остальные $k - 1$ ЗЭ *соседними* ЗЭ (*neighborhood cells*), или соседними ячейками, выступающими в роли агрессоров, количество и местоположение которых может быть произвольным. Поэтому на практике обычно используют модели кодочувствительных неисправностей *NPSFk* с числом k, равным 3, 5 и 9, конфигурации и обозначение которых приведены на рис. 1.8.

Как видно из рис. 1.8, в каждой из приведенных неисправностей в явном виде выделяется базовый запоминающий элемент – b и соседние запоминающие элементы n. Соседние запоминающие ячейки ЗУ являются физическими соседями по отношению к базовому запоминающему элементу.

Существуют и другие конфигурации *NPSFk* как по взаимному расположению ЗЭ, так и по их количеству. Наиболее распространенной разновидностью таких неисправностей является *NPSFk*, для которых все k запоминающих элемента принадлежат одному столбцу или одной строке матрицы ЗЭ [73], а максимальное значение k равняется 25 [42]. Подобное описание таких неисправностей является идеальным с точки зрения решения проблемы тестирования ЗУ. Однако на практике физическое месторасположение запоминающих элементов ЗУ и их логические адреса в основном не совпадают и на уровне пользователя являются недоступной информацией [42, 43, 44]. Как правило, логически соседние ЗЭ физически располагаются удаленно друг от друга, и наоборот физически соседние ЗЭ имеют адреса, удаленные друг от друга в адресном пространстве ЗУ. Причиной данного факта является ряд технологических приемов, используемых при производстве ЗУ [64, 73].

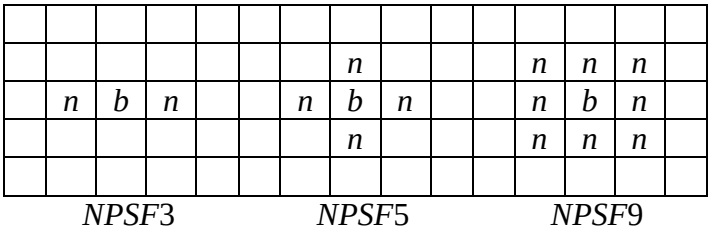


Рисунок 1.8 – *Модели кодочувствительных неисправностей*

В зависимости от эффекта влияния на базовый ЗЭ различают несколько классических типов кодочувствительных неисправностей *NPSFk* [73, 303, 331].

Пассивными кодочувствительными неисправностями (*passive NPSF – PNPSF*) являются неисправности, при которых состояние базового ЗЭ не может быть изменено для определенного кода в $k - 1$ соседних ЗЭ запоминающего устройства.

Под *активными кодочувствительными неисправностями* (*active NPSF – ANPSF*) понимают неисправности, в которых базовый ЗЭ изменяет свое состояние из-за изменения кода в соседних ЗЭ. Изменение кода для подобных неисправностей происходит в результате изменения состояния на противоположное только в одном соседнем элементе, в то время как остальные ячейки сохраняют предыдущее состояние.

Статические кодочувствительные неисправности (*static NPSF – SNPSF*) характеризуются тем, что для определенной комбинации значений в соседних ЗЭ состояние базового ЗЭ принудительно устанавливается в состояние 0 или состояние 1. Главным отличием статических неисправностей от активных кодочувствительных неисправностей является длительность процесса установления неверного значения в базовой ячейке. Для статических неисправностей это время существенно больше [73].

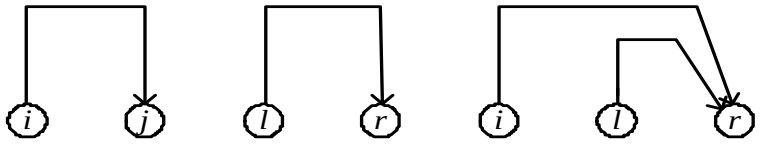
В качестве объекта исследования чаще всего рассматриваются пассивные кодочувствительные неисправности (*PNPSFk*), где k обозначает количество произвольных ЗЭ ЗУ емкостью N бит, участвующих в конкретной неисправности, один из которых является базовым, а $k-1$ остальных – соседними элементами. Отметим, что результаты, полученные для *PNPSFk*, легко обобщаются и для других классов кодочувствительных неисправностей, в силу того что *PNPSFk* является наиболее трудно обнаруживаемой моделью неисправностей ЗУ, покрывающей другие виды неисправностей [73, 303, 331].

Выделяют k различных классов *PNPSFk* в зависимости от местоположения в адресном пространстве памяти базового элемента по отношению к соседним ЗЭ. Например, в случае $k = 5$ существует пять следующих классов *PNPSF*5: $b_{i0}\, n_{i1}\, n_{i2}\, n_{i3}\, n_{i4}$; $n_{i0}\, b_{i1}\, n_{i2}\, n_{i3}\, n_{i4}$; $n_{i0}\, n_{i1}\, b_{i2}\, n_{i3}\, n_{i4}$; $n_{i0}\, n_{i1}\, n_{i2}\, b_{i3}\, n_{i4}$; $n_{i0}\, n_{i1}\, n_{i2}\, n_{i3}\, b_{i4}$, где адреса ячеек имеют следующее соотношение $i0 < i1 < i2 < \ldots < i4$. Каждый класс включает две *PNPSFk* в зависимости от состояния базового элемента. В соседних ячейках возможны любые из 2^{k-1} двоичных наборов, каждый из которых определяет конкретную неисправность. Тогда общее количество возможных *PNPSFk* для памяти емкостью N бит определяется согласно выражению [331]:

$$Q(PNPSFk) = 2k2^{k-1}\binom{N}{k} = k2^{k}\binom{N}{k}. \qquad (1.2)$$

Емкость современных запоминающих устройств, как правило, велика и является существенно большей по сравнению с количеством ячеек k, участвующих в кодочувствительной неисправности. Поэтому всегда справедливо следующее неравенство $N >> k$, которое позволяет оценить количество *PNPSFk*, определяемое соотношением (1.2). Тогда для $k = 3$ получим, что $Q(PNPSF3) \approx N^3$, для $k = 5$ – $Q(PNPSF5) \approx N^5$, а для $k = 9$ – $Q(PNPSF3) \approx N^9$. Для $N = 10^9$ приведенные оценки принимают нереально большие значения, показывающие сложность проблемы синтеза тестов запоминающих устройств для обнаружения кодочувствительных неисправностей.

В структуре классических моделей неисправностей запоминающих устройств *связные неисправности* (*linked faults*) занимают одно из самых видных мест в силу сложности их обнаружения тестами памяти [71, 72, 73, 131, 303, 311, 331]. Под связными неисправностями понимают неисправности различных типов, которые включают в себе общие ячейки. Пример подобной неисправности приведен на следующем рис. 1.9 [303].



а) *б*) *в*)

Рисунок 1.8 – *Пример связной неисправности*

На рис. 1.9, *а* и рис. 1.9, *б* представлены две неисправности взаимного влияния $\langle\uparrow, 1\rangle$ и $\langle\uparrow ,0\rangle$ причем адреса ячеек *i, j, l* и *r,* участвующих в этих неисправностях расположены по возрастанию $i < j < l < r$. В случае, когда $j = r$ две неисправности взаимного влияния, имеющие общую ячейку жертву, трансформируются в связную неисправность (рис. 1.9, *в*). Возможны любые сочетания типов и подтипов неисправностей, участвующих в конкретной связной неисправности. Их многообразие и количество всегда являлись препятствием для анализа их обнаружения конкретным тестом.

Рассмотренные неисправности являются классическими и наиболее часто используются при тестировании памяти вычислительных систем [73, 303, 331]. Однако существует и ряд других моделей неисправностей, также обсуждаемых и применяемых при тестировании памяти. В первую очередь здесь необходимо отметить *неисправности дешифратора адреса* (*address decoder faults – AF*), или *адресные неисправности* [73]. Однако как было показано ранее, обнаружение подобных неисправностей обеспечивается тестами, обнаруживающими неисправности матрицы запоминающих элементов [73].

Также к электронному обрамлению в большей мере относятся неисправности типа обрыв (*stuck open fault – SOF*), которые также могут быть обнаружены при тестировании матрицы запоминающих элементов [43, 44, 64, 73, 252, 254].

Логика чтения/записи обеспечивает интерфейс между массивом ячеек и внешней шиной данных. Данный функциональный блок может содержать те же типы неисправностей, что и массив ячеек памяти (*SAF*, *TF*, *CFin*, *CFid*). Как показано в ряде работ [73], для обнаружения неисправностей логики чтения/записи нет необходимости разрабатывать специальные тесты, в силу того что тесты, обнаруживающие неисправности массива ячеек памяти, обнаруживают и соответствующие неисправности логики чтения/записи.

Модель типа *разрушающая операция чтения* (*read disturb fault – RDF*) практически аналогична неисправности *слабый запоминающий элемент*

(*weak cell fault – WCF*) и обнаруживается путем внесения в классический маршевый тест временной задержки перед выполнением следующего маршевого элемента [73]. Аналогично проявляют себя и неисправности, характерные для динамических запоминающих устройств (*pre-charge faults – PCF*) [24,48,50].

Иным типом неисправности логики чтения/записи является неисправность ошибочная запись (*false write through fault – FWTF*), которая применяется для описания дефектов в статическом запоминающем устройстве [73, 303, 331]. Для динамических запоминающих устройств характерна неисправность сохранения данных (*data retention fault – DRF*), которая так же, как и неисправности *RDF* и *WCF*, обнаруживается путем внесения временных задержек в тест памяти [73].

Существуют и другие модели неисправностей памяти современных вычислительных систем, которые еще в большей мере зависят от технологии ее изготовления, их технических характеристик и внешних условий применения. В то же время наличие многообразия моделей неисправностей памяти не решает проблемы классификации ее дефектных состояний. Различают только два вида дефектов, а именно: *глобальные дефекты* (*global defect*) и *точечные дефекты* (*spot defect*). На практике очень сложно соотнести конкретный физический дефект памяти с одной из моделей его неисправностей. Для этих целей применяется *индуктивный анализ* (*inductive fault analyses*), позволяющий соотносить физические дефекты памяти с их математическими моделями.

В случае, когда физический дефект запоминающего устройства неизвестен, его отображение на множество возможных моделей неисправностей содержит большую степень неопределенности [73].

В таблице 1.2 представлены данные по 1192 кристаллам ЗУ, содержащим физические дефекты [73]. Данные, приведенные в последней таблице, свидетельствуют о невозможности описания многообразных и зачастую непредсказуемых физических дефектов памяти ограниченным числом их математических моделей.

Таблица 1.2 – *Распределение неисправностей*

Модель неисправности	Количество	%
Stuck-At Faults (*SA*)	714	59,8
Stuck-Open Faults (*SOF*)	169	14,1
Data Retention Faults (*DRF*)	26	2,10
Multiple Access Fault (*AF*)	18	1,50
State Coupling Faults (*CF*)	9	0,75
?	–	21,5

Неадекватность классических математических моделей неисправностей предопределила появление так называемых обобщающих моделей неисправностей.

1.6. Обобщающие модели неисправностей

В предыдущих разделах приведены основные математические модели неисправностей вычислительных систем, представленные тремя множествами. А именно, неисправности программного обеспечения, неисправности аппаратной составляющей системы и неисправности ее запоминающих устройств. Характерной особенностью приведенных разновидностей неисправностей является их широкое разнообразие и чрезвычайно большое количество, что не позволяет строить тестовые процедуры для обнаружения каждой из обозначенных неисправностей. В области технической диагностики современных вычислительных систем используются различные приемы, позволяющие существенно уменьшить как разновидность рассматриваемых неисправностей при построении тестовых процедур, так и их количество. Последовательно рассмотрим некоторые из них [21, 50, 208, 290, 302].

Первоначально выделим класс *необнаруживаемых неисправностей* (*non-detectable faults*), которые невозможно обнаружить с применением тестовых процедур, основанных на проверке правильности логики функционирования системы и ее составных компонент [206, 331]. Примером подобных неисправностей могут быть неисправности аппаратной части вычислительных систем, наличие которых приводит к эквивалентному преобразованию алгоритмов, либо функций реализованных цифровым устройством. Один из примеров подобных неисправностей приведен на рис. 1.5 для случая положительной логики, реализуемой элементом 3И. Действительно, при наличии данной неисправности функционирование элемента 3И не отличается от случая его исправного состояния [331].

Выделяют две основные причины появления необнаруживаемых неисправностей, первая из которых определяется технологическими особенностями производства цифровых устройств, что иллюстрируется примером, приведенным на рис. 1.5. Вторая причина связана с наличием избыточности в цифровых устройствах, которая в основном связана с невозможностью решить задачу оптимизации структуры цифрового устройства, в силу ее сложности, либо отсутствия необходимости в такой оптимизации. Поэтому, как правило, цифровые устройства содержат избыточность в виде избыточных модулей, элементов и соединений между ними. Простейший пример избыточной комбинационной схемы и эквивалентной ей неизбыточной схемы приведен на рис. 1.10.

Как видно из приведенного рисунка, наличие константной неисправности *s*-*a*-1 на втором входе элемента 2И-НЕ, например, как результат отсутствия соединения с входом элемента, невозможно определить путем формирования входного тестового воздействия. Действительно, для активизации

данной неисправности необходимо чтобы $x_2 = 0$, так как только в этом случае возникнет ошибка на втором входе элемента 2И-НЕ. Однако $x_2 = 0$ всегда определяет правильное значение функции $f = \overline{x_1}\, x_2$, реализуемой комбинационной схемой, независимо от значения формируемого на втором входе элемента 2И-НЕ. Отметим, что не для всех технологий обрыв входного соединения эквивалентен генерированию по данному входу единичного значения.

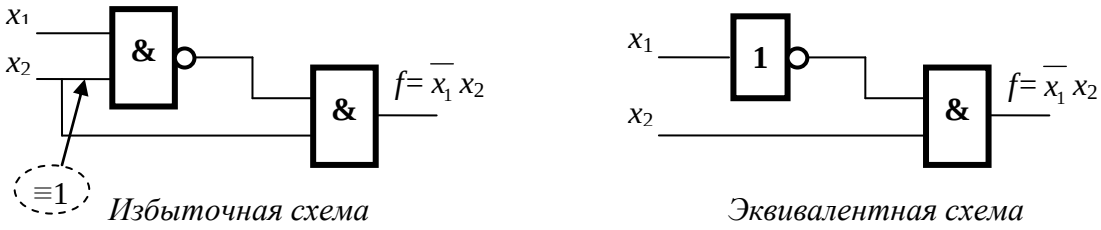


Рисунок 1.10 – *Избыточная комбинационная схема и ее эквивалентная реализация*

Вторым примером существенного уменьшения рассматриваемых неисправностей является использование так называемых *эквивалентных неисправностей* (*equivalent faults*) [50, 286, 289]. Для случая комбинационных цифровых устройств под эквивалентными неисправностями понимают неисправности удовлетворяющие следующему определению [120, 207].

Определение 1.5. Две неисправности α и β комбинационного цифрового устройства называются эквивалентными, если булевы функции, описывающие оба неисправных состояния устройства, эквивалентны.

Отметим, что приведенное определение расширяется как в части количества неисправностей, так и в части объектов тестирования, которые могут включать и программные средства вычислительных систем. Простейшие примеры эквивалентных одиночных константных неисправностей, приведены в таблице 1.3.

Важным свойством эквивалентных неисправностей является эквивалентность тестов и тестовых процедур для их обнаружения. Например, для случая элемента И с n входами тестовый набор $x_1\, x_2\, x_3 \ldots x_n = 1\ 1\ 1\ \ldots\ 1$ обнаруживает любую из его эквивалентных неисправностей приведенных в таблице 1.3.

Таблица 1.3 – *Эквивалентные одиночные константные неисправности*

Элемент И	*Элемент* ИЛИ	*Элемент* И-НЕ	*Элемент* ИЛИ-НЕ
*s-a-*0 на всех полюсах элемента	*s-a-*1 на всех полюсах элемента	*s-a-*0 на входных полюсах и *s-a-*1 выходном полюсе	*s-a-*1 на входных полюсах и *s-a-*0 выходном полюсе

Существенное влияние на количественное сокращение рассматриваемых неисправностей оказывают так называемые *доминирующие неисправно-*

сти (*dominant faults*), которые соответствуют следующему определению [50, 120, 302].

Определение 1.6. Неисправность α называется *доминирующей неисправностью* по отношению к неисправности β, если все множество тестов, обнаруживающих неисправность α, является подмножеством тестов, обнаруживающих неисправность β.

Таким образом, неисправность β может быть исключена из рассмотрения при построении тестов, так как любой тест обнаруживающий неисправность α будет обнаруживать и неисправность β. Очень важным следствием приведенного определения является его справедливость для случая, когда одна неисправность является доминирующей по отношению к некоторому подмножеству неисправностей. Так же, как и определение 1.5, приведенное определение справедливо, как для случая большего количества неисправностей, над которыми доминирует рассматриваемая неисправность, так и для различных объектов тестирования, которые могут включать и программные средства вычислительных систем. Простейшие примеры доминирующих одиночных константных неисправностей приведены в таблице 1.4.

Таблица 1.4 – *Доминирующие одиночные константные неисправности*

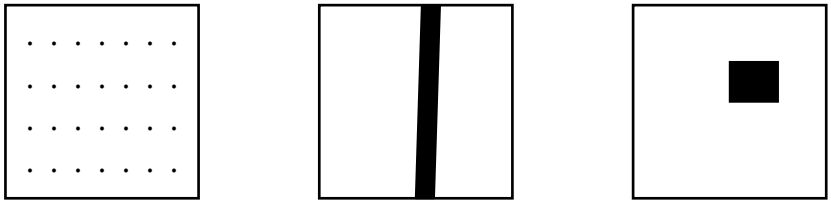
Элемент И	*Элемент* ИЛИ	*Элемент* И-НЕ	*Элемент* ИЛИ-НЕ
s-*a*-1 на выходе доминирует над *s*-*a*-1 по любому входу	*s*-*a*-0 на выходе доминирует над *s*-*a*-0 по любому входу	*s*-*a*-0 на выходе доминирует над *s*-*a*-1 по любому входу	*s*-*a*-1 на выходе доминирует над *s*-*a*-0 по любому входу

Из приведенной таблицы следует, что, например, в случае элемента ИЛИ тестовый набор 0 0 0 … 0 1 0 … 0, обнаруживающий *s*-*a*-0 по одному из его входов, автоматически обнаруживает и неисправность *s*-*a*-0 по выходу данного элемента.

Весьма оригинальным и заслуживающим внимания подходом является полное абстрагирование от неисправностей вычислительных систем и ее составных частей. В данном случае предлагается рассмотрение множества входных тестовых воздействий, которые приводят к наблюдаемому неисправному поведению систем. Очевидно, что данные множества являются уникальными для каждого из объектов тестирования и зависят от многих факторов (см. рис. 1.4) и, в первую очередь, от его архитектуры и функциональности. В тоже время структуры входных тестовых воздействий характеризуются рядом закономерностей, отмеченных в последних современных публикациях [29, 172, 173, 176, 197].

Большой объем эмпирических и экспериментальных исследований, в первую очередь, различного рода программного обеспечения показал обобщенность (структурированность) входных воздействий (наборов), которые инициируют неисправное поведение систем, т. е. являются тестовыми воздействиями [29, 172, 176, 197]. В основополагающей работе [29] была пред-

ставлена классификация обобщенных входных тестовых воздействий. Их три категории: *точечные тестовые наборы* (*point patterns*), *узкополосные тестовые наборы* (*strip patterns*) и *блочные тестовые наборы* (*block patterns*). Для иллюстрации данной классификации пространство входных наборов рассматривается как двухмерное пространство, хотя на практике пространство входных воздействий может иметь произвольную размерность. Тогда для двухмерного случая три категории входных тестовых наборов имеют простую геометрическую интерпретацию (рис. 1.11) [29, 172, 176, 197].



а) *б*) *в*)

Рисунок 1.11 – *Типовые входные тестовые воздействия*

На рис. 1.11 области, выделенные черным цветом, соответствуют входным воздействиям, приводящим к наблюдаемому неисправному поведению вычислительной системы, а фигуры квадратов соответствуют областям входных воздействий [29, 176, 201]. Соответственно, точечные тестовые наборы (рис. 1.11, *а*) соответствуют уникальным входным наборам либо небольшим их количествам, распределенным по всему пространству входных наборов, причем их распределение в большинстве случаев имеет регулярную структуру. Узкополосные тестовые наборы (рис. 1.11, *б*) характеризуются непрерывными множествами входных воздействий имеющих вид узких полос. Тестовые наборы блочного типа (рис. 1.11, *в*) представляют собой одну либо несколько непрерывных областей входных воздействий, приводящих к наблюдаемому неисправному поведению системы.

Как правило, эти типовые входные воздействия выбираются в качестве моделей входных тестовых воздействий, поскольку многочисленные эмпирические исследования подтверждают целесообразность их использования в качестве приближения к реальным входным тестовым воздействиям [29, 172, 176, 201]. Несмотря на то, что эти модели не являются неким стандартом для всего множества вычислительных систем, их применение, а также применение их комбинаций оказывается весьма эффективным при построении тестовых последовательностей для реальных вычислительных систем [29, 172, 176, 201].

Значительная часть аппаратной составляющей вычислительных систем, особенно систем на кристалле и встроенных систем, включает в себя встроенную память, которая, в соответствии с прогнозами будет занимать доминирующую часть на кристаллах вычислительных систем. Поэтому эффективное

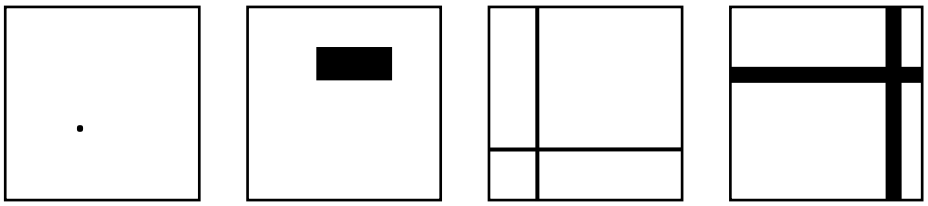
тестирование памяти и совершенная методология анализа неисправностей запоминающих устройств будет способствовать повышению надежности и выхода годных встроенных систем и систем на кристалле, особенно при ускоренных процессах их разработки и применении передовых технологий изготовления [21, 163, 331].

Причинно-следственный подход может быть использован для прогнозирования неисправного поведения запоминающих устройств, в случае возникновения физических дефектов. Неисправное поведение памяти представляется растровым изображением физического топологического представления результатов тестирования, показывающим расположение неисправных запоминающих ячеек. Чаще всего возникают одиночные неисправности запоминающих ячеек памяти; неисправности групп запоминающих ячеек; неисправности дешифратора адреса ячейки памяти; неисправности шин данных и другие неисправности запоминающих устройств [197, 176, 331].

Согласно устоявшейся классификации неисправностей запоминающих устройств различают два их множества, а именно: простые, в которых участвуют одна либо две ячейки памяти, и сложные, включающие в себя множество ячеек [73, 331]. К неисправностям, затрагивающим одну ячейку, относятся константные неисправности и переходные неисправности, а к неисправностям, в которых участвуют две ячейки, – неисправности взаимного влияния [73, 331]. В обоих случаях обобщающей моделью входных тестовых воздействий (адресов ячеек памяти) является точечная модель (рис. 1.12, *а*) либо, для случая многократных неисправностей данных типов, блоковая (рис. 1.12, *б*) [7, 143]. Кодочувствительные неисправности, неисправности дешифратора адреса и неисправности типа *разрушающая операция чтения* и типа *ошибочная запись* покрываются моделью узкополосных входных воздействий (рис. 1.12, *в*) [73, 173, 197, 331]. В случае комбинации физических дефектов памяти, а также их многократных разновидностей входные тестовые воздействия описываются комбинированной моделью (рис. 1.12, *г*), объединяющей блоковый и узкополосный типы [173].

Таким образом, все множество моделей неисправного поведения запоминающих устройств описывается четырьмя обобщенными моделями входных тестовых воздействий, представленных на рис. 1.12 [173].

В сравнении с входными тестовыми наборами для программного обеспечения тестовые воздействия запоминающих устройств в большинстве случаев имеют реальную двухмерную структуру в силу физического двухмерного представления матрицы запоминающих ячеек [73]. В данном случае входными тестовыми данными являются адреса запоминающих ячеек, состоящие из адресов по горизонтальной и вертикальной оси матрицы ячеек памяти.



а) *б*) *в*) *г*)

Рисунок 1.12 – *Типовые входные тестовые воздействия запоминающих устройств*

Как видно из приведенного анализа, типовые входные тестовые воздействия для программного обеспечения вычислительных систем и для большей части аппаратных средств вычислительных систем описываются аналогичными моделями, что, очевидно, позволяет использовать единый подход для тестирования вычислительных систем в целом. Следующим важным выводом анализа типовых входных тестовых воздействий является их разнородность по структуре (см. рис. 1.10 и 1.11), что затрудняет выбор наиболее эффективного метода для тестирования вычислительных систем.

Глава 2. Методы автоматизированного построения тестов

2.1. Виды процедур тестирования

В настоящее время существует большое многообразие критериев классификации тестовых процедур современных вычислительных систем по их видам [13, 15, 57, 60, 61, 82, 87, 271, 274]. Приведем наиболее часто встречающиеся критерии и признаки классификации и соответствующие им виды процедур тестирования [271, 274].

По ***степени знания*** *вычислительной системы* различают тестирование *черного ящика* (*black box testing*), *тестирования белого ящика* (*white box testing*) и *тестирование серого ящика* (*grey box testing*). Наиболее распространенным из приведенных видов тестирования является метод черного ящика.

Тестирование по методу *черного ящика* основывается на знании только интерфейса вычислительной системы и ее функциональности без деталей реализации системы. Данный вид тестирования, как правило, проводится на уровне заказчика либо потребителя вычислительной системы и позволяет проводить как верификацию системы в целом, так и ее валидацию.

Метод *белого ящика* основывается на полном знании о системе, вплоть до деталей архитектуры системы, принципиальных схем аппаратной части системы, кодов программного обеспечения и форматов данных, а также организации запоминающих устройств на всех уровнях. Подобный вид тестирования в большей своей части входит в процесс проектирования и, как правило, выполняется разработчиками системы.

Тестирование по методу *серого ящика* по своей сути включает два предыдущих метода и является методом, который позволяет проводить исследование системы как потребителю, который при необходимости может анализировать некоторые детали реализации системы, так и разработчику вычислительной системы.

По ***степени автоматизации*** процедуры тестирования разделяют *ручное* (*manual testing*), *автоматизированное* (*automated testing*) и *полуавтоматизированное* (*semi-automated testing*).

Ручное тестирование полностью выполняется специалистом по тестированию или разработчиком системы без применения специальных средств автоматизации этого процесса.

Автоматизированное тестирование выполняется с помощью отдельно написанных специальных программ, устройств и стендового оборудования, которые выполняя запрограммированные в них алгоритмы и методы тестирования, проверяют работоспособность системы.

Полуавтоматизированное тестирование подразумевает, что автоматизирована лишь малая часть всех тестовых процедур, либо применяются средства, которые частично способны тестировать продукт, но для их работы требуется постоянный контроль со стороны специалиста.

По виду ***позитивности*** разделяют проверку вычислительной системы на предмет правильности ее функционирования в штатном режиме (*позитивное тестирование*) и в режиме, когда система должна адекватно среагировать, если от неё потребуют чего-то неожиданного и ей несвойственного (*негативное тестирование*).

Позитивные тесты (*positive tests*) проверяют факт того, что при подаче вычислительной системе входных параметров, соответствующих требованиям, она функционирует нормально.

Негативные тесты (*negative tests*) проверяют факт того, что система корректно отработает нештатную ситуацию в случае появления неожиданных некорректных входных данных и параметров.

По ***степени подготовленности*** специалиста по тестированию выделяют *тестирование по документации* (*formal testing*), *разведывательное тестирование* (*exploratory testing*) и *интуитивное тестирование* (*ad hoc testing*).

Тестирование по документации проводится в соответствии со специальной документацией, так называемыми, *тест кейсами* (*test cases*), в которых описаны необходимые действия и их регламент для процедуры тестирования.

Разведывательное тестирование предполагает полное отсутствие знаний о тестируемом объекте.

Интуитивное тестирование во многом повторяет разведывательное, но здесь считается, что специалист по тестированию уже работал с тестируемой системой и знает, как она устроена.

Следующим критерием классификации является ***время проведения тестирования*** вычислительной системы в процессе ее разработки.

На каждой фазе разработки системы требуется проверять разные аспекты ее работы и тестировать разную ее функциональность. По времени проведения тестирования выделяют следующие его виды: *альфа тестирование* (*alpha testing*); *бета тестирование* (*beta testing*); *дымное тестирование* (*smoke testing*); *регрессионное тестирование* (*regression testing*); и *приемочное тестирование* (*user acceptance testing*).

Альфа тестирование реализуется разработчиками системы для определения ее соответствия предъявляемым требованиям. Альфа тестирование проводится для того, чтобы выявить и устранить несоответствия спецификации полностью реализованной системы перед ее передачей заказчику.

Бета тестирование системы проводится ограниченным кругом лиц, которые не являются сотрудниками компании и привлекаются со стороны для независимой оценки и свежего взгляда на готовую систему.

Дымное тестирование представляет поверхностный анализ системы на предмет ее способности функционировать и выполнять основные функции. Это тестирование проводится для того, чтобы какой-то модуль системы или всю систему в дальнейшем подвергнуть более тщательному анализу.

Регрессионное тестирование проводится на новой версии системы с целью проверить отсутствие исправленных неисправностей и ошибок, а также отсутствие новых неисправностей и ошибок, которые могут появиться в процессе устранения уже выявленных.

Приемочное тестирование проводится персоналом заказчика для подтверждения того, что он получил заказанный им продукт, который удовлетворяет предъявляемым к нему требованиям.

При тестировании можно проверять какую-то одну компоненту вычислительной системы, либо проверить сразу несколько ее компонент или же всю систему в целом. Соответственно, по ***степени изолированности компонентов*** выделяют: *модульное тестирование* (*unit testing*); *компонентное тестирование* (*component/unit testing*); *интеграционное тестированием* (*integration testing*); *системное тестирование* (*system/end-to-end testing*).

Под *модульным тестированием* понимают проверку минимально функционально автономной составляющей вычислительной системы.

Компонентное тестирование представляет собой тестирование одного отдельного компонента в системе.

Интеграционное тестирование состоит в проверке взаимодействия между различными компонентами системы.

Системное тестирование представляет собой тестирование полностью собранной системы в том виде, в котором она будет поставляться потребителю.

Виды тестирования ещё разделяют на ***статическое*** *тестирование* (*static testing*) и ***динамическое*** *тестирование* (*dynamic testing*).

Статическое тестирование заключается в тестировании системы без исполнения ею своих функций. Система не запускается, а выполняется проверка исходного кода программ, документации, требований и технических обзоров.

Динамическое тестирование – это работа непосредственно с вычислительной системой. Для данного вида тестирования система должна быть сконфигурирована, работоспособна и запущена для выполнения своих функций.

Наибольшее число видов тестирования определяется по ***объекту тестирования***. Наиболее значимыми из них являются: *функциональное тестирование* (*functional testing*); *нефункциональное тестирование* (*non functional testing*); *нагрузочное тестирование* (*load testing*); *тестирование производительности* (*performance testing*); *стрессовое тестирование* (*stress testing*); *тестирование стабильности* (*stability/endurance testing*); *тестирование безопасности* (*security testing*); *тестирование локализации* (*localization testing*); *тестирование интернационализации* (*internalization testing*); *тестирование совместимости* (*compatibility testing*); *тестирование удобства использования* (*usability testing*); *тестирование пользовательского интерфейса* (*user interface testing*).

Приведенный перечень видов тестирования, показывает их большое разнообразие и специфику применения. При реализации всех видов тестирования, в том числе и приведенных выше, используются так называемые *тесты* (*tests, test patterns set*), которые имеют ряд альтернативных названий, таких как: *тестовые наборы* (*test patterns*), *тест-кейсы* (*test cases*) и *наборы тест-кейсов* (*test case suites*). Очевидно, что свойства и основные характеристики тестов, применяемых при различных видах тестирования, определяют как качество самой процедуры тестирования, так и качество разрабатываемых вычислительных систем. Поэтому методы построения тестов являются ключевой проблемой в области тестирования вычислительных систем. Отметим, что ручное неавтоматизированное построение тестов в случае вычислительных систем малоэффективно и практически не применяется на практике. Поэтому дальнейшее внимание сконцентрируем на автоматизированном построении тестов, которое является основой современного тестирования вычислительных систем.

2.2. Автоматизированное построение тестов для программного обеспечения

Тестирование является неотъемлемой частью общей методологии разработки программного обеспечения и характеризуется высокой трудоемкостью и соответственно стоимостью, которая, как правило, достигает более 50 % общих затрат на разработку. Снижение затрат и повышение эффективности тестирования программного обеспечения возможно, в основном, за счет автоматизации процесса тестирования и в первую очередь автоматизации процессов построения тестов [8, 18, 160, 241, 274].

Среди многих проблем и задач по тестированию построение тестов является одной из самых сложных интеллектуальных задач, поскольку качество тестовых последовательностей и трудоемкость их построения оказывают сильное влияние на эффективность разработки программного обеспечения [8, 18, 160, 241, 271, 274].

В общем случае тесты, как важный *артефакт* (*artifact*) программного обеспечения, должны строиться на основании некоторой информации, то есть некоторых других типов артефактов программ [8]. Типы артефактов, которые используются в качестве входной информации для построения тестов, включают в себя: структуру программы и/или исходный код; спецификацию программного обеспечения и/или модели его проектирования; информацию о пространстве данных ввода/вывода и информацию, динамически получаемую в результате выполнения программы. Основываясь на указанных артефактах, все множество методов автоматизированного построения тестов программного обеспечения представляется в виде следующих пяти множеств [8]:

- ***символьное выполнение*** и ***тестирование покрытия структуры программы*** (*symbolic execution and program structural coverage testing*);

- ***построение теста на основе модели*** (*model-based test case generation*);

- ***комбинаторное тестирование*** (*combinatorial testing*);

- ***адаптивное вероятностное тестирование*** (*adaptive random testing*);

- ***поисковое тестирование*** (*search-based testing*).

Методы ***символьного выполнения*** основаны на анализе кода программы для автоматического генерирования тестовых данных программы. В отличие от подходов к генерированию тестовых данных черного ящика, в данном случае используется подход белого ящика, основанный на анализе программы ее исходного кода или двоичного кода. Символьное выполнение использует символьные значения вместо конкретных значений входных данных и представляет значения программных переменных как символьные выражения этих входных данных [112]. Несмотря на то, что методы символьного выполнения вычислительно существенно сложнее, чем другие методы построения тестов, в настоящее время они получили большое внимание исследователей по двум причинам. Во-первых, применение символьного выполнения для крупных программ реального мира требует решения сложных нерешенных теоретических задач с различными ограничениями. Во-вторых, сегодняшние доступные вычислительные мощности, и в частности современные средства для облачных вычислений (*cloud computing*), предоставляют практически неограниченные возможности.

Символьное выполнение использовалось для генерирования тестов различного назначения, однако наиболее известное использование этого подхода заключается в создании тестовых данных для улучшения *покрытия кода* (*code coverage*) программы [24, 25], автоматического построения *нагрузочных тестов* (*load tests*) [239], регрессионных тестов (*regression tests*) [167], тестов для анализа надежности программ (*robustness testing*) [124], а также тестов для баз данных с конфиденциальными тестовыми данными (*data privacy testing*) [76]. Несмотря на свою очевидную эффективность, методы символьного выполнения требуют решения, по крайней мере, трех фундаментальных проблем для современного программного обеспечения, которые заключаются в: чрезвычайно большом числе путей в современных программах (*path explosion*); дивергенции путей (*path divergence*); и наличии сложных ограничений для реализации данного метода (*complex constraints*) [8].

Построение тестов на основе модели представляет собой формальную методологию, которая использует различные модели программных систем для получения тестовых наборов. Акцент ставится на поведенческом уровне программ, иногда также называемым функциональным черным ящиком, который тестирует программу по ее наблюдаемому поведению. Выделяют два основных направления в методологии построения тестов на основе модели: *аксиоматические подходы* (*axiomatic approaches*) и подходы с использованием моделей *конечных автоматов* (*finite state machine − FSM*) [8].

Аксиоматические основы методов построения тестов, использующих модели, базируются на некоторой форме логического исчисления. Наиболее

значимые результаты в данной области проанализированы в работе [66], где также приводится обобщение методов построения тестов по моделям, задаваемым алгебраической спецификацией. В рамках аксиоматических подходов были разработаны различные понятия тестовых гипотез, в частности *регулярность* (*regularity*) и однородность (*uniformity*) [8]. В соответствии с гипотезой о регулярности все возможные значения x для входных тестовых воздействий до некоторой сложности n считаются достаточными для достижения необходимой эффективности теста. В соответствии с гипотезой однородности одно значение для каждого класса входных данных считается достаточным для достижения требуемой полноты покрытия теста [8, 66, 95].

Основой методов построения тестов в соответствии с их моделями в большинстве случаев являются конечные автоматы и различные их модификации. *Конечный автомат* представляет собой систему с конечным множеством состояний, среди которых выделяются начальное состояние и конечные множества воспринимаемых извне стимулов (входных данных), а также формируемых им реакций (выходных данных) [272, 299]. Кроме этого, в конечном автомате определен набор переходов между состояниями, причем каждый переход помечен вызывающим его входным значением (стимулом) и генерируемой выходной реакцией.

Пример конечного автомата с множеством состояний $\{a_0, a_1, a_2\}$, множеством стимулов $\{x_1, x_2\}$ и множеством реакций $\{y_1, y_2\}$ изображен на рис. 2.1.

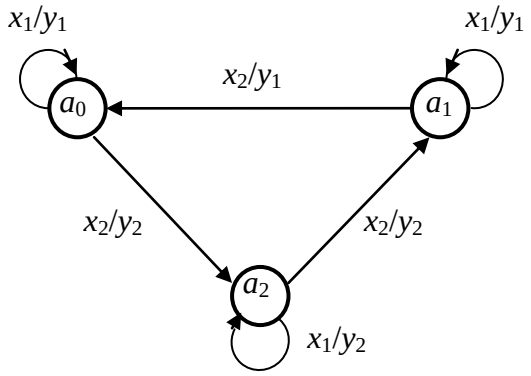


Рисунок 2.1 – *Конечный автомат*

Функционирование конечного автомата осуществляется путем последовательной подачи на его входы стимулов. Тогда автомат, в зависимости от подаваемых стимулов, выполняет переходы, начиная с начального состояния. Выполнение перехода заключается в изменении текущего состояния и выдачи реакции, которой помечен данный переход. Так, если считать начальным состоянием, показанного на рис. 2.1, автомата состояние a_0, то после подачи на его вход последовательности $x_1\ x_2\ x_1\ x_2\ x_1\ x_2$ он перейдет в состояние a_0 и сгенерирует последовательность реакций $y_1\ y_2\ y_2\ y_2\ y_1\ y_1$. Таким образом, конечный автомат реализует некоторое преобразование конечных последова-

тельностей входных стимулов в конечные последовательности выходных реакций.

Одним из обобщений конечных автоматов являются *конечные системы помеченных переходов* (*labeled transition systems − LTS*) [273]. В системе переходов каждый переход помечен только одним символом, а именно символом стимула, либо символом реакции (*input output LTS*), кроме того, он может вообще не иметь метки (выполнение такого перехода никак не проявляется вовне).

Построение тестов на основе модели представляет собой методологию, основанную на различных разновидностях формального описания программных продуктов, среди которых выделяют три основных, а именно: *ориентированные на сценарий* (*scenario-oriented*), ориентированные на состояние (*state-oriented*) и ориентированные на процесс (*process-oriented*) [8].

Формальные описания, ориентированные на *сценарий*, непосредственно описывают входные/выходные последовательности между тестируемой программой и внешней средой [51, 89]. Подобное описание тестируемого объекта (программы) весьма понятно по своей сути и поэтому наиболее часто используется для построения тестов на основании формальных моделей.

Описания, ориентированные на *состояния*, описывают исследуемое программное приложение по реакции на входное воздействие в заданном его состоянии [8, 77, 153]. Подобные описания обычно используют, так называемые *диаграммы состояний* (*state chart*) [153] или текстовые описания (*textual form*) [77].

Нотации, ориентированные на *процесс*, описывают программное приложение в процедурном стиле, где входные воздействия и выходные реакции рассматриваются как сообщения в каналах связи [8, 100, 193]. Примером подобных описаний может быть процедурно-алгебраические языки, такие как *LOTOS* [193], а также языки программирования, которые внедряют различные примитивы каналов связи [100].

Автоматизированное построение тестов для ***комбинаторного тестирования*** основано на выборе входных параметров или конфигураций настройки, которые охватывают заданное подмножество комбинаций элементов программы, подлежащих тестированию. Наиболее распространенной разновидностью комбинаторного тестирования является *комбинаторное взаимосвязанное тестирование* (*combinatorial interaction testing*), где все (*t-way*) комбинации значений параметров (или разновидностей конфигурации) содержатся в тесте.

Основы комбинаторного тестирования происходят из раздела математической статистики, который называется *планирование эксперимента* (*design of experiments*) [41, 62]. На основании идей планирования эксперимента в 1985 году *Мандл* (*Mandl*) использовал комбинации значений параметров в программном обеспечении и описал их ортогональными латинскими квадратами [128]. Развитием этой работы явилась ортогональная система тестирования массивов, *OATS*, в которых использовались ортогональные

массивы для определения комбинаций системы электронной почты AT&T, в которой все пары значений фактора проверяются ровно один раз [20]. Основой результатов, изложенных в работах [20, 128], является латинский квадрат n-го порядка, которым является двумерная матрица $Q = (q_{ij})$ размером $n \times n$, заполненная n элементами множества M таким образом, что в каждой строке и в каждом столбце таблицы каждый элемент из M встречается в точности один раз. Пример латинского квадрата 3-го порядка для множества целых чисел {1, 2, 3} приведен на рис. 2.2.

$$Q = \begin{vmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{vmatrix}.$$

Рисунок 2.2 – *Латинский квадрат*

Два латинских квадрата $Q = (q_{ij})$ и $E = (e_{ij})$ n-го порядка называются ортогональными, если все упорядоченные пары (q_{ij}, e_{ij}) различны. Пример двух ортогональных латинских квадратов и соответствующие им упорядоченные пары элементов для $n = 3$ приведен на рис 2.3.

$$Q = \begin{vmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{vmatrix}, \quad E = \begin{vmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{vmatrix}, \quad \begin{vmatrix} (1,1) & (2,2) & (3,3) \\ (2,3) & (3,1) & (1,2) \\ (3,2) & (1,3) & (2,1) \end{vmatrix}.$$

Рисунок 2.3 – *Ортогональные латинские квадраты*

Применение ортогональных латинских квадратов позволяет систематическим образом формировать, например, всевозможные пары многозначных данных при построении тестовой процедуры. Как видно из рис. 2.3, ортогональные квадраты Q и E позволяют сгенерировать всевозможные пары чисел 1, 2, и 3, причем каждая пара формируется только один раз.

В работах *Коэна* (*Cohen*), посвященных методам формирования тестовых воздействий, было сформулировано ключевое свойство для тестовых наборов, которое в отличие от тестов, построенных на основании ортогональных массивов вместо ограничения *точно один раз* (*exactly once*), использовало ограничение *не менее одного раза* (*at least once*) [45, 88, 186]. В результате была сформулировано понятие так называемых *покрывающих массивов* (*covering arrays − CA*), использование которых гарантировало, что проверяемые параметры программы будут проверены не реже одного раза. Покрывающий массив $CA(t, k, v)$ представляет собой двухмерный $N \times k$ массив, состоящий из N строк и построенный из v параметров таким образом, что любые $t < k$ сочетания из v параметров встретятся в данном массиве не

менее одного раза. В качестве примера на рис. 2.4 приведен покрывающий массив $CA(3, 4, 2)$.

Как видно из данного рисунка, массив содержит $N = 9$ строк и обеспечивает всевозможные $2^3 = 8$ двоичные ($v = 2$) комбинации на любых $t = 3$ из $k = 4$ столбцах матрицы.

0 0 0 1
0 0 1 0
0 1 0 0
0 1 1 1
1 0 0 0
1 0 1 0
1 1 0 1
1 1 1 0
1 0 1 1

Рисунок 2.4 – *Покрывающий массив* $CA(3, 4, 2)$

Понятие покрывающих массивов часто отождествляется с *псевдо-исчерпывающими тестами* (*pseudo-exhaustive tests*) [88, 105, 121, 186, 187], которые являются развитием *исчерпывающих тестов* (*exhaustive tests*) [14], первоначально рассматриваемых только для случая тестирования комбинационных цифровых устройств и оперативных запоминающих устройств. Псевдо-исчерпывающие тесты $B(t, k)$ являются реальной альтернативой исчерпывающим тестам и основываются на формировании множества тестовых наборов в виде покрывающего массива $CA(t, k, v)$ для двоичного случая ($v = 2$). Подобные тесты обеспечивают всевозможные 2^t двоичные комбинации на любых t из k входах тестируемого комбинационного устройства или в любых t из k ячейках запоминающего устройства. Характерной особенностью псевдо-исчерпывающих тестов $B(t, k)$ является то, что их сложность $O(B(t, k))$ существенно меньше по сравнению со сложностью исчерпывающих тестов. Так, для теста $B(t, k) = B(2, 6) = \{000000, 000011, 011100, 101101, 110110, 111011\}$, приведенного в [105], сложность (количество тестовых наборов) $O(B(2, 6))$ псевдо-исчерпывающего теста $B(2, 6)$ будет равна 6, что заметно меньше, чем сложность исчерпывающего теста, которая для $k = 6$ равняется $2^k = 2^6 = 64$. Для общего случая сложность $O(B(N, k))$ псевдо-исчерпывающего теста $B(t, k)$ оценивается неравенством $2^t \le O(B(t, k)) \le 2^k$.

Адаптивное вероятностное тестирование относится к достаточно новому направлению в автоматическом построении тестов и является альтернативой вероятностному тестированию. Также, как и вероятностное тестирование, адаптивное вероятностное тестирование реализует подход черного ящика [126]. В этом случае структура и функциональность объекта тестирования принимаются неизвестной, либо считаются таковой, и никак не

учитываются при формировании тестовых наборов. Более того вероятностное тестирование не использует информацию, которая доступна в процессе генерирования очередного тестового набора. Эта информация может быть получена из предыдущих тестовых наборов и использована при формировании очередного тестового воздействия [126]. Альтернативным развитием вероятностного тестирования также является метод, называемый *анти-вероятностным тестированием* (*anti-random testing*), который по своей сути соответствует адаптивному вероятностному тестированию. Он основан на том, что каждый последующий тестовый набор формируется с использованием некоторой характеристики, либо нескольких характеристик, получаемых на основании предыдущих тестовых наборов [126, 202, 203]. Зачастую эти два вида тестирования отождествляют, хотя более широким понятием считается адаптивное вероятностное тестирование.

Основная предпосылка анти-вероятностного тестирования заключается в том, что в целях достижения более высокого покрытия неисправностей (ошибок), обнаруживаемых тестом, и минимизации его сложности, то есть в повышении эффективности вероятностного теста, необходимо целенаправленно выбирать очередной тестовый набор в зависимости от ранее сгенерированных наборов. Очевидным примером повышения эффективности вероятностного теста является простейшее исключение повторяющихся случайных наборов, которые возможны при реализации случайного тестирования. В общем случае критерием выбора очередного случайного тестового набора является нахождение максимально отличного (максимально удаленного) тестового набора от ранее сгенерированных наборов. Для количественной оценки отличия (расстояния) текущего тестового набора от предыдущих наборов были предложены методы, основанные на применении *расстояния Хемминга* и *декартова* (*евклидова*) *расстояния* [126, 202]. Новый тестовый набор согласно указанным методам выбирался таким образом, чтобы метрики различия принимали максимальное значение [126, 202, 203]. Этот подход оказался более эффективным по сравнению с вероятностным тестированием [126, 202]. К сожалению, основным недостатком анти-вероятностного тестирования является его большая вычислительная сложность. По существу реализация анти-вероятностного метода построения тестов требует перечисления всевозможных входных тестовых воздействий и вычисления расстояния для каждого потенциального кандидата в очередные тестовые наборы [126].

Различные модификации анти-вероятностного тестирования, такие как: *быстрое анти-вероятностное* тестирование (*fast antirandom − FAR*) [130]; *адаптивное вероятностное* тестирование (*adaptive random testing*) [34, 102, 240]; *эволюционное вероятностное* тестирование (*evolutionary random testing*) [188]; *эффективное вероятностное* тестирование (*good random testing*) [30]; *ограниченное вероятностное* тестирование (*restricted random testing*) [31, 32]; *зеркальное вероятностное* тестирование (*mirror random testing*) [119]; *упорядоченное вероятностное* тестирование (*orderly random testing*) [204]; *управляемое вероятностное* тестирование (*control random testing*)

[334, 346, 347], и другие решения также основаны на вычислении характеристик для предыдущих тестовых наборов, и также характеризуются существенной вычислительной сложностью.

Поисковое тестирование основано на использовании программного обеспечения, в котором применяются алгоритмы оптимизации для автоматизации поиска тестовых данных, которые максимизируют достижение целей тестирования, одновременно сводя к минимуму расходы на построение тестов. Подходы поискового тестирования охватывают широкий круг прикладных областей, включая: *эволюционное тестирование*, основанное на программных агентах [146]; *эмпирический подход* к поисковому тестированию [85]; *структурный подход* к тестированию взаимодействия [48]; *мульти-эволюционный подход* к интеграционному тестированию [49]; *высокоуровневый мутационный подход* к формированию тестовых данных [86]; формирования тестовых данных для *стрессового тестирования* [79]; генерирование тестов для *web приложений* [6], а также для *регрессионного* тестирования [196].

2.3. Автоматизированное построение тестов для аппаратного обеспечения

При автоматизированном генерировании тестов для вычислительных систем в общем случае решаются две основные задачи, а именно задача обеспечения наблюдаемости неисправности в точке возникновения и задача ее транспортировки на один из внешних, либо наблюдаемых извне выходов системы. Для этих целей в настоящее время используются различные подходы и методы, реализующие как эвристические, так и строгие формальные методы и подходы.

В случае аппаратного обеспечения, исторически одним из первых подходов, применяемых для генерирования тестов, является метод *активизации одномерного пути* (*one dimensional path sensitizing*) [207, 302]. Сущность его заключается в том, что для транспортировки неисправности от места ее возникновения до одного из выходов цифровой схемы активизируется одномерный путь. Процедура активизации осуществляется в следующей последовательности.

1. Определяется условие, при котором заданная неисправность проявляется в точке ее возникновения. Так, например, для неисправности ≡1 условием наблюдаемости будет обеспечение значения логического нуля для полюса схемы, на котором возникла данная неисправность.

2. Выбирается путь, по которому неисправность транспортируется на выход, который задается в виде последовательности элементов, лежащих на нем.

3. Определяются условия активности выбранного пути в терминах значений входных переменных элементов, образующих его. Для элемента выбранного пути входные переменные задаются таким образом, чтобы его

выходное значение определяется только входом, подключенным к выходу предыдущего элемента, лежащего на данном пути.

4. Вычисляются значения входных переменных цифровой схемы, которые определяют условия наблюдаемости неисправности и ее транспортировки на выход схемы, т. е. получается условие обнаружения заданной неисправности в терминах входных переменных схемы. Результатом вычислений и будет искомый тест.

Пункты 1-3 приведенной процедуры построения теста часто называют *прямой фазой* метода активизации одномерного пути, а пункт 4 – *обратной фазой*. В качестве примера, иллюстрирующего метод активизации одномерного пути, рассмотрим цифровую схему, приведенную на рис. 2.5 [207, 302].

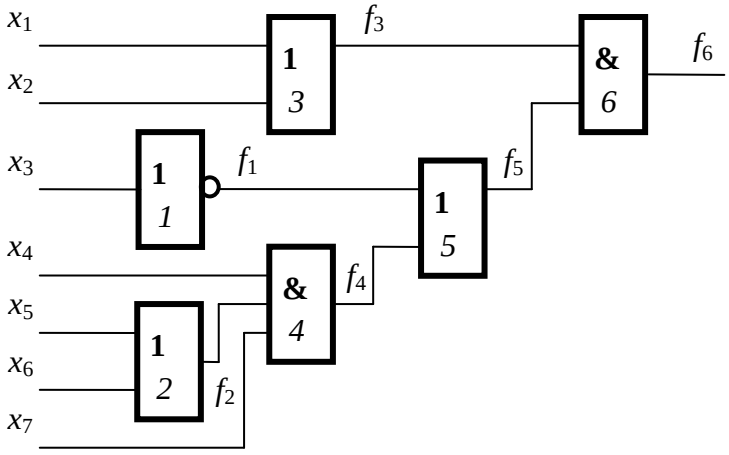


Рисунок 2.5 – *Пример цифровой схемы, иллюстрирующий метод активизации одномерного пути*

Построим для нее тест, позволяющий обнаруживать неисправность ≡1 по выходу элемента *4*. Согласно приведенной процедуре, заданная неисправность будет наблюдаема в месте ее возникновения при выполнении условия $f_4 = 0$ или, учитывая функцию, выполняемую элементом *4*, данное условие можно записать как: $x_4 (x_5 + x_6) x_7 = 0$. Далее выбирается путь, проходящий через элементы *5* и *6* от места возникновения неисправности до выхода схемы. Активность пути через элемент *5* определяется значением $f_1 = 0$ на втором его входе, в этом случае изменение f_4 приведет к изменению f_5. Выполнение равенства $f_3 = 1$ обеспечивает условие активности пути через элемент *6*. В результате применения прямой фазы метода активизации одномерного пути, формируется условие наблюдаемости неисправности в точке ее возникновения и транспортировки на выход в виде системы логических уравнений:

$$f_4 = 0, \quad f_1 = 0, \quad f_3 = 1. \tag{2.1}$$

Решение системы уравнений (2.1) содержательно представляет собой процедуру, выполняемую в процессе обратной фазы метода активизации одномерного пути. С учетом функций, реализуемых элементами схемы, данная система примет вид:

$$f_4 = x_4x_5x_7 + x_4x_6x_7 = 0, \quad f_1 = \overline{x_3} = 0, \quad f_3 = x_1 + x_2 = 1.$$

Уравнениям полученной системы будет удовлетворять множество решений, каждое из которых является тестом, позволяющим обнаруживать заданную неисправность. Действительно, для вектора переменных $x_1\,x_2\,x_3\,x_4\,x_5\,x_6\,x_7$ = 1 1 1 0 1 1 1, представляющего одно из возможных решений, значение f_6 на выходе схемы (рис. 2.5) в случае наличия неисправности ≡1 на полюсе f_4 равняется 1, а в случае ее отсутствия $f_6 = 0$. Это свидетельствует о наблюдаемости неисправности по выходу схемы.

Однако, несмотря на очевидную простоту реализации, метод активизации одномерного пути не нашел широкого практического применения, так как по приведенным процедурам прямой и обратной фазы метода активизации одномерного пути не всегда возможно получить искомый тест. Этот факт иллюстрируется на классическом примере Шнейдера (*Schneider*) показывающем необходимость активизации многомерного пути [171,207, 302].

Метод *активизации многомерного пути* (*multi-dimensional path sensitizing*), предложенный *Ротом* (*Roth*), является, по сути, первым алгоритмическим методом построения тестов для неизбыточных комбинационных цифровых схем. Часто этот метод называют *d-алгоритмом* (*d-algorithm*) Рота и рассматривается как универсальный метод для автоматизации построения тестов цифровых устройств [165, 207, 302,]. Данный метод гарантирует нахождение теста для заданной неисправности, если такой тест существует. Он основан на применении кубического описания булевых функций, которое является одной из возможных форм их задания. Среди кубических представлений булевых функций, используемых в указанном алгоритме, можно выделить так называемые *сингулярные кубы* (*singular cubes*), совокупность которых является *сингулярным покрытием* (*singular cover*) схемы, реализующей булеву функцию [120]. Для задания сингулярных кубов применяются символы 0, 1 и *X*, где $X \in \{0, 1\}$ означает, что переменная по данной координате может произвольно принимать как нулевое, так и единичное значение [120]. Формально сингулярные кубы строятся путем задания минимального количества определенных значений 0 или 1 по входам элемента для получения на его выходе 0 либо 1. По остальным входам задается значение *X*. На рис. 2.6 приведено множество сингулярных кубов для элементов простейшей комбинационной схемы.

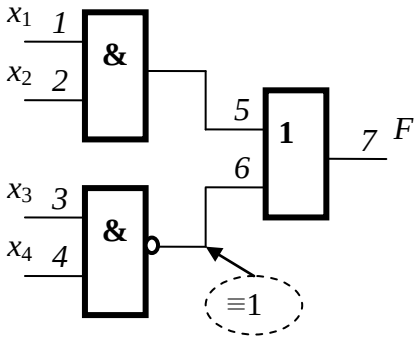
Рассмотренные сингулярные кубы используются для построения *d-кубов* (*d-cubes*), для задания которых применяются символы 0, 1, *X*, *d* и $\overline{d}$ [120]. Символ *d* может принимать значение, как нуля, так и единицы, а $\overline{d}$

противоположное значение. Для конкретной цифровой схемы все символы d принимают одно и то же значение 0 или 1, а $\bar{d}$ соответственно 1 или 0.

d-Куб, описывающий поведение цифровой схемы, получается в результате выполнения покоординатной операции пересечения двух сингулярных кубов из сингулярного покрытия схемы с различным значением по выходной координате на основании соотношений (2.2).

$$\begin{aligned} &0 \cap 0 = 0;\ 0 \cap X = 0;\ X \cap 0 = 0;\ 1 \cap 1 = 1;\ 1 \cap X = 1; \\ &X \cap 1 = 1;\ X \cap X = X;\ 0 \cap 1 = d;\ 1 \cap 0 = \bar{d}. \end{aligned} \qquad (2.2)$$

Так, например, на основании кубов X 0 1 и 1 1 0 из сингулярного покрытия элемента 2И-НЕ (см. рис. 2.6) в результате их покоординатного пересечения получим: $X\ 01 \cap 110 = 1\ d\ \bar{d}$.



2И			2И-НЕ			ИЛИ		
1	*2*	*5*	*3*	*4*	*6*	*5*	*6*	*7*
0	*X*	0	0	*X*	1	1	*X*	1
X	0	0	*X*	0	1	*X*	1	1
1	1	1	1	1	0	0	0	0

Рисунок 2.6 – *Комбинационная схема и сингулярные покрытия ее элементов*

Для описания заданной неисправности используются так называемые *примитивные d-кубы неисправности*, которые состоят из входного набора, позволяющего наблюдать неисправность по выходу элемента, и символа d или $\bar{d}$ в выходной координате. В данном случае символ d означает, что в исправном состоянии по выходной координате должен быть символ 1, а в неисправном 0. В тоже время символ $\bar{d}$ свидетельствует о том, что в исправном состоянии будет 0, а неисправном – 1. Для примера, приведенного на рис. 2.6 и неисправности ≡1 по выходу элемента 2И-НЕ, соответствующий примитивный d-куб будет иметь вид:

3	*4*	*6*
1	1	$\bar{d}$

d-Куб неисправности сравнительно легко строиться для одного элемента схемы, на выходе которого присутствует константная неисправность. Однако основной задачей *d*-алгоритма является построение *d*-куба неисправности всей рассматриваемой схемы. Для этого используется операция *d-пересечения* (*d-intersections* ($\cap_d$)), введенная Ротом с целью формализации процедуры активизации многомерного пути [120, 165, 207, 302]. Применительно к *d*-алгоритму процедура *d*-пересечения двух *d*-кубов $A = (a_1, a_2, \ldots, a_n)$ и $B = (b_1, b_2, \ldots, b_n)$, где $a_i, b_i \in \{0, 1, X, d, \bar{d}\}$, $i = \overline{1,n}$, выполняется по соотношениям (2.3):

1. $X \cap_d b_i = b_i$; $a_i \cap_d X = a_i$.

2. Если $a_i \neq X$ и $b_i \neq X$ то $a_i \cap_d b_i = \begin{cases} a_i, \text{ если } a_i = b_i, \\ \varnothing, \text{ в противном случае.} \end{cases}$ (2.3)

Символ $\varnothing$ означает пустое множество, либо отсутствие результата пересечения. Окончательным результатом *d*-пересечения двух кубов *A* и *B* будет являться куб $A \cap_d B = (a_1 \cap_d b_1, a_2 \cap_d b_2, \ldots, a_n \cap_d b_n)$, если для любого i $a_i \cap_d b_i \neq \varnothing$, в противном случае $A \cap_d B = \varnothing$.

Непосредственно *d*-алгоритм во многом повторяет метод активизации одномерного пути, используя при этом формальный подход построения описания неисправного поведения схемы, который основывается на операции *d*-пересечения [120, 207, 302,]. В случае примера, приведенного на рис. 2.6, описание неисправного поведения схемы, содержащей неисправность $\equiv 1$ по выходу элемента 2И-НЕ, представляется результирующим *d*-кубом:

1	*2*	*3*	*4*	*5*	*6*	*7*
0	*X*	1	1	0	$\bar{d}$	$\bar{d}$

Соответственно, тестом x_1 x_2 x_3 x_4 будет являться один из двоичных наборов, 0 0 1 1 или 0 1 1 1. При подаче на входы схемы любого из указанных наборов в исправном состоянии значение *F* на выходе схемы будет ровняться 0, а при наличии неисправности – 1.

Доказано, что *d*-алгоритм позволяет формировать тест, если такой существует для каждой неисправности комбинационной схемы [207, 302]. Кроме того, данный алгоритм позволяет идентифицировать наличие избыточности цифровой схемы, требует сравнительно небольшого объема памяти и отличается практически реальной трудоемкостью, что позволяет широко использовать его в системах автоматизированного построения тестов [207, 302].

Наличие избыточности в комбинационной схеме позволяет идентифицировать и *булево-дифференциальный* (*boolean differences*) метод построения тестов [207, 302, 174]. Сущность данного метода состоит в использовании

булевых разностей (*булевых частных производных*), которые определяются следующим образом [207, 302].

Булевой частной производной, булевой функции $F(x_1, x_2, \ldots, x_n)$, по булевой переменной x_i, $i \in \{1, 2, \ldots, n\}$ называется булева функция:

$$\frac{\partial F(x_1, x_2, ..., x_n)}{\partial x_i} = F(x_1, x_2, ..., x_i, ..., x_n) \oplus F(x_1, x_2, ..., \overline{x_i}, ..., x_n). \qquad (2.4)$$

Аналогично определяется и частная производная от функции $F(x_1, x_2, \ldots, x_n)$ относительно переменной f_k, формируемой на внутреннем k-ом полюсе цифровой схемы, реализующей данную функцию. Используя определение булевой производной для функции $F(x_1, x_2, \ldots, x_n)$ по переменной x_i, $i \in \{1, 2, \ldots, n\}$, можно показать, что при выполнении равенства:

$$\frac{\partial F(x_1, x_2, ..., x_n)}{\partial x_i} = 1 \qquad (2.5)$$

изменение переменной x_i приводит к изменению значения функции $F(x_1, x_2, \ldots, x_n)$, в противном случае значение x_i не влияет на значение функции. Другими словами, соотношение (2.5) есть условие наблюдаемости изменения переменной x_i по выходу схемы, реализующей функцию $F(x_1, x_2, \ldots, x_n)$. Указанное свойство булевой разности используется для установления условий транспортировки неисправности при формировании тестового набора. Условием наблюдаемости неисправности в точке ее возникновения будет равенство функции f_k, реализуемой на данном полюсе, единице, если неисправность $\equiv 0$, и нулю, если $\equiv 1$. При объединении двух условий можно построить систему логических уравнений для неисправности $\equiv 0$:

$$f_k(x_1, x_2, ..., x_n) = 1, \quad \frac{\partial F(x_1, x_2, ..., x_n, f_k)}{\partial f_k} = 1,$$

и для неисправности $\equiv 1$:

$$f_k(x_1, x_2, ..., x_n) = 0, \quad \frac{\partial F(x_1, x_2, ..., x_n, f_k)}{\partial f_k} = 1.$$

На практике указанные системы уравнений имеют вид одного уравнения:

$$f_k(x_1, x_2, ..., x_n) \frac{\partial F(x_1, x_2, ..., x_n, f_k)}{\partial f_k} = 1; \qquad (2.6)$$

и

$$\overline{f_k(x_1, x_2, ..., x_n)} \frac{\partial F(x_1, x_2, ..., x_n, f_k)}{\partial f_k} = 1. \qquad (2.7)$$

Любое решение, удовлетворяющее уравнению (2.6), является тестом для обнаружения неисправности ≡0, а решение уравнения (2.7) является тестом для неисправности ≡1, возникшей на полюсе k рассматриваемой цифровой схемы, описываемой функцией $F(x_1, x_2, \ldots, x_n)$. Для примера, приведенного на рис. 2.6, тестом, позволяющим обнаруживать неисправность $f_6 \equiv 1$, будет решение уравнения (2.7) для $f_6 = \overline{x_3 x_4}$ и $F = x_1 x_2 + f_6$. Учитывая, что $\partial F(x_1 x_2 + f_6) / \partial f_6 = \overline{x_1 x_2}$ получаем:

$$\overline{f_6} \frac{\partial (x_1 x_2 + f_6)}{\partial f_6} = x_3 x_4 \overline{x_1 x_2} = 1.$$

Любое решение последнего уравнения будет тестом для неисправности $f_6 \equiv 1$.

Рассмотренный метод построения тестов позволяет формировать тесты для различных неисправностей цифровой схемы [120], которые включают и многократные неисправности [174], причем заданная неисправность может транспортироваться на выход схемы по любому предварительно определенному пути. К недостаткам метода следует отнести вычислительную сложность и использование больших объемов памяти.

Большой вычислительной сложностью и применением существенных объемов памяти характеризуется метод генерирования тестов, с помощью которого можно построить эффективную, почти минимальную, их последовательность [9, 256]. Данный метод, широко известный как метод *эквивалентных нормальных форм* (*ЭНФ*), основан на построении ЭНФ для исследуемой одно-выходной комбинационной схемы, процедура получения которой состоит из следующих основных этапов:

1. Всем элементам схемы присваиваются последовательные номера (индексы).

2. Записывается булево выражение, реализуемое схемой таким образом, что каждому i-му элементу схемы соответствует пара скобок с индексом i, а все переменные в выражении являются входными переменными.

3. Полученное выражение преобразуется в дизъюнктивную нормальную форму. Удаление пары скобок связано с приписыванием ее индекса каждой переменной внутри этой скобки. Кроме того, избыточные члены не удаляются.

В качестве примера построения *ЭНФ* рассмотрим цифровую схему (рис. 2.6), в которой элементу И определим индекс 1, элементу И-НЕ индекс 2 и элементу ИЛИ – 3. Тогда $F = ((x1\ x2)_1 + (x3\ x4)_2)_3$. Далее раскрывая скоб-

ки, получим $F = x1_{13}\, x2_{13} + x3_{23}\, x4_{23}$, что представляет собой *ЭНФ* схемы, приведенной на рис. 2.6.

В общем случае *ЭНФ* любой цифровой схемы характеризуется таким свойством, что каждая ее переменная содержит полную информацию об одном из возможных путей в схеме. Конкретный путь начинается в точке приложения переменной *ЭНФ* и проходит через элементы, номера которых являются ее индексами. Так, например, для *ЭНФ* схемы, показанной на рис. 2.6, переменная $x3_{23}$ описывает путь, начинающийся в точке приложения $x3$ и проходящий через элементы с индексами 2 и 3 до выхода схемы.

Алгоритм построения тестов по методу эквивалентных нормальных форм состоит из следующих этапов.

1. Строится *ЭНФ* для исследуемой схемы.

2. Определяется путь в виде последовательности элементов, на котором лежит рассматриваемая неисправность.

3. Выбранный путь задается переменной *ЭНФ*, цепочка индексов которой соответствует номерам элементов пути.

4. Обеспечивается условие существенности выбранного пути. Для этого в одном из термов, в который входит выбранная переменная *ЭНФ*, все остальные сомножители доопределяются до единицы, а в других термах хотя бы один сомножитель до нуля.

В качестве примера построения теста, согласно рассмотренному методу, исследуем схему, приведенную на рис. 2.6, для случая неисправности $f_6 \equiv 1$. При этом последовательно получим:

1. $F = x1_{13}\, x2_{13} + x3_{23}\, x4_{23}$.

2. Выбирается путь, проходящий через элементы 2 и 3, на котором лежит неисправность $f_6 \equiv 1$.

3. Выбранный путь задается в виде переменной $x3_{23}$.

4. Обеспечивается условие существенности данного пути. Соответственно будем иметь: $x4 = 1$, $x1 = 0$, $x2 = 1$.

В результате применения метода эквивалентных нормальных для рассматриваемого примера получаем входной тестовый набор $x_1\, x_2\, x_3\, x_4 = 0\ 1\ 1\ 1$.

Определяющим недостатком данного метода является трудоемкость его реализации для цифровых устройств современных вычислительных систем, содержащих большое количество элементов. Данный недостаток относиться ко всем выше рассмотренным методам построения тестов для аппаратной составляющей вычислительных систем.

Более экономичным является подход, основанный на принципе моделирования неисправностей цифровых устройств [33, 251, 286]. Сущность его состоит в том, что случайные наборы входных воздействий подаются на две идентичные цифровые схемы или их модели. Одна из них является эталонной, а во вторую последовательно вносятся неисправности [286]. В случае отличия выходных реакций эталонной схемы и схемы с неисправностью входной случайный набор считается тестом, проверяющим данную неис-

правность. Последовательно внося все неисправности, строится полный тест, при этом для моделирования неисправностей используется *последовательное*, *параллельное* или *дедуктивное* моделирование [286]. Эффективность данного подхода, который часто называется *случайным поиском* (*random search*) [251], объясняется тем, что многие неисправности обнаруживаются большим множеством тестов. Однако данный метод мало эффективен в случае неисправностей, для которых существует малое количество тестовых наборов. Наличие двух подмножеств неисправностей, к первому из которых относятся неисправности, обнаруживаемые большим множеством тестовых наборов, а ко второму относятся неисправности, имеющие малое число тестовых наборов, хорошо иллюстрируется примером схемы, приведенной на рис. 2.6. Действительно для неисправности $x_3 \equiv 1$ данной схемы множество тестовых наборов $x_1\, x_2\, x_3\, x_4$ состоит из трех наборов 0 0 0 1, 0 1 0 1, 1 0 0 1, а для неисправности $x_1 \equiv 1$ один уникальный набор 0 1 1 1. Поэтому на практике, как правило, используется *комбинированный подход* построения тестов [207, 302]. В этом случае на первом этапе применяется метод случайного поиска, позволяющий сравнительно быстро построить тест для неисправностей, обнаруживаемых множеством тестовых наборов, а для неисправностей, требующих для своего обнаружения конкретного входного набора, используется один из методов *направленного поиска* (*d*-алгоритм, булево-дифференциальный метод и др.).

Несмотря на сравнительно высокий процент обнаружения неисправностей в результате применения традиционных подходов построения тестов, проблема синтеза эффективных тестов еще не решена, так как практически все реализованные методы формирования тестов ориентированы на класс одиночных константных неисправностей, и соответствующие оценки их эффективности приводятся относительно указанного класса неисправностей. Ранее отмечалось, что реальные неисправности современных вычислительных машин и в особенности их аппаратной составляющей, как правило, далеко не исчерпываются моделью одиночных неисправностей.

Таким образом, учет реальных неисправностей требует новых подходов для решения проблемы автоматизированного построения тестов цифровых схем. Кроме того, принадлежность задачи построения тестов к классу *универсальных переборных* (*NP-complete*) приводит к тому, что примерно 10^5 элементов является практическим пределом эффективного применения традиционных методов автоматизированного построения тестов [207, 257, 302].

Необходимость применения качественно новых методов и подходов для решения задачи автоматизированного построения тестов обусловлено высокой трудоемкостью построения тестов для цифровых устройств современных систем. Одним из самых реальных подходов для решения проблемы автоматизации построения тестов является учет задач тестового диагностирования на этапе проектирования цифровых устройств. В данном случае применяются как различные правила логического и конструкторского проек-

тирования, упрощающие задачу тестирования, так и новые принципы схемотехнического построения цифровых устройств.

2.4. Построение тестов для запоминающих устройств

Среди множества тестов запоминающих устройств выделяют так называемые традиционные тесты. Под ним подразумевают тесты, которые были разработаны ориентировочно до 1980 года. Все традиционные тесты не были нацелены на выявление конкретных неисправностей запоминающих устройств, а позволяли обеспечивать некоторую степень покрытия возможных неисправностей при соответствующих временных затратах. К широко известным традиционным тестам запоминающих устройств относят тест типа *шахматная доска* (*checkerboard*), тест *бегущая* 1/0 (*walking* 1/0), тест *бегущая диагональ* (*sliding diagonal*), тест *пинг-понг* (*ping-pong*), тест *галопирующая* 1/0 (*galloping* 1/0) и другие [69, 73, 144, 254, 303]. Наиболее часто применяемым на практике в те времена является тест типа шахматная доска. При использовании данного теста логическая структура двумерного массива из N запоминающих элементов разбивается на два класса ячеек, а именно: класс A и класс B (рис. 2.7).

Сам тест может быть описан как последовательность следующих шагов.

1. Записать 1 во все ячейки класса A, а в ячейки класса B записать 0.
2. Прочитать содержимое всех ячеек памяти.
3. Записать 0 во все ячейки класса A, а в ячейки класса B записать 1.
4. Прочитать содержимое всех ячеек памяти.

A	*B*	*A*	*B*
B	*A*	*B*	*A*
A	*B*	*A*	*B*
B	*A*	*B*	*A*

Рисунок 2.7 – *Логическая структура массива ЗЭ для теста шахматная доска*

Отметим, что во всех тестах ЗУ, в том числе и описанном выше, операция чтения содержимого ячейки сопряжена с проверкой прочитанного значения 0/1 с ранее записанным значением 0/1 в данную ячейку.

Одним из основных критериев оценки эффективности тестов ЗУ является время выполнения теста, которое зависит от количества операций обращения к тестируемому устройству и от времени отклика устройства. В силу различия временных параметров тестируемых ЗУ была введена обобщающая характеристика их эффективности под названием сложность теста Q(тест). Эта характеристика оценивается как количество всех элементарных операций

(чтение/запись), необходимых для реализации теста. Так, сложность теста типа шахматная доска может быть оценена по следующей формуле:

$$Q(\textit{шахматная доска}) = N + N + N + N = 4N. \qquad (2.8)$$

Величина $4N$ представляет собой количество элементарных операций, необходимых для выполнения данного теста.

Все множество традиционных тестов делиться на четыре подмножества. Различают тесты типа N, $N\log_2 N$, $N^{3/2}$ и N^2 циклов обращения к памяти, где приведенные значения представляют собой порядок сложности O теста как приблизительную оценку количества операций чтения/записи, необходимых для реализации теста. Тест типа шахматная доска относится к тестам первого подмножества, так как его сложность оценивается как $O(N)$. В таблице 2.1 приведены величины сложности и порядки сложности наиболее известных традиционных тестов ЗУ [303].

Таблица 2.1 – *Оценки сложности традиционных тестов ЗУ*

Название теста	Сложность теста Q	Порядок сложности O
Checkerboard	$4N$	N
Galloping 1/0	$2N^2 + 6N$	N^2
Walking 1/0	$4N^2 + 2N$	N^2
Sliding diagonal	$2N\sqrt{N} + 6N$	$N^{3/2}$
Butterfly	$2(3N + 5N(N/2 - 1))$	$N\log_2 N$

Тенденция к постоянному росту информационного объема современных ЗУ привела не только к экономической, но и к физической невозможности применения функциональных тестов, сложность которых оценивается величинами $O(N^2), O(N\log_2 N), O(N^{3/2})$. Приведенная ниже таблица 2.2 наглядно демонстрирует зависимость времени тестирования ЗУ от его информационного объема и от сложности применяемого теста. Время доступа к ЗУ принято считать равным 60 *нс*, а временные затраты приведены в секундах (*с*), часах (*ч*), днях (*д*) и годах (*г*) [73, 303, 331].

Из таблицы 2.2 видно, что последние два класса тестов для запоминающих устройств не находят применения в силу большой временной сложности, определяемой емкостью современной памяти. В тоже время широкое применение находят тесты, имеющие линейную зависимость сложности $O(N)$ от емкости N памяти, получившие название *маршевые тесты* (*march tests*) [73, 74, 166, 254, 303, 331]. Эти тесты обладают рядом достоинств, среди которых можно выделить приемлемую покрывающую способность неисправностей запоминающих устройств и простоту реализации тестовой процедуры как временную, так и аппаратурную. Это особенно важно для *встро-*

енных средств самотестирования (*Built-In Self-Test – BIST*) [72, 260, 268, 303, 331].

Таблица 2.2 – *Зависимость времени тестирования ЗУ от информационного объема и сложности теста*

Информационная емкость, в битах (N)	Сложность теста			
	$O(N)$	$O(N\log_2 N)$	$O(N^{3/2})$	$O(N^2)$
1*Мбит*	0,06*с*	1,26*с*	64,5*с*	18,3*ч*
4*Мбит*	0,25*с*	5,54*с*	515,4*с*	293,2*ч*
16*Мбит*	1,01*с*	24,16*с*	1,2*ч*	196*д*
64*Мбит*	4,03*с*	104,7*с*	9,2*ч*	8,5*л*
256*Мбит*	16,11*с*	451,0*с*	73,3*ч*	137*л*
1*Гбит*	64,43*с*	1932,8*с*	586,4*ч*	2194*г*
2*Гбит*	128,9*с*	3994,4*с*	1658,6*ч*	∞

В общем случае маршевый тест состоит из конечного числа *маршевых элементов* (*march elements*) [73, 166, 254, 303, 331]. В свою очередь, каждый маршевый элемент содержит символ, определяющий порядок формирования *адресной последовательности* (*address sequence*) для тестируемого запоминающего устройства, где символ $\Uparrow$ определяет последовательный перебор адресов ЗУ по возрастанию, символ $\Downarrow$ определяет последовательный перебор адресов по убыванию и сочетание двух символов $\Uparrow\Downarrow$ означает формирование адресов по убыванию либо по возрастанию. Убывающая последовательность адресов представляет собой последовательность, которая формируется в обратном порядке по сравнению с возрастающей последовательностью. При этом возрастающая последовательность может быть любой последовательностью адресов. Кроме этого, маршевый элемент содержит последовательность *операций чтения* (*read – r*) и *записи* (*write – w*), заключенных в круглые скобки и разделяемых точкой с запятой. Каждая операция представляет собой элемент из следующего набора: $r0$ – операция чтения содержимого ЗЭ с ожидаемым значением 0, $r1$ – операция чтения ЗЭ с ожидаемым значением 1, $w0$ – операция записи 0 в ЗЭ, $w1$ – операция записи 1 в ЗЭ. Одна или несколько операций в маршевом элементе используются последовательно для адресуемой ячейки ОЗУ. Переход к следующей ячейке будет осуществлен только после выполнения всех операций в текущем маршевом элементе [331].

Маршевый тест MATS (*modified algorithmic test sequence* – MATS) имеет следующую запись: $\{\Uparrow\Downarrow(w0);\ \Uparrow(r0,w1);\ \Downarrow(r1)\}$, согласно которой данный тест состоит из трех маршевых элементов и имеет сложность $4N$. Первый маршевый элемент $\Uparrow\Downarrow(w0)$ называется *фазой инициализации* (*initialization phase*), применяемой для записи *начального состояния* (*background*) запоминающего устройства. Эта фаза, как правило, выполняется при изменении адресов от младшего адреса к старшему адресу или, наоборот, путем записи во

все ячейки нулевого значения. Вторая фаза теста MATS ⇑(*r*0, *w*1) определяет возрастающий порядок адресов и для каждого ЗЭ состоит из операции чтения (*r*0), когда ожидаемое значение содержимого ЗЭ 0, и записи в данную ячейку 1 (*w*1). В третьей фазе (⇓(*r*1)) содержимое ЗУ считывается последовательно, при этом порядок изменения адресов обратный по отношению к предыдущей фазе, а ожидаемое считываемое значение равно 1.

Количество маршевых элементов, их взаимное расположение, а также вид каждого из этих элементов определяют *покрывающую способность маршевого теста* (*fault coverage*) [73, 74, 166, 254, 303, 331], т. е. эффективность обнаружения всех видов возможных неисправностей запоминающего устройства. До настоящего времени процедура создания новых маршевых тестов практически не формализована из-за многообразия факторов, влияющих на их эффективность [28, 73, 74, 90, 91, 166, 199, 254, 303, 331]. Во-первых, это требование минимальной временной сложности. Во-вторых, высокой покрывающей способности, а также ряда других требований, зависящих от технологических особенностей запоминающего устройства и специфики процедуры его тестирования. Наиболее часто используемые маршевые тесты приведены в таблице 2.3 [73, 74, 166, 254, 303, 331].

Таблица 2.3 – *Классические маршевые тесты*

№	Название теста	Сложность теста	Описание теста
1	Scan	4*N*	{⇑⇓ (*w*0); ⇑⇓ (*r*0); ⇑⇓ (*w*1); ⇑⇓ (*r*1)}
2	MATS	4*N*	{⇑⇓ (*w*0); ⇑(*r*0,*w*1); ⇓⇑(*r*1)}
3	MATS+	5*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0)}
4	MATS++	6*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0,*r*0)}
5	Marching 1/0	14*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1); ⇓(*r*1,*w*0,*r*0); ⇑⇓(*w*1); ⇑(*r*1,*w*0,*r*0); ⇓(*r*0,*w*1,*r*1)}
6	March *X*	6*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0); ⇑⇓(*r*0)}
7	March *Y*	8*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1); ⇓(*r*1,*w*0,*r*0); ⇑⇓(*r*0)}
8	March *C*	11*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1); ⇑(*r*1,*w*0); ⇑⇓(*r*0); ⇓(*r*0,*w*1); ⇓(*r*1,*w*0); ⇑⇓(*r*0)}
9	March *C*–	10*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1); ⇑(*r*1,*w*0); ⇓(*r*0,*w*1); ⇓(*r*1,*w*0); ⇑⇓(*r*0)}
10	March *A*	15*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*w*0,*w*1); ⇑(*r*1,*w*0,*w*1); ⇓(*r*1,*w*0,*w*1,*w*0); ⇓(*r*0,*w*1,*w*0)}
11	March *B*	17*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1,*w*0,*r*0,*w*1); ⇑(*r*1,*w*0,*w*1); ⇓(*r*1,*w*0,*w*1,*w*0); ⇓(*r*0,*w*1,*w*0)}
12	Algorithm *B*	17*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*w*0,*w*1); ⇑(*r*1,*w*0,*r*0,*w*1); ⇓(*r*1,*w*0,*w*1,*w*0); ⇓(*r*0,*w*1,*r*1,*w*0)}

Продолжение таблицы 2.3

13	March *C-R*	15*N*	{⇑⇓(*w*0); ⇑(r0,r0,w1); ⇑(r1,r1,w0); ⇓(r0,r0,w1); ⇓(r1,r1,w0); ⇓⇑(r0,r0)}
14	PMOVI	13*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1); ⇑(*r*1,*w*0,*r*0); ⇓(*r*0,*w*1,*r*1); ⇓(*r*1,*w*0,*r*0)}
15	PMOVI-*R*	17*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1,*r*1); ⇑(*r*1,*w*0,*r*0,*r*0); ⇓(*r*0,*w*1, *r*1,*r*1); ⇓ (*r*1,*w*0,*r*0,*r*0)}
16	March *G*	23*N*+*2*D	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1,*w*0,*r*0,*w*1); ⇑(*r*1,*w*0,*w*1); ⇓(*r*1, *w*0,*w*1,*w*0); ⇓(*r*0,*w*1,*w*0); *D*; ⇑⇓(*r*0,*w*1,*r*1); *D*; ⇑⇓(*r*1,*w*0,*r*0)}
17	March *U*	13*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1,*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0,*r*0, *w*1); ⇓ (*r*1,*w*0)}
18	March *UD*	13*N*+2*D*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1,*w*0); *D*; ⇑(*r*0,*w*1); *D*; ⇓(*r*1,*w*0, *r*0,*w*1); ⇓(*r*1,*w*0)}
19	March *U-R*	15*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1,*r*1,*w*0); ⇑(*r*0,*w*1); ⇓(*r*1,*w*0,*r*0, *r*0,*w*1); ⇓(*r*1,*w*0)}
20	March *LR*	14*N*	{⇑⇓(*w*0); ⇓(*r*0,*w*1); ⇑(*r*1,*w*0,*r*0,*w*1); ⇑(*r*1,*w*0); ⇑ (*r*0,*w*1,*r*1,*w*0); ⇓(*r*0)}
21	March *LA*	22*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*w*0,*w*1,*r*1); ⇑(*r*1,*w*0,*w*1,*w*0,*r*0); ⇓(*r*0,*w*1,*w*0,*w*1,*r*1); ⇓(*r*1,*w*0,*w*1,*w*0,*r*0); ⇓(*r*0)}
22	March *M*	16*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1,*w*0); ⇑⇓(*r*0); ⇑(*r*0,*w*1); ⇑⇓(*r*1); ⇓(*r*1,*w*0,*r*0,*w*1); ⇑⇓(*r*1); ⇓(*r*1,*w*0)}
23	March *PS*	23*N*	{⇑⇓(*w*0); ⇑(*r*0,*w*1,*r*1,*w*0,*r*0,*w*1); ⇑(*r*1,*w*0,*r*0,*w*1,*r*1); ⇑(*r*1,*w*0,*r*0,*w*1,*r*1,*w*0); ⇑(*r*0,*w*1,*r*1,*w*0,*r*0)}
24	March *PNPSFk*	18*N*	{⇑⇓(*w*0);⇑(*r*0,*w*1,*r*1,*w*0); ⇓(*r*0,*w*1); ⇑(*r*1,*w*0,*r*0,*w*1); ⇑(*r*1,*w*0);⇑(*r*0,*w*1); ⇓(*r*1,*w*0,*r*0)}
25	*Cheng* test	17*N*	{⇑⇓(*w*0);⇑(*w*1,*r*1,*w*0); ⇑(*r*0,*w*1); ⇑(*r*1,*w*0,*r*0,*w*1); ⇑(*r*1,*w*0); ⇓(*r*0,*w*1); ⇓(*r*1,*w*0); ⇑⇓(*r*0)}

В работах [1, 54, 55, 56, 113, 129, 144, 145, 185] были проведены оценки обнаруживающих способностей маршевых тестов относительно одиночных неисправностей различных классов. В таблице 2.4 приведены маршевые тесты, специально разработанные для обнаружения неисправностей, характерных для статических запоминающих устройств. Как показывает анализ покрывающих способностей, не все маршевые тесты способны в полной мере обнаруживать различные классы неисправностей. Из приведенной таблицы покрывающих способностей тестов видно, что, несмотря на различные структуру и сложность маршевых тестов, открытой проблемой остается обнаружение в полной мере неисправностей взаимного влияния и сложных связанных неисправностей [303, 331].

Для изучения природы различных покрывающих способностей маршевых тестов в рамках представленных классов неисправностей был проведен анализ структуры и последовательности маршевых элементов с целью выявления условий активизации и условий обнаружения неисправностей [73, 303,

331]. При определении обнаруживающей способности маршевых тестов используются условия, полученные в работах [73, 303, 331], которые являются необходимыми для обнаружения основных классов неисправностей запоминающих устройств. В качестве примеров рассмотрим некоторые из них.

Таблица 2.4 – *Сводная таблица обнаруживающей способности различных тестов ЗУ*

Название теста	*Q*	Обнаруживаемые неисправности								
		AF	*SAF*	*TF*	*CFin*	*CFid*	*TF-CF*	*CFid-CFid*	*CFin-CFin*	*CFid-CFin*
MATS	4*N*		+							
MATS+	5*N*	+	+							
MATS++	6*N*	+	+	+						
March *X*	6*N*	+	+	+	+					
March *Y*	8*N*	+	+	+	+		+			
Marching 1/0	14*N*	+	+	+	+		+			
March *C*	11*N*	+	+	+	+	+			+	
March *C*−	10*N*	+	+	+	+	+			+	
March *A*	15*N*	+	+	+	+	+		+		
March *B*	17*N*	+	+	+	+	+	+	+		
Algorithm *B*	17*N*	+	+	+	+	+	+	+		

Условие 2.1. Для обнаружения константных неисправностей *SAF* необходимо наличие в маршевых тестах операций *r*0 и *r*1.

Например, тест MATS (таблица 2.3) является тестом минимальной сложности, позволяющий обнаруживать все неисправности типа *SAF*. Первый маршевый элемент ⇑⇓(*w*0) является условием проявления всех константных неисправностей типа *SAF*1, а операция *r*0 второго маршевого элемента является условием обнаружения *SAF*1. Аналогично, вторая операция *w*1 второго маршевого элемента является условием проявления неисправности *SAF*0, а последний маршевый элемент – условием обнаружения *SAF*0.

Условие 2.2. Для обнаружения переходных неисправностей *TF* необходимо, чтобы маршевый тест содержал в себе операции записи, которые осуществляют всевозможные логические переходы в каждой ячейке и операции чтения содержимого ячейки после каждого перехода.

Примером теста, позволяющим обнаруживать все переходные неисправности, является тест MATS++ (таблица 2.3). Данный тест является тестом минимальной сложности, который удовлетворяет условиям, позволяющим обнаруживать все неисправности типов *SAF, TF* и *AF*. Более сложную природу имеют неисправности взаимного влияния. Их обнаружение обеспечивается маршевым тестом March *C*−, который удовлетворяет условию обнаружения всех восьми типов неисправностей *CFid*. Два указанных теста MATS++ и March *C*− являются наиболее часто используемыми на практике тестами запоминающих устройств.

Глава 3. Вероятностное тестирование

3.1. Сущность и достоинства вероятностного тестирования

Одной из самых распространенных технологий тестирования по методу черного ящика является реализация так называемого *вероятностного тестирования* (*random testing*), которое зачастую применяется для тестирования вычислительных систем. В случае вероятностного тестирования структура и функциональность объекта тестирования принимаются неизвестной, либо считаются таковой, и практически не учитываются при формировании тестовых наборов. Несмотря на широкое применение вероятностного тестирования на практике, сведения о нем носят разрозненный и зачастую противоречивый характер. Так, например, в области тестирования программного обеспечения вероятностное тестирование часто интерпретируется как *обезьянье тестирование* (*monkey testing*) [10, 143].

Обезьянье тестирование представляет собой метод тестирования программного обеспечения, когда пользователь тестирует приложение, формируя случайные входные данные, случайные последовательности данных и команд, проверяя при этом поведение тестируемого объекта. Для этого метода не существует определенных правил, он не следует за предопределенными тестовыми примерами или стратегиями и, таким образом, работает на настроении и чувствах специалиста по тестированию.

Название данного метода тестирования можно объяснить тем, что иногда поведение специалиста по тестированию, разработчика или пользователя напоминают поведение обезьяны, которая работает с приложением неосознанно и без определенных целей и смысла вводит произвольные данные. Различают три типа обезьяньего тестирования, а именно тестирование *тупой обезьяны* (*dumb monkey*), *умной обезьяны* (*smart monkey*) и *блестящей обезьяны* (*brilliant monkey*) [10, 143].

В тестах *тупой обезьяны* процедура тестирования выполняется при полном незнании о системе или приложении. Пользователь, проводящий тестирование, не имеет ни малейшего представления о входных данных приложения и о том, являются ли они действительными, нежелательными или запрещенными.

Тесты *умной обезьяны* выполняются при хорошем представлении о системе или приложении. Специалист по тестированию точно знает функциональность продукта и формирует действительные входные данные для проведения тестирования.

Реализация тестов *блестящей обезьяны* основана на знании функциональности приложения его назначении и особенностях применения на практике. Подобное тестирование проводится с точки зрения пользователя.

Определение вероятностного тестирования впервые было сформулировано для случая тестирования цифровых устройств [190, 207, 245, 246, 247, 302].

При вероятностном тестировании на исследуемый объект подаются случайные или псевдослучайные входные последовательности. Главное достоинство такого подхода заключается в том, что исключается необходимость предварительного построения детерминированного теста, процедура построения которого может оказаться слишком сложной, а в случае современных вычислительных систем и нереализуемой. Подобная ситуация возникает при больших функциональных возможностях испытуемой системы или в том случае, когда нет достаточных сведений о ее внутренней структуре. Тогда целесообразным является применение вероятностного тестирования.

Существуют два основных способа организации вероятностного тестирования. Первый способ использует эталонный объект тестирования (см. рис. 3.1).

Входные наборы случайных и независимых значений вероятностного теста, вырабатываемые генератором тестовой последовательности, одновременно поступают на входы тестируемого и эталонного объектов. Их выходные реакции (значения) сравниваются схемой сравнения. В случае расхождения выходных значений делается вывод о неисправности испытуемого объекта, в противном случае объект считается исправным. Основным недостатком классической схемы вероятностного тестирования, приведенной на рис. 3.1, является необходимость наличия эталонного объекта, который считается заведомо исправным.

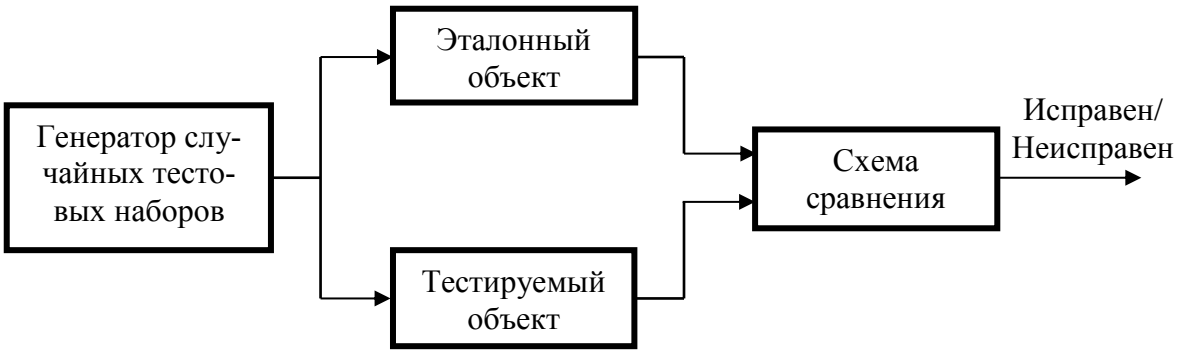


Рисунок 3.1 – *Схема вероятностного тестирования*

Второй способ отличается методом сравнения выходных реакций и не предполагает наличия исправного объекта тестирования. В этом случае реакция испытуемого объекта приводится к компактному виду с помощью схемы сжатия, а результат сравнивается со сжатым эталоном (см. рис. 3.2). Часто подобный вид тестирования называют *компактным тестированием* (*compact testing*) [207, 302].

Второй способ вероятностного тестирования вычислительных систем широко применяется для реализации встроенного их самотестирования. Среди несомненных достоинств вероятностного тестирования выделяют:

1. Отсутствие сложных процедур построения тестов и тестовых сценариев. Соответственно, подготовительный этап не требуют больших временных затрат.

2. Непосредственно процедура вероятностного тестирования базируется на использовании небольших объемов дополнительных программно/аппаратурных средств.

3. Такой подход помогает обнаружить новые неисправности тестируемой системы, которые нельзя было бы найти с помощью специальных тестовых процедур и сценариев.

4. С помощью специальных инструментов вероятностное тестирование можно легко автоматизировать.

5. Вероятностное тестирование является весьма эффективным для выполнения стресс тестирования и нагрузочного тестирования.

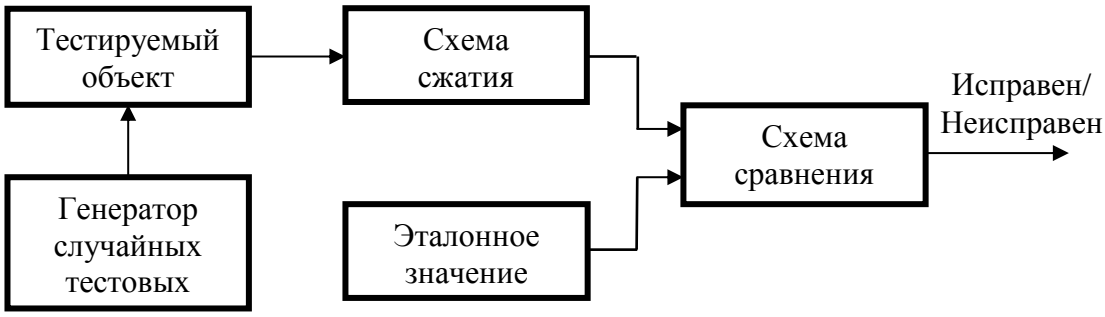


Рисунок 3.2 – *Схема вероятностного компактного тестирования*

Тем не менее, вероятностное тестирование не дает возможности воспроизвести неисправность и убедится, что это действительно некий дефект, а не особенность функционала системы. Анализ найденных проблем в ходе вероятностного тестирования занимает много времени и сил, так как нет точного их описания и шагов воспроизведения. Таким образом, к наиболее существенным недостаткам вероятностного тестирования относят:

1. Тест, проведенный во время вероятностного тестирования, настолько случаен, что очень сложно либо невозможно воссоздать условия для обнаружения конкретной неисправности.

2. Необходимы большие временные затраты для анализа нестандартных и неожиданных проблем, возникающих во время вероятностного тестирования.

3. Специалисты по тестированию испытывают трудности с определением сценариев тестирования даже в части законов распределения вероятностей входных тестовых данных и их длины.

4. Отсутствует гарантия эффективности вероятностных тестов.

5. Вероятностное тестирование характеризуется большой временной сложностью. Как правило, количество тестовых наборов вероятностного теста существенно превышает количество наборов детерминированного теста.

Попытки увеличения эффективности вероятностного тестирования основываются на учете особенностей объекта тестирования. Наиболее успеш-

ными являются результаты, полученные для цифровых устройств вычислительных систем. В данном случае оказывается возможным определение вероятностного описания их поведения, что позволяет строить эффективные вероятностные тесты.

3.2. Вероятностное описание цифровых устройств

Основная отличительная особенность вероятностного тестирования цифровых устройств состоит в применении последовательностей случайных независимых двоичных цифр, подаваемых на входы проверяемого цифрового устройства. При этом случайная и независимая двоичная переменная $x_i \in \{0, 1\}$, $i = \overline{1,n}$, подаваемая на i-й вход, описывается вероятностью $p(x_i = 1)$ ее единичного значения.

Классической является схема вероятностного тестирования, состоящего из генератора последовательностей случайных независимых двоичных цифр, проверяемой и эталонной цифровых схем, а также блока сравнения выходных реакций (рис. 3.1). Генератор последовательностей двоичных цифр предназначен для формирования случайных независимых цифр с заданным законом распределения вероятностей $p(x_i = 1)$. Формируемые на выходе генератора двоичные наборы из n бит подаются одновременно на входы проверяемой и эталонной цифровых схем, выходные реакции которых анализируются в блоке сравнения выходных реакций. В случае совпадения полученных реакций обеих схем проверяемая схема считается исправной, в противном случае принимается гипотеза об ее неисправном состоянии и решается задача поиска неисправности [207, 302].

В первых работах, посвященных вероятностному тестированию [190, 245], рассматривалась задача вероятностного описания поведения цифровых устройств. Такая задача заключалась в определении эталонных значений вероятностей появления единичных символов на промежуточных и выходных полюсах схемы. Согласно предложенным методам эталонные значения вероятностей, определенные аналитическим путем, использовались для сравнения с реально полученными вероятностями в рамках второго метода реализации вероятностного тестирования (см. рис. 3.2). Таким образом, процедура вероятностного тестирования заключалась в нахождении реальных значений вероятностей и сравнении их с эталонными величинами.

Рассмотрим зависимость вероятностей выходных значений $f(x_1, x_2, \ldots, x_n)$, для простейших n-входовых логических элементов, от вероятностей $p(x_i = 1)$ и $p(x_i = 0) = 1 - p(x_i = 1)$ их входных двоичных переменных $x_1, x_2, \ldots, x_n$.

Первоначально исследуем логический элемент НЕ, входная переменная x_1 которого однозначно определяет выходное значение f, где $f = \overline{x_1}$. Тогда:

$$p(f=1)=p(x_1=0)=1-p(x_1=1). \qquad (3.1)$$

Для исследования зависимости вероятности $p(f = 1)$ элемента И подадим на его входы последовательности случайных двоичных цифр, которые являются независимыми величинами. Выходное значение элемента И будет равно единице только в случае равенства единице всех входных переменных. Вероятность этого события определяется как [246]:

$$p(f=1)=p(x_1=1, x_2=1, \ldots, x_n=1)=\prod_{i=1}^{n} p(x_i=1), \qquad (3.2)$$

где $f = x_1 x_2 \ldots x_n$.

В случае многовходового элемента ИЛИ, выходное значение которого определяется выражением $f = x_1 + x_2 + \ldots + x_n$, в результате простейших преобразований, применяя (3.1) и (3.2), а также учитывая независимость событий $\overline{x_1}$, $\overline{x_2}$, ..., и $\overline{x_n}$ получим:

$$\begin{aligned} p(f=1)&=p(x_1+x_2+\ldots+x_n=1)=p(\overline{\overline{x_1}\,\overline{x_2}\ldots\overline{x_n}}=1)=1-p(\overline{x_1}\,\overline{x_2}\ldots\overline{x_n}=1)= \\ &=1-\prod_{i=1}^{n} p(\overline{x_i}=1)=1-\prod_{i=1}^{n}[1-p(x_i=1)]. \end{aligned} \qquad (3.3)$$

Более полный перечень соотношений, связывающих выходные вероятности логических элементов как функций вероятностей входных переменных и их корреляционных функций, приведен в [158, 300]. В этих соотношениях выражения, описывающие логические элементы И, ИЛИ, могут принимать совершенно отличный вид от соотношений (3.2) и (3.3). Так, например, для двухвходового элемента И, на оба входа которого подается переменная x_1 выходная вероятность будет иметь вид [158]:

$$p(f=1)=p(x_1x_1=1)=p(x_1=1).$$

Подобным образом можно показать, что:

$$p(x_1\overline{x_1}=1)=0,\ p(x_1+x_1=1)=p(x_1=1),\ p(x_1+\overline{x_1}=1)=1.$$

В простейших случаях комбинационных древовидных цифровых схем достаточно применять полученные соотношения для элементов НЕ, И и ИЛИ, на основании которых можно найти значения вероятностей на промежуточных и выходных полюсах произвольной цифровой схемы. В качестве примера рассмотрим цифровую схему, приведенную на рис. 3.3, на входы которой подаются последовательности случайных независимых двоичных цифр с вероятностями $p(x_1 = 1) = p(x_2 = 1) = p(x_3 = 1) = p(x_4 = 1) = p(x_5 = 1) = p$.

Применяя соотношения (3.3) и (3.2), соответственно будем иметь $p(f_1 = 1) = p^3$, $p(f_2 = 1) = 1 - (1 - (1 - p)^2) = 1 - 2p + p^2$. Окончательно для выходного значения схемы получим:

$$p(f_3 = 1) = p(f_1 = 1)p(f_2 = 1) = p^3(1 - 2p + p^2) = p^3 - 2p^4 + p^5.$$

При $p = 0{,}5$ эталонные значения вероятностей будут иметь вид $p(f_1 = 1) = 0{,}125$, $p(f_2 = 1) = 0{,}25$, $p(f_3 = 1) = 0{,}03125$. При реализации вероятностного тестирования рассматриваемой схемы реально полученные оценки вероятностей $p'(f_1 = 1)$, $p'(f_2 = 1)$ и $p'(f_3 = 1)$ сравниваются с эталонными значениями, затем принимается решение о состоянии схемы.

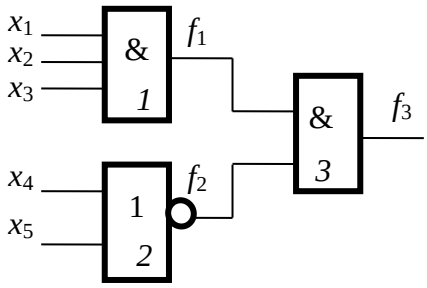


Рисунок 3.3 – *Пример комбинационной схемы*

В случае возникновения неисправностей реальные значения вероятностей, как правило, отличаются от их эталонных величин. Так, например, при возникновении неисправности $f_1 \equiv 1$ будем иметь $p'(f_1 = 1) = 1$, $p'(f_2 = 1) = 0{,}25$, и $p'(f_3 = 1) = 0{,}25$, что свидетельствует о неисправном состоянии схемы.

В общем случае определение эталонных вероятностей представляет собой сложную аналитическую задачу, в особенности для цифровых схем произвольного вида, что требует использования специальных методик.

Одними из первых методик аналитического определения вероятностного описания цифровых схем являются два алгоритма предложенные в работе [158]. Они используют полученные ранее соотношения (3.1), (3.2) и (3.3) и применяются для комбинационных схем. Первый алгоритм состоит из двух последовательных этапов [156, 158]. Первоначально выполняется подготовительный этап, состоящий из следующих шагов [158].

1. Булева функция, описывающая произвольный полюс комбинационной цифровой схемы, представляется в виде логической суммы произведений входных переменных и их отрицаний.

2. Булева функция, записанная как сумма произведений, представляется в виде канонической суммы произведений $f = \pi_1 + \pi_2 + \ldots + \pi_k$, где $\pi_i \pi_j = 0$ для $i \neq j \in \{1, 2, \ldots, k\}$.

Непосредственно алгоритм вычисления вероятности появления единичного значения на произвольном полюсе цифровой схемы состоит из следующих шагов.

Для заданного полюса цифровой схемы получается аналитическое выражение в виде канонической суммы произведений $f = \pi_1 + \pi_2 + ... + \pi_k$. Для каждого произведения π_i, $i \in \{1, 2, \ldots, k\}$, используя соотношения (3.1) и (3.2), определяется значение вероятности $p(\pi_i = 1)$.

3. В силу несовместности событий $\pi_i = 1$ и $\pi_j = 1$ значение $p(f = 1)$ вычисляется по выражению (3.4):

$$p(f = 1) = \sum_{i=1}^{k} p(\pi_i = 1). \qquad (3.4)$$

В качестве примера использования рассмотренного алгоритма возьмем цифровую схему (рис. 3.4), описываемую выражением $f = (x_1 + x_2)\, x_3 + x_1\, x_2$, для которой найдем значение $p(f = 1)$ [15, 16, 154]. Первоначально булева функция $f = (x_1 + x_2)\, x_3 + x_1\, x_2$ представляется в виде канонической суммы произведений:

$$f = (x_1 + x_2)x_3 + x_1\, x_2 = x_1\, x_2\, x_3 + \overline{x_1}\, x_2\, x_3 + x_1\, \overline{x_2}\, x_3 + x_1\, x_2\, \overline{x_3}.$$

Последовательно вычисляются значения $p(x_1\, x_2\, x_3 = 1)$, $p(\overline{x_1}\, x_2\, x_3 = 1)$, $p(x_1 \overline{x_2} x_3 = 1)$ и $p(x_1\, x_2\, \overline{x_3} = 1)$.

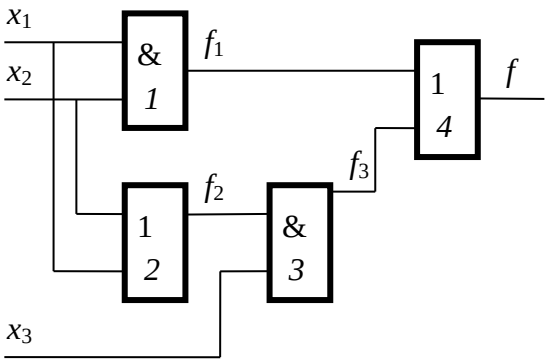


Рисунок 3.4 – *Комбинационная схема*

На основании (3.4) определяется значение искомой вероятности:

$$p(f = 1) = p(x_1\, x_2\, x_3 = 1) + p(\overline{x_1}\, x_2\, x_3 = 1) + p(x_1\, \overline{x_2}\, x_3 = 1) + p(x_1\, x_2\, \overline{x_3} = 1) =$$
$$p(x_1 = 1)\, p(x_2 = 1) + p(x_1 = 1)\, p(x_3 = 1) + p(x_2 = 1)\, p(x_3 = 1) -$$
$$2\, p(x_1 = 1)\, p(x_2 = 1)\, p(x_3 = 1).$$

Для $p(x_1 = 1) = p(x_2 = 1) = p(x_3 = 1) = p$, получим, что $p(f = 1) = 3p^2 - 2p^3$.

Существенным недостатком рассмотренной методики является сложность получения аналитического выражения в виде канонической суммы произведений. Для практического применения наиболее прост второй алгоритм, позволяющий получить вероятностное описание цифровой схемы на

основании ее функциональной схемы [302, 207, 208]. Сущность данного алгоритма основывается на свойстве вероятности зависимых событий. Поясним указанное свойство на примере элемента И с пятью входами. Пусть на три его первых входа подается переменная x_1, для которой вероятность единичного значения составляет $p(x_1 = 1)$ и соответственно x_2 и x_3 на четвертый и пятый входы, описываемые вероятностями $p(x_2 = 1)$ и $p(x_3 = 1)$. Согласно соотношению (3.2) вероятность $p(f = 1)$ единичного значения на выходе данного элемента И будет определяться выражением:

$$p(f = 1) = p^3(x_1 = 1)\, p(x_2 = 1)\, p(x_3 = 1),$$

где $p(x_1 = 1)$ входит в последнее выражение в третьей степени. С другой стороны, рассматривая логику функционирования элемента И, можно заключить, что вероятность единичного значения на его выходе характеризуется вероятностью совместного события $x_1 = 1$, $x_2 = 1$ и $x_3 = 1$, которая для независимых переменных x_1, x_2 и x_3 вычисляется как произведение вероятностей:

$$p(f = 1) = p(x_1 = 1)\, p(x_2 = 1)\, p(x_3 = 1),$$

где все вероятности имеют только первую степень. Таким образом, для получения правильной оценки вероятности, описывающей поведение комбинационной схемы, содержащей сходящиеся разветвления (зависимые события), в выражении вероятности, полученном по соотношениям (3.2), (3.3) и (3.4) показатели степеней вероятностей уменьшаются до первой степени [158].

Алгоритм, использующий данное свойство, состоит из следующих этапов:

1. Все полюса рассматриваемой цифровой схемы обозначаются различными переменными.

2. Для каждого элемента схемы вычисляется выходная вероятность как функция вероятностей входных переменных, причем данная процедура выполняется последовательно от элементов, к которым подключены входы схемы к элементу, на выходе которого формируется функция, реализуемая схемой. Для получения правильного выражения вероятностного описания схемы все показатели степени уменьшаются до единичного значения.

В качестве примера, иллюстрирующего рассмотренный алгоритм, определим $p(f = 1)$ для схемы, приведенной на рис. 3.4. Применяя соотношения (3.2), (3.3) и (3.4), получим:

$$p(f_1 = 1) = p(x_1 = 1)\, p(x_2 = 1);$$

$$p(f_2 = 1) = p(x_1 = 1) + p(x_2 = 1) - p(x_1 = 1)\, p(x_2 = 1);$$

$$p(f_3 = 1) = p(f_2 = 1)\, p(x_3 = 1) = p(x_1 = 1)\, p(x_3 = 1) + p(x_2 = 1)\, p(x_3 = 1) -$$
$$- p(x_1 = 1)\, p(x_2 = 1)\, p(x_3 = 1);$$

$$p(f=1)=p(f_1=1)+p(f_3=1)-p(f_1=1)p(f_3=1)=p(x_1=1)p(x_2=1)+$$
$$p(x_1=1)p(x_3=1)+p(x_2=1)p(x_3=1)-p(x_1=1)p(x_2=1)p(x_3=1)-$$
$$-p^2(x_1=1)p(x_2=1)p(x_3=1)-p(x_1=1)p^2(x_2=1)p(x_3=1)+$$
$$+p^2(x_1=1)p^2(x_2=1)p(x_3=1).$$

Уменьшив показатели степеней до единицы, окончательно имеем:

$$p(f=1)=p(x_1=1)p(x_2=1)+p(x_1=1)p(x_3=1)+p(x_2=1)p(x_3=1)-$$
$$2p(x_1=1)p(x_2=1)p(x_3=1).$$

Задача определения вероятностного описания цифровой схемы может быть значительно упрощена для случая реализации последней на однотипных логических элементах. Так, в [3, 4] показано, что для древовидной структуры цифровой схемы, реализованной на элементах И-НЕ, входы каждого из которых подключены к выходам элементов предыдущего уровня, вероятностное описание будет иметь вид [3]:

$$p_l(y=1)=p_{l-1}^n(y=1)=\left[1-p_{l-1}(y=0)\right]^n,$$

где $p_l(y=0)$ и $p_l(y=1)$ вероятности появления нулевого и единичного значений на выходах элементов l-го уровня.

Оригинальный алгоритм, основанный на преобразовании исходной цифровой схемы в древовидную структуру, рассматривается в [169]. Сущность данного метода заключается в определении пределов, в которых находится искомая вероятность, за счет искусственного задания граничных вероятностей в точках разветвления. В этом случае разветвление представляется в виде независимых входов, вероятность появления единицы на которых лежит в пределах от 0 до 1. Так, например, комбинационная схема, приведенная на рис. 3.4, имеет разветвления. Для этой схемы, согласно указанному методу, на входах элемента ИЛИ (2) вероятности появления единичных сигналов принимаются равными величинам в пределах от 0 до 1 (0 - 1). Соответственно $p(f_2=1)=0-1$ и $p(f_3=1)=p(f_2=1)p(x_3=1)=0-p(x_3=1)$. Вероятность появления единичного сигнала на выходе элемента И (1) равна $p(x_1=1)\,p(x_2=1)$, а выходная вероятность соответственно $p(f=1)=p(f_1=1)+p(f_3=1)-p(f_1=1)\,p(f_3=1)$. Подставляя верхний ($p(x_3=1)$) и нижний (0) пределы изменения величины $p(f_3=1)$ в последнее выражение, окончательно получаем:

$$p(f=1)=\left[p(x_1=1)p(x_2=1)\right]-$$
$$-\left[p(x_1=1)p(x_2=1)+p(x_3=1)-p(x_1=1)p(x_2=1)p(x_3=1)\right].$$

При $p(x_1 = 1) = p(x_2 = 1) = p(x_3 = 1) = p$ будем иметь:

$$p(f = 1) = (p^2) - (p + p^2 - p^3),$$

где p^2 и $p + p^2 - p^3$, соответственно нижняя и верхняя оценки вероятности $p(f = 1)$.

Более сложной является задача определения вероятностного описания последовательностных цифровых схем. В данном случае аналитические подходы практически неприемлемы. Единственной альтернативой может быть применение статистического эксперимента. Примером реализации статистического подхода для определения вероятностного описания цифровых схем служит приложение *STAFAN* (*Statistic Fault Analysis*) [101], представляющее собой систему математического обеспечения, позволяющую, в частности, оценить вероятности появления единичного сигнала на любом из полюсов схемы.

Вероятностное описание цифровой схемы, полученное либо аналитически, либо по экспериментальным данным, необходимо лишь для формирования эталонных вероятностей, используемых при сравнении их с реально определенными значениями в результате тестирования цифровых схем. Такие вероятности часто называют сигнальными [3, 169]. С их помощью можно оценить вероятность появления единичного сигнала на заданных полюсах схемы.

Задача определения вероятностей обнаружения неисправностей на заданных полюсах цифровой схемы наиболее важная, что дает возможность оценить эффективность вероятностного тестирования в каждом конкретном случае и наметить пути ее увеличения. Рассмотрим ряд практических подходов, используемых для оценки вероятности обнаружения заданных неисправностей.

3.3. Оценка вероятности обнаружения неисправности

Вероятность $p_d(f \equiv \chi)$, где $\chi \in \{0, 1\}$, обнаружения константной неисправности $\equiv \chi$ на полюсе f цифровой схемы представляет собой меру оценки эффективности вероятностного тестирования и характеризует сложность определения заданной неисправности цифровой схемы. Более детально содержание понятия вероятности обнаружения неисправности рассмотрим на примере цифровой схемы, имеющей вид n-входового элемента И.

Как и для общего случая, процедура тестирования элемента И состоит в подаче на его входы последовательностей случайных двоичных цифр с вероятностями $p(x_1 = 1)$, $p(x_2 = 1)$, ..., $p(x_n = 1)$ и анализе выходной реакции. Не нарушая общности рассуждений, предположим, что $p(x_1 = 1) = p(x_2 = 1) = ... = p(x_n = 1) = p$.

Вероятность $p_d(f \equiv \chi)$ обнаружения неисправности $\equiv \chi$ характеризуется вероятностью совместного выполнения двух событий, а именно проявления

неисправности в точке ее возникновения и транспортировки на один из выходов схемы. При этом событие проявления неисправности $\equiv\chi$ можно оценить вероятностью $p_e(f \equiv\chi)$ проявления ее на заданном полюсе f, а событие транспортировки, соответственно вероятностью $p_t(f \equiv\chi)$ наблюдения этой неисправности хотя бы по одному из выходов схемы.

Для n-входового элемента И вероятность проявления неисправности $\equiv 0$ по первому входу будет равна вероятности появления на данном входе единичного символа, т. е. $p_e(x_1 \equiv 0) = p(x_1 = 1) = p$, а для $p_e(x_1 \equiv 1)$, соответственно вероятности появления на данном входе нулевого уровня. Тогда $p_e(x_1 \equiv 1) = p(x_1 = 0) = 1 - p(x_1 = 1) = 1 - p$. Условием транспортировки любой неисправности первого входа элемента И будет активность пути от данного входа к выходу элемента. Указанное условие определяется событием $x_2 = 1$, $x_3 = 1$, ..., $x_n = 1$ с вероятностью $p(x_2 = 1, x_3 = 1, ..., x_n = 1)$. Учитывая независимость входных случайных двоичных последовательностей, окончательно получаем $p_t(f \equiv\chi) = p(x_2 = 1)\, p(x_3 = 1) \ldots p(x_n = 1) = p^{n-1}$. Вероятность $p_e(y \equiv\chi)$ проявления неисправности $\equiv\chi$ по выходу элемента И для $\equiv 0$ определяется вероятностью $p(y = 1)$, а для $\equiv 1$ вероятностью $p(y = 0)$, при этом вероятность $p_t(y \equiv\chi)$ строго равна единице, так как выход элемента И является выходным наблюдаемым полюсом схемы.

Для рассмотренного простейшего случая цифровой схемы события проявления неисправности и ее транспортировки независимые, поэтому в данном случае вероятность $p_d(f \equiv\chi)$ вычисляется согласно следующему соотношению:

$$p_d(f \equiv \chi) = p_e(f \equiv \chi)\, p_t(f \equiv \chi).$$

В таблице 3.1 приведены вероятности $p_e(f \equiv\chi)$, $p_t(f \equiv\chi)$ и $p_d(f \equiv\chi)$ для константных неисправностей i-х входов $i \in \{1, 2, \ldots, n\}$, и выходов элементов И и ИЛИ для случая $p(x_1 = 1) = p(x_2 = 1) = ... = p(x_n = 1) = p$.

Таблица 3.1 – *Значения вероятностей проявления, транспортировки и обнаружения для элементов* И *и* ИЛИ *с n входами*

$f \equiv\chi$	И			ИЛИ		
	$p_e(f \equiv\chi)$	$p_t(f \equiv\chi)$	$p_d(f \equiv\chi)$	$p_e(f \equiv\chi)$	$p_t(f \equiv\chi)$	$p_d(f \equiv\chi)$
$x_i \equiv 0$	p	p^{n-1}	p^n	p	$(1-p)^{n-1}$	$p(1-p)^{n-1}$
$x_i \equiv 1$	$1-p$	p^{n-1}	$p^{n-1}-p^n$	$1-p$	$(1-p)^{n-1}$	$(1-p)^n$
$y \equiv 0$	p^n	1	p^n	$1-(1-p)^n$	1	$1-(1-p)^n$
$y \equiv 1$	$1-p^n$	1	$1-p^n$	$(1-p)^n$	1	$(1-p)^n$

Для $p = 1/2$, согласно соотношениям, рассмотренным в таблице 3.1, вероятность $p_d(x_i \equiv 0)$ для $n = 4$ и элемента И равна $p = 1/2^4$, а $p_d(y \equiv 1) = 1 - 1/2^4$.

Понятия вероятностей обнаружения, проявления и транспортировки неисправности в значительной степени аналогичны понятиям управляемости, наблюдаемости и тестируемости цифровой схемы [208, 302]. Отличие состо-

ит лишь в том, что данные характеристики вероятностей обнаружения, проявления и транспортировки вычисляются не только для равномерного распределения тестовых наборов, а и для произвольного распределения входных вероятностей.

Аналитическое определение значений вероятностей p_d, p_e и p_t для цифровых схем общего вида значительно усложняется зависимостью событий проявления неисправности и ее транспортировки на один из выходов схемы. Поэтому в большинстве случаев вычисляется лишь некоторая оценка вероятности обнаружения неисправности. Так, в [169] задача определения указанной вероятности сводится к задаче вычисления сигнальной вероятности единичного значения на выходе искусственно введённого элемента И (φ), как это показано на приведенном примере (рис. 3.5).

Фрагмент цифровой схемы, приведенной на рисунке, состоит из последовательно соединенных элементов И-НЕ (*1*), И (*2*), ИЛИ (*3*), ИЛИ-НЕ (*4*). В качестве неисправности рассмотрим константную неисправность $f_1 \equiv 0$, которая транспортируется на выход F по пути, проходящему через элементы *2*, *3*, и *4*.

Условием проявления неисправности $f_1 \equiv 0$ в точке ее возникновения является появление единичного сигнала на выходе первого элемента И (1). В тоже время условием ее транспортировки на выход F является наличие значения единицы на первом входе элемента И (2) и нуля на первых входах элементов ИЛИ (3) и (4). Приведенное множество условий, определяющих наблюдаемость неисправности $f_1 \equiv 0$, объединяется путем введения фиктивного элемента И (φ), выходное значение которого равняется единице только в случае выполнения всех предыдущих условий [208, 302].

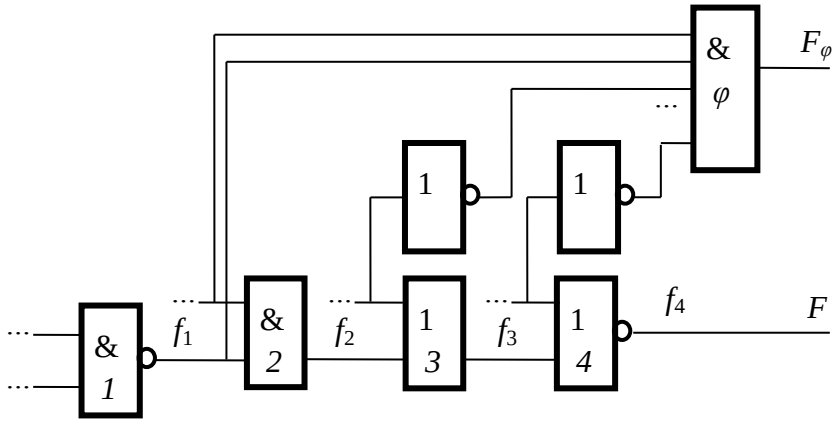


Рисунок 3.5 –*Пример сведения задачи определения вероятности обнаружения неисправности* $f_1 \equiv 0$ *к задаче определения сигнальной вероятности на выходе* F_φ *искусственно введенного элемента* И (φ)

Следовательно, вероятность $p(F_\varphi = 1)$ равенства единице выходного значения фиктивного элемента И (φ) будет соответствовать вероятности обнаружения заданной неисправности $f_1 \equiv 0$, то есть:

$$p_d(f_1 \equiv 0) = p(F_\varphi \equiv 1).$$

Таким образом, вычисление вероятности обнаружения неисправности сводится к определению сигнальной вероятности. Для этого могут использоваться любые из ранее рассмотренных методов. В том случае, когда существует множество возможных путей транспортировки неисправности на выходы цифровой схемы, сигнальная вероятность для выходного полюса фиктивного элемента И будет служить нижней оценкой искомой вероятности обнаружения неисправности. Кроме того, нет необходимости в определении вероятностей обнаружения всевозможных неисправностей схемы. Достаточно вычислить только вероятности обнаружения характеристических неисправностей, т. е. неисправностей, лежащих на входных полюсах схемы и на полюсах, соответствующих узлам разветвления [15]. Анализ вероятностей обнаружения неисправностей позволяет оценить эффективность вероятностного тестирования цифровой схемы, которая в первую очередь характеризуется его сложностью, определяемой длиной вероятностного теста.

3.4. Определение длины вероятностного теста

Основным параметром систем контроля и диагностики вычислительных систем является время, необходимое для проведения процедуры тестирования, которое однозначно определяется длиной тестовых последовательностей. В случае детерминированных методов синтеза тестов их длина определяется требуемой полнотой покрытия возможных неисправностей схемы и в среднем принимает приемлемые величины. Следует отметить, что с увеличением количества наборов возрастает полнота покрытия. Аналогичная зависимость, очевидно, справедлива и для вероятностного тестирования. Об этом свидетельствует пример использования случайных тестовых наборов для тестирования двух достаточно сложных цифровых устройств, рассмотренных в [58].

Как видно из таблицы 3.2, полнота покрытия неисправностей вероятностным тестом для обоих цифровых устройств повышается с увеличением числа тестовых наборов, представляющих собой равномерно распределенные случайные числа [58, 207, 302]. Сравниваемая полнота покрытия с полнотой покрытия для детерминированных тестов достигается при длине вероятностного теста, равной 10 000 наборов. В тоже время для сравнительно небольшого числа наборов, равного 100 значениям полноты покрытия вероятностного теста, достаточно высокие.

Таблица 3.2 – *Значения полноты покрытия вероятностного теста в зависимости от его длины*

Цифровое устройство	Количество входов	Количество элементов	Зависимость полноты покрытия (%) от количества тестовых наборов			Полнота покрытия детерминированного теста
			100	1000	10000	
№1	63	926	86,1	94,1	96,3	96,7
№2	54	1103	75,2	92,3	95,9	97,1

Зависимость полноты покрытия неисправностей от количества тестовых наборов, представляющих собой случайные значения, впервые была экспериментально исследована в работе [58]. Показано, что полнота покрытия F_C, которая достигается при формировании N случайных тестовых наборов, может быть аппроксимирована экспонентой следующего вида [58]:

$$F_C = (1 - e^{-\lambda \log_{10} N}) \times 100\%, \qquad (3.5)$$

где λ представляет собой постоянную величину, которая принимает конкретные значения для каждого определенного типа цифровых устройств.

Аналогичная зависимость может быть получена в результате следующих рассуждений относительно неисправности, имеющей вероятность обнаружения, равную p_d. Как и в предыдущих разделах, данная вероятность характеризует вероятность обнаружения заданной неисправности при подаче одного тестового набора вероятностного теста. Отметим, что вероятностный тест состоит из N случайных и независимых наборов. Таким образом, при подаче одного набора рассматриваемая неисправность будет обнаружена с вероятностью $p_d(1) = p_d$ и соответственно не обнаружена с вероятностью $p_n(1) = 1 - p_d$. При подаче двух тестовых наборов вероятность того, что неисправность будет не обнаружена, равняется $p_n(2) = (1 - p_d)^2$. Соответственно вероятность обнаружения неисправности при подаче двух тестовых наборов примет значение $p_d(2) = 1 - p_n(2) = 1 - (1 - p_d)^2$. При подаче N независимых случайных наборов указанная вероятность будет возрастать и в общем случае определяться соотношением:

$$p_d(N) = 1 - p_n(N). \qquad (3.6)$$

где $p_d(N)$ представляет собой вероятность обнаружения неисправности при подаче N тестовых наборов, а $p_n(N)$ – вероятность необнаружения указанной неисправности. Учитывая независимость формируемых тестовых наборов, вероятность $p_n(N)$ может быть вычислена как произведение N сомножителей $(1 - p_d)$, представляющих собой вероятность необнаружения неисправности при подаче одного тестового набора. С учетом последнего замечания соотношение (3.6) примет вид:

$$p_d(N) = 1 - (1 - p_d)^N,$$

откуда для заданной достоверности тестирования, определяемой вероятностью $p_d(N)$, необходимое количество случайных тестовых наборов вероятностного теста можно найти следующим образом:

$$N = \frac{\ln(1 - p_d(N))}{\ln(1 - p_d)}. \qquad (3.7)$$

Для n-входового элемента И количество тестовых наборов N вероятностного теста, необходимое для определения его неисправности $y \equiv 0$ (см. таблицу 3.1) с достоверностью, определяемой доверительной вероятностью $p_d(N) = 0{,}99$, вычисляется согласно (3.7). Учитывая равенство $p_d = p^n$ для рассматриваемой неисправности, получим:

$$N = \ln(1 - 0{,}99) / \ln(1 - p^n),$$

где p есть вероятность появления единичного значения в случайных последовательностях, двоичных наборов теста подаваемых на входы n-входового элемента И.

Анализ последней зависимости показывает, что длина N тестовой последовательности значительно зависит от вида тестируемой цифровой схемы (в данном случае от количества n входов элемента И) и распределения входных вероятностей p. Конкретный пример, рассмотренный в [3] для случая цифровых схем, реализованных на элементах И-НЕ, показывает, что для входных вероятностей $p = 0{,}5$, использованных для тестирования 13-уровневых цифровых схем с доверительной вероятностью $p_d(N) = 0{,}99$, вероятностный тест должен состоять из $N = 3 \times 10^8$ двоичных наборов. Количество необходимых тестовых наборов N уменьшается до величины 2×10^4 при задании входных вероятностей p, равных 0,617.

Приведенные примеры свидетельствуют о необходимости решения задачи определения значений входных вероятностей, позволяющих минимизировать длину вероятностных тестовых последовательностей.

3.5. Методы определения оптимальных значений вероятностей

Под оптимальными значениями вероятностей входных переменных вероятностных тестов понимаются такие их значения, которые позволяют достичь минимальной длины N тестовой последовательности, обеспечивающей заданную вероятность обнаружения *наиболее трудной для обнаружения неисправности* (*hard to detect fault*) [296, 302]. Наиболее трудной для обнаружения неисправностью считается такая неисправность, которая имеет минимальное значение вероятности p_d ее обнаружения. Таким образом, основной проблемой при определении оптимальных значений вероятностей входных

переменных является обеспечение максимально возможного значения минимальной вероятности обнаружения неисправности цифровой схемы.

В качестве примера, иллюстрирующего определение оптимальных значений входных вероятностей, рассмотрим случай n-входового элемента И, вероятности обнаружения неисправностей которого приведены в таблице 3.1. Значения вероятностей обнаружения неисправностей $p_d(x_i \equiv 0) = p^n$, $p_d(x_i \equiv 1) = p^{n-1} - p^n$, $p_d(y \equiv 0) = p^n$ и $p_d(y \equiv 1) = 1 - p^n$ зависят как от вероятности p появления единичных значений на входах элемента И, так и от количества его входов n. Анализ зависимостей p_d от p показывает, что:

$$\max_p \min(p^n, 1 - p^n, p^{n-1} p^n),$$

для $n = 2$ достигается при $p = 0{,}5$, а для $n = 3$ при $p = 2/3 = 0{,}666$ [207, 302]. Для произвольного n оптимальную вероятность p_0 появления единичных символов вероятностного теста на входах элемента И определяется из уравнения:

$$\frac{\partial(p^{n-1} - p^n)}{\partial p} = 0.$$

Откуда получаем, что $p_0 = (n - 1)/n$ [207, 302]. Аналогичным образом можно определить p_0 для цифровой схемы, представляющей собой n-входовой элемент ИЛИ. В этом случае p_0 для, например, $n = 2$ принимает значение 0,617.

В то же время необходимо отметить, что для общего случая задача определения оптимальных значений входных вероятностей является нелинейной задачей оптимизации, требующей сложных математических вычислений.

В работе [296] исследован один из возможных методов определения оптимальных значений входных вероятностей для реализации вероятностного тестирования. Сущность предложенного метода заключается в максимальном увеличении минимальной вероятности обнаружения неисправностей схемы, являющихся самыми трудными для нахождения (неисправностей с минимальными вероятностями обнаружения). Показано, что возможными местами неисправностей с минимальной вероятностью обнаружения являются входные полюса схемы, а также полюса, берущие свое начало в точках разветвления. Определение вектора оптимальных входных вероятностей основано на максимизации целевой функции, представляющей собой минимальную вероятность обнаружения неисправности. Для этого используется так называемый релаксационный метод, который не во всех случаях дает оптимальное решение. Однако решение, полученное на основании указанного метода, всегда лучше тривиального случая, когда входные вероятности равны 0,5.

Аналогичная задача определения оптимальных распределений входных вероятностей приведена в [5], где также рассматривается случай комбинационных цифровых схем. Показано, что для произвольной цифровой схемы, содержащей элементы памяти, задача вычисления оптимальных значений входных вероятностей сложнее задачи построения детерминированных тестовых последовательностей. В связи с этим применение вероятностных принципов оказывается неэффективным.

Определение эффективности использования вероятностного тестирования для цифровых схем по минимальной информации о проверяемой цифровой схеме дано в [5]. Целесообразность применения вероятностного тестирования устанавливается по трем основным параметрам схемы, а именно количеству n входов схемы, максимальному числу L элементов, последовательно включенных между входами и выходами схемы, и среднему количеству n_{av} входов элементов схемы. Несмотря на упрощенный подход, использованный автором, результаты работы [5], представленные в виде диаграммы, могут быть полезны для разработки практических рекомендаций по использованию вероятностного тестирования.

Дальнейшим развитием вероятностного тестирования является применение различных модификаций как общей схемы вероятностного тестирования, так и отдельных его компонентов. Весьма интересным и перспективным является применение для современных вычислительных систем управляемого вероятностного тестирования.

Глава 4. Управляемое вероятностное тестирование

4.1. Сущность и виды анти-вероятностного тестирования

Как отмечалось ранее, одной из самых распространенных технологий применения вероятностного тестирования цифровых устройств и программного обеспечения, является метод черного ящика [78, 125, 127, 175]. В этом случае структура и функциональность объекта тестирования принимаются неизвестной, либо считаются таковой, и никак не учитываются при формировании тестовых наборов. Более того вероятностное тестирование не использует информацию, которая доступна в процессе генерирования очередного тестового набора. Эта информация может быть получена из предыдущих тестовых наборов и использована при формировании очередного тестового воздействия [126]. Развитием вероятностного тестирования является новый подход, называемый *анти-вероятностным тестированием* (*antirandom testing −AT*), который основан на том, что каждый последующий тестовый набор формируется с использованием некоторой характеристики, либо нескольких характеристик, получаемых на основании предыдущих тестовых наборов [126, 202, 203].

Основная предпосылка анти-вероятностного тестирования заключается в том, что в целях достижения более высокого покрытия неисправностей, обнаруживаемых тестом, и минимизации его сложности, то есть в повышении эффективности вероятностного теста, необходимо целенаправленно выбирать очередной тестовый набор в зависимости от ранее сгенерированных наборов. Очевидным примером повышения эффективности вероятностного теста является простейшее исключение повторяющихся случайных наборов, которые возможны при реализации вероятностного тестирования. В общем случае критерием выбора очередного случайного тестового набора является нахождение максимально отличного (максимально удаленного) тестового набора от ранее сгенерированных наборов. Для количественной оценки отличия (расстояния) текущего тестового набора от предыдущих наборов, были предложены методы, основанные на применении *расстояния Хэмминга* (*Hamming distance*) и *евклидова расстояния* (*Euclidean distance*) [126, 202]. Новый тестовый набор согласно указанным методам выбирается таким образом, чтобы метрики различия принимали максимальное значение [126, 202, 203]. Этот подход оказался более эффективным по сравнению с вероятностным тестированием [126, 202, 203]. К сожалению, основным недостатком анти-вероятностного тестирования является его большая вычислительная сложность. По существу реализация анти-вероятностного тестирования требует перечисления всевозможных входных тестовых воздействий и вычисления расстояния для каждого потенциального кандидата в очередные тестовые наборы [126].

Различные модификации анти-вероятностного тестирования, такие как: *быстрое анти-вероятностное тестирование* (*fast antirandom − FAR*) [130];

адаптивное вероятностное тестирование (*adaptive random testing*) [34, 102, 240]; *эволюционное вероятностное тестирование* (*evolutionary random testing*) [188]; *эффективное вероятностное тестирование* (*good random testing*) [30]; *ограниченное вероятностное тестирование* (*restricted random testing*) [31, 32]; *зеркальное вероятностное тестирование* (*mirror random testing*) [119]; *упорядоченное вероятностное тестирование* (*orderly random testing*) [204] и другие решения, также основаны на вычислении характеристик для предыдущих тестовых наборов, и также характеризуются большой вычислительной сложностью.

Под *управляемым вероятностным тестированием* (*controlled random testing*) в дальнейшем будем понимать случайную тестовую последовательность, в которой очередной тестовый набор формируется с учетом ранее сгенерированных предыдущих наборов [228, 334, 347]. Ключевой особенностью контролируемого генерирования случайных тестовых наборов является информация, которая извлекается в виде некоторых характеристик (метрик) из ранее сгенерированных тестовых наборов и используется для формирования очередного тестового набора [228, 334]. Для всех методов управляемого случайного тестирования, используемого для тестового диагностирования цифровых устройств и программных приложений с N входами и пространством входных наборов, состоящим из 2^N двоичных наборов (векторов), справедливы следующие определения.

Определение 4.1. *Тест* (*T*) представляет собой множество из $q < 2^N$ тестовых наборов $\{T_0, T_1, T_2, \ldots, T_{q-1}\}$, где $T_i = t_{i,N-1}, t_{i,N-2}, \ldots, t_{i,2}, t_{i,1}, t_{i,0}$ и $t_{i,l} \in \{0, 1\}$, а N является размером набора в битах.

Определение 4.2. *Вероятностный тест* (*RT*) представляет собой тест $RT = \{T_0, T_1, T_2, \ldots, T_{q-1}\}$, который состоит из $q < 2^N$ случайных и независимых тестовых наборов T_i, $i \in \{0, 1, 2, ..., q-1\}$.

Определение 4.3. *Управляемым вероятностным тестом* $CRT = \{T_0, T_1, T_2, \ldots, T_{q-1}\}$ является тест T, состоящий из сгенерированных случайным образом тестовых наборов T_i, $i \in \{0, 1, 2, ..., q-1\}$, таких, что T_i удовлетворяет какому-либо критерию, либо критериям, полученным на основании множества $\{T_0, T_1, T_2, \ldots, T_{i-1}\}$ предыдущих наборов.

Одним из первых подходов к контролируемому формированию случайных тестовых наборов является анти-вероятностное тестирование, приведенное в [102]. Предложенный подход использовал тот факт, что если очередной тестовый набор будет сформирован в зависимости от ранее сгенерированных наборов, это позволит достичь большей эффективности теста. Подобная зависимость существенно изменяет структуру вероятностного теста, свойства которого весьма далеки от свойств случайных последовательностей. Поэтому одним из первых названий управляемых вероятностных тестов был использован термин *анти-вероятностный тест* (*AT*), который соответствует определению 4.3 [102]. Для того чтобы очередной тестовый набор T_i являлся максимально отличным от предыдущих наборов $T_0, T_1, T_2, \ldots, T_{i-1,}$ в качестве критериев использовались расстояние Хэмминга и декартово (евклидово)

расстояние [102]. Данные характеристики определялись для двоичных тестовых наборов T_i и T_j , где расстояние Хэмминга $HD(T_i, T_j)$ вычислялось как вес $w(T_i \oplus T_j)$ вектора $T_i \oplus T_j$ согласно соотношению:

$$HD(T_i, T_j) = w(T_i \oplus T_j) = \sum_{l=0}^{N-1} (t_{i,l} \oplus t_{j,l}). \quad (4.1)$$

Для оценки различия между двумя тестовыми наборами $T_i = t_{i,N-1}, t_{i,N-2}, \ldots, t_{i,2}, t_{i,1}, t_{i,0}$ и $T_j = t_{j,N-1}, t_{j,N-2}, \ldots, t_{j,2}, t_{j,1}, t_{j,0}$ определяется характеристика $S_{qg}(T_i, T_j)$, где q, $g \in \{0, 1\}$ является числом пар компонент (t_{ik}, t_{jk}), таких, что $t_{ik} = q$ и $t_{jk} = g$. Различают четыре величины $S_{qg}(T_i, T_j)$, а именно: $S_{00}(T_i, T_j)$, $S_{01}(T_i, T_j)$, $S_{10}(T_i, T_j)$ и $S_{11}(T_i, T_j)$, которые используются для определения различия или подобия двоичных векторов [194]. Отметим, что $HD(T_i, T_j) = S_{01}(T_i, T_j) + S_{10}(T_i, T_j)$.

Декартово расстояние $CD(T_i, T_j)$ определяется в соответствии с выражением:

$$CD(T_i, T_j) = \sqrt{\sum_{l=0}^{N-1} (t_{i,l} - t_{j,l})^2} = \sqrt{\sum_{l=0}^{N-1} |t_{i,l} - t_{j,l}|} = \sqrt{\sum_{l=0}^{N-1} (t_{i,l} \oplus t_{j,l})} = \sqrt{HD(T_i, T_j)}. \quad (4.2)$$

Пример 4.1. Для двух двоичных наборов T_i = 0 0 0 0 0 0 и T_j = 1 0 1 0 1 0 получим $HD(T_i, T_j) = 3$ и $CD(T_i, T_j) = \sqrt{3}$ = 1,732.

Очередной тестовый набор T_i формируется таким образом, чтобы быть максимально отличным от всех ранее сгенерированных наборов. В данном случае принимается гипотеза, что для двух тестовых наборов, имеющих минимальное расстояние (4.1) или (4.2), количество обнаруживаемых неисправностей (ошибок) вторым набором будет минимальным и, наоборот, для максимальных значений указанных характеристик обнаруживающая способность второго набора максимальна [202, 203]. Для количества рассматриваемых тестовых наборов более чем два, при формировании набора T_i применяются суммарные значения расстояний T_i по отношению к T_0, T_1, T_2, …, T_{i-1} [202]. Тогда для очередного набора T_i суммарное значение расстояний относительно предыдущих наборов T_0, T_1, T_2, …, T_{i-1} вычисляется как:

$$THD(T_i) = \sum_{j=0}^{i-1} HD(T_i, T_j); \quad TCD(T_i) = \sum_{j=0}^{i-1} CD(T_i, T_j). \quad (4.3)$$

Здесь $THD(T_i)$ и $TCD(T_i)$ представляют собой *суммарное расстояние Хэмминга* (*total hamming distance − THD*) и *суммарное декартово расстояние* (*total cartesian distance − TCD*), соответственно.

Пример 4.2. Для анти-вероятностного теста AT = {0 0 0, 1 1 1, 0 1 0, 1 0 1} с параметрами N = 3 и q = 4 значения ранее определенных характеристик приведены в таблице 4.1.

Таблица 4.1 – *Значения расстояний* $THD(T_i)$ и $TCD(T_i)$

i	Набор (T_i)	$t_{i,2}\ t_{i,1}\ t_{i,0}$	$THD(T_i)$	$TCD(T_i)$
0	T_0	0 0 0	-	-
1	T_1	1 1 1	3	1,7320
2	T_2	0 1 0	3	2,4142
3	T_3	1 0 1	6	4,1460

Различают тесты с *максимальным суммарным расстоянием Хэмминга* (*maximal hamming distance antirandom test − MHDAT*) и *максимальным декартовым расстоянием* (*maximal cartesian distance antirandom test − MCDAT*) [126, 202, 203]. В приведенном примере $AT = \{0\ 0\ 0,\ 1\ 1\ 1,\ 0\ 1\ 0,\ 1\ 0\ 1\}$ является одновременно *MHDAT* и *MCDAT*.

В общем случае для *MHDAT* и *MCDAT* выполняются следующие свойства.

Результатом произвольной перестановки бит $t_{i,j}$ во всех наборах теста *MHDAT* (*MCDAT*) одновременно является тест *MHDAT* (*MCDAT*) [126, 202].

Результатом инвертирования всех бит наборов теста *MHDAT* (*MCDAT*) является тест *MHDAT* (*MCDAT*) [126, 202].

Любой *MHDAT* (*MCDAT*) всегда содержит инверсные наборы, такие, что за набором T_{2k} всегда будет следовать набор $T_{2k+1} = \overline{T}_{2k}$ для $k = 0, 1, 2, \ldots$ [333].

Приведенные свойства позволяют уменьшать сложность генерирования *MHDAT* (*MCDAT*), однако в общем случае, к сожалению, не исключают процедуры перечисления пространства возможных тестовых наборов и вычисления расстояний для каждого потенциального кандидата в тестовые наборы [333]. Даже для улучшенных версий методов генерирования анти-вероятностных тестов вычислительная сложность для реальных значений N является чрезвычайно большой.

Генерирование быстрых анти-вероятностных тестов (*fast antirandom − FAR*) основано на вычислении так называемых *центроидов* (*centroid pattern*), определяемых на основании предыдущих наборов T_0, T_1, T_2, …, T_{i-1} для $T_i = t_{i,N-1}, t_{i,N-2}, \ldots t_{i,2}, t_{i,1}, t_{i,0}$ [130]. Согласно *FAR* алгоритму тестовые наборы T_0, T_1, T_2, …, $T_{i\text{-}1}$ рассматриваются как двухмерные массивые $i \times N$ бит. Для каждого столбца $l \in \{N-1, N-2, N-3, \ldots, 1, 0\}$ значение l-го элемента c_l *центроида* $C = c_{N-1}, c_{N-2}, \ldots, c_2, c_1, c_0$ определяется как сумма $t_{i,l} \in \{0, 1\}$, деленная на количество i строк. В результате получается центроид, элементы которого c_l принимают значения в интервале от 0 до 1. Далее согласно *FAR* алгоритму значение c_l округляется до 0, если $c_l < 0{,}5$ или до 1, если $c_l > 0{,}5$. В случае, когда $c_l = 0{,}5$ *FAR* алгоритм регламентирует равновероятное округление c_l до 0 или 1 [130]. Как результат приведенных преобразований формируется двоичный центроид C_b. На финальном этапе генерируется T_i как результат инвертирования C_b, то есть $T_i = \overline{C}_b$.

Пример 4.3. В качестве примера возьмем тестовые наборы T_0 = 0 0 0, T_1 = 1 1 1 и T_2 = 0 1 0, тогда элементы центроида примут следующие значения $C = c_2, c_1, c_0$ = 1/3, 2/3, 1/3 = 0,333, 0,666, 0,333. Соответствующий бинарный центроид C_b = 0 1 0, а его инверсное значение 1 0 1 и будет тестовым набором T_3 = 1 0 1. Следует отметить идентичность полученного результата T_3 с аналогичным набором, сформированным согласно классической интерпретации анти-вероятностного тестирования (см. пример 4.2).

Применение вероятностного округления для случая, когда c_l = 0,5, может привести к ошибкам, снижающим эффективность данного алгоритма. Действительно, для предыдущего примера, когда рассматриваются два набора T_0 = 0 0 0, T_1 = 1 1 1 центроид $C = c_2, c_1, c_0$ = 0,5, 0,5, 0,5, тогда очередным набором T_2 может быть любой набор, состоящий из трех бит, в том числе T_0 = 0 0 0 и T_1 = 1 1 1.

Концепция упорядоченных вероятностных тестов впервые была предложена в [204] как *анти-вероятностные тесты с полу-максимальным расстоянием* (*semi-maximum distance testing sequences − SMDTS*), для которых каждый тестовый набор, входящий в тест, имеет свое инверсное значение. Например, T = {0 0 0, 1 1 1, 0 1 0, 1 0 1} есть *SMDTS*, так как {0 0 0, 1 1 1} и {0 1 0, 1 0 1} являются инверсными значениями тестовых наборов. Для *SMDTS* были предложены новые характеристики для оценки расстояния при генерировании очередного тестового набора T_i [205]. По сути, предложенные в [204, 205] характеристики *THD*(T) и *TCD*(T) представляют собой *суммарное расстояние Хэмминга* (*total hamming distance − THD*) и *суммарное декартово расстояние* (*total cartesian distance − TCD*), соответственно, для теста T, состоящего из q тестовых наборов, и вычисляется, как:

$$THD(T) = \sum_{i=1}^{q-1}\sum_{j=0}^{i-1} HD(T_i, T_j); \qquad TCD(T) = \sum_{i=1}^{q-1}\sum_{j=0}^{i-1} CD(T_i, T_j). \tag{4.4}$$

Здесь $HD(T_i, T_j)$ и $CD(T_i, T_j)$ вычисляются для всех $i \neq j$; $i, j \in \{0, 1, \ldots, q-1\}$.

Процедура генерирования *SMDTS*, содержащего $q = 2k$, k = 0, 1, 2, … наборов, состоит из случайных тестовых наборов с четными индексами. То есть T_0, T_2, T_4, … генерируются как равновероятные двоичные векторы, состоящие из N бит, а тестовые наборы с нечетными индексами T_1, T_3, T_5, … являются инверсными значениями по отношению к T_0, T_2, T_4, … . Тогда *THD*(T) для *SMDTS* = T_0, T_1, T_2, …, T_{2k-2}, T_{2k-1}, равняется k^2N [205]. Для ранее рассмотренного примера 4.2 теста *SMDTS* = {0 0 0, 1 1 1, 0 1 0, 1 0 1} k = 2 и N = 3, тогда $THD(T) = k^2N = 2^23 = 12$, что соответствует соотношению (4.4). В силу того, что только половина тестовых наборов достигает максимального расстояния Хемминга, такие последовательности и получили название тестов с полу-максимальным расстоянием [204, 205].

Тесты с полным максимальным расстоянием (*total maximum distance test sequences − TMDTS*) являются дальнейшим развитием упорядоченных ве-

роятностных тестов [204]. Для их генерирования одновременно используются обе метрики *THD* и *TCD* в соответствии со следующей процедурой [204].

Процедура 4.1. Построение *TMDTS* для $q = 2k$.

1. Первый тестовый набор T_0 генерируется как случайный двоичный N разрядный вектор.

2. Для получения очередного i-го набора T_1, T_3, T_5, … T_{2k-1} с нечетным индексом используется соотношение $T_{2k+1} = \overline{T}_{2k}$, которое для $k = 0$ принимает вид: $T_1 = \overline{T}_0$.

3. Для получения каждого нового набора T_2, T_4, T_6, … T_{2k-2} с четным индексом из множества возможных кандидатов в тесты выбирается такой, для которого $THD(T)$ и $TCD(T)$ принимают максимальные значения, причем для их вычисления используются ранее сгенерированные тестовые наборы.

4. Этапы 2 и 3 повторяются до тех пор, пока все q наборов не будут сгенерированы.

Очевидно, что наиболее трудоемким является этап 3, который требует большого объема вычислений.

Основным недостатком *TMDTS* является ограничение длины таких тестов, ровно как и всех упорядоченных случайных тестов. Это следует из того факта, что, чем больше минимальное расстояние $minCD(T_i, T_j)$, используемое как критерий включения T_i в тест, тем меньше будет количество Q кандидатов в тесты, это следует из предельной *оценки Хэмминга* (*Hamming bound*) [83]. Эта оценка для $minHD(T_i, T_j) = 2r + 1$ и $minCD(T_i, T_j) = \sqrt{2r+1}$ может быть представлена как неравенство:

$$Q \le \frac{2^N}{\sum_{l=0}^{r} \binom{N}{l}}. \qquad (4.5)$$

Например, в случае, когда $N = 15$ и $minHD(T_i, T_j) = 5 = 2 \times 2 + 1$ декартово расстояние удовлетворяет неравенству $CD(T_i, T_j) \ge \sqrt{5}$. Таким образом, существует не более чем:

$$Q \le \frac{2^{15}}{\sum_{l=0}^{2} \binom{15}{l}} = \frac{2^{15}}{\binom{15}{0} + \binom{15}{1} + \binom{15}{2}} = \frac{2^{15}}{1+15+105} = 270 = 2^8$$

возможных кандидатов в тесты с декартовым расстоянием $CD(T_i, T_j)$ больше или равным $\sqrt{5}$. Учитывая инверсные значения наборов, максимальная длина *TMDTS* равняется $q = 2^9$. Однако увеличение декартового расстояния,

например, до значения $\sqrt{7}$ существенно уменьшает длину *TMDTS*. В этом случае количество наборов *TMDTS* равняется $2^6 = 64$.

4.2. Оптимальные управляемые вероятностные тесты

Обобщая многообразные модификации управляемых вероятностных тестов, следует отметить единообразие в процедуре генерирования очередного тестового набора T_i [228]. Во всех случаях критерием является максимальное или минимальное значение некоторой характеристики (характеристик), как правило, имеющей небольшую вычислительную сложность и определяемой на основании предыдущих тестовых наборов T_0, T_1, T_2, …, T_{i-1} [228]. К числу таких характеристик относится расстояние Хэмминга и декартово расстояние. Во всех ранее рассмотренных методах находится лучшее, или локально лучшее решение на основании простейших метрик по *жадному алгоритму* (*Greedy algorithm*), который заключается в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. Если глобальная оптимальность алгоритма имеет место практически всегда, жадный алгоритм является предпочтительным по отношению к другим методам оптимизации, таким как, например, динамическое программирование. В случае управляемых случайных тестов жадный алгоритм является единственно возможным для практического использования в силу его существенно меньшей вычислительной сложности по сравнению с другими оптимизационными алгоритмами.

Используя метрики, рассмотренные в предыдущем разделе, а именно расстояние Хэмминга $HD(T_i, T_j)$ для тестовых наборов T_i и T_j (4.1); декартово расстояние $CD(T_i, T_j)$ (4.2); суммарное расстояние Хэмминга $THD(T_i)$ для следующего набора T_i (4.3); суммарное декартово расстояние $TCD(T_i)$ (4.3); суммарное расстояние Хэмминга $THD(T)$ для теста T (4.4) и суммарное декартово расстояние $TCD(T)$ (4.4) [228], синтезируем оптимальный управляемый вероятностный тест.

На первом шаге, следуя концепции черного ящика, генерируем произвольный тестовый набор T_0, состоящий из N бит. Отметим, что в качестве T_0 может быть любой из 2^N тестовых наборов. Не нарушая общности дальнейших рассуждений, предположим, что $T_0 = 0\ 0\ 0\ \ldots\ 0$.

В качестве второго тестового набора T_1, в соответствии со всеми ранее приведенными метриками (4.1) − (4.4), оптимальным будет инверсный вектор по отношению к T_0, то есть $T_1 = \overline{T_0}$. Тогда *оптимальный управляемый случайный тест* (*optimal controlled random test − OCRT*), состоящий из двух $q = 2$ наборов, принимает вид $OCRT = \{T_0, T_1\}$, где $T_1 = \overline{T_0}$. Для ранее приведенного примера $T_1 = \overline{T}_0 = \overline{0\,0\,0\,\ldots\,0} = 111\ldots1$. Оптимальность выбора второго тестового набора T_1, как инверсию первого T_0 подтверждается максимальны-

ми значениями всех ранее рассмотренных метрик. Действительно, $HD(T_0, T_1) = THD(T_1) = THD(T) = N$, и $CD(T_0, T_1) = TCD(T_1) = TCD(T) = \sqrt{N}$.

Для получения третьего набора T_2 *OCRT* нельзя использовать характеристики $HD(T_i, T_j)$ и $CD(T_i, T_j)$. Это объясняется тем фактом, что максимизация их значений $HD(T_2, T_1)$ и $CD(T_2, T_1)$ приводит к противоречивому результату, а именно, что $T_2 = T_0$. Поэтому при дальнейших исследованиях данные характеристики не будут использованы. Аналогичное заключение может быть сделано по отношению метрик $THD(T_i)$ и $THD(T)$. Это следует из того факта, что для $OCRT = \{T_0, T_1, T_2\}$, где $T_1 = \overline{T_0}$, любой третий набор T_2 позволяет максимизировать данные характеристики. Действительно, для произвольного T_2 $THD(T_2) = N$ и $THD(T) = 2N$. Следует отметить, что даже для случаев, когда $T_2 = T_0$ и $T_2 = T_1$ указанные метрики принимают максимально возможные значения $THD(T_2) = N$ и $THD(T) = 2N$, что свидетельствует об оптимальности выбора T_2 в обоих случаях.

С учетом приведенного анализа кандидатом в качестве третьего набора T_2 *OCRT* может быть любой набор T_2, который удовлетворяет следующим соотношениям $T_2 \neq T_0$ и $T_2 \neq T_1$. Предположим, что в качестве T_2 выбран набор, для которого $HD(T_i, T_j)$ принимает значение $HD(T_0, T_2)$, тогда $HD(T_1, T_2) = N - HD(T_0,T_2)$. Характеристики, основанные на декартовом расстоянии, принимают значения:

$$TCD(T_i) = CD(T_0,T_2)+CD(T_1,T_2) = \sqrt{HD(T_0,T_2)} + \sqrt{N - HD(T_0,T_2)}$$

и

$$TCD(T) = CD(T_0,T_1) + CD(T_0,T_2) + CD(T_1,T_2) =$$
$$= \sqrt{N} + \sqrt{HD(T_0,T_2)} + \sqrt{N - HD(T_0,T_2)}.$$

Тогда $TCD(T_i)$ и $TCD(T)$ принимают максимальное значение для оптимальной величины $HD(T_0, T_2) = N/2$, полученной как решение следующего уравнения:

$$\frac{\partial(\sqrt{H(T_0,T_2)} + \sqrt{N - H(T_0,T_2)})}{\partial(H(T_0,T_2))} = 0.$$

Приняв допущение, что величина N является четной величиной, для примера, рассмотренного выше оптимальным тестовым набором T_2 для двух ранее полученных векторов T_0 = 0 0 0 … 0 0 0 0 … 0 и T_1 = 1 1 1 … 1 1 1 1 … 1 является набор, состоящий из $N/2$ бит, принимающих значение 0 и из $N/2$ бит, равных 1, то есть T_2 = 0 0 0 … 0 1 1 1 … 1. Таким образом, для $q = 3$ $OCRT = \{T_0, T_1, T_2\}$ = {0 0 0 … 0 0 0 0 … 0, 1 1 1 … 1 1 1 1 … 1, 0 0 0 … 0 1 1 1 … 1}.

Для следующего кандидата в тестовые наборы необходимо также достичь максимальных значений $TCD(T_i)$ и $TCD(T)$. С целью максимизации

$HD(T_0, T_3)$ и $HD(T_1, T_3)$ необходимо выбирать T_3 из множества наборов, для которых сумма $HD(T_0, T_i) + HD(T_1, T_i)$ принимает максимальное значение, то есть для кандидата в четвертый тестовый набор должно выполняться равенство $w(T_i) = N/2$. Для получения максимальных значений метрик $TCD(T_3)$ и $TCD(T)$ очевидным и единственным решением является $T_3 = \overline{T_2}$. Для ранее рассмотренного примера получим T_3 = 1 1 1 … 1 0 0 0 … 0 и $OCRT = \{T_0, T_1, T_2, T_3\}$ = {0 0 0 … 0 0 0 0 … 0, 1 1 1 …1 1 1 1 … 1, 0 0 0 … 0 1 1 1 … 1, 1 1 1 … 1 0 0 0 … 0}.

Аналогичный результат будет получен с использованием *FAR* алгоритма [130]. Согласно данному алгоритму, основываясь на трех предыдущих наборах T_0 = 0 0 0 … 0 0 0 0 … 0, T_1 = 1 1 1 …1 1 1 1 … 1 и T_2 = 0 0 0 … 0 1 1 1 … 1, может быть получен центроид $C = c_{N-1}, c_{N-2}, \ldots, c_2, c_1, c_0$ =1/3, 1/3, 1/3, …, 1/3, 2/3, 2/3, 2/3, …, 2/3, а на его основе бинарный центроид C_b = 0 0 0 … 0 1 1 1 … 1 [130]. И наконец, инвертируя значения элементов бинарного центроида, окончательно получим T_3 = 1 1 1 … 1 0 0 0 … 0. Для набора T_3 декартовы метрики принимают максимальные, значения равные соответственно $TCD(T_3) = \sqrt{N} + 2\sqrt{N/2}$ и $TCD(T) = 2\sqrt{N} + 4\sqrt{N/2}$.

Для процедуры формирования *OCRT* с учетом предыдущих этапов может быть использовано следующее условие. На всех последующих этапах формирования очередного тестового набора необходимо максимизировать значения метрик $TCD(T_i)$ и $TCD(T)$. Тогда для набора T_i с четным значением $i \in \{0, 2, 4, \ldots, 2k-2\}$ должны выполняться равенства:

$$\begin{aligned} &maxTCD(T_i) = i \times \sqrt{N/2}; \\ &maxTCD(T) = (i/2) \times \sqrt{N} + (i^2/2) \times \sqrt{N/2}, \quad i \in \{0, 2, 4, \ldots, 2k-2\}. \end{aligned} \tag{4.6}$$

Отметим, что приведенные равенства представляют максимально возможные значения метрик $TCD(T_i)$ и $TCD(T)$, которые обеспечиваются путем выбора набора T_i с четным индексом i такого, что $HD(T_i, T_j) = N/2$ между набором T_i и всеми предыдущими наборами T_j, $j < i$.

В тоже время набор с нечетными индексами $i \in \{1, 3, 5, \ldots, 2k-1\}$ является инверсным значением по отношению к предыдущему набору, то есть $T_i = \overline{T_{i-1}}$, что позволяет максимизировать значения метрик $TCD(T_i)$ и $TCD(T)$:

$$\begin{aligned} &maxTCD(T_i) = \sqrt{N} + (i-1) \times \sqrt{N/2}; \\ &maxTCD(T) = ((i+1)/2) \times \sqrt{N} + ((i^2-1)/2) \times \sqrt{N/2}, \\ &i \in \{0, 2, 4, \ldots, 2k-2\}. \end{aligned} \tag{4.7}$$

В качестве примера рассмотрим случай, когда $N = 2^m$, тогда количество q наборов $OCRT = \{T_0, T_1, T_2, \ldots, T_{q-1}\}$ равняется $2(m + 1)$. Для общего случая количество наборов *OCRT* определяется как: $q = 2(\lceil \log_2 N \rceil + 1)$, а конструктивный алгоритм для формирования тестовых наборов представлен в [186].

При $N = 2^m$ и $m = 3$ *OCRT*, предполагая, что $T_0 = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$, будет состоять из $2(m + 1) = 2(3 + 1) = 8$ тестовых наборов, приведенных в таблице 4.2. В этой же таблице представлены соответствующие значения (4.6) и (4.7) для $maxTCD(T_i)$ и $maxTCD(T)$.

Следует отметить, что в результате синтеза оптимального управляемого теста *OCRT*, используя жадный алгоритм, был получен достаточно тривиальный результат. Такой же результат легко может быть получен на основании классического алгоритма бинарного поиска (*divide and conquer algorithm*) [114]. Тест *OCRT* также может быть получен как результат применения одного из известных методов генерирования управляемых вероятностных тестов, а именно при использовании процедуры генерирования анти-вероятностных тестов [126, 202, 203]; процедуры формирования быстрых антивероятностных тестов [130]; ограниченных случайных тестов [31, 32], и многочисленных их модификаций. Все перечисленные методы используют критерии выбора очередного тестового набора на основе расстояния Хемминга и декартова расстояния.

Таблица 4.2 – *Оптимальный управляемый вероятностный тест для* $q = 8$

T_i	$t_{i,7}$	$t_{i,6}$	$t_{i,5}$	$t_{i,4}$	$t_{i,3}$	$t_{i,2}$	$t_{i,1}$	$t_{i,0}$	$maxTCD(T_i)$	$maxTCD(T)$
T_0	0	0	0	0	0	0	0	0	-	-
T_1	1	1	1	1	1	1	1	1	$\sqrt{8}$	$\sqrt{8}$
T_2	0	0	0	0	1	1	1	1	$2\sqrt{4}$	$\sqrt{8}+2\sqrt{4}$
T_3	1	1	1	1	0	0	0	0	$\sqrt{8}+2\sqrt{4}$	$2\sqrt{8}+4\sqrt{4}$
T_4	0	0	1	1	0	0	1	1	$4\sqrt{4}$	$2\sqrt{8}+8\sqrt{4}$
T_5	1	1	0	0	1	1	0	0	$\sqrt{8}+4\sqrt{4}$	$3\sqrt{8}+12\sqrt{4}$
T_6	0	1	0	1	0	1	0	1	$6\sqrt{4}$	$3\sqrt{8}+18\sqrt{4}$
T_7	1	0	1	0	1	0	1	0	$\sqrt{8}+6\sqrt{4}$	$4\sqrt{8}+24\sqrt{4}$

В качестве универсальной процедуры формирования оптимальных управляемых случайных тестов *OCRT* применима следующая процедура [228].

Процедура 4.2. Формирование *OCRT* для $N = 2^m$.

1. Первоначально генерируется матрица *M*, содержащая *N* столбцов и $q = 2(m + 1)$ строк, используя, например, алгоритм бинарного поиска.

Первая строка M_0 матрицы *M* принимает нулевое значение, вторая строка M_1 состоит из *N* единиц. Последующие строки матрицы *M* с четными индексами $i > 0$ представляют собой множество, состоящее из четного количества блоков одинаковой размерности, половина которых представляет собой нулевые блоки, а вторая половина – единичные. На каждой итерации очередная строка с четным индексом формируется из предыдущей строки для четного *i*, при этом все блоки предыдущей строки делятся на два блока, первый из которых является нулевым блоком, а второй – единичным. Каждая

строка с нечетным индексом формируется как результат инвертирования элементов предыдущей строки с четным индексом.

2. Основываясь на одном из алгоритмов перестановок, столбцы матрицы M перемешиваются для получения матрицы M^*, так называемых векторов масок.

3. Случайным образом формируется двоичный вектор, состоящий из N бит, который принимается как первый набор T_0 *OCRT* теста.

4. Каждый последующий набор T_1, T_2, T_3, …, T_{2m+1} определяется как поразрядная сумма вектора T_0 и соответствующей строки матрицы M^*, то есть $T_i = T_0 \oplus M_i^*$.

5. Этап 4 повторяется до тех пор, пока все $2(m + 1)$ наборы не будут сгенерированы.

Следует отметить, что перестановка столбцов матрицы не изменяет значения расстояния Хемминга между двумя строками (наборами), что следует из соотношения (4.1).

Пример 4.4. В качестве примера рассмотрим *OCRT* для $N = 2^m = 2^3$.

1. Матрица M с $N = 8$ столбцами и $q = 2(3 + 1) = 8$ строками строиться таким образом, что $M_0 = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$, а $M_1 = 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1$. Следующая строка с четным индексом M_2 формируется на основании M_0 путем деления M_0 на два равных блока, первый из которых содержит нулевые значения, а второй – единичные, то есть $M_2 = 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1$. $M_3 = 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0$ получается путем инвертирования значений строки M_2. Аналогично получаются две следующие строки $M_4 = 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1$ и $M_5 = 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0$ матрицы M. Последние две строки принимают вид $M_6 = 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1$ $M_7 = 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0$ (см. таблицу 4.2).

2. Матрица M^* векторов масок как результат перестановок столбцов матрицы M (таблица 4.2) представлена в таблице 4.3.

3. Случайным образом формируется $T_0 = 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0$.

4. Каждый новый i-ый набор T_1, T_2, T_3, …, T_7 получается как сумма $T_i = T_0 \oplus M_i^*$, где M_i^* выбирается из таблицы 4.3.

Таблица 4.3 – *Матрица M^* векторов масок для $m = 3$*

M_i^*	$t_{i,2}$	$t_{i,6}$	$t_{i,1}$	$t_{i,4}$	$t_{i,3}$	$t_{i,7}$	$t_{i,5}$	$t_{i,0}$
M_0^*	0	0	0	0	0	0	0	0
M_1^*	1	1	1	1	1	1	1	1
M_2^*	1	0	1	0	1	0	0	1
M_3^*	0	1	0	1	0	1	1	0
M_4^*	0	0	1	1	0	0	1	1
M_5^*	1	1	0	0	1	1	0	0
M_6^*	1	1	0	1	0	0	0	1
M_7^*	0	0	1	0	1	1	1	0

5. Все $q = 2(m + 1) = 8$ наборы для *OCRT* представлены в таблицы 4.4 с соответствующими значениями характеристик $maxTCD(T_i)$ и $maxTCD(T)$.

Таблица 4.4 – *Управляемый вероятностный тест OCRT для* $m = 3$

T_i	$t_{i,7}$	$t_{i,6}$	$t_{i,5}$	$t_{i,4}$	$t_{i,3}$	$t_{i,2}$	$t_{i,1}$	$t_{i,0}$	$maxTCD(T_i)$	$maxTCD(T)$
T_0	0	1	1	1	1	0	1	0	-	-
T_1	1	0	0	0	0	1	0	1	$\sqrt{8}$	$\sqrt{8}$
T_2	1	1	0	1	0	0	1	1	$2\sqrt{4}$	$\sqrt{8}+2\sqrt{4}$
T_3	0	0	1	0	1	1	0	0	$\sqrt{8}+2\sqrt{4}$	$2\sqrt{8}+4\sqrt{4}$
T_4	0	1	0	0	1	0	0	1	$4\sqrt{4}$	$2\sqrt{8}+8\sqrt{4}$
T_5	1	0	1	1	0	1	1	0	$\sqrt{8}+4\sqrt{4}$	$3\sqrt{8}+12\sqrt{4}$
T_6	1	1	0	1	0	0	0	1	$6\sqrt{4}$	$3\sqrt{8}+18\sqrt{4}$
T_7	0	0	1	0	1	1	1	0	$\sqrt{8}+6\sqrt{4}$	$4\sqrt{8}+24\sqrt{4}$

В данном разделе представлен метод генерирования оптимальных управляемых вероятностных тестов, как обобщение известных алгоритмов, использующих в своей основе жадный алгоритм, а также расстояние Хемминга и декартово расстояние в качестве критерия выбора очередного тестового набора. В отличие от известных решений предложенный метод гарантированно обеспечивает максимально возможные значения расстояний $TCD(T_i)$ и $TCD(T)$. Следует отметить, что аналогичный результат может быть получен с использованием известных решений, однако его достижение не является гарантированным и требует большого объема вычислений, связанных с определением характеристик расстояния для каждого потенциального кандидата в тестовые наборы. В свою очередь оптимальные управляемые случайные тесты (*OCRT*) имеют очевидное преимущество по отношению ко всем известным методам формирования управляемых случайных тестов. Они характеризуются минимальной вычислительной сложностью в силу отсутствия процедур перечисления всевозможных кандидатов в тестовые наборы и вычисления для них соответствующих характеристик. Для произвольного N наиболее трудоемкой процедурой при формировании *OCRT* является процедура построения матрицы M, состоящей из $q = 2(m + 1)$ строк и выполнения перемешивания ее столбцов для получения M^*.

4.3. Универсальная метрика для управляемых вероятностных тестов

В качестве меры эффективности управляемых вероятностных тестов рассмотрим их степень приближения к исчерпывающим и псевдо-исчерпывающим тестам [334]. Известно, что псевдо-исчерпывающий тест $T(N, k)$ исчерпывающе покрывает всевозможные k из N подпространств N-мерного пространства [52, 186, 187].

Так, тест $T(4, 2) = \{0\ 0\ 0\ 0, 0\ 1\ 1\ 1, 1\ 0\ 1\ 1, 1\ 1\ 0\ 1, 1\ 1\ 1\ 0\}$ формирует всевозможные $2^k = 2^2$ двоичные комбинации для всех $k = 2$ из $N = 4$ разрядов тестовых наборов $T_i = t_{i,3}, t_{i,2}, t_{i,1}, t_{i,0}$, что следует из таблицы 4.5.

Таблица 4.5 – *Оценка покрывающей способности теста* $T(4, 2)$

T_i	$t_{i,3}t_{i,2}t_{i,1}t_{i,0}$	$t_{i,1},t_{i,0}$	$t_{i,2},t_{i,0}$	$t_{i,3}t_{i,0}$	$t_{i,2}t_{i,1}$	$t_{i,3}t_{i,1}$	$t_{i,3}t_{i,2}$	$t_{i,2}t_{i,1}t_{i,0}$	$t_{i,3}t_{i,1}t_{i,0}$	$t_{i,3}t_{i,2}t_{i,0}$	$t_{i,3}t_{i,2}t_{i,1}$
T_0	0000	xx00	x0x0	0xx0	x00x	0x0x	00xx	x000	0x00	00x0	000x
T_1	0111	xx11	x1x1	0xx1	x11x	0x1x	01xx	x111	0x11	01x1	011x
T_2	1011	xxxx	x0x1	1xx1	x01x	1x1x	10xx	x011	1x11	10x1	101x
T_3	1101	xx01	xxxx	xxxx	x10x	1x0x	11xx	x101	1x01	11x1	110x
T_4	1110	xx10	x1x0	1xx0	xxxx	xxxx	xxxx	x110	1x10	11x0	111x

В то же время тест $T(4, 2)$ генерирует 5 из $2^k = 2^3 = 8$ двоичных комбинаций для $k = 3$ и такое же количество комбинаций для $k = 4$ из 16 возможных (см. таблицу 4.5).

По определению псевдо-исчерпывающий тест $T(N, k)$ обеспечивает 100% покрытие для конкретного значения $k < N$, и, соответственно, гарантирует 100% покрытие для всевозможных $r \le k$ из N подпространств N-мерного пространства [88, 333].

Для $r > k$ эффективность $E(T(N, k), r)$ псевдо-исчерпывающего теста, в процентном исчислении, может быть оценена как количество комбинаций $C(T(N, k), r)$ из r бит, формируемых тестом $T(N, k)$ на всевозможных r из N разрядах деленное на общее количество возможных комбинаций:

$$E(T(N,k),r) = \frac{C(T(N,k),r)}{2^r \begin{pmatrix} N \\ r \end{pmatrix}} 100\%. \qquad (4.8)$$

Для примера, приведенного в таблице 4.5, согласно (4.8) получим, что $E(T(4, 2), 3) = ((5 + 5 + 5 + 5)/(8 \times 4))100\% = 62{,}5\%$, и $E(T(4, 2), 4) = 5/(16 \times 1)100\% = 31{,}25\%$.

Мера эффективности (4.8) псевдо-исчерпывающего теста $T(N, k)$ может быть использована для оценки эффективности управляемых вероятностных тестов. Подобная характеристика формулируется для следующего тестового набора T_i и некоторого теста, в том числе и *CRT* [334].

Определение 4.4. Метрикой $E(T_i, r)$ для очередного тестового набора T_i теста $T = \{T_0, T_1, T_2, \ldots, T_i\}$ является дополнительное количество двоичных комбинаций на всевозможных r из N разрядах генерируемых тестовым набором T_i по отношению к предыдущим наборам $T_0, T_1, T_2, \ldots, T_{i-1}$.

Определение 4.5. Метрикой $E(T, r)$ теста $T = \{T_0, T_1, T_2, \ldots, T_i\}$ является количество двоичных комбинаций на всевозможных r из N разрядах, генерируемых тестом T.

Для обеих метрик $E(T_i, r)$ и $E(T, r)$, подобно, как и (4.8), могут быть использованы их взвешенные процентные оценки $WE(T_i, r)$ и $WE(T, r)$. Так, для

вероятностного теста RT, состоящего из q случайного набора RT_0, RT_1, …, RT_{q-1}, согласно определению 4.5 получим [346]:

$$WE(RT,r)=\left(1-\left(1-\frac{1}{2^r}\right)^q\right)100\%. \qquad (4.9)$$

Метрика $WE(RT_i, r)$ для очередного i-го тестового набора RT определяется, как:

$$WE(RT_i,r)=\left(1-\left(1-\frac{1}{2^r}\right)^{i+1}\right)100\%-\left(1-\left(1-\frac{1}{2^r}\right)^{i}\right)100\%= \qquad (4.10)$$
$$=\frac{1}{2^r}\left(1-\frac{1}{2^r}\right)^i 100\%.$$

Значения $WE(RT, r)$ для небольших величин r приведены в таблице 4.6.

Таблица 4.6 – *Численные значения* $WE(RT, r)$ *для вероятностного теста RT*

r	T_0	T_0,T_1	$T_0,\ldots,T_3$	$T_0,\ldots,T_5$	$T_0,\ldots,T_7$	$T_0,\ldots,T_9$	$T_0,\ldots,T_{11}$	$T_0,\ldots,T_{13}$
2	25,0	43,7	68,3	82,2	89,9	94,3	96,8	98,2
3	12,5	23,4	41,3	55,1	65,6	73,7	79,8	84,5
4	6,25	12,1	22,7	32,1	40,3	47,6	53,9	59,4
5	3,12	6,15	11,9	17,3	22,4	27,2	31,7	35,8

Очевидно, что максимальная покрывающая способность вероятностного теста *RT*, равная 100%, достигается для $r << N$ и $q \to \infty$.

Управляемые вероятностные тесты *CRT*, по сравнению с *RT*, позволяют повысить эффективность формирования всевозможных двоичных комбинаций на любых r из N разрядах генерируемых тестовых наборов. Так, для случая оптимальных управляемых вероятностных тестов (*OCRT*), принимая допущение, что $N^2 >> N$ и $r << N$, можно показать, что $WE(OCRT, r)$ будет вычисляться в соответствии с выражением:

$$WE(OCRT,r)=\left(1-\left(\frac{2^{r-1}-1}{2^{r-1}}\right)^{q/2}\right)100\%; \quad q=2,4,6,... . \qquad (4.11)$$

Приведенное соотношение вытекает из оценок, полученных в [226] для случая, когда *OCRT* состоит из четного числа $q = 2(\lceil \log_2 N \rceil + 1)$ наборов и включает последовательные инверсные пары наборов $T_i = \overline{T}_{i-1}$, $i \in \{1, 3, 5, \ldots, q-1\}$.

Взвешенное количество дополнительных двоичных комбинаций $WE(OCRT(T_{i-1}, T_i), r)$ для любых r из N бит генерируемых парой T_{i-1}, T_i тестовых наборов $OCRT$ вычисляется, как:

$$WE(OCRT(T_{i-1},T_i),r) = \left(1-\left(\frac{2^{r-1}-1}{2^{r-1}}\right)^{(i+1)/2}\right)100\% - \\ -\left(1-\left(\frac{2^{r-1}-1}{2^{r-1}}\right)^{(i-1)/2}\right)100\% = \frac{1}{2^{r-1}}\left(\frac{2^{r-1}-1}{2^{r-1}}\right)^{(i-1)/2} 100\%. \quad (4.12)$$

Очевидно, что в силу инверсного соотношения между T_i и T_{i-1}, $i \in \{1, 3, 5, \ldots, q-1\}$:

$$WE(OCRT_i, r) = WE(OCRT_{i-1}, r) = \frac{1}{2^r}\left(\frac{2^{r-1}-1}{2^{r-1}}\right)^{(i-1)/2} 100\%. \quad (4.13)$$

В табл. 4.7 приведены численные значения для $WE(OCRT, r)$.

Таблица 4.7 – *Численные значения* $WE(OCRT, r)$ *для* $OCRT$ *для* $q = 2, 4, \ldots, 14$

r	T_0, T_1	$T_0, \ldots, T_3$	$T_0, \ldots, T_5$	$T_0, \ldots, T_7$	$T_0, \ldots, T_9$	$T_0, \ldots, T_{11}$	$T_0, \ldots, T_{13}$
2	50,0	75,00	87,5	93,75	96,87	98,43	99,2
3	25,0	43,75	57,8	68,35	76,26	82,20	86,65
4	12,5	23,43	33,0	41,4	48,71	55,1	60,73
5	6,25	12,10	17,6	22,8	27,6	32,1	36,35

Сравнительная эффективность $OCRT$ по отношению к RT для $i \in \{1, 3, 5, \ldots, q-1\}$ оценивается соотношением (4.14), численные значения которого приведены на рис. 4.1.

$$WE = WE(OCRT_i, r) - WE(RT_i, r) = \frac{1}{2^r}\left(\left(1-\frac{1}{2^{r-1}}\right)^{(i-1)/2} - \left(1-\frac{1}{2^r}\right)^i\right)100\%. \quad (4.14)$$

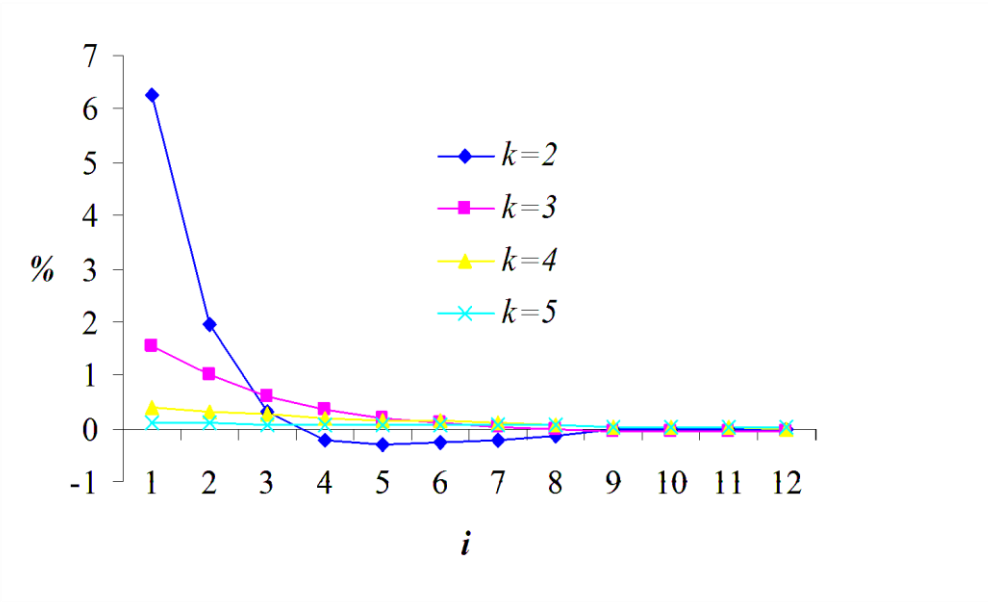


Рисунок 4.1 – *Оценка WE сравнительной эффективности OCRT по отношению к RT*

Анализ приведенных результатов показывает высокую эффективность теста *OCRT* по отношению к тесту *RT* только для небольших значений k и i, что следует из данных, приведенных в таблицах 4.6 и 4.7, а также на рис. 4.1. Этот факт предопределяет интерес к разработке оптимальных управляемых вероятностных тестов малой длины.

4.4. Оптимальные управляемые вероятностные тесты малой длины

Как было показано в предыдущем разделе, характеристики $E(T_i, r)$ и $E(T, r)$, так же, как и их взвешенные процентные аналоги $WE(T_i, r)$ и $WE(T, r)$, позволяют однозначно оценить эффективность каждого кандидата в управляемые вероятностные тесты. Однако в сравнении с метриками, использованными для известных управляемых вероятностных тестов, универсальные характеристики $E(T_i, r)$ и $E(T, r)$ требуют больших объемов вычислений для каждого кандидата в тесты. Только в случае фиксированной структуры управляемых вероятностных тестов, например *RT* и *OCRT*, оказывается возможным получение аналитических оценок указанных метрик. Как отмечалось в разделе 4.2, для *OCRT* выполняется неравенство $HD(T_i, T_j) \geq N/2$, из которого можно заключить, что в случае *OCRT* $minHD(T_i, T_j) = N/2$. В то же время для *RT* данное условие не выполняется; это позволяет предположить, что для достижения большей эффективности управляемых вероятностных тестов необходимо максимизировать $minHD(T_i, T_j)$, допуская, что с увеличением $minHD(T_i, T_j)$ длина q *CRT* будет уменьшаться. Достижение максимального значения $minHD(T_i, T_j)$ также максимизирует характеристики $THD(T_i)$ и $TCD(T_i)$ (4.3), и их суммы $THD(T)$ и $TCD(T)$ [112].

Для генерирования *оптимальных управляемых вероятностных тестов малой длины* (*short optimal controlled random tests − SOCRT*) с малым количеством наборов q первоначально рассмотрим коды с максимальным минимальным расстоянием Хемминга. Теорема *Плоткина* позволяет определить максимально возможное количество q кодовых слов в двоичном коде длины N для минимального кодового расстояния d, а *граница Плоткина* (*Plotkin bound*) дает верхний предел этого количества [159].

Теорема 4.1. (*Граница Плоткина*) Если $d \geq N/2$, то для q выполняется неравенство:

$$q \leq \begin{cases} \dfrac{2d}{2d-N}, & \text{для} \quad 2d-N>0; \\ 4d, & \text{для} \quad 2d-N=0. \end{cases} \tag{4.15}$$

Для случая, когда $2d - N > 0$, справедливо следующее соотношение:

$$d \leq qN/(2(q-1)). \tag{4.16}$$

Основываясь на теореме 4.1, определим новую характеристику для построения $SOCRT(q, d) = \{T_0, T_1, T_2, \ldots, T_{q-1}\}$ с ограниченным количеством q наборов и расстоянием равным d.

Определение 4.6. Максимальное минимальное расстояние Хэмминга $max\_minHD(T_i, T_j)$ определяется как максимальное минимальное расстояние между любыми двумя наборами T_i и T_j теста $SOCRT=\{T_0, T_1, T_2, \ldots, T_{q-1}\}$.

Соотношение (4.16) позволяет получить максимальное минимальное расстояние $d = max\_minHD(T_i, T_j)$ для различных значений $q \in \{2, 3, \ldots\}$. Для $q = 2$, согласно (4.16), $max\_minHD(T_i, T_j) \leq N$. В предыдущем разделе при построении *OCRT* для $q = 2$ в качестве второго тестового набора T_1 использовалась инверсия первого T_0 набора, т. е. $T_1 = \overline{T}_0$. В этом случае достигается максимальное значение $HD(T_0, T_1) = N$, тогда $SOCRT(2, N) = OCRT = \{T_0, \overline{T}_0\}$.

Для $q = 3$, согласно (4.16), $max\_minHD(T_i, T_j) \leq 3N/4$. Ближайшее оптимальное решение может быть получено только для $max\_minHD(T_i, T_j) = 2N/3$, здесь $2N/3 < 3N/4$. Докажем, что для случая $q = 3$, $max\_minHD(T_i, T_j)$ не может быть больше чем $2N/3$ [334].

Теорема 4.2. *Для* $SOCRT = \{T_0, T_1, T_2\}$ $max\_minHD(T_i, T_j) = 2N/3$.

Доказательство теоремы 4.2. Докажем от обратного. Предположим, что для целого $2N/3$ $max\_minHD(T_i, T_j) = 2N/3 + \Delta$, где $\Delta > 0$, т. е. $max\_minHD(T_i,T_j) > 2N/3$. Тогда для $SOCRT = \{T_0, T_1, T_2\}$ расстояния $HD(T_0, T_1)$, $HD(T_0, T_2)$ и $HD(T_1, T_2)$ между тестовыми наборами должны быть больше или равняться $2N/3 + \Delta$. Пусть расстояние $HD(T_0, T_1) = S_{01}(T_0, T_1) + S_{10}(T_0, T_1) = 2N/3 + \Delta$, следовательно, $S_{00}(T_0, T_1) + S_{11}(T_0, T_1) = N - 2N/3 - \Delta = N/3 - \Delta$. Для обеспечения максимального отличия третьего набора T_2 от первого T_0 и второго T_1 наборов, он должен отличаться от них в позициях с одинаковыми

символами 0 или 1, т.е. в $S_{00}(T_0, T_1) + S_{11}(T_0, T_1) = N/3 - \Delta$ позициях. Тогда $HD(T_0, T_2) = N/3 - \Delta + Q$, и с учетом того, что $S_{01}(T_0, T_1) + S_{10}(T_0, T_1) = 2N/3 + \Delta$, получим, что $HD(T_1, T_2) = N/3 - \Delta + (2N/3 + \Delta - Q) = N - Q$. Для обеспечения неравенства $HD(T_0, T_2) \geq 2N/3 + \Delta$ значение Q должно удовлетворять соотношению $Q \geq N/3 + 2\Delta$, тогда $HD(T_1, T_2) = N - Q \leq N - (N/3 + 2\Delta) = 2N/3 - 2\Delta$, что меньше, чем $2N/3 + \Delta$, следовательно, предположение, что $max\_minHD(T_i,T_j) > 2N/3$ неверно. Что и требовалось доказать.

Тест $SOCRT(3, 2N/3)$, состоящий из трех наборов, имеет вид:

$$
\begin{aligned}
T_0 &= t_{0,N-1}, t_{0,N-2}, ..., t_{0,2N/3}, t_{0,2N/3-1}, ..., t_{0,N/3}, t_{0,N/3-1}, ..., t_{0,2}, t_{0,1}, t_{0,0};\\
T_1 &= \overline{t_{1,N-1}}, \overline{t_{1,N-2}}, ..., \overline{t_{1,2N/3}}, \overline{t_{1,2N/3-1}}, ..., \overline{t_{1,N/3}}, t_{0,N/3-1}, ..., t_{1,2}, t_{1,1}, t_{1,0}; \qquad (4.17)\\
T_2 &= t_{2,N-1}, t_{2,N-2}, ..., t_{2,2N/3}, \overline{t_{2,2N/3-1}}, ..., \overline{t_{2,N/3}}, \overline{t_{2,N/3-1}}, ..., \overline{t_{2,2}}, \overline{t_{2,1}}, \overline{t_{2,0}}.
\end{aligned}
$$

Для $N = 6$ соответствующий тест $SOCRT(3, 4) = \{T_0, T_1, T_2\} = \{0\ 0\ 0\ 0\ 0\ 0, 1\ 1\ 1\ 1\ 0\ 0, 0\ 0\ 1\ 1\ 1\ 1\}$.

Для случая $q = 4$ в соответствии с (4.16) $max\_minHD(T_i, T_j) \leq 2N/3$. Оптимальный тест $SOCRT(4, 2N/3)$ имеет $max\_minHD(T_i,T_j) = 2N/3$, что следует из теоремы 4.3.

Теорема 4.3. Для $SOCRT = \{T_0, T_1, T_2, T_3\}$ $max\_minHD(T_i,T_j) = 2N/3$.

Доказательство теоремы 4.3. Учитывая, что согласно теореме 4.2 для $q = 3$ $max\_minHD(T_i, T_j) = 2N/3$, можно заключить, что для теста *SOCRT,* состоящего из $q = 4$ наборов T_0,T_1,T_2 и T_3 также нельзя получить $max\_minHD(T_i,T_j) > 2N/3$. Это значит, что наилучшим решением для *SOCRT* состоящего из наборов T_0, T_1, T_2 и T_3, будет три набора $SOCRT(3, 2N/3)$ и набор T_3, равноудаленный от первых трех T_0, T_1 и T_2 на расстояние $HD(T_0, T_3) = HD(T_1, T_3) = HD(T_2, T_3) = 2N/3$. В случае $SOCRT(3, 2N/3)$, T_0 и T_1 имеют $S_{01}(T_0, T_1) + S_{10}(T_0, T_1) = 2N/3$ различных бит. Третий набор T_2 формируется с использованием половины этих бит из набора T_0 и другой половины бит из T_1, а также с помощью инверсии всех $S_{00}(T_0, T_1) + S_{11}(T_0, T_1) = N/3$ одинаковых бит для T_0 и T_1. Для формирования четвертого набора T_3 для него из $S_{01}(T_0, T_1) + S_{10}(T_0, T_1) = 2N/3$ различных бит наборов T_0 и T_1 выбираются другие половины в сравнении с T_2 и инверсия $S_{00}(T_0, T_1) + S_{11}(T_0, T_1) = N/3$ битов наборов T_0 и T_1. Исходя из описанной процедуры формирования T_2 и T_3 можно заключить, что в $S_{01}(T_0, T_1) + S_{10}(T_0, T_1) = 2N/3$ позициях с различными битами для T_0 и T_1 наборы T_2 и T_3 имеют инверсные значения бит. Тогда $HD(T_2, T_3) = 2N/3$ и соответственно $HD(T_0, T_1) = HD(T_0, T_2) = HD(T_0, T_3) = HD(T_1, T_2) = HD(T_1, T_3) = HD(T_2, T_3) = 2N/3$. Что и требовалось доказать.

Тест $SOCRT(4, 2N/3)$ имеет вид (4.18).

Для $N = 6$ получим, что $SOCRT(4, 4) = \{T_0, T_1, T_2, T_3\} = \{0\ 0\ 0\ 0\ 0\ 0, 1\ 1\ 1\ 1\ 0\ 0, 0\ 0\ 1\ 1\ 1\ 1, 1\ 1\ 0\ 0\ 1\ 1\}$. Следует отметить, что приведенные примеры $SOCRT(4, 4)$ и $SOCRT(3, 4)$ являются не единственными решениями для $N = 6$.

$$
\begin{aligned}
T_0 &= t_{0,N-1}, t_{0,N-2}, ..., t_{0,2N/3}, t_{0,2N/3-1}, ..., t_{0,N/3}, t_{0,N/3-1}, ..., t_{0,2}, t_{0,1}, t_{0,0};\\
T_1 &= \overline{t_{1,N-1}}, \overline{t_{1,N-2}}, ..., \overline{t_{1,2N/3}}, \overline{t_{1,2N/3-1}}, ..., \overline{t_{1,N/3}}, t_{0,N/3-1}, ..., t_{1,2}, t_{1,1}, t_{1,0};\\
T_2 &= t_{2,N-1}, t_{2,N-2}, ..., t_{2,2N/3}, \overline{t_{2,2N/3-1}}, ..., \overline{t_{2,N/3}}, \overline{t_{2,N/3-1}}, ..., \overline{t_{2,2}}, \overline{t_{2,1}}, \overline{t_{2,0}};\\
T_3 &= \overline{t_{3,N-1}}, \overline{t_{3,N-2}}, ..., \overline{t_{3,2N/3}}, t_{3,2N/3-1}, ..., t_{3,N/3}, \overline{t_{3,N/3-1}}, ..., \overline{t_{3,2}}, \overline{t_{3,1}}, \overline{t_{3,0}}.
\end{aligned}
\tag{4.18}
$$

Кроме того, для заданного q = 4 возможны и другие решения для меньших значений $max\_minHD(T_i, T_j)$, чем $2N/3$, но также близких к оптимальным значениям, определяемым соотношением (4.16). Одним из таких решений является тест $SOCRT(4, 5N/8)$ с $max\_minHD(T_i, T_j) = 5N/8 < 2N/3$, имеющий следующий вид [334]:

$$
\begin{aligned}
T_0 &= t_{0,N-1}, t_{0,N-2}, ..., t_{0,3N/8}, t_{0,3N/8-1}, ..., t_{0,3N/8}, t_{0,3N/8-1}, ..., t_{0,1}, t_{0,0};\\
T_1 &= \overline{t_{1,N-1}}, \overline{t_{1,N-2}}, ..., \overline{t_{1,3N/8}}, t_{1,3N/8-1}, ..., t_{1,2}, t_{1,1}, t_{1,0};\\
T_2 &= t_{2,N-1}, t_{2,N-2}, ..., t_{2,5N/8}\, \overline{t_{2,5N/8-1}}, ..., \overline{t_{2,2}}, \overline{t_{2,1}}, \overline{t_{2,0}};\\
T_3 &= \overline{t_{3,N-1}}, \overline{t_{3,N-2}}, ..., \overline{t_{3,5N/8}}, t_{3,5N/8-1}, ..., t_{3,3N/8}, \overline{t_{3,3N/8-1}}, ..., \overline{t_{3,1}}, \overline{t_{3,0}}.
\end{aligned}
\tag{4.19}
$$

В силу того, что $HD(T_0, T_1) = HD(T_0, T_2) = HD(T_1, T_3) = HD(T_2, T_3) = 5N/8$ и $HD(T_0, T_3) = HD(T_1, T_2) = 6N/8$ для (4.19), получим, что $max\_minHD(T_i, T_j) = 5N/8$. Отметим, что для реализации данного теста необходимо обеспечить делимость N на 8, что является более практическим случаем по сравнению с делимостью N на 3, необходимой при использовании (4.17) и (4.18).

Примерами $SOCRT(4, 5N/8)$ (4.19) могут быть $SOCRT(4, 5) = \{T_0, T_1, T_2, T_3\}$ = {0 0 0 0 0 0 0 0, 0 0 0 1 1 1 1 1, 1 1 1 1 1 0 0 0, 1 1 1 0 0 1 1 1} и $SOCRT(4, 10) = \{T_0, T_1, T_2, T_3\}$ = {0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0, 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1, 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0, 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1} для N = 8 и 16. Два приведенных примера $SOCRT(4, 2N/3)$ (4.18) и $SOCRT(4, 5N/8)$ (4.19), очевидно, являются наилучшими решениями, соответствующими условиям теоремы 4.1 и соотношению (4.16) для q = 4.

Для q = 5, согласно (4.16), $max\_minHD(T_i, T_j) \le 5N/8$, однако наилучшим решением, очевидно, является тест:

$$
\begin{aligned}
T_0 &= t_{0,N-1}, t_{0,N-2}, ..., t_{0,3N/7}, t_{0,3N/7-1}, ..., t_{0,N/3+1}, t_{0,N/3}, ..., t_{0,1}, t_{0,0};\\
T_1 &= \overline{t_{1,N-1}}, \overline{t_{1,N-2}}, ..., \overline{t_{1,3N/7}}, t_{1,3N/7-1}, ..., t_{1,2}, t_{1,1}, t_{1,0};\\
T_2 &= t_{2,N-1}, t_{2,N-2}, ..., t_{2,4N/7}\, \overline{t_{2,4N/7-1}}, ..., \overline{t_{2,2}}, \overline{t_{2,1}}, \overline{t_{2,0}};\\
T_3 &= \overline{t_{3,N-1}}, \overline{t_{3,N-2}}, ..., \overline{t_{3,5N/7}}, t_{3,5N/7-1}, ..., t_{3,2N/7}, \overline{t_{3,2N/7-1}}, ..., \overline{t_{3,1}}, \overline{t_{3,0}};\\
T_4 &= \overline{t_{4,N-1}}, \overline{t_{4,N-2}}, ..., \overline{t_{4,6N/7}}, t_{4,6N/7-1}, ..., t_{4,5N/7}, \overline{t_{4,5N/7-1}}, ..., \overline{t_{4,4N/7}},\\
&\quad t_{4,4N/7-1}, ..., t_{4,3N/7}, \overline{t_{4,3N/7-1}}, ..., \overline{t_{4,2N/7}}, t_{4,2N/7-1}, ..., t_{4,N/7}, \overline{t_{4,N/7-1}}, ..., \overline{t_{1,0}}.
\end{aligned}
\tag{4.20}
$$

Для приведенного теста $SOCRT(5, 4N/7) = \{T_0, T_1, T_2, T_3, T_4\}$, величина $max\_minHD(T_i, T_j) = 4N/7 < 5N/8$. Примером теста $SOCRT(5, 4N/7)$ для $N = 7$ может быть $SOCRT(5, 4) = \{T_0, T_1, T_2, T_3, T_4\}$ = {0 0 0 0 0 0 0, 1 1 1 1 0 0 0, 0 0 0 1 1 1 1, 1 1 0 0 0 1 1, 1 0 1 0 1 0 1}. Примеры теста $SOCRT(5,4N/7)$ для $N = 7$, 14 и 21 приведены в таблице 4.8.

Таблица 4.8 – *Тест* $SOCRT(5, 4N/7)$ *для различных значений* N

MMHD(5)			
N	7	14	21
T_0	0000000	00000000000000	000000000000000000000
T_1	1111000	11111111000000	11111111111000000000
T_2	0001111	00000011111111	00000000111111111111
T_3	1100011	11110000001111	11111100000000000111111
T_4	1010101	11001100110011	11100011100011100011 1

Как видно из приведенных примеров и общей структуры теста $SOCRT(5, 4N/7)$ (4.20), $HD(T_0, T_1) = 4N/7$, $HD(T_0, T_2) = 4N/7$, $HD(T_0, T_3) = 4N/7$, $HD(T_0, T_4) = 4N/7$, $HD(T_1, T_2) = 6N/7$, $HD(T_1, T_3) = 4N/7$, $HD(T_1, T_4) = 4N/7$, $HD(T_2, T_3) = 4N/7$, $HD(T_2, T_4) = 4N/7$ и $HD(T_3, T_4) = 4N/7$. С учетом того, что $4N/7 < 6N/7$, тест $SOCRT(5, 4N/7)$ имеет $max\_min\{SOCRT(5, 4N/7)\} = 4N/7$.

Для теста $SOCRT(5, 4N/7)$ так же, как и для предыдущих тестов $SOCRT(3, 2N/3)$ и $SOCRT(4, 2N/3)$, накладывается ограничение на размерность N теста. В случае $SOCRT(5, 4N/7)$, необходимо выполнение условия делимости N на 7.

Для $q = 6$, в соответствии с (4.16), $max\_minHD(T_i, T_j) \le 3N/5$. Так же, как и в предыдущем случае, оказывается невозможным построение теста из шести наборов и минимальным максимальным расстоянием Хемминга, равным $3N/5$. Наиболее близким результатом к оптимальному решению, определяемому теоремой 4.1, является $SOCRT(6, 8N/15)$, для которого $max\_minHD(T_i, T_j) \le 8N/15$. Примером $SOCRT(6, 8N/15)$ может быть тест, построенный по аналогичной процедуре, как и предыдущие тесты $SOCRT(4, 2N/3)$ и $SOCRT(5, 4N/7)$, у которых $max\_minHD(T_i, T_j)$ равняется $2N/3$ и $4N/7$, соответственно. Тогда для $SOCRT(6, 8N/15)$ первые четыре набора повторяют структуру теста $SOCRT(4, 2N/3)$ [227, 340]. Для $N = 15$ получим T_0 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0, T_1 = 1 1 1 1 1 1 1 0 0 0 0 0 0 0, T_2 = 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 и T_3 = 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1. Далее, используя принцип построения *OCRT*, основанный на половинном делении, получим два оставшиеся тестовых набора T_4 = 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 и T_5 = 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1. Для приведенного теста соответствующие расстояния Хэмминга принимают значения $HD(T_0, T_1) = 8$, $HD(T_0, T_2) = 8$, $HD(T_0, T_3) = 8$, $HD(T_0, T_4) = 8$, $HD(T_0, T_5) = 8$, $HD(T_1, T_2) = 14$, $HD(T_1, T_3) = 8$, $HD(T_1, T_4) = 8$, $HD(T_1, T_5) = 8$, $HD(T_2, T_3) = 8$, $HD(T_2, T_4) = 8$, $HD(T_2, T_5) = 8$, $HD(T_3, T_4) = 8$, $HD(T_3, T_5) = 8$ и $HD(T_4, T_5) = 8$. Для N, равного $15m$, где m − целое, все значения расстояний $HD(T_i, T_j)$, $i \ne j$ и $i \ne 1$, $j \ne 2$; $i, j \in \{0, 1, 2, 3, 4, 5\}$ равняются $8N/15$, кроме $HD(T_1, T_2) = 14N/15$.

Обобщая рассмотренную эвристическую процедуру нахождения $SOCRT(q)$ для малых значений q, представим формальный алгоритм синтеза теста $SOCRT(q)$ для заданного $q \geq 4$. В соответствии с данным алгоритмом будет сформирован тест $SOCRT(q)$, состоящий из q наборов с $max\_min\{SOCRT(q)\} = 2^{q-3}N/(2^{q-2}-1)$. Отметим, что для любого целого q расстояние $2^{q-3}N/2^{q-2}-1$ больше, чем $N/2$, однако с ростом q, стремящееся к $N/2$. Количество разрядов N тестовых наборов в битах должно принимать значения, кратные величине $2^{q-2}-1$, причем минимальное его значение равно $2^{q-2}-1$.

Рассмотрим процедуру формирования теста $SOCRT(q)$ с $max\_min\{SOCRT(q)\} = 2^{q-3}N/(2^{q-2}-1)$ и $N = 2^{q-2}-1$ для заданного количества q тестовых наборов.

Процедура 4.3. Генерирование $SOCRT(q)$ для $q \in \{4, 5, 6, \ldots\}$ и $N = 2^{q-2}-1$.

1. Первоначально формируется первый тестовый набор T_0. В общем случае в качестве T_0 может быть использован случайный вектор, состоящий из $2^{q-2}-1$ бит. Не нарушая общности дальнейших рассуждений, в качестве T_0 используем вектор, включающий $2^{q-2}-1$ нулей.

2. Второй тестовый набор T_1 формируется как 2^{q-3} единиц и $2^{q-3}-1$ последующих нулей.

3. Третий тестовый набор T_2 формируется как $2^{q-3}-1$ нулей и 2^{q-3} последующих единиц.

4. Четвертый тестовый набор T_3 формируется как 2^{q-4} единиц и 2^{q-4} нулей, слева направо в первой половине из 2^{q-3} бит и 2^{q-4} единиц и 2^{q-4} нулей, справа налево во второй половине из 2^{q-3} бит вектора длиной $2^{q-2}-1$. Все четыре блока единиц и нулей имеют одинаковую размерность 2^{q-4} и располагаются в такой последовательности, что два нулевых блока строятся из $2^{q-3}-1$ бит путем использования центрального бита вектора в обоих нулевых блоках.

5. Последующие $q-4$ наборы формируются единообразно, используя итерационную процедуру, причем для построения следующего тестового набора T_i используется предыдущий T_{i-1} набор. Для генерирования T_4 используется T_3 и так далее. На каждой итерации очередной тестовый набор T_i формируется из предыдущего T_{i-1}, при этом все блоки первой половины из 2^{q-3} бит предыдущего набора делятся на два блока, первый из которых является единичным блоком, а второй – нулевым (слева направо). Аналогичным преобразованиям подвергаются блоки второй половины из 2^{q-3} бит предыдущего набора T_{i-1}, с той лишь разницей, что единичные и нулевые блоки строятся справа налево.

Обобщением данной процедуры 4.3 является использование на его первом шаге произвольного случайного набора T_0. Тогда единичные значения в последующих наборах, согласно данной процедуре, будут соответствовать инвертированию бит набора T_0, а нулевые значения отсутствию инверсии бит набора T_0.

Пример 4.5. В качестве примера рассмотрим процедуру синтеза $SOCRT(q)$ для $q = 7$ и $N = 2^{q-2} - 1 = 2^5 - 1 = 31$.

1. Первоначально формируется первый тестовый набор T_0 = 0000000000000000 000000000000, состоящий из 31 нулей.

2. Второй тестовый набор T_1 формируется как $2^{7-3} = 16$ единиц и $2^{7-3}-1 = 15$ нулей, т. е. T_1 = 1111111111111110000000000000000.

3. Третий тестовый набор T_2 формируется как $2^{7-3} - 1 = 15$ нулей и $2^{7-3} = 16$ единиц, т. е. T_2 = 0000000000000001111111111111111.

4. Четвертый тестовый набор T_3 формируется как $2^{7-4} = 8$ единиц и $2^{7-4} = 8$ нулей, слева направо в первой половине из $2^{7-3} = 16$ бит и $2^{7-4} = 8$ единиц и $2^{7-4} = 8$ нулей, справа налево во второй половине из 2^{7-3} бит вектора длиной $2^{7-2} - 1$. В результате получим T_3 = 1111111000000000000011111111. Выделенный центральный нулевой символ используется в двух восьми битных нулевых блоках.

5. Для генерирования T_4 используется T_3 путем деления четырех блоков T_3 на два блока и формирования нулевых и единичных блоков для первой половины T_3 слева направо и для второй половины справа налево. В результате получим T_4 = 111100001111000**0**000 111100001111. Аналогичным образом формируется T_5 = 1100110011001**0**011001100110 011 и T_6 = 1010101010 10101**0**101010101010101.

Подобным образом синтезируется $SOCRT(q)$ для произвольного значения q. При увеличении числа наборов q, значение $max\_min\{SOCRT(q)\} = 2^{q-3}N/2^{q-2} - 1 = N/(2 - 1/2^{q-3})$ быстро сходится к величине $N/2$, которая характерна для тестов $SOCRT(q)$. Поэтому данный алгоритм целесообразно применять для синтеза вероятностных тестов с малым числом наборов $q < 10$. Результат синтеза, согласно процедуре 4.3 для $N = 2^{q-2} - 1$, легко обобщается на произвольный случай, в том числе и с использованием следующих свойств [126, 159]. Известны четыре основных алгоритма построения множества тестов (кодов) с заданными свойствами на основании исходного теста (кода) [126, 159]. Эти алгоритмы используют следующие четыре свойства, которые сформулируем для случая $SOCRT(q)$.

Свойство 4.1. Результатом перестановки бит $t_{i,j}$ одновременно во всех q тестовых наборах $T_i = t_{i,N-1}t_{i,N-2}\ldots t_{i,2}t_{i,1}t_{i,0}$ теста $SOCRT(q) = \{T_0, T_1, \ldots, T_{q-1}\}$, является тест $SOCRT(q)$.

Свойство 4.2. Результатом инвертирования бит $t_{i,j}$ во всех q тестовых наборах $T_i = t_{i,N-1}\, t_{i,N-2}\ldots t_{i,2}t_{i,1}t_{i,0}$ теста $SOCRT(q) = \{T_0, T_1, \ldots, T_{q-1}\}$ также является тест $SOCRT(q)$.

Свойство 4.3. Тест $SOCRT(q) = \{T_0, T_1, \ldots, T_{q-1}\}$ с тестовыми наборами $T_i = t_{i,2N-1}\, t_{i,2N-2}\ldots t_{i,2}t_{i,1}t_{i,0,}$ состоящими из $2N$ бит, формируется из тестовых наборов $T_i = t_{i,N-1}t_{i,N-2}\ldots t_{i,2}t_{i,1}t_{i,0}$ исходного теста $SOCRT(q)$ путем их конкатенации (повторения), т.е. $T_i = t_{i,N-1}\, t_{i,N-2}\ldots t_{i,2}t_{i,1}\, t_{i,0}t_{i,N-1}t_{i,N-2}\ldots t_{i,2}t_{i,1}t_{i,0.}$

Свойство 4.4. Результатом масштабирования (увеличения) в v раз теста $SOCRT(q) = \{T_0, T_1, \ldots, T_{q-1}\}$ является тест $SOCRT(q)$, состоящий из тестовых

наборов $T_i = (t_{i,N-1})^v (t_{i,N-2})^v \ldots (t_{i,2})^v (t_{i,1})^v (t_{i,0})^v$, где $T_i = t_{i,N-1}t_{i,N-2}\ldots t_{i,2}t_{i,1}t_{i,0}$ – тестовый набор теста *SOCRT*(*q*).

Взяв за основу, например, тест *SOCRT*(4) = *SOCRT*(4,2) = {000, 110, 011, 101} для $N = 3$ с *max_min*{*SOCRT* (4)} = $2N/3 = 2$, можно получить тест *SOCRT*(4) = {000, 011, 110, 101} как результат перестанови первого и третьего бита всех наборов. Применив инверсию ко всем наборам исходного теста *SOCRT*(4), также получим *SOCRT*(4) = {111, 001, 100, 010}. Для $N = 6$, на основании свойства 4.3, сформируем тест *SOCRT*(4) = {000000, 110110, 011011, 101101}. Используя свойство 4.4 для увеличения исходного теста *SOCRT*(4) = {000, 110, 011, 101} в $v = 3$ раза, получим *SOCRT*(4) = $\{(0)^3\ (0)^3(0)^3, (1)^3(1)^3(0)^3, (0)^3(1)^3(1)^3, (1)^3(0)^3(1)^3\}$ = {000000000, 111111000, 000111111, 111000111}. Отметим, что все вновь полученные тесты принадлежат одному и тому же семейству тестов *SOCRT*(*q*), состоящих из $q = 4$ тестовых наборов. Каждый из тестов данного семейства имеет одно и то же значение *max_min*{*SOCRT*(4)} = $2N/3$.

Совместное использование всех четырех свойств позволяет сформировать семейство тестовых наборов *SOCRT*(*q*).

4.5. Анализ эффективности управляемых вероятностных тестов

Как было показано в разделе 4.3, в качестве меры эффективности управляемых вероятностных тестов рассматривается их близость к тестам с максимальной эффективностью, в качестве которых выступают псевдо-исчерпывающие тесты *T*(*N*, *k*). По определению *T*(*N*, *k*) обеспечивает 100 % покрытие для конкретного значения $k < N$ и для всевозможных $r \le k$ из *N* подпространств *N*-мерного пространства [340]. Для $r > k$ эффективность *E*(*T*(*N*, *k*), *r*) псевдо-исчерпывающего теста, в процентном исчислении, может быть оценена как количество комбинаций *C*(*T*(*N*, *k*), *r*) из *r* бит, формируемых тестом *T*(*N*, *k*) на всевозможных *r* из *N* разрядах, деленное на общее количество возможных комбинаций (4.8).

Используем характеристику (4.8) для оценки качества вероятностных тестов оценим эффективность тестов *OCRT*(*q*) и *SOCRT*(*q*). Первоначально рассмотрим случаи тестов *OCRT*(*q*) и *SOCRT*(*q*), состоящих из $q = 3$ и $q = 4$ тестовых наборов, каждый из которых состоит из *N* бит. На рис. 4.2 представлены тесты *OCRT*(3) и *SOCRT*(3) для $q = 3$, а на рис. 4.3 тесты *OCRT*(4) и *SOCRT*(4).

Как видно из *OCRT*(3), приведенного на рис. 4.2, $HD(T_0, T_1) = N$, $HD(T_0, T_2) = N/2$ и $HD(T_1, T_2) = N/2$, а для *SOCRT*(3) $HD(T_0, T_1) = d$, $HD(T_0, T_2) = d$ и $HD(T_1, T_2) = 2N - 2d$. Максимальное значение *max_min*{*SOCRT*(3)} = *max_min*{$HD(T_0, T_1)$, $HD(T_0, T_2)$, $HD(T_1, T_2)$} = *max_min*{d, $2N - 2d$} для *SOCRT*(3) достигается при выполнении равенства $d = 2N - 2d$. Предполагая, что *N* делится на 3, получим, что *max_min*{*SOCRT*(3)} = $2N/3$ [340]. В тоже время *max_min*{*OCRT* (3)} = $N/2$.

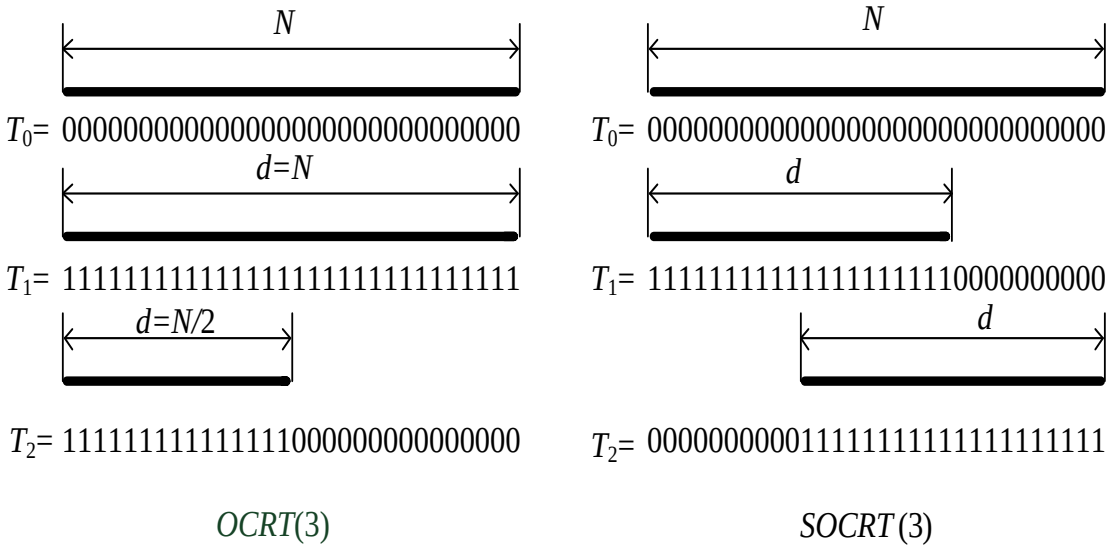


Рисунок 4.2 – *Тесты OCRT*(3) *и SOCRT*(3)

Из рис. 4.3 следует, что для *OCRT*(4) $HD(T_0, T_1) = N$, $HD(T_0, T_2) = N/2$, $HD(T_0, T_3) = N/2$, $HD(T_1, T_3) = N/2$ и $HD(T_2, T_3) = N$, а для *SOCRT*(4) - $HD(T_0, T_1) = d$, $HD(T_0, T_2) = d$, $HD(T_0, T_3) = 2v$, $HD(T_1, T_2) = 2N - 2d$, $HD(T_1, T_3) = d$ и $HD(T_2, T_3) = d$.

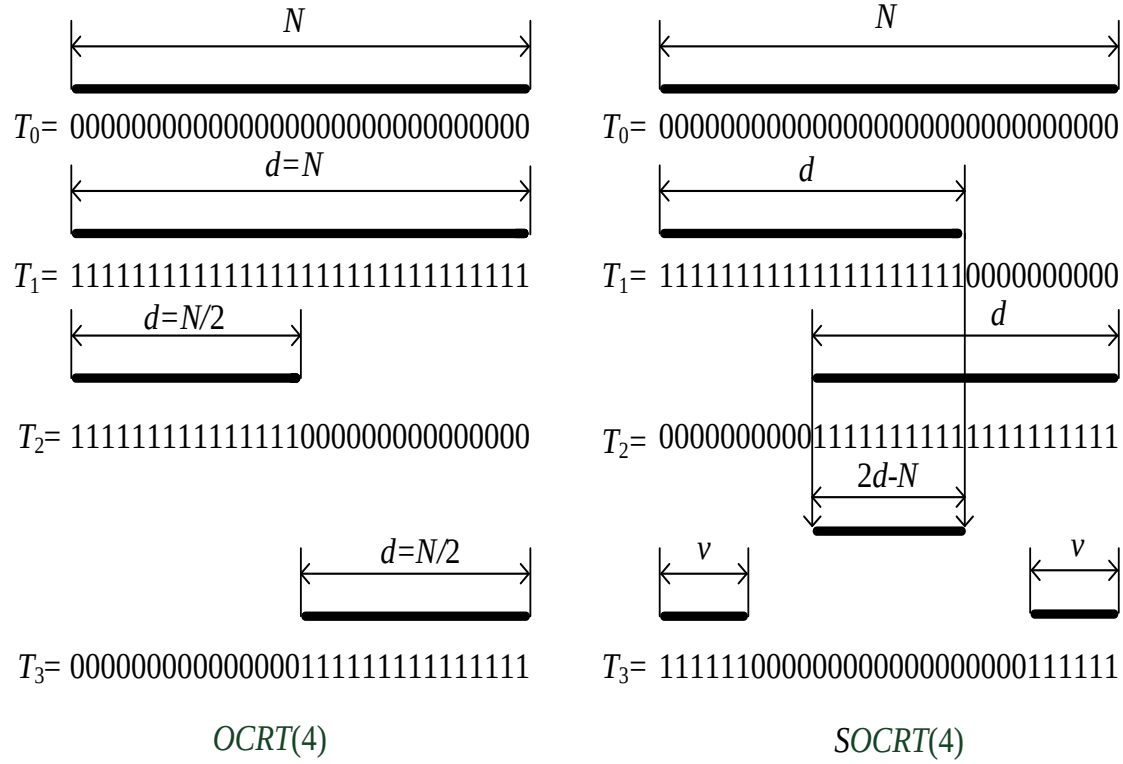


Рисунок 4.3 – *Тесты OCRT*(4) *и SOCRT*(4)

Максимальное значение $max\_min\{MMHD(4)\} = max\_min\{d, 2N - 2d, 2v\}$ для *SOCRT*(3) достигается при выполнении равенства $d = 2N - 2d$ и $2v =$

d. Предполагая, что N делится на 3, получим, что $max\_min\{MMHD(4)\} = 2N/3$ [340]. В то же время $max\_min\{OCRT(4)\} = N/2$.

Количество комбинаций $C(OCRT(3), r)$ из r бит, формируемых тестом $OCRT(3)$ на всевозможных r из N разрядах наборов T_0, T_1 и T_2, предполагая, что N четное, определяется как:

$$C(OCRT(3),r) = 2\times\binom{N}{r} + \sum_{i=1}^{r-1}\left(\binom{N/2}{r-i}\times\binom{N/2}{i}\right). \quad (4.21)$$

Значение $C(OCRT(4),r)$ для теста $OCRT(4)$ вычисляется согласно соотношению:

$$C(OCRT(4),r) = 2\times\binom{N}{r} + 2\times\sum_{i=1}^{r-1}\left(\binom{N/2}{r-i}\times\binom{N/2}{i}\right). \quad (4.22)$$

Для r=2 получим, что $C(OCRT(3),2)=N(N-1)+(N/2)(N/2)=5N^2/4-N$, $C(OCRT(4),2) = N(N-1) + 2(N/2)(N/2) = 3N^2/2-N$. А для случая, когда r=3, получим $C(OCRT(3),3) = (N(N-1)(N-2))/3 + (N/2)^2(N/2-1) = 11N^3/24 - 5N^2/4 + 2N/3$, $C(OCRT(4),3) = (N(N-1)(N-2))/3 + 2(N/2)^2(N/2-1) = 7N^3/12-3N^2/2 + 2N/3$.

Аналогичные характеристики для $SOCRT(3)$ и $SOCRT(4)$ (см. рис. 4.2 и рис. 4.3) с $max\_minHD(T_i,T_j) = 2N/3$ имеют вид:

$$C(SOCRT(3),r) = \binom{N}{r} + \binom{2N/3}{r} + \binom{N/3}{r} + 2\times\sum_{i=1}^{r-1}\left(\binom{2N/3}{r-i}\times\binom{N/3}{i}\right) + \sum_{i=1}^{r-1}\left(\binom{N/3}{r-i}\times\binom{N/3}{i}\right) \quad (4.23)$$

$$C(SOCRT(4),r) = \binom{N}{r} + \binom{2N/3}{r} + \binom{N/3}{r} + 3\times\sum_{i=1}^{r-1}\left(\binom{2N/3}{r-i}\times\binom{N/3}{i}\right) + 2\times\sum_{i=1}^{r-1}\left(\binom{N/3}{r-i}\times\binom{N/3}{i}\right) \quad (4.24)$$

Для $r = 2$ и $r = 3$ согласно (4.23) и (4.24) получим, что

$C(SOCRT(3), 2) = (N(N - 1))/2 + ((2N/3)(2N/3 - 1))/2 + ((N/3)(N/3 - 1))/2 + 2(2N/3) \times (N/3)) + (N/3)(N/3)) = 4N^2/3 - N$,

$C(SOCRT(4), 2) = (N(N - 1))/2 + ((2N/3)(2N/3 - 1))/2 + ((N/3)(N/3 - 1))/2 + 3(2N/3) \times (N/3)) + 2(N/3)(N/3)) = 5N^2/3 - N$,

$C(SOCRT(3), 3) = (N(N-1)(N-2))/6 + ((2N/3)(2N/3-1)(2N/3-2))/6 + ((N/3)(N/3-1)(N/3-2))/6 + 2((2N/3)(N/3)N/3-1)) + (N/3)(2N/3-1)(N/3)) + (N/3)(N/3)(N/3) = 13N^3/27 - 4N^2/3 + 2N/3.$

$C(SOCRT(4), 3) = (N(N-1)(N-2))/6 + ((2N/3)(2N/3-1)(2N/3-2))/6 + ((N/3)(N/3-1)(N/3-2))/6 + 3((2N/3)(N/3)N/3-1)) + (N/3)(2N/3-1)(N/3)) + 2(N/3)(N/3)(N/3) = 17N^3/27 - 5N^2/3 + 2N/3.$

Основываясь на значениях $C(OCRT(q), r)$ и $C(SOCRT(q),r)$ для конкретных величии q, сопоставительная эффективность может быть оценена разностью данных величин:

$$\Delta(q,r)=C(SOCRT(q),r)-C(OCRT(q),r). \qquad (4.25)$$

При $q = 3$ для $r = 2$ и $r = 3$ получим $\Delta(3, 2) = C(SOCRT(3), 2) - C(OCRT(3), 2) = N^2/12$, $\Delta(3, 3) = C(SOCRT(3), 3) - C(OCRT(3), 3) = 5N^3/216 - N^2/12$.

При $q = 4$ для $r = 2$ и $r = 3$ получим $\Delta(4, 2) = C(SOCRT(4), 2) - C(OCRT(4), 2) = N^2/6$, $\Delta(4,3) = C(SOCRT(4), 3) - C(OCRT(4), 3) = 5N^3/108 - N^2/6$.

Для общего случая, используя соотношение (4.8), применяемое для оценки эффективности исчерпывающего тестирования, введем относительную характеристику (4.26) для сравнительного анализа тестов $OCRT(4)$ и $SOCRT(4)$.

$$\delta(q,r) = \frac{\Delta(q,r)}{2^r \binom{N}{r}} 100\%. \qquad (4.26)$$

При $q = 3$ для $r = 2$ и $r = 3$ получим:

$$\delta(3,2) = \frac{1}{24(1-1/N)} 100\%; \qquad \delta(3,3) = \frac{5-18/N}{288(1-3/N+2/N^2)} 100\%.$$

При $q = 4$ для $r = 2$ и $r = 3$ получим:

$$\delta(4,2) = \frac{1}{12(1-1/N)} 100\%; \qquad \delta(4,3) = \frac{5-18/N}{144(1-3/N+2/N^2)} 100\%.$$

Для $N \to \infty$ $\delta(3, 2) = 4{,}17\%$, $\delta(4, 2) = 8{,}33\%$, $\delta(3, 3) = 1{,}74\%$ и $\delta(4, 3) = 3{,}47\%$. Зависимость $\delta(4, r)$ при $N \to \infty$ для произвольного значения r показана на рис. 4.4.

Анализ приведенных результатов позволяет сделать следующие выводы. Как видно из приведенных выше соотношений и, в особенности, рис. 4.4, эффективность управляемых вероятностных тестов существенно больше для малых значений r, причем их эффективность увеличивается с увеличением

max_minHD(T_i, T_j), применяемого при синтезе теста как критерий эффективности [227, 340].

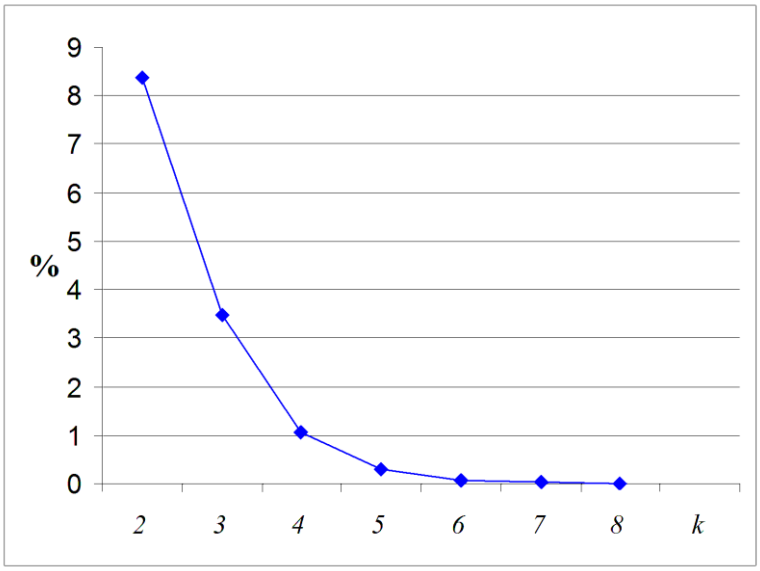


Рисунок 4.4 – *Зависимость* $\delta(4, r)$ *при* $N \to \infty$

В то же время увеличение метрики *max_minHD*(T_i, T_j) существенно уменьшает количество q тестовых наборов, которое, как следует из (4.5), ограничивается несколькими наборами [227, 340].

Для значений q = 2, 3 и 4 тесты *SOCRT*(2), *SOCRT*(3) и *SOCRT*(4) с максимальным значением *max_minHD*(T_i, T_j) получены в [337], а их эффективность экспериментально подтверждена в [331] на примере тестирования ОЗУ.

Для сопоставительного анализа эффективности вероятностных тестов *RT*, оптимальных управляемых вероятностных тестов *OCRT* и оптимальных управляемых вероятностных тестов малой длины *SOCRT* используем q = 4 набора *OCRT* и *SOCRT*(4, 2*N*/3), которые для каждой итерации будем применять в качестве масок M для модификации $M \oplus T_R$ случайного набора T_R. Например, для N = 6 *OCRT* = {0 0 0 0 0 0, 1 1 1 1 1 1, 0 0 0 1 1 1, 1 1 1 0 0 0} и *SOCRT*(4,4) = {0 0 0 0 0 0, 1 1 1 1 0 0, 0 0 1 1 1 1, 1 1 0 0 1 1}. Тогда для случайного набора T_R = 0 1 0 1 0 1 текущей итерации, четыре набора *OCRT* примут вид 0 1 0 1 0 1, 1 0 1 0 1 0, 0 1 0 0 1 0 и 1 0 1 1 0 1, а для *SOCRT* – 0 1 0 1 0 1, 1 0 1 0 0 1, 0 1 1 0 1 0 и 1 0 0 1 1 0.

Как было показано ранее [331, 334], все существующие модификации вероятностных тестов являются аппроксимациями псевдо-исчерпывающих тестов, позволяющих достичь 100 % полноты покрытия для заданного количества разрядов r тестовых наборов. Поэтому в качестве критерия эффективности для сравнения *RT*, *OCRT* и *SOCRT* используем среднее количество наборов для обеспечения всевозможных комбинаций в r = 3, 4 и 5 разрядах тестовых наборов из N = 6, 9 и 15 бит. Результаты экспериментальных исследований приведены в таблице 4.9.

Таблица 4.9 – *Среднее количество наборов для обеспечения* 2^r *комбинаций*

r	*RT*			*OCRT*			*SOCRT*(4, 2*N*/3)		
	$N = 6$	$N = 9$	$N = 15$	$N = 6$	$N = 9$	$N = 15$	$N = 6$	$N = 9$	$N = 15$
3	21,9	21,4	21,8	15,0	16,9	16,4	11,7	12,8	13,1
4	53,9	54,4	53,8	32,6	42,1	40,7	33,0	33,7	33,9
5	127,8	128,4	129,6	91,8	104,5	100,5	90,8	86,2	86,7

Как видно из приведенной таблицы, наиболее эффективными являются *SOCRT*, так как при их итерационном применении обеспечение всех двоичных комбинаций для всевозможных $r = 3$, 4 и 5 из $N = 6$, 9 и 15 бит достигается для меньшего количества тестовых наборов. Соответственно, наихудший результат достигается при использовании классических вероятностных тестов.

Глава 5. Многократное вероятностное тестирование

5.1. Определение многократных управляемых вероятностных тестов

Идея многократного тестирования изначально была сформулирована в рамках *неразрушающего тестирования* (*transparent testing*) [149], а затем исчерпывающего и псевдо-исчерпывающего тестирования запоминающих устройств [105], основанного на использовании повторяющейся тестовой процедуры для различных первоначальных состояний запоминающих устройств. Как отмечалось в ряде источников [52, 134, 346], неразрушающее тестирование теоретически позволяет обнаружить любые неисправные состояния запоминающих устройств, что на практике требует многократного повторения одного либо нескольких тестов с изменяемыми начальными параметрами. Эффективность многократного тестирования зависит как от эффективности используемых тестов и их количества, так и от изменяемых перед каждым повторным применением теста начальных условий.

Многократное применение исчерпывающих, псевдо-исчерпывающих и почти псевдо-исчерпывающих вероятностных тестов для целей тестирования с различными начальными параметрами, такими как: исходное состояние вычислительной системы или начальный случайно сгенерированный тестовый набор позволяют достичь высокой эффективности процедуры тестирования [8].

Обобщением существующих модификаций управляемых вероятностных тестов (см. раздел 4), основанных на использовании различных метрик расстояния, является оптимальный управляемый вероятностный тест [334, 347]. Подобный тест, очевидно, является единственным представителем из множества управляемых вероятностных тестов, который по своей сути предполагает многократное применение.

Проведенный анализ многократных тестов показывает их общность в части подходов для их формирования. Во-первых, основные усилия разработчиков многократных тестов прилагаются для построения базового теста для однократного его применения, при этом активно используются всевозможные меры различия между тестовыми наборами. Во-вторых, в большинстве случаев при формировании последующих итераций базового теста применяется некий случайный параметр, как правило, начальный тестовый набор. В-третьих, все итерации многократных тестов не требуют значительных вычислительных затрат и формируются согласно детерминированным алгоритмам.

Основываясь на методологии управляемых вероятностных тестов, дадим определение многократного управляемого теста.

Определение 5.1. *Многократный управляемый вероятностный тест* $MCRT_r$ состоит из r однократных управляемых вероятностных тестов CRT_0,

CRT_1, CRT_2, …, CRT_{r-1}, где CRT_0 удовлетворяет определению 4.3, а последующие тесты CRT_j, $j \in \{1, 2, 3, ..., r-1\}$ формируются согласно некоторым алгоритмам, таким образом, чтобы они удовлетворяли определенному критерию, либо критериям, полученным на основании предыдущих тестов CRT_0, CRT_1, CRT_2, …, CRT_{j-1}.

Часто на практике в качестве меры расстояния используется расстояние Евклида, вычисляемое как корень квадратный из суммы квадратов разностей координат (элементов) двух многозначных векторов. Более общей мерой расстояния (различия) является *расстояние Минковского* (*Minkowski distance −MD*), которое для случая двух управляемых вероятностных тестов $CRT_k = \{T_{k,0}, T_{k,1}, T_{k,2}, \ldots, T_{k,q-1}\}$ и $CRT_l = \{T_{l,0}, T_{l,1}, T_{l,2}, \ldots, T_{l,q-1}\}$, вычисляется согласно выражению:

$$MD(CRT_k, CRT_l) = \sqrt[\lambda]{\sum_{i=0}^{q-1} \left| T_{k,i} - T_{l,i} \right|^{\lambda}}. \qquad (5.1)$$

Для $\lambda = 2$ расстояние Минковского является расстоянием Евклида, при $\lambda = \infty$ – расстоянием Чебышева (*Chebyshev distance*), а при $\lambda = 1$ принимает вид арифметического расстояния $AD(CRT_k, CRT_l)$ [331, 333, 342]. Отметим, что CRT_k и CRT_l являются p-ичными векторами, где $p = 2^m$, а их тестовые наборы $T_{k,i}$ и $T_{l,i}$, $i \in \{0, 1, 2, ..., q-1\}$ представляют собой p-ичные коды.

По аналогии с (4.1) и (4.2) рассмотрим *расстояние Хэмминга* и *расстояние Евклида* для двух тестов CRT_k и CRT_l. Первоначально отметим, что расстояние Хэмминга $HD(CRT_k, CRT_l)$ (5.2), которое равняется числу несовпадающих компонент $T_{k,i}$ и $T_{l,i}$, исходного теста CRT_k и формируемого CRT_l, может рассматриваться как необходимое условие, которому должен удовлетворять тест CRT_l:

$$HD(CRT_k, CRT_l) = \sum_{i=0}^{q-1} f(T_{k,i}, T_{l,i}); \quad f(T_{k,i}, T_{l,i}) = \begin{cases} 1, \text{если } T_{k,i} \neq T_{l,i}; \\ 0, \text{если } T_{k,i} = T_{l,i}. \end{cases} \qquad (5.2)$$

В этом случае расстояние Хэмминга $HD(CRT_k, CRT_l)$ равняется числу несовпадающих компонент $T_{k,i}$ и $T_{l,i}$, векторов CRT_k и CRT_l, максимальное значение которого $\max HD(CRT_k, CRT_l) = q$, а минимальное $\min HD(CRT_k, CRT_l) = 0$. Очевидно, что требованием с точки зрения максимального различия, которому должны соответствовать CRT_k и CRT_l, является отсутствие у них совпадающих компонент $T_{k,i}$ и $T_{l,i}$, что эквивалентно выполнению неравенства $T_{l,i} \neq T_{k,i}$, $i \in \{0, 1, 2, ..., q-1\}$.

Расстояние Евклида для CRT_k и CRT_l определяется, как:

$$CD(CRT_k, CRT_l) = \sqrt{\sum_{i=0}^{q-1} (T_{i,k} - T_{i,l})^2}. \qquad (5.3)$$

Аналогично, как и суммарные значения расстояний $THD(T_i)$ и $TCD(T_i)$ (4.3) для тестового набора T_i, по отношению к предыдущим наборам T_0, T_1, T_2, …, T_{i-1} управляемого вероятностного теста CRT, введем аналогичную меру расстояния между двумя управляемыми вероятностными тестами CRT_k и CRT_l. При этом каждая пара компонент $T_{k,i}$ и $T_{l,i}$ двух тестов рассматривается как двоичные m-разрядные векторы:

$$THD(CRT_j) = \sum_{i=0}^{j-1} HD(CRT_i, CRT_j); \quad TCD(CRT_j) = \sum_{i=0}^{j-1} CD(CRT_i, CRT_j). \qquad (5.4)$$

Пример 5.1. В качестве примера теста, формирующего p-ичные ($p = 2^4$) данные, используем управляемый вероятностный тест CRT_0= {0 0 1 0 (2), 0 1 1 1 (7), 1 1 0 1 (13), 0 1 0 0 (4), 1 0 0 1 (9)}. Тест состоит из $q = 5$ тестовых наборов, разрядность m каждого из которых равняется 4. Здесь в скобках указано десятичное значение тестовых данных.

Отметим, что тест CRT_0 может быть получен на основании одного из известных методов построения подобных тестов, рассмотренных в предыдущем разделе. Для формирования теста CRT_1 будем использовать исходный тест CRT_0 и, например, операцию инвертирования для получения наборов теста $T_{1,i}$ на основании $T_{0,i}$. Так, инвертируя только младший бит тестовых наборов теста CRT_0, будет получен управляемый вероятностный тест CRT_1 = {0 0 1 1 (3), 0 1 1 0 (6), 1 1 0 0 (12), 0 1 0 1 (5), 1 0 0 0 (8)}. Значения покомпонентных расстояний для тестов CRT_0 и CRT_1 приведены в таблице 5.1.

Так как $HD(CRT_k, CRT_l)$ равняется числу несовпадающих компонент, то на основании (5.2) $HD(CRT_0, CRT_1) = 5$, то есть принимает максимально возможное значение. Расстояние Евклида определяется в соответствии с (5.3), как $CD(CRT_0, CRT_1) = (1^2 + 1^2 + 1^2 + 1^2 + 1^2)^{1/2} = 2{,}23$.

Таблица 5.1 – *Значения покомпонентных расстояний для CRT_0 и CRT_1*

i	$T_{0,i}= t_{0,3}t_{0,2}t_{0,1}t_{0,0}$	$T_{1,i}= t_{1,3}t_{1,2}t_{1,1}t_{1,0}$	$HD(T_{0,i}, T_{1,i})$	$(T_{0,i} - T_{1,i})^2$
0	0 0 1 0	0 0 1 1	1	1
1	0 1 1 1	0 1 1 0	1	1
2	1 1 0 1	1 1 0 0	1	1
3	0 1 0 0	0 1 0 1	1	1
4	1 0 0 1	1 0 0 0	1	1

Совместное применение двух тестов CRT_0 и CRT_1 для формирования входных тестовых данных показывает неравномерность покрытия входного пространства p-ичных ($p = 2^4$) данных, как это видно из таблицы 5.2.

Пример 5.2. Используя тот же исходный управляемый вероятностный тест CRT_0 = {0 0 1 0 (2), 0 1 1 1 (7), 1 1 0 1 (13), 0 1 0 0 (4), 1 0 0 1 (9)} и применив ту же операцию отрицания, но уже старшего разряда тестовых наборов

CRT_0, получим CRT_2 = {1 0 1 0 (10), 1 1 1 1 (15), 0 1 0 1 (5), 1 1 0 0 (12), 0 0 0 1 (1)}.

Таблица 5.2 – *Оценка покрывающей способности тестов CRT_0 и CRT_1*

Входные данные	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CRT_0			+		+			+		+				+		
CRT_1				+		+	+		+				+			
CRT_0&CRT_1			+	+	+	+	+	+	+	+			+	+		

Таблица 5.3 – *Значения покомпонентных расстояний для CRT_0 и CRT_2*

i	$T_{0,i} = t_{0,3}t_{0,2}t_{0,1}t_{0,0}$	$T_{1,i} = t_{1,3}t_{1,2}t_{1,1}t_{1,0}$	$HD(T_{0,i}, T_{1,i})$	$(T_{0,i} - T_{1,i})^2$
0	0 0 1 0	1 0 1 0	1	8
1	0 1 1 1	1 1 1 1	1	8
2	1 1 0 1	0 1 0 1	1	8
3	0 1 0 0	1 1 0 0	1	8
4	1 0 0 1	0 0 0 1	1	8

На основании (5.2) так же, как и в предыдущем случае $HD(CRT_0, CRT_2) = 5$, то есть принимает максимально возможное значение. Расстояние Евклида определяется в соответствии с (5.3), как $CD(CRT_0, CRT_2) = (8^2 + 8^2 + 8^2 + 8^2 + 8^2)^{1/2} = 17{,}88$. Совместное применение двух тестов CRT_0 и CRT_2 для формирования входных тестовых данных показывает заметно лучшую равномерность покрытия входного пространства p-ичных ($p = 2^4$) данных, как это видно из таблицы 5.4.

Таблица 5.4 – *Оценка покрывающей способности тестов CRT_0 и CRT_2*

Входные данные	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CRT_0			+		+			+		+				+		
CRT_2		+				+					+		+			+
CRT_0&CRT_2		+	+		+	+		+		+	+		+	+		+

Сравнивая два примера (5.1) и (5.2) для тестов CRT_1 и CRT_2, оба из которых получены на основании одного и того же исходного теста CRT_0, можно отметить, что во втором случае (CRT_2) достигается существенно большая равномерность покрытия входного пространства тестовыми данными, формируемыми тестами CRT_0 и CRT_2, по сравнению с CRT_0 и CRT_1 (см. таблицу 5.4, и таблицу 5.2).

В качестве более эффективных критериев оценки качества многократных управляемых вероятностных тестов определим максимальное минимальное значение расстояния Хэмминга $MHD(CRT_i(v))$ и максимальное минимальное значение расстояния Евклида $MCD(CRT_i(v))$ (5.5).

Значение $v \in \{0, 1, \ldots, w - 1\}$, а w представляет собой количество рассматриваемых кандидатов в очередной вероятностный тест CRT_i, а сами характеристики (5.5) используются при построении теста CRT_i.

$$MHD(CRT_i(v)) = \max_{CRT_i(v),\, v\in\{0,1,\ldots,w-1\}} \{ \min_{j\in\{0,1,\ldots,i-1\}} HD(CRT_i(v), CRT_j)\};$$
$$MCD(CRT_i(v)) = \max_{CRT_i(v),\, v\in\{0,1,\ldots,w-1\}} \{ \min_{j\in\{0,1,\ldots,i-1\}} ED(CRT_i(v), CRT_j)\}. \quad (5.5)$$

Согласно приведенным метрикам (5.5) очередной управляемый вероятностный тест CRT_i выбирается из множества $\{CRT_i(0), CRT_i(1), CRT_i(2), \ldots, CRT_i(w-1)\}$ кандидатов в тесты по критерию максимальности минимального расстояний Хэмминга и Евклида по отношению к ранее сгенерированным управляемым вероятностным тестам $CRT_0, CRT_1, \ldots, CRT_{i-1}$.

5.2. Характеристики различия для многократных тестов

Первоначально отметим, что расстояние Хэмминга $HD(CRT_k, CRT_l)$, которое равняется числу несовпадающих компонент $T_{k,i}$ и $T_{l,i}$, исходного теста CRT_k и формируемого CRT_l, может рассматриваться как необходимое условие, которому должен удовлетворять тест CRT_l. Очевидно, что условием, которому должны соответствовать CRT_k и CRT_l, является отсутствие у них совпадающих компонент $T_{k,i}$ и $T_{l,i}$,, что эквивалентно выполнению неравенства $T_{l,i} \neq T_{k,i}$, $i \in \{0, 1, 2, ..., q-1\}$. Формально это требование обеспечивается путем максимизации расстояния Хэмминга $HD(CRT_k, CRT_l)$, которое должно равняться q.

Основой для вычисления характеристик различия, приведенных в разделе 5.1, является соотношение пар тестовых наборов $T_{k,i}$ и $T_{l,i}$, $i\in\{0, 1, 2, ..., q-1\}$, для двух управляемых вероятностных тестов, а именно – исходного CRT_k и формируемого CRT_l. Чем более различными, несовпадающими являются коды наборов $T_{k,i}$ и $T_{l,i}$, тем, очевидно, более эффективным будет использование тестов CRT_k и CRT_l при реализации многократного теста $MCRT_r$, состоящего из r однократных тестов.

В качестве основного алгоритма формирования многократных тестов используем хорошо апробированный метод, основанный на применении масок двоичных векторов $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 \neq 0, 0, \ldots, 0, 0$, определяющих инверсии бит исходного базового теста CRT_k по отношению к новому тесту CRT_l [331, 333].

Предположив, что исходный тест CRT_k состоит из тестовых наборов $T_{k,i} = t_{k,m-1}, t_{k,m-2}, \ldots, t_{k,2}, t_{k,1}, t_{k,0}$, где $t_{k,j} \in \{0, 1\}$, для $j \in \{0, 1, 2, \ldots, m-1\}$, тогда выражение для наборов $T_{l,i} \neq T_{k,i}$, теста CRT_l будет иметь вид:

$$T_{l,i} = t_{k,m-1}^{\lambda m-1}, t_{k,m-2}^{\lambda m-2}, ..., t_{k,0}^{\lambda 0} = (\lambda_{m-1} \oplus t_{k,m-1}), (\lambda_{m-2} \oplus t_{k,m-2}), ..., (\lambda_0 \oplus t_{k,0}), \quad (5.6)$$

где при $\lambda_j = 1$ отрицание над $t_{k,j}$ присутствует, а при $\lambda_j = 0$ – отсутствует. Отметим, что в качестве исходного значения $T_{k,i}$ может быть любая m-разрядная двоичная комбинация. Отличие $T_{l,i}$ от $T_{k,i}$ определяется двоичным вектором

$\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$. Из неравенства $T_{l,i} \neq T_{k,i}$, следует, что $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 \neq 0, 0, \ldots, 0, 0$.

Для произвольной пары тестовых наборов $T_{k,i}$ и $T_{l,i}$, из двух управляемых вероятностных тестов CRT_k и CRT_l докажем следующую теорему [333].

Теорема 5.1. Значение $T_{k,i}-T_{l,i}$, для $T_{k,i} = t_{k,m-1}, t_{k,m-2}, \ldots, t_{k,2}, t_{k,1}, t_{k,0}$, где $t_{k,j} \in \{0, 1\}$, при $j \in \{0, 1, 2, \ldots, m-1\}$ и $T_{l,i} = t_{k,m-1}^{\lambda m-1}, t_{k,m-2}^{\lambda m-2}, \ldots, t_{k,1}^{\lambda 1}, t_{k,0}^{\lambda 0} = (\lambda_{m-1} \oplus t_{k,m-1}), (\lambda_{m-2} \oplus t_{k,m-2}), \ldots, (\lambda_1 \oplus t_{k,1}), (\lambda_0 \oplus t_{k,0})$, для которых g значений $\lambda_\alpha, \lambda_\beta, \lambda_\chi, \ldots, \lambda_\delta$, $(\alpha > \beta > \chi > \ldots > \delta)$ равняются 1, а остальные $m - g$ значения λ_k для $k \neq \alpha \neq \beta \neq \chi \neq \ldots \neq \delta$, где $k, \alpha, \beta, \chi, \ldots, \delta \in \{0, 1, 2, \ldots, m-1\}$ равняются 0, вычисляется как:

$$T_{k,i} - T_{l,i} = \sum_{c \in \{\alpha, \beta, \chi, \ldots, \delta\}} (t_{k,c} - \left| t_{k,c} - 1 \right|) 2^c. \tag{5.7}$$

Доказательство теоремы 5.1. Первоначально тестовые наборы $T_{k,i}$ и $T_{l,\text{i}}$ представим в позиционном коде, для которого каждый j-ый, $j \in \{0, 1, 2, \ldots, m-1\}$ разряд имеет вес 2^j, определяемый степенью двойки. Тогда величина разности $T_{k,i}-T_{l,i}$ для произвольного $i \in \{0, 1, 2, \ldots, q-1\}$ будет вычисляться, как: $(t_{k,\alpha} - (1 \oplus t_{k,\alpha}))2^\alpha + (t_{k,\beta} - (1 \oplus t_{k,\beta}))2^\beta + (t_{k,\chi} - (1 \oplus t_{k,\chi}))2^\chi + \ldots + (t_{k,\delta} - (1 \oplus t_{k,\delta}))2^\delta$, где знак (–) означает арифметическую операцию вычитания, а знак ($\oplus$) логическую операцию сложения по модулю два. Отметим, что согласно условию теоремы, выполняется неравенство $\alpha > \beta > \chi > \ldots > \delta$, кроме того, каждое слагаемое $(t_{k,c} - (1 \oplus t_{k,c}))2^c$, где $c \in \{\alpha, \beta, \chi, \ldots, \delta\}$ можно представить, как: $(t_{k,c} - |t_{k,c} - 1|)2^c$. В зависимости от величины $t_{k,c} \in \{0, 1\}$ выражение $(t_{k,c} - |t_{k,c} - 1|)2^c$ принимает значение -2^c или $+2^c$. Действительно, если $t_{k,c} = 0$, то $(t_{k,c} - |t_{k,c}-1|)2^c$ равняется -2^c, а для $t_{k,c} = 1$ принимает значение $+2^c$. Тогда, окончательно, значение разности $T_{k,i} - T_{l,i}$ равняется $(t_{k,\alpha} - |t_{k,\alpha} - 1|)2^\alpha + (t_{k,\beta} - |t_{k,\beta} - 1|)2^\beta + (t_{k,\chi} - |t_{k,\chi} - 1|)2^\chi + \ldots + (t_{k,\delta} - |t_{k,\delta} - 1|)2^\delta$. Что и требовалось доказать.

Пример 5.3. Для тестовых наборов $T_{0,2} = 1\ 1\ 0$ и $T_{1,2} = 0\ 0\ 1$, учитывая, что $\lambda_2, \lambda_1, \lambda_0 = 1, 1, 1$, получим, что $T_{0,2} - T_{1,2} = (t_{0,2} - |t_{0,2} - 1|)2^2 + (t_{0,1} - |t_{0,1} - 1|)2^1 + (t_{0,0} - |t_{0,0} - 1|)2^0 = 2^2 + 2^1 - 2^0 = 5$.

Теорема 5.1 позволяет сформулировать ряд следствий для случая, когда все тестовые наборы $T_{l,0}, T_{l,1}, T_{l,2}, \ldots, T_{l,q-1}$, теста CRT_l сформированы из тестовых наборов $T_{k,0}, T_{k,1}, T_{k,2}, \ldots, T_{k,q-1}$ на основании соотношения (5.6), при использовании одного и того же двоичного вектора отрицаний $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$. Отметим, что $\lambda_j \in \{0,1\}$, а $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 \neq 0, 0, \ldots, 0, 0$. Первоначально рассмотрим *арифметическое расстояние*, чаще называемое *манхэттенским расстоянием* (*Manhattan distance*, или *city block distance*) [191].

$$AD(CRT_k, CRT_l) = \sum_{i=0}^{q-1} \left| T_{k,i} - T_{l,i} \right|. \tag{5.8}$$

Сформулируем две следующие теоремы для арифметического и евклидова расстояний между тестами CRT_k и CRT_l [210, 315, 317, 331].

Теорема 5.2. Арифметическое расстояние $AD(CRT_k, CRT_l)$ для тестов CRT_k и CRT_l, где $CRT_k = \{T_{k,0}, T_{k,1}, T_{k,2}, \ldots, T_{k,q-1}\}$ включает $q = 2^m$ m-разрядных, неповторяющихся, сгенерированных случайным образом, тестовых наборов $T_{k,i}$, а тестовые наборы $T_{l,i}$ получены согласно (5.6) на основании вектора отрицаний $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$, для которого g значений $\lambda_\alpha, \lambda_\beta, \lambda_\chi, \ldots, \lambda_\delta$, ($\alpha > \beta > \chi > \ldots > \delta$) равняются 1, вычисляется как $2^{m+\alpha}$.

Пример 5.4. Для теста CRT_0 = {1 0 0, 1 1 1, 1 1 0, 0 1 1, 0 0 0, 0 0 1, 0 1 0, 1 0 1}, где $m = 3$, а $q = 2^m = 8$, используя вектор отрицаний $\lambda_2, \lambda_1, \lambda_0 = 0, 1, 1$ получим, CRT_1 = {1 1 1, 1 0 0, 1 0 1, 0 0 0, 0 1 1, 0 10, 0 0 1, 1 1 0}. Тогда в соответствии с (5.8), $AD(CRT_0, CRT_1) = |4 - 7| + |7 - 4| + |6 - 5| + |3 - 0| + |0 - 3| + |1 - 2| + |2 - 1| + |5 - 6| = 16$. Отметим, что такой же результат может быть получен на основании теоремы 5.2. Действительно, $AD(CRT_0, CRT_1) = 2^{m+\alpha} = 2^{3+1} = 16$.

Как видно из теоремы 5.2 и примера 5.4, арифметическое расстояние $AD(CRT_k, CRT_l)$ не зависит от вида двоичного вектора отрицаний $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$, точнее от всех его ненулевых компонент $\lambda_\alpha, \lambda_\beta, \lambda_\chi, \ldots, \lambda_\delta$, где $\alpha > \beta > \chi > \ldots > \delta$, а определяется только индексом α старшей ненулевой компоненты λ_α. Действительно, если CRT_k состоит из 2^m m-разрядных, неповторяющихся, сгенерированных случайным образом, тестовых наборов $T_{k,i}$, а CRT_l получен согласно (5.6) на основании $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$ со старшей ненулевой компонентой λ_α, то в этом случае $AD(CRT_k, CRT_l) = 2^{m+\alpha}$.

Теорема 5.2 позволяет сделать два важных вывода. Во-первых, значение арифметического расстояния $AD(CRT_k, CRT_l)$ для случая, когда количество тестовых наборов $q = 2^m$, может быть использовано как среднее значение для арифметического расстояния тестов CRT_k и CRT_l произвольной длины $q < 2^m$, сформированных согласно (5.6). Во-вторых, абсолютное значение $AD(CRT_k, CRT_l)$ возрастает с ростом индекса α старшей ненулевой компоненты вектора отрицаний $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$, примененного в (5.6) для получения CRT_l на основании CRT_k.

Следующая теорема справедлива для случая расстояния Евклида [317].

Теорема 5.3. Расстояние Евклида $CD(CRT_k, CRT_l)$ для тестов CRT_k и CRT_l, где $CRT_k = \{T_{k,0}, T_{k,1}, T_{k,2}, \ldots, T_{k,q-1}\}$ включает $q = 2^m$ m-разрядных, неповторяющихся, сгенерированных случайным образом, тестовых наборов $T_{k,i}$, а тестовые наборы $T_{l,i}$ получены согласно (5.6) на основании вектора отрицаний $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$, для которого g значений $\lambda_\alpha, \lambda_\beta, \lambda_\chi, \ldots, \lambda_\delta$, ($\alpha > \beta > \chi > \ldots > \delta$) равняются 1, вычисляется, как (5.9).

Пример 5.5. Для тестов CRT_0 и CRT_1, приведенных в примере 5.4, используя (5.3), получим $CD(CRT_0, CRT_1) = ((4 - 7)^2 + (7 - 4)^2 + (6 - 5)^2 + (3 - 0)^2 + (0 - 3)^2 + (1 - 2)^2 + (2 - 1)^2 + (5 - 6)^2)^{1/2} = 40^{1/2}$. Аналогичный результат мо-

жет быть получен на основании теоремы 5.3. Действительно, $CD(CRT_0, CRT_1) = (2^{3-2}(-2^1 - 2^0)^2 + (-2^1 + 2^0)^2 + (+2^1 - 2^0)^2 + (+2^1 + 2^0)^2)^{1/2} = (2(9 + 1 + 1 + 9))^{1/2} = 40^{1/2}$.

$$CD(CRT_k, CRT_l) = \sqrt{2^{m-g} \sum_{t_{k,\alpha} t_{k,\beta} \ldots t_{k,\delta} = 00\ldots0}^{11\ldots1} \left[(t_{k,\alpha} - \overline{t_{k.\alpha}}) 2^{\alpha} + (t_{k,\beta} - \overline{t_{k,\beta}}) 2^{\beta} + \ldots + (t_{k,\delta} - \overline{t_{k,\delta}}) 2^{\delta} \right]^2}. \quad (5.9)$$

В отличие от арифметического расстояния, расстояние Евклида, согласно 5.9, зависит от всех ненулевых компонент вектора отрицаний λ_{m-1}, λ_{m-2}, …, λ_1, λ_0. Нетрудно показать, что, с ростом количества ненулевых компонент вектора отрицаний, а также индекса старшего ненулевого индекса α, значение метрики расстояния $CD(CRT_k, CRT_l)$ возрастает. Действительно, предположив, что количество отрицаний (ненулевых компонент вектора λ_{m-1}, λ_{m-2}, …, λ_1, λ_0) увеличивается на одно отрицание $\lambda_\mu = 1$, получим новые значения слагаемых, которыми являются тестовые наборы $T_{k,i}$ и $T_{l,i}$, в выражении (5.9). В этом случае, для 2^{m-1} тестовых наборов $T_{l,i}$ их значения увеличатся на 2^μ, а для второй их половины, также состоящих из 2^{m-1} наборов, уменьшаться на эту же величину. Таким образом, для любого индекса i, всегда существует индекс j, где $i \neq j \in \{0, 1, 2, \ldots, 2^m - 1\}$, для которых разность тестовых наборов примет значения $T_{k,i} - T_{l,i} + 2^\mu = \Delta + 2^\mu$ и $T_{k,j} - T_{l,j} - 2^\mu = \Delta - 2^\mu$. Учитывая, что $\Delta^2 + \Delta^2 < (\Delta + 2^\mu)^2 + (\Delta - 2^\mu)^2 = \Delta^2 + \Delta^2 + 2^{2\mu+1}$, можно заключить, что с увеличением на одно отрицание $\lambda_\mu = 1$, значения вектора отрицаний, увеличивается сумма квадратов разностей тестовых наборов, и соответственно значение расстояния Евклида.

Приведенный анализ и соответствующие утверждения подтверждаются численными значениями указанных метрик расстояния для всевозможных векторов отрицания λ_2, λ_1, $\lambda_0 \neq 0, 0, 0$ при получении согласно (5.6) теста CRT_l и произвольного исходного теста $CRT_k = a_2 a_1 a_0$ подтверждается данными приведенными в таблице 5.5.

Таблица 5.5 – *Значения арифметического расстояния и расстояния Евклида для* $m = 3$

$CRT_k = a_2 a_1 a_0$	CRT_l						
	$a_2 a_1 \overline{a_0}$	$a_2 \overline{a_1} a_0$	$a_2 \overline{a_1} \overline{a_0}$	$\overline{a_2} a_1 a_0$	$\overline{a_2} a_1 \overline{a_0}$	$\overline{a_2} \overline{a_1} a_0$	$\overline{a_2} \overline{a_1} \overline{a_0}$
$AD(CRT_k, CRT_l)$	8	16	16	32	32	32	32
$CD(CRT_k, CRT_l)$	$\sqrt{8}$	$\sqrt{32}$	$\sqrt{40}$	$\sqrt{128}$	$\sqrt{136}$	$\sqrt{160}$	$\sqrt{168}$

Соотношение (5.9) для расстояния Евклида $CD(CRT_k, CRT_l)$ может быть заметно упрощено, основываясь на следующей теореме.

Теорема 5.4. Расстояние Евклида $CD(CRT_k, CRT_l)$ для тестов CRT_k и CRT_l, где $CRT_k = \{T_{k,0}, T_{k,1}, T_{k,2}, \ldots, T_{k,q-1}\}$ включает $q = 2^m$ m-разрядных, непо-

вторяющихся, сгенерированных случайным образом, тестовых наборов $T_{k,i}$, а тестовые наборы $T_{l,i}$ получены согласно (5.6) на основании вектора отрицаний λ_{m-1}, λ_{m-2}, …, λ_1, λ_0, для которого g значений λ_α, λ_β, …, λ_χ, λ_δ, ($\alpha > \beta > \ldots > \chi > \delta$) равняются 1, вычисляется как:

$$CD(CRT_k, CRT_l) = \sqrt{2^m\left(2^{2\alpha} + 2^{2\beta} + \ldots + 2^{2\chi} + 2^{2\delta}\right)}. \qquad (5.10)$$

Доказательство теоремы 5.4. Выражение

$$\left[(t_{k,\alpha} - \overline{t_{k.\alpha}})2^\alpha + (t_{k,\beta} - \overline{t_{k,\beta}})2^\beta + \ldots + (t_{k,\chi} - \overline{t_{k,\chi}})2^\chi + (t_{k,\delta} - \overline{t_{k,\delta}})2^\delta\right]^2$$

под знаком суммирования в соотношении (5.9) можно представить как два слагаемые a и b, возведенные в квадрат, т.е. $(a+b)^2$. Например, в качестве слагаемого b рассмотрим компоненту с индексами k и δ, тогда $b = (t_{k,\delta} - \overline{t_{k,\delta}})2^\delta$. Оставшаяся часть соотношения (5.9) будет представлять собой слагаемое a. Отметим, что для случая $q = 2^m$ под знаком суммирования будут использованы всевозможные m - разрядные комбинации, соответственно двоичная переменная $t_{k,\delta}$ в половине случаев примет значение 0, а для второй половины – значение 1. Откуда следует, что слагаемое b будет иметь знак плюс для $t_{k,\delta} = 1$, а для $t_{k,\delta} = 0$ знак минус. Тогда:

$$\sum_{t_{k,\alpha}t_{k,\beta}\ldots t_{k,\chi}t_{k,\delta}=00\ldots00}^{11\ldots11}\left[(t_{k,\alpha} - \overline{t_{k,\alpha}})2^\alpha + (t_{k,\beta} - \overline{t_{k,\beta}})2^\beta + \ldots + (t_{k,\chi} - \overline{t_{k,\chi}})2^\chi + (t_{k,\delta} - \overline{t_{k,\delta}})2^\delta\right]^2 =$$

$$= \sum_{t_{k,\alpha}t_{k,\beta}\ldots t_{k,\chi}=00\ldots0}^{11\ldots1}\{\left[(t_{k,\alpha} - \overline{t_{k,\alpha}})2^\alpha + (t_{k,\beta} - \overline{t_{k,\beta}})2^\beta + \ldots + (t_{k,\chi} - \overline{t_{k,\chi}})2^\chi + 2^\delta\right]^2 +$$

$$+ \left[(t_{k,\alpha} - \overline{t_{k,\alpha}})2^\alpha + (t_{k,\beta} - \overline{t_{k,\beta}})2^\beta + \ldots + (t_{k,\chi} - \overline{t_{k,\chi}})2^\chi - 2^\delta\right]^2\}.$$

Используя соотношение $(a + b)^2 + (a - b)^2 = 2(a^2 + b^2)$, получим:

$$2\times \sum_{t_{k,\alpha}t_{k,\beta}\ldots t_{k,\chi}=00\ldots0}^{11\ldots1}\{\left[(t_{k,\alpha} - \overline{t_{k,\alpha}})2^\alpha + (t_{k,\beta} - \overline{t_{k,\beta}})2^\beta + \ldots + (t_{k,\chi} - \overline{t_{k,\chi}})2^\chi\right]^2 + 2^{2\delta}\} = 2^g 2^{2\delta} +$$

$$+ 2\times \sum_{t_{k,\alpha}t_{k,\beta}\ldots t_{k,\chi}=00\ldots0}^{11\ldots1}\left[(t_{k,\alpha} - \overline{t_{k,\alpha}})2^\alpha + (t_{k,\beta} - \overline{t_{k,\beta}})2^\beta + \ldots + (t_{k,\chi} - \overline{t_{k,\chi}})2^\chi\right]^2.$$

Подобным образом, последовательно рассматривая оставшиеся слагаемые под знаком суммы, окончательно получим:

$$CD(CRT_k, CRT_l) = \sqrt{2^m(2^{2\alpha} + 2^{2\beta} + \ldots + 2^{2\chi} + 2^{2\delta})}.$$

Что и требовалось доказать.

Для теоремы 5.4 справедливы следующие следствия.

Следствие 5.1. Численное значение $CD(CRT_k, CRT_l)$, в соответствии с теоремой 5.3, для двоичного вектора $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_{i+1}, \lambda_i, \lambda_{i-1}, \ldots, \lambda_1, \lambda_0 = 0, 0, \ldots, 0, 1, 1, \ldots, 1, 1$ и использовании всевозможных m-разрядных двоичных комбинаций в тестах CRT_l и CRT_k, принимает следующий вид:

$$CD(CRT_k, CRT_l) = \sqrt{2^m(2^{2i} + 2^{2i-2} + \ldots + 2^2 + 2)} = \sqrt{\frac{2^m(2^{2(i+1)} - 1)}{3}}. \qquad (5.11)$$

Максимальное значение $CD(CRT_k, CRT_l)$, в соответствии со следствием 5.1, достигается для двоичного вектора $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 = 1, 1, \ldots, 1, 1$ и принимает вид:

$$CD(CRT_k, CRT_l) == \sqrt{\frac{2^m\left(2^{2m} - 1\right)}{3}}. \qquad (5.12)$$

Отметим, что, в случае максимального значения $CD(CRT_k, CRT_l)$, второй тест CRT_l представляет собой инверсные значения тестовых наборов исходного теста CRT_k. Например, для случая, когда тест $CRT_k = t_{k,2}t_{k,1}t_{k,0} = \{1\ 1\ 1, 1\ 1\ 0, 1\ 0\ 1, 1\ 0\ 0, 0\ 1\ 1, 0\ 1\ 0, 0\ 0\ 1, 000\}$, а тест $CRT_l = \overline{t_{l.2}}\,\overline{t_{l,1}}\,\overline{t_{l,0}} = \{0\ 0\ 0, 0\ 0\ 1, 0\ 1\ 0, 0\ 1\ 1, 1\ 0\ 0, 1\ 0\ 1, 1\ 1\ 0, 1\ 1\ 1\}$. Количество тестовых наборов в CRT_k и CRT_l равняется $q = 2^m = 2^3 = 8$. Из таблицы 5.5 и соотношения (5.12) следует, что $CD(CRT_k, CRT_l) = \sqrt{168}$. Применив соотношения (5.9), получим аналогичный результат $CD(CRT_k, CRT_l) = ((7-0)^2 + (6-1)^2 + (5-2)^2 + (4-3)^2 + (3-4)^2 + (2-5)^2 + (1-6)^2 + (0-7)^2)^{1/2} = \sqrt{168}$.

Следствие 5.2. Численное значение $CD(CRT_k, CRT_l)$, в соответствии с теоремой 5.3, для двоичного вектора масок $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_{i+1}, \lambda_i, \lambda_{i-1}, \ldots, \lambda_1, \lambda_0 = 0, 0, \ldots, 0, 1, 0, \ldots, 0, 0$ и использовании всевозможных m-разрядных двоичных комбинаций в тестах CRT_l и CRT_k, в соответствии с (5.6) принимает следующий вид:

$$CD(CRT_k, CRT_l) = \sqrt{2^{2i+m}}. \qquad (5.13)$$

Минимальное значение $CD(CRT_k, CRT_l)$ в соответствии с теоремой 5.4 достигается для двоичного вектора масок $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 = 0, 0, \ldots, 0, 1$ и равняется $\sqrt{2^m}$. Для соотношения (5.13), в отличие от (5.11) и (5.12), справедливо обобщение для произвольного значения $q \le 2^m$.

$$ED(CRT_k, CRT_l) = \sqrt{q2^{2i}}. \qquad (5.14)$$

Действительно, для тестов CRT_0, CRT_1 и CRT_2 согласно (5.14) получим, соответственно $CD(CRT_0, CRT_1) = \sqrt{q2^{2i}} = \sqrt{5 \times 2^{2\times 0}} = \sqrt{5}$ и

$$CD(CRT_0, CRT_2) = \sqrt{q2^{2i}} = \sqrt{5 \times 2^{2\times 3}} = \sqrt{320}.$$

Отметим, что выражения (5.10) – (5.13) справедливы для $q = 2^m$ и могут быть использованы как среднее значение для произвольного значения $q < 2^m$.

Таким образом, абсолютное значение $CD(CRT_k, CRT_l)$ возрастает с ростом как индекса α старшей ненулевой компоненты и последующих компонент вектора отрицаний $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$, так и общего количества его ненулевых компонент при получении CRT_l на основании CRT_k.

Как отмечалось ранее, суммарное покомпонентное хэммингово расстояние $THD(CRT_k, CRT_l)$ для двух управляемых вероятностных тестов CRT_k и CRT_l возрастает с ростом числа ненулевых компонент вектора отрицаний, используемого для получения CRT_l на основании (5.6).

С учетом последних рассуждений целесообразным представляется применение арифметического расстояния $AD(CRT_k, CRT_l)$ и расстояния Евклида $CD(CRT_k, CRT_l)$ для целей построения многократных управляемых вероятностных тестов. Совместное применение суммарного покомпонентного хэммингового расстояния $THD(CRT_k, CRT_l)$, которое также не требует сложных вычислений, позволяет одновременно максимизировать и расстояние Евклида.

5.3. Многократные управляемые вероятностные тесты

Одним из первых примеров многократных тестов являются многократные маршевые тесты, использованные для тестирования запоминающих устройств [331, 333, 342]. При реализации многократных маршевых тестов запоминающих устройств в качестве меры отличия адресных последовательностей использовалось арифметическое расстояние, позволяющее определить набор адресных последовательностей (тестов), имеющих максимальное отличие между собой [331, 333].

В [333] показано, что чем старше инверсный бит исходной адресной последовательности при формировании второй адресной последовательности согласно (5.6), тем больше арифметическое расстояние и тем, соответственно, больше полнота покрытия теста. В тоже время отмечается, что при равенстве значений арифметического расстояния наблюдается незначительное, но все-таки отличие покрывающей способности теста (его эффективности), которая однозначно зависит от другой метрики, а именно Евклидова расстояния $CD(CRT_0, CRT_1)$. Как показано в [331, 333], при использовании для второго теста CRT_1 двоичного вектора $\lambda_5, \lambda_4, \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 0, 1, 1, 1, 1, 1$, суммарное значение полноты покрытия сложных неисправностей равняется 12,07%. Использование вектора $\lambda_5, \lambda_4, \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 0, 1, 0, 0, 0, 0$ характеризуется полнотой покрытия 10,67%. Для первого и второго случев арифметическое расстояние принимает одинаковое значение, равное 2^{6+4}, а $CD(CRT_0,$

CRT_1) принимает различные значения, в первом случае оно равняется 147,73, а во втором – 128,00.

Как видно из полученных в [333] экспериментальных результатов для случая многократных маршевых тестов, метрика $CD(CRT_0, CRT_1)$ позволяет определять вид двоичного вектора $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$, при котором достигается максимальная эффективность многократных маршевых тестов запоминающих устройств.

Соотношение (5.6) применяемое для формирования многократных тестов, основано на использовании двоичных векторов масок $M = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$; $\lambda_i \in \{0, 1\}$, $i \in \{0, 1, 2, \ldots, m-1\}$. Так, если для получения теста CRT_1, на основании теста CRT_0, используется вектор $M(0, 1) = \lambda_{m-1}(0, 1), \lambda_{m-2}(0, 1), \ldots, \lambda_1(0, 1), \lambda_0(0, 1)$, тогда для получения CRT_2 на основании CRT_0 будет использоваться другой вектор масок $M(0, 2) = \lambda_{m-1}(0, 2), \lambda_{m-2}(0, 2), \ldots, \lambda_1(0, 2), \lambda_0(0, 2)$. Таким образом, для всех последующих тестов многократного теста $MCRT_r = \{CRT_0, CRT_1, CRT_2, \ldots, CRT_{r-1}\}$ используется множество масок $\{M(0, 1), M(0, 2), M(0, 3), \ldots, M(0, r-1)\}$.

Используя свойство $\overline{\overline{\lambda_i}} = \lambda_i$ двоичной операции отрицания, применяемой для метода формирования многократных тестов (5.6), сформулируем ряд утверждений.

Утверждение 5.1. Если тест CRT_l получен из исходного теста CRT_k, на основании двоичного вектора масок $M(k, l) = \lambda_{m-1}(k, l), \lambda_{m-2}(k, l), \ldots, \lambda_1(k, l), \lambda_0(k, l)$, в соответствии с (5.6), то используя этот же вектор масок $M(k, l)$, а в качестве исходного теста CRT_l, на основании того же алгоритма (5.6), будет получен тест CRT_k.

Формально это утверждение описывается равенством $M(k, l) = M(l, k)$.

Утверждение 5.2. Если тест CRT_l сформирован из исходного теста CRT_k на основании двоичного вектора масок $M(k, l) = \lambda_{m-1}(k, l), \lambda_{m-2}(k, l), \ldots, \lambda_1(k, l), \lambda_0(k, l)$, а тест CRT_j получен из CRT_k на основании маски $M(k, j) = \lambda_{m-1}(k, j), \lambda_{m-2}(k, j), \ldots, \lambda_1(k, j), \lambda_0(k, j)$, в соответствии с (5.6), тогда тест CRT_j может быть получен на основании теста CRT_l, используя двоичный вектор масок $M(j, l) = M(k, j) \oplus M(k, l) = \lambda_{m-1}(k, j) \oplus \lambda_{m-1}(k, l), \lambda_{m-2}(k, j) \oplus \lambda_{m-2}(k, l), \ldots, \lambda_1(k, j) \oplus \lambda_1(k, l), \lambda_0(k, j) \oplus \lambda_0(k, l)$.

Пример 5.6. Если тест CRT_1 получен на основании теста CRT_0 и вектора маски $M(0, 1) = \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 0, 0, 0, 1$, а тест CRT_2 на базе CRT_0 с использованием вектора маски $M(0, 2) = \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 1, 0, 0,0$. Согласно утверждению 5.2, тест CRT_2 на основании CRT_1 может быть получен в соответствии с (5.6) при использовании вектора $M(1,2) = \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 1 \oplus 0, 0 \oplus 0, 0 \oplus 0, 0 \oplus 1 = 1, 0, 0, 1$. Справедливо и обратное (см. утверждение 5.1), тест CRT_1 на основании CRT_2 может быть получен при использовании $M(2, 1) = \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 0 \oplus 1, 0 \oplus 0, 0 \oplus 0, 1 \oplus 0 = 1, 0, 0, 1$.

Следующее утверждение сформулируем в виде теоремы.

Теорема 5.5. Если тест CRT_l сформирован из исходного теста CRT_k на основании двоичного вектора масок $M(k, l)$, а тест CRT_j получен из CRT_k на основании $M(k, j) > M(k, l)$ в соответствии с (5.6), то выполняется неравенство $CD(CRT_k, CRT_j) > CD(CRT_k, CRT_l)$.

Доказательство теоремы 5.5. Двоичный унитарный вектор масок $\lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$, где $\lambda_i \in \{0, 1\}$, $i \in \{0, 1, 2, \ldots, m-1\}$, применяемый в методе генерирования управляемых вероятностных тестов (5.6), можно интерпретировать как числовое значение, представленное в позиционном коде как $M = \lambda_{m-1} \times 2^{m-1} + \lambda_{m-2} \times 2^{m-2} + \ldots + \lambda_1 \times 2^1 + \lambda_0 \times 2^0$. Тогда, $M(k, l) = \lambda_{m-1}(k, l), \lambda_{m-2}(k, l), \ldots, \lambda_1(k, l), \lambda_0(k, l)$ можно записать в виде числа $M(k, l) = \lambda_{m-1}(k, l) \times 2^{m-1} + \lambda_{m-2}(k, l) \times 2^{m-2} + \ldots + \lambda_{i-1}(k, l) \times 2^{i-1} + \lambda_i(k, l) \times 2^i + \lambda_{i+1}(k, l) \times 2^{i+1} + \ldots + \lambda_1(k, l) \times 2^1 + \lambda_0(k, l) \times 2^0$, а $M(k, j) = \lambda_{m-1}(k, j), \lambda_{m-2}(k, j), \ldots, \lambda_1(k, j), \lambda_0(k, j)$ как число $M(k, j) = \lambda_{m-1}(k, j) \times 2^{m-1} + \lambda_{m-2}(k, j) \times 2^{m-2} + \ldots + \lambda_{i-1}(k, j) \times 2^{i-1} + \lambda_i(k, j) \times 2^i + \lambda_{i+1}(k, j) \times 2^{i+1} + \ldots + \lambda_1(k, j) \times 2^1 + \lambda_0(k, j) \times 2^0$. Отметим, что $M(k, l)$ используется в (5.6) для получения CRT_l на базе исходного теста CRT_k, а $M(k, j)$ для получения теста CRT_j при использовании того же исходного теста CRT_k. На основании неравенства $M(k, j) > M(k, l)$, можно заключить, что всегда существует такое $i \in \{0, 1, 2, \ldots, m-1\}$ для которого $\lambda_i(k, j) = 1$, а $\lambda_i(k, l) = 0$, кроме того $\lambda_c(k, j) = \lambda_c(k, l)$ для $c \in \{i+1, i+2, \ldots, m-1\}$. Остальные разряды $\lambda_e(k, l)$ и $\lambda_e(k, j)$, $e \in \{0, 1, 2, \ldots, i-1\}$ масок $\Lambda(k, l)$ и $\Lambda(k, j)$ принимают произвольные значения. Максимально близкое числовое значение $M(k, l)$ к значению $M(k, j)$ достигается для случая, когда $\lambda_e(k, j) = 0$, а $\lambda_e(k, l) = 1$ для всех e. Тогда разность $M(k, j) - M(k, l)$ принимает минимальное значение, равное $2^0 = 1$. Примером, когда $M(k, j) > M(k, l)$ могут быть значения масок $M(k, j) = \lambda_6, \lambda_5, \lambda_4, \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 0, 1, 0, 1, 0, 0, 0$ и $M(k, l) = \lambda_6, \lambda_5, \lambda_4, \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 0, 1, 0, 0, 1, 1, 1$, где $i = 3$, так как $\lambda_3(k, j) = 1$, а $\lambda_3(k, l) = 0$, а также $\lambda_c(k, j) = \lambda_c(k, l)$ для $c \in \{4, 5, 6\}$ и $\lambda_e(k, j) = 0$, а $\lambda_e(k, l) = 1$ для $e \in \{0, 1, 2\}$.

Определим расстояния Евклида $CD(CRT_k, CRT_j)$ и $CD(CRT_k, CRT_l)$ для тестов CRT_j и CRT_l по отношению к CRT_k. Рассмотрим случай, когда вектора масок $M(k, j)$ и $M(k, l)$), для которых выполняется неравенство $M(k, j) > M(k, l)$, принимают максимально близкие численные значения, т.е. когда $\lambda_c(k, j) = \lambda_c(k, l)$ для $c \in \{i+1, i+2, \ldots, m-1\}$, $\lambda_i(k, j) = 1$, а $\lambda_i(k, l) = 0$, а также $\lambda_e(k, j) = 0$, а $\lambda_e(k, l) = 1$ для $e \in \{0, 1, 2, \ldots, i-1\}$. Предположим, что среди $m - i - 1$ старших разрядов векторов масок $M(k, j)$ и $M(k, l)$ только $\lambda_\alpha, \lambda_\beta, \ldots, \lambda_\delta$, ($\alpha > \beta > \ldots > \delta$) принимают единичное значение тогда согласно (5.10) получим, что:

$$CD(CRT_k, CRT_j) = \sqrt{2^m(2^{2\alpha} + 2^{2\beta} + \ldots + 2^{2\delta} + 2^{2i})};$$
$$CD(CRT_k, CRT_l) = \sqrt{2^m\left(2^{2\alpha} + 2^{2\beta} + \ldots + 2^{2\delta} + \sum_{n=0}^{i-1} 2^{2n}\right)}. \qquad (5.15)$$

Учитывая соотношение $\sum_{n=0}^{i-1} 2^{2n} = \frac{2^{2i}-1}{3} < 2^{2i}$, которое использовалось при получении выражения (5.11) следствия 5.1, можно заключить, что для приведенного наихудшего соотношения векторов масок $M(k, j)$ и $M(k, l)$, когда $M(k, j) = M(k, l) + 1$ выполняется неравенство $CD(CRT_k, CRT_j) > CD(CRT_k, CRT_l)$. В остальных случаях, когда $M(k, j)$ больше чем на единицу превышает численное значение $M(k, l)$, аналогичным образом показывается, что $CD(CRT_k, CRT_j) > CD(CRT_k, CRT_l)$. Что и требовалось доказать.

Численные значения $CD(CRT_k, CRT_j)$ для $m = 3$ (см. таблицу 5.5) и других m подтверждают результаты теоремы 5.5. Действительно, чем больше значение $M(k, l) = \lambda_2, \lambda_1, \lambda_0 = 1, 1, 1$, использованное для получения CRT_l согласно (5.6), тем больше расстояние Евклида $CD(CRT_k, CRT_l) = 12{,}96$.

Важным следствием теоремы 5.5 является возможность использования вектора масок $M = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$ в качестве метрики, позволяющей строить многократные управляемые вероятностные тесты с использованием соотношения (5.6).

Последовательно рассмотрим многократные управляемые вероятностные тесты $MCRT_r$ различной кратности, начиная с двукратных тестов $MCRT_2$ состоящих из CRT_0 и CRT_1, где CRT_1, формируется на основании CRT_0 согласно (5.6).

В случае двукратных тестов максимальное значение $M(0, 1)$ достигается для двоичного вектора масок $M = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 = 1, 1, \ldots, 1, 1$, представляющего собой максимальное числовое значение m-разрядного двоичного вектора масок. Отметим, что в данном случае второй тест CRT_1 представляет собой инверсные значения тестовых наборов исходного теста CRT_0. Например, для $m = 4$ значение $M(0, 1)$ равняется 15 и соответственно $CD(CRT_0, CRT_1) = 36{,}68$, что равно максимальному значению.

В случае трехкратного вероятностного теста $MCRT_3$ для получения второго CRT_1 и третьего CRT_2 теста на основании исходного теста CRT_0 необходимо использовать оптимальные сочетания двоичных векторов $M = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$ руководствуясь теоремой 5.4. Используя $M(0, 1) = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 = 1, 1, \ldots, 1, 1$ для получения CRT_1 на основании CRT_0, получим максимально возможное расстояние $CD(CRT_0, CRT_1)$ между первым и вторым тестами. Для формирования управляемого теста CRT_2 на основании CRT_0, согласно (5.6), в соответствии с теоремой 5.5, необходимо максимизировать расстояние теста CRT_2 от CRT_0 и CRT_1. Это означает, что вектор маски $M(0, 1) = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$ используемый для получения CRT_1 на основании CRT_0 и векторы масок $M(0, 2)$ и $M(1, 2)$, определяющие расстояния CRT_2 от CRT_0 и CRT_1 одновременно принимали максимально возможные значения. То есть, чтобы минимальное числовое значение для элементов множества $\{M(0, 1), M(0, 2), M(1, 2)\}$ было максимально возможным. Отметим, что согласно утверждению 5.2 $M(1, 2) = M(0, 1) \oplus M(0, 2)$. С учетом того, что $M(0, 1) = 1$,

1, …, 1, 1, возможны два варианта значений для $M(0, 2)$, а именно $M(0, 2) = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 = 0, 1, \ldots, 1, 1$, либо $M(0, 2) = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 = 1, 0, \ldots, 0, 0$. В первом случае, $M(1, 2) = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 = 1, 0, \ldots, 0, 0$, а во втором $M(1, 2) = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0 = 0, 1, \ldots, 1, 1$. Для примера, когда $m = 4$ множество масок имеет вид $\{M(0, 1), M(0, 2), M(1, 2)\} = \{1\ 1\ 1\ 1, 1\ 0\ 0\ 0, 0\ 1\ 1\ 1\} = \{15, 8, 7\}$ и соответственно минимальное численное значение элементов множества $\{M(0, 1), M(0, 2), M(1, 2)\}$ принимает максимально возможное значение равное 0111 (7). Анализ результатов, для случая $m = 4$, показывает, что для $M(0, 1) = 1\ 1\ 1\ 1$ и $M(0,2) = 1\ 0\ 0\ 0$, соотношение значений расстояний Евклида $CD(CRT_0, CRT_1) = 36{,}68$, $CD(CRT_0, CRT_2) = 32$ и $CD(CRT_1, CRT_2) = 18{,}33$ обеспечивают максимальную удаленность каждого из тестов CRT_0, CRT_1 и CRT_2 друг от друга, причем минимальная удаленность в терминах расстояния Евклида равняется 18,33. При других сочетаниях значений $M(0, 1)$ и $M(0, 2)$ кроме $M(0, 1) = 1\ 1\ 1\ 1$ и $M(0, 2) = 0\ 1\ 1\ 1$ минимальное значение расстояния Евклида всегда будет меньше чем 18,33.

Обобщая приведенный анализ для случая $MCRT_3$, можно заключить, что оптимальным сочетанием векторов $M(0, 1)$ и $M(0, 2)$ для получения CRT_1 и CRT_2 согласно (5.6) являются два возможных набора (1 1 1 … 1 1 1, 0 1 1 … 1 1 1) либо (1 1 1 … 1 1 1, 1 0 0 … 0 0 0).

Для случая четырехкратных вероятностных тестов $MCRT_4$ набор двоичных векторов $\{M(0, 1), M(0, 2), M(0, 3)\}$ обеспечивающих максимальные значения минимального расстояния Евклида $CD(CRT_k, CRT_l)$ между любыми двумя тестами из четырех CRT_0, CRT_1, CRT_2 и CRT_3 имеет вид {1 1 1 … 1 1 1, 0 1 1 … 1 1 1, 1 0 0 … 0 0 0}. Так, для примера, когда $m = 4$ набор масок $\{M(0, 1), M(0, 2), M(0, 3), M(1, 2), M(1, 3), M(2, 3)\} = \{1\ 1\ 1\ 1, 0\ 1\ 1\ 1, 1\ 0\ 0\ 0, 1\ 0\ 0\ 0, 0\ 1\ 1\ 1, 1\ 1\ 1\ 1\}$, а соответствующие расстояния Евклида больше либо равняются значению 18,33.

Для общего случая многократных управляемых вероятностных тестов $MCRT_r$ оптимальным сочетанием набора двоичных векторов масок $M = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$ для тестов CRT_0, CRT_1, CRT_2, …, CRT_r, будут вектора сформированные следующим образом. Первые старшие двоичные значения векторов масок $M = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$, состоящие из $\lceil \log_2 r \rceil$ бит вектора соответствуют двоичному коду индексу i, $i \in \{1, 2, 3, \ldots, r-1\}$ теста CRT_i, а остальные повторяют значение младшего бита кода индекса i. Например, для многократного теста $MCRT_{11}$, включающего вероятностные тесты CRT_0, CRT_1, CRT_2, CRT_3, …, CRT_{10} их двоичные векторы принимают вид: 0 0 0 ***0*** 0 0 … 0 0 0, 0 0 0 ***1*** 1 1 … 1 1 1, 0 0 1 ***0*** 0 0 … 0 0 0, 0 0 1 ***1*** 1 1 … 1 1 1, …, 1 0 1 ***0*** 0 0 … 0 0 0. В данном случае $\lceil \log_2 r \rceil = \lceil \log_2 11 \rceil = 4$.

В качестве меры эффективности многократных управляемых вероятностных тестов $MCRT_r$ используем метрику $E(k, 2^m)$, введенную в [334], для формирования очередных тестовых наборов для однократного управляемого

вероятностного теста. Данная метрика подробно исследована в предыдущем разделе и соответствует определениям 4.4 и 4.5.

В случае многократных тестов подобная характеристика применяется для очередного теста CRT_j, и может быть определена следующим образом.

Определение 5.2. Мерой эффективности $E(k, 2^m)$ для очередного управляемого теста CRT_j является дополнительное количество двоичных комбинаций на всевозможных k из 2^m разрядах генерируемых тестовыми наборами теста CRT_j по отношению к множеству k из 2^m двоичных комбинаций сгенерированные предыдущими тестами CRT_0, CRT_1, CRT_2, …, CRT_{j-1} многократного теста $MCRT_r$.

Очевидно, что чем больше значение данной метрики, тем более эффективным является очередной управляемый тест CRT_j, который в совокупности с предыдущими тестами позволяет достичь максимальной эффективности. Отметим, что в предыдущих разделах было показано, что для достижения максимальной эффективности многократных управляемых вероятностных тестов $MCRT_r$ необходимо, чтобы расстояние Евклида для теста CRT_j было максимальным по отношению к ранее сформированным тестам CRT_0, CRT_1, CRT_2, …, CRT_{j-1}, что обеспечивается максимизацией двоичного вектора масок $M = \lambda_{m-1}, \lambda_{m-2}, \ldots, \lambda_1, \lambda_0$.

Для сопоставительного анализа эффективности рассмотренных многократных управляемых вероятностных тестов $MCRT_r$, используем задачу тестирования запоминающих устройств. Первоначально рассмотрим запоминающее устройство, состоящее из $2^3 = 8$ ячеек, для которого применим в качестве первого теста тест CRT_0 состоящий из формирования всевозможных адресов $a_2a_1a_0$ = {0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1} запоминающего устройства по схеме простейших маршевых тестов [331]. При формировании очередного адреса первоначальное нулевое состояние ячейки памяти изменяется на единичное значение. Таким образом, начальное нулевое состояние всех ячеек запоминающего устройства изменяется на единичное состояние. Отметим, что значения $CD(CRT_0, CRT_1)$ для двух тестов CRT_0, CRT_1 и m=3 приведены в таблице 5.5. В качестве второго управляемого вероятностного теста CRT_1 используем тест, полученный согласно (5.6) для всевозможных значений двоичных векторов $\lambda_2, \lambda_1, \lambda_0 \neq 0, 0, 0$. Результирующие значения метрики $E(k, 2^m)$ для двукратного теста, состоящего из тестов $CRT_0 = a_2a_1a_0$ и $CRT_1 = (a_2 \oplus \lambda_2)(a_1 \oplus \lambda_1)(a_0 \oplus \lambda_0)$, приведены в таблице 5.6.

Таблица 5.6. *Оценка эффективности двукратного теста для запоминающего устройства, состоящего из 8 запоминающих ячеек*

$\lambda_2, \lambda_1, \lambda_0$	$E(k, 2^m)$ - дополнительное количество двоичных комбинаций			
	$E(3, 8)$	$E(4, 8)$	$E(5, 8)$	$E(6, 8)$
0 0 1	24	60	80	60

0 1 0	48	118	152	108
0 1 1	64	142	162	112
1 0 0	96	204	224	140
1 0 1	104	208	224	140
1 1 0	112	210	224	140
1 1 1	112	210	224	140

Как видно из приведенных численных значений эффективность двукратного теста находится в строгом соответствии со значением $CD(CRT_0, CRT_1)$, приведенным в таблице 5.5. Действительно, для $\lambda_2, \lambda_1, \lambda_0 = 0, 0, 1$ расстояние Евклида между CRT_0 и CRT_1 равняется $\sqrt{8}$ (см. таблицу 5.5) и количество дополнительных двоичных комбинаций равняется 24. В тоже время для $\lambda_2, \lambda_1, \lambda_0 = 1, 1, 1$ и соответственно максимального значения $CD(CRT_0, CRT_1) = \sqrt{168}$ число дополнительных комбинаций равняется 112.

Аналогичные результаты, подтверждающие работоспособность расстояния Евклида в качестве меры эффективности многократного теста, для запоминающего устройства, состоящего из 16 ячеек, приведены в таблице 5.7.

Таблица 5.7. *Оценка эффективности двукратного теста запоминающего устройства состоящего из 16 запоминающих ячеек*

$\lambda_2, \lambda_2, \lambda_1, \lambda_0$	$E(k, 2^m)$ - дополнительное количество двоичных комбинаций			
	$E(3, 8)$	$E(4, 8)$	$E(5, 8)$	$E(6, 8)$
0 0 0 0	2240	9100	26208	56056
0 0 0 1	112	728	2912	8008
0 0 1 0	224	1452	5776	15752
0 0 1 1	320	1980	7536	19712
0 1 0 0	448	2840	10880	28280
0 1 0 1	528	3200	11840	29960
0 1 1 0	608	3556	12768	31528
0 1 1 1	672	3780	13216	32088
1 0 0 0	896	5040	17024	39760
1 0 0 1	944	5160	17184	39880
1 0 1 0	992	5276	17328	39976
1 0 1 1	1024	5324	17360	39984
1 1 0 0	1088	5448	17472	40040
1 1 0 1	1104	5456	17472	40040
1 1 1 0	1120	5460	17472	40040
1 1 1 1	1120	5460	17472	40040

В графическом виде данное утверждение подтверждается и для запоминающего устройства емкостью 32 бита. На рис. 5.1 приведено значение метрики $E(k, 2^m)$ для двукратного теста. Как видно из приведенного рисунка

максимальная эффективность двукратного теста состоящего из CRT_0 и CRT_1 достигается для двоичного вектора $M(0, 1) = \lambda_4, \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 1, 1, 1, 1, 1$.

Отметим, что в этом случае и расстояние Евклида $CD(CRT_0, CRT_1)$, принимает максимальное значение равное $\sqrt{1360}$. В тоже время для вектора $M(0, 1) = \lambda_4\ \lambda_3\ \lambda_2\ \lambda_1\ \lambda_0 = 0\ 0\ 0\ 0\ 1$ имеем минимальную эффективность двукратного теста, о чем и свидетельствует минимальное значение $CD(CRT_0, CRT_1)$ равное $\sqrt{32}$. Графически видно существенное отличие эффективности двукратного теста для двух различных векторов $M(0, 1) = \lambda_4, \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 0, 1, 1, 1, 1$ и $M(0, 1) = \lambda_4, \lambda_3, \lambda_2, \lambda_1, \lambda_0 = 1, 0, 0, 0, 0$. Этот факт подтверждается значениями расстояния Евклида $CD(CRT_0, CRT_1)$. Действительно в первом случае при $M(0, 1) = 01111$ имеем $CD(CRT_0, CRT_1) = \sqrt{336}$, а во втором $CD(CRT_0, CRT_1) = \sqrt{1024}$, что соответствует результатам, приведенным на рис. 5.1.

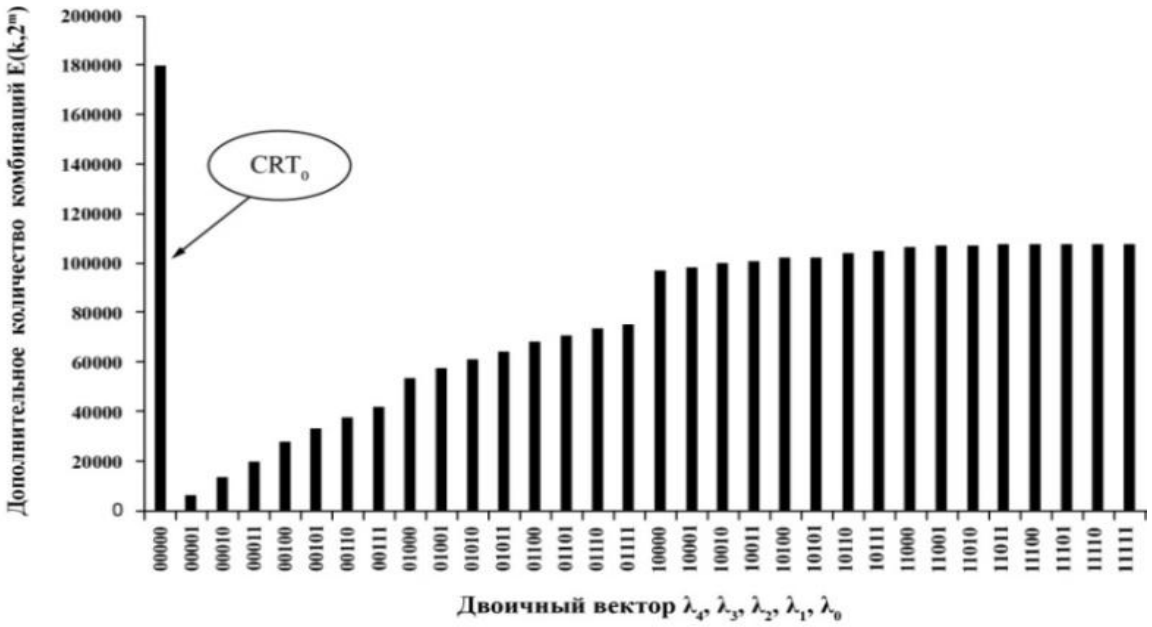


Рис. 5.1. *Количество дополнительных двоичных комбинаций* $E(k,2^4)$ *формируемых тестом* CRT_1 *на произвольных* k=5 *из* 32 *ячеек запоминающего устройства.*

Рассмотренный метод построения многократных тестов может рассматриваться как альтернатива классическим управляемым вероятностным тестам. Показано, что используя исходный вероятностный тест, без существенных вычислительных затрат, можно сформировать его модификации как последующие тесты многократного теста. Подтверждена применимость расстояния Евклида для целей построения многократных тестов. Эффективность предложенных решений согласуется с экспериментальными результа-

тами, приведенными в заключительной части для случая многократных тестов запоминающих устройств.

5.4. Построение многократных управляемых вероятностных тестов

В качестве основной операции, используемой для построения многократных вероятностных тестов $MCRT_r$, применим операцию сложения, что позволит обеспечить минимальную вычислительную сложность при их формировании. Действительно, все последующие тесты CRT_0, CRT_1…, CRT_{r-1} могут быть легко сформированы на базе CRT_0 путем однократного применения для каждого тестового набора операции сложения.

Согласно определению 4.3, управляемый вероятностный тест CRT состоит из q тестовых наборов T_i, $i \in \{0, 1, 2, ..., q-1\}$, каждый из которых представляет собой m-разрядный двоичный вектор $T_i = t_{i,m-1}, t_{i,m-2}, \ldots, t_{i,2}, t_{i,1}, t_{i,0}$, где $t_{i,l} \in \{0,1\}$. Таким образом, тестовые наборы T_i управляемого вероятностного теста CRT могут интерпретироваться как $g = 2^m$-ичные данные $T_i \in \{0, 1, 2, ..., 2^m - 1\}$. Тогда, например, тест CRT = {0 0 1 1, 0 1 1 0, 1 1 0 0, 0 1 0 1, 1 0 0 0} можно представить как набор 16-ичных данных CRT = {3, 6, 12, 5, 8}, приведенных в десятичной системе счисления. В случае исходного теста $CRT_k = \{T_0(k), T_1(k), T_2(k), \ldots, T_{q-1}(k)\}$ соотношение для получения нового теста $CRT_l = \{T_0(l), T_1(l), T_2(l), \ldots, T_{q-1}(l)\}$ примет вид.

$$T_i(l) = T_i(k) + d \mod 2^m; \quad i = \overline{0, q-1}. \tag{5.16}$$

В приведенном соотношении параметр $d \in \{1, 2, 3, ..., 2^m - 1\}$ используется для достижения различия между тестовыми наборами и, соответственно, тестами CRT_l и CRT_k. Значение этого параметра является определяющим для достижения максимального отличия теста CRT_l от теста CRT_k в терминах определенных ранее метрик. Для соотношения (5.16) справедливо следующее утверждение.

Утверждение 5.3. Если тест CRT_l получен из исходного теста CRT_k на основании соотношения (5.16) для параметра $d \in \{1, 2, 3, ..., 2^m - 1\}$, то используя для теста CRT_l в качестве параметра значение $2^m - d$ и, применяя тоже соотношение (5.16), будет получен исходный тест CRT_k. Это утверждение вытекает из равенства $d + 2^m - d \mod 2^m = 0$.

Пример 5.7. При $m = 4$ для исходного теста CRT_k = {3, 6, 12, 5, 8} и параметра $d = 8$, согласно (5.16), получим CRT_l = {11, 14, 4, 13, 0}. Используя в качестве исходного теста CRT_l = {11, 14, 4, 13, 0} и тоже значение $d = 8$ получим тест CRT_k = {3, 6, 12, 5, 8}, что соответствует утверждению 5.3. Для того же исходного теста CRT_k = {3, 6, 12, 5, 8} и другого параметра $d = 5$ будем иметь иной результат, а именно CRT_l = {8, 11, 1, 10, 13}.

Пример 5.8. Для теста CRT_k = {3, 7, 0, 6, 2, 5, 1, 4}, построенного для $m = 3$ и параметра $d = 4$, согласно (5.16) получим что CRT_l = {7, 3, 4, 2, 6, 1, 5,

0}. Для того же исходного теста, и параметра d = 5 будем иметь другой результат CRT_l = {0, 4, 5, 3, 7, 2, 6, 1}. Отметим, что в данном примере приведенные тесты включают всевозможные восьмеричные значения данных.

Анализируя приведенные примеры, в каждом из которых представлено по два новых теста, полученных в соответствии с (5.16), возникает вопрос, какой из этих двух тестов является более эффективным для целей многократного тестирования. Таким образом, стоит задача определения оптимального значения параметра d, при использовании, в качестве метрики качества для многократных тестов, расстояния Евклида. Первоначально для $CD(CRT_k, CRT_l)$, где тест CRT_l получен согласно (5.16), докажем теорему.

Теорема 5.6. Расстояние Евклида $CD(CRT_k, CRT_l)$ для тестов CRT_k и CRT_l, где $CRT_k = \{T_0(k), T_1(k), T_2(k), \ldots, T_{q-1}(k)\}$ состоит из $q=2^m$ m-разрядных, неповторяющихся, сгенерированных случайным образом тестовых наборов $T_i(k)\in\{0, 1, 2, ..., 2^m-1\}$, а тестовые наборы $T_i(l)$ получены согласно $T_i(l) = T_i(k) + d \mod 2^m;\quad i = \overline{0, q-1}$, вычисляется как

$$CD(CRT_k, CRT_l) = \sqrt{2^m d(2^m - d)}. \qquad (5.17)$$

Доказательство теоремы 5.6. Выражение для расстояния Евклида $CD(CRT(k), CRT(l))$ принимает вид

$$CD(CRT_k, CRT_l) = \sqrt{\sum_{i=0}^{2^m-1}\left[T_i(k) - T_i(l)\right]^2} = \sqrt{\sum_{i=0}^{2^m-1}\left[T_i(k) - (T_i(k) + d \mod 2^m)\right]^2}.$$

Учитывая тот факт, что тестовые наборы $T_i(k)$ состоят из $q = 2^m$ m-разрядных, неповторяющихся данных {0, 1, 2, ..., $2^m - 1$}, можно сделать следующие заключения. Значения $T_i(l)$ согласно (5.16) в $2^m - d$ случаях будут принимать вид $T_i(l) = T_i(k) + d$. Как видно из примера 5.8, для d = 5 в $2^m - d = 2^3 - 5 = 3$ случаях $T_i(l) = T_i(k) + 5$, а именно, для $T_i(k) = \{0, 1, 2\}$. Кроме того, значения $T_i(l)$ согласно (5.16) в d случаях будут принимать вид $T_i(l) = T_i(k) + 2^m - d$. С учетом приведенных соотношений выражение для расстояния Евклида примет вид.

$$CD(CRT_k, CRT_l) = \sqrt{\sum_{i=0}^{2^m-d-1} d^2 + \sum_{i=0}^{d-1}(2^m - d)^2} = \sqrt{(2^m - d)d^2 + d(2^m - d)^2} = \sqrt{2^m d(2^m - d)}.$$

Что и требовалось доказать.

Пример 5.9. Расстояние Евклида для тестов CRT_k = {3, 7, 0, 6, 2, 5, 1, 4} и CRT_l = {7, 3, 4, 2, 6, 1, 5, 0} из примера 5.8 определяется как $CD(CRT_k, CRT_l) = [(3 - 7)^2 + (7 - 3)^2 + (0 - 4)^2 + (6 - 2)^2 + (2 - 6)^2 + (5 - 1)^2 + (1 - 5)^2 + (4 - 0)^2]^{1/2} = \sqrt{128}$. Такое же значение может быть получено на основании теоремы 5.6 $\sqrt{2^3 \times 4 \times (2^3 - 4)} = \sqrt{128}$.

Значения расстояний Евклида для случая $m = 3$ и всевозможных значений d приведены в таблице 5.8.

Таблица 5.8. *Значения расстояний Евклида для* $m = 3$

d	1	2	3	4	5	6	7
$CD(CRT_k, CRT_l)$	$\sqrt{56}$	$\sqrt{96}$	$\sqrt{120}$	$\sqrt{128}$	$\sqrt{120}$	$\sqrt{96}$	$\sqrt{56}$

Для приведенной теоремы 5.6 справедливы следующие следствия.

Следствие 5.3. Значение расстояния Евклида $CD(CRT_k, CRT_l)$ принимает максимальное значение при $d = 2^{m-1}$, что соответствует решению уравнения

$$\frac{\partial\sqrt{2^m d(2^m - d)}}{\partial d} = 0.$$

Справедливость данного следствия подтверждается результатами, приведенными в таблице 5.8, где для $d = 2^{m-1} = 2^{3-1} = 4$ расстояние Евклида принимает максимальное значение равное $\sqrt{128}$.

Следствие 5.4. Расстояние Евклида $CD(CRT_k, CRT_k)$, полученное для параметра d равняется расстоянию Евклида для параметра равного $2^m - d$, что следует из равенства

$$\sqrt{2^m d(2^m - d)} = \sqrt{2^m (2^m - d)(2^m - (2^m - d))}.$$

Иллюстрацией данного свойства являются численные значения расстояния Евклида, приведенные в таблице 5.8.

Следствие 5.5 Значение расстояния Евклида $CD(CRT_k, CRT_l) = \sqrt{2^m d(2^m - d)}$ (5.17), полученное для тестов CRT_k и CRT_l, состоящих из $q = 2^m$ m-разрядных данных $\{0, 1, 2, ..., 2^m - 1\}$, может быть использовано как среднее значение расстояния Евклида $ACD(CRT_k, CRT_l)$ равное $\sqrt{qd(2^m - d)}$ между тестами CRT_k и CRT_l, включающими $q < 2^m$ тестовых наборов.

Для примера 5.7 и теста $CRT_l = \{11, 14, 4, 13, 0\}$, полученного на основании исходного теста $CRT_k = \{3, 6, 12, 5, 8\}$ при $d = 8$, согласно (5.16) получим, что $ACD(CRT_k, CRT_l) = \sqrt{5 \times 8 \times (2^3 - 8)} = \sqrt{320}$. Отметим, что для этих тестов расстояние Евклида строго равняется его среднему значению. Действительно, $CD(CRT_k, CRT_l) = [(3 - 11)^2 + (6 - 14)^2 + (12 - 4)^2 + (5 - 13)^2 + (8 - 0)^2]^{1/2} = \sqrt{320}$.

Следствие 5.6. Если расстояние Евклида $CD(CRT_k, CRT_l)$ между управляемыми вероятностными тестами CRT_k и CRT_l согласно теореме 5.6 равня-

ется $\sqrt{2^m d_l (2^m - d_l)}$, а для тестов CRT_k и CRT_n, $CD(CRT_k, CRT_n) = \sqrt{2^m d_n (2^m - d_n)}$, то $CD(CRT_l, CRT_n) = \sqrt{2^m d_c (2^m - d_c)}$, где $d_c = d_l - d_n \mod 2^m$.

В соответствии с примером 5.8 $CRT_l = \{0, 4, 5, 3, 7, 2, 6, 1\}$, а $CRT_n = \{7, 3, 4, 2, 6, 1, 5, 0\}$, из следствия 5.6 получим, что $d_c = d_l - d_n \mod 2^m = 5 - 4 \mod 2^3 = 1$ и $CD(CRT_k, CRT_n) = \sqrt{2^m d_c (2^m - d_c)} = \sqrt{2^3 \times 1 \times (2^3 - 1)} = \sqrt{56}$. Используя классическое определение расстояния Евклида, получим, что $CD(CRT_l, CRT_n) = [(0-7)^2 + (4-3)^2 + (5-4)^2 + (3-2)^2 + (7-6)^2 + (2-1)^2 + (6-5)^2 + (1-0)^2]^{1/2} = \sqrt{56}$.

В качестве основы построения многократных управляемых вероятностных тестов $MCRT_r = \{CRT_0, CRT_1, CRT_2, \ldots, CRT_{r-1}\}$ будет использоваться соотношение (5.16), которое характеризуется минимальной вычислительной сложностью при получении последующих тестов $CRT_1, CRT_2, \ldots, CRT_{r-1}$, на основании исходного CRT_0.

При формировании многократных вероятностных тестов $MCRT_r = \{CRT_0, CRT_1, CRT_2, \ldots, CRT_{r-1}\}$, в качестве мер эффективности, в дальнейшем будут использоваться максимальность минимального расстояния Хэмминга $MHD(CRT_k, CRT_l)$ и максимальность минимального расстояния Евклида $MCD(CRT_k, CRT_l)$, $k \neq l \in \{0, 1, 2, \ldots, r-1\}$ (5.5).

Первоначально отметим, что максимальность значения $MHD(CRT_k, CRT_l)$, достигается за счет выполнения неравенства $d_1 \neq d_2 \neq d_3 \neq \ldots \neq d_{r-1}$. Действительно, как отмечалось ранее, необходимым условием с точки зрения максимального значения расстояния Хэмминга $MHD(CRT_k, CRT_l)$, которому должны соответствовать тесты CRT_k и CRT_l, $k \neq l \in \{0, 1, 2, \ldots, r-1\}$, является отсутствие у этих тестов совпадающих тестовых наборов $T_i(k)$ и $T_i(l)$, $i \in \{0, 1, 2, \ldots, q-1\}$, что эквивалентно выполнению неравенства $T_i(k) \neq T_i(l)$. Так как тестовые наборы $T_i(k)$ и $T_i(l)$ взаимосвязаны соотношением (5.16), то выполнение неравенства $T_i(k) \neq T_i(l)$ достигается за счет применения ненулевого значения параметра $d \neq 0$ для получения тестовых наборов $T_i(l)$ теста CRT_l на основании тестовых наборов $T_i(k)$ исходного теста CRT_k.

Последовательно рассмотрим многократные управляемые вероятностные тесты $MCRT_r$ различной кратности, начиная с двукратных тестов $MCRT_2$ состоящих из CRT_0 и CRT_1, где второй тест CRT_1 формируется на основании исходного теста CRT_0 согласно (5.16). Оптимальное значение параметра d для получения CRT_1, согласно следствию 5.3, равняется 2^{m-1}. В этом случае расстояние Евклида между тестами CRT_0 и CRT_1 принимает максимальное значение, что обеспечивает максимальное различие между этими тестами и максимальную эффективность их совместного применения.

Для тестов $MCRT_r$, кратность которых $r > 2$, докажем следующую теорему.

Теорема 5.7. Максимальное значение $MHD(CRT_k, CRT_l)$, которому должны соответствовать тесты CRT_k и CRT_l, $k \neq l \in \{0, 1, 2, \ldots, r-1\}$, много-

кратного управляемого вероятностного теста $MCRT_r$, состоящего из $r > 2$ вероятностных тестов $\{CRT_0, CRT_1, CRT_2, \ldots, CRT_{r-1}\}$, каждый из которых содержит $q \le 2^m$ m-разрядных тестовых наборов, достигается при максимальном минимальном значении $d_k - d_l$, $k \ne l \in \{0, 1, 2, \ldots, r-1\}$, а $d_k \ne d_l \in \{1, 2, \ldots, 2^m - 1\}$.

Доказательство теоремы 5.7. При построении многократного управляемого вероятностного теста $MCRT_r = \{CRT_0, CRT_1, CRT_2, \ldots, CRT_{r-1}\}$, последующие тесты $CRT_1, CRT_2, \ldots, CRT_{r-1}$ строятся на основании CRT_0, используя соотношение (5.16) и множество параметров $d \in \{d_1, d_2, d_3, \ldots, d_{r-1}\}$. В случае, когда $r > 2$, значения параметра d выбираются таким образом, чтобы максимизировать расстояние Хэмминга, а именно чтобы выполнялось неравенство $d_1 \ne d_2 \ne d_3 \ne \ldots \ne d_{r-1}$.

Предположив, что $q = 2^m$, можно заключить, что для двух произвольных тестов CRT_k и CRT_l, $k \ne l \in \{0, 1, 2, \ldots, r-1\}$, согласно (5.17), расстояние Евклида равняется выражению $\sqrt{2^m d(2^m - d)}$ для параметра d равного $d_l - d_k \bmod 2^m$. При $r > 2$ для произвольных пар параметров d_l и d_k, $k \ne l \in \{0, 1, 2, \ldots, r-1\}$, минимальная их разность $d_l - d_k \bmod 2^m$ всегда больше нуля и меньше 2^{m-1}. Отметим, что функция расстояния Евклида $\sqrt{2^m d(2^m - d)}$ является возрастающей функцией для $d = \overline{0, 2^{m-1}}$. Тогда можно заключить, что, чем больше минимальная разность $d_l - d_k \bmod 2^m$, тем больше будет значение $MCD(CRT_k, CRT_l)$, $k \ne l \in \{0, 1, 2, \ldots, r-1\}$ (5.17). Что и требовалось доказать.

Основываясь на доказанной теореме можно заключить, что для общего случая многократного теста $MCRT_r$, оптимальными значениями параметров $d_1, d_2, \ldots, d_{r-1}$ являются значения, которые делят диапазон целых чисел от 0 до 2^m на равные интервалы и вычисляются согласно соотношению.

$$d_i = \left\lfloor \frac{i 2^m}{r} + 0{,}5 \right\rfloor, i \in \{1, 2, \ldots, r-1\}. \qquad (5.18)$$

В случае трехкратного вероятностного теста $MCRT_3$, для получения второго CRT_1 и третьего CRT_2 теста, на основании исходного теста CRT_0, необходимо использовать оптимальные сочетания параметров d_1 и d_2 (5.18), применяемых для получения тестов CRT_1 и CRT_2 согласно (5.16). Соответственно для трехкратных вероятностных тестов $d_1 = \lfloor 1 \times 2^m / 3 + 0{,}5 \rfloor$, а $d_2 = \lfloor 2 \times 2^m / 3 + 0{,}5 \rfloor$. Для $m = 3$ получим, что $d_1 = 3$, а $d_2 = 5$, а для $m = 4$ - $d_1 = 5$, а $d_2 = 11$.

Рассмотрим $MCRT_3 = \{CRT_0, CRT_1, CRT_2\}$ для $m = 4$, при использовании $d_1 = 5$ и $d_2 = 11$. Расстояние Евклида между тестами CRT_0 и CRT_1 вычисляется следующим образом $CD(CRT_0, CRT_1) = \sqrt{16 \times 5 \times (16 - 5)} = \sqrt{880} = 29{,}7$. Осталь-

ные значения расстояний Евклида, для произвольного значения d, приведены в таблице 5.9.

Таблица 5.9. *Значения расстояний Евклида для* m=4

d	1	2	3	4	5	6	7	8
$CD(CRT_k, CRT_l)$	15,5	21,2	24,9	27,7	29,7	30,9	31,7	32,0
d	9	10	11	12	13	14	15	16
$CD(CRT_k, CRT_l)$	31,7	30,9	29,7	27,7	24,9	21,2	15,5	0

Согласно указанной таблице значение расстояния Евклида $CD(CRT_0, CRT_2) = 29{,}7$. В тоже время расстояние между тестами CRT_1 и CRT_2, в соответствии со следствием 5.6, для d определяемого как $d_2 - d_1 = 11 - 5 = 6$ равняется $CD(CRT_1, CRT_2) = 29{,}7$.

Анализ приведенных значений расстояний Евклида, для рассматриваемого $MCRT_3$, показывает, что $MCD(CRT_k, CRT_l) = 29{,}7$ для $k \neq l \in \{0, 1, 2\}$ (5.5), и $TCD(CRT_2) = CD(CRT_2, CRT_0) + CD(CRT_2, CRT_1) = 29{,}7 + 29{,}7 = 59{,}4$ (5.4) принимают максимальные значения.

Для четырехкратного теста $MCRT_4 = \{CRT_0, CRT_1, CRT_2, CRT_3\}$, используя соотношение (5.18), например, для $m = 4$ получим, что $d_1 = 4$, $d_2 = 8$ и $d_3 = 12$. Значения расстояний между любыми двумя тестами $MCRT_4$ приведены в таблице 5.10.

Таблица 5.10. *Значения расстояний Евклида для теста* $MCRT_4$

	CRT_0	CRT_1	CRT_2	CRT_3
CRT_0	-	27,7	32,0	27,7
CRT_1	27,7	-	27,7	32,0
CRT_2	32,0	27,7	-	27,7
CRT_3	27,7	32,0	27,7	-

Как видно из таблицы 5.10 значение $MCD(CRT_k, CRT_l)$, $k \neq l \in \{0, 1, 2, 3\}$ для $MCRT_4$ принимает максимально возможное значение равное 27,7.

5.5. Эффективность многократных вероятностных тестов

В качестве меры эффективности многократных управляемых вероятностных тестов $MCRT_r$ используем метрику $E(k, 2^m)$, введенную в [334] для целей формирования очередных тестовых наборов при генерировании однократного управляемого вероятностного теста. Можно предположить, что чем больше значение данной метрики, тем более эффективным является очередной управляемый тест CRT_i, который в совокупности с предыдущими тестами позволяет достичь максимальной эффективности. Отметим, что в предыдущих разделах было показано, что для достижения максимальной эффективности многократных управляемых вероятностных тестов $MCRT_r$ необхо-

димо, чтобы расстояние Евклида для теста CRT_i было максимальным по отношению к ранее сформированным тестам CRT_0, CRT_1, CRT_2, …, CRT_{i-1}.

Для сопоставительного анализа эффективности многократных управляемых вероятностных тестов $MCRT_r$, используем задачу тестирования запоминающих устройств [331, 333]. Первоначально рассмотрим запоминающее устройство, состоящее из $2^3 = 8$ запоминающих ячеек. Для его тестирования применим тест CRT_0, представляющий собой всевозможные трехразрядные адреса, формируемые согласно методологии маршевых тестов [331, 333]. При формировании очередного адреса первоначальное нулевое состояние ячейки памяти изменяется на единичное состояние. Таким образом, начальное нулевое состояние всех ячеек запоминающего устройства изменяется на их единичное состояние. Отметим, что значения $CD(CRT_0, CRT_1)$ для двух тестов CRT_0, CRT_1 и $m = 3$ приведены в таблице 5.8. В качестве второго управляемого вероятностного теста CRT_1 используем тест, полученный согласно (5.16) для всевозможных значений параметра d. Результирующие значения метрики $E(k, 2^m)$ для двукратного теста $MCRT_2$, состоящего из тестов CRT_0 и CRT_1, приведены в таблице 5.11.

Как видно из приведенных численных значений, эффективность двукратного теста находится в строгом соответствии со значениями $CD(CRT_0, CRT_1)$, приведенными в таблице 5.8. Действительно, для $d = 1$ и $d = 7$ расстояние Евклида между CRT_0 и CRT_1 равняется минимальному значению $\sqrt{56}$ (см. таблицу 5.8), соответственно и количество дополнительных двоичных комбинаций минимально для всех значений k. В тоже время для $d = 4$ и соответственно максимального значения $CD(CRT_0, CRT_1) = \sqrt{128}$ число дополнительных комбинаций максимально (см. таблицу 5.11).

Таблица 5.11. *Оценка эффективности двукратного теста для запоминающего устройства, состоящего из восьми запоминающих ячеек* ($2^m = 8$), *для* $k = 3, 4, 5, 6$

d	$E(k, 2^m)$ - дополнительное количество двоичных комбинаций			
	$E(3, 8)$	$E(4, 8)$	$E(5, 8)$	$E(6, 8)$
1	42	105	140	105
2	72	165	200	135
3	90	195	220	140
4	96	204	224	140
5	90	195	220	140
6	72	165	200	135
7	42	105	140	105

Аналогичные результаты, подтверждающие работоспособность Евклидова расстояния в качестве меры эффективности многократного теста на примере двукратного теста $MCRT_2$, для запоминающего устройства, состоящего из 128 ячеек и $k = 3$, приведены на рис. 5.2.

Приведенные в таблице 5.11 и на рис. 5.2 результаты подтверждают справедливость полученных теоретических положений, и в первую очередь правомерность теоремы 5.7.

При использовании управляемых вероятностных тестов чаще всего, количество q тестовых наборов меньше общего количества 2^m m-разрядных входных наборов [331, 333]. Соответственно существенное значение для предложенного метода формирования управляемых вероятностных тестов имеет справедливость результатов теоремы 5.6 для случая $q < 2^m$ и, в первую очередь, следствия 5.6. Согласно указанному следствию расстояние Евклида $CD(CRT_k, CRT_l) = \sqrt{2^m d(2^m - d)}$, полученное для $q = 2^m$, может быть использовано как среднее значение для $q < 2^m$ и определяться соотношением $ACD(CRT_k, CRT_l) = \sqrt{qd(2^m - d)}$.

Для подтверждения данного следствия были проведены статистические эксперименты. В частности, для $m = 10$ и различных $q < 2^m$, генерировались по 5 000 исходных управляемых вероятностных тестов CRT_k. Затем, используя всевозможные значения $d \in \{0, 1, \ldots, 1023\}$, строились тесты CRT_l. Далее определялось экспериментальное значение $ACD(CRT_k, CRT_l)$ как результат усреднения по 5 000 парам тестов CRT_k и CRT_l и, согласно следствию 5.6, по формуле $\sqrt{qd(2^m - d)}$. На рис. 5.3 приведены результаты для случая малого значения $q = 5$, когда погрешность эксперимента по отношению к аналитическому результату максимальна.

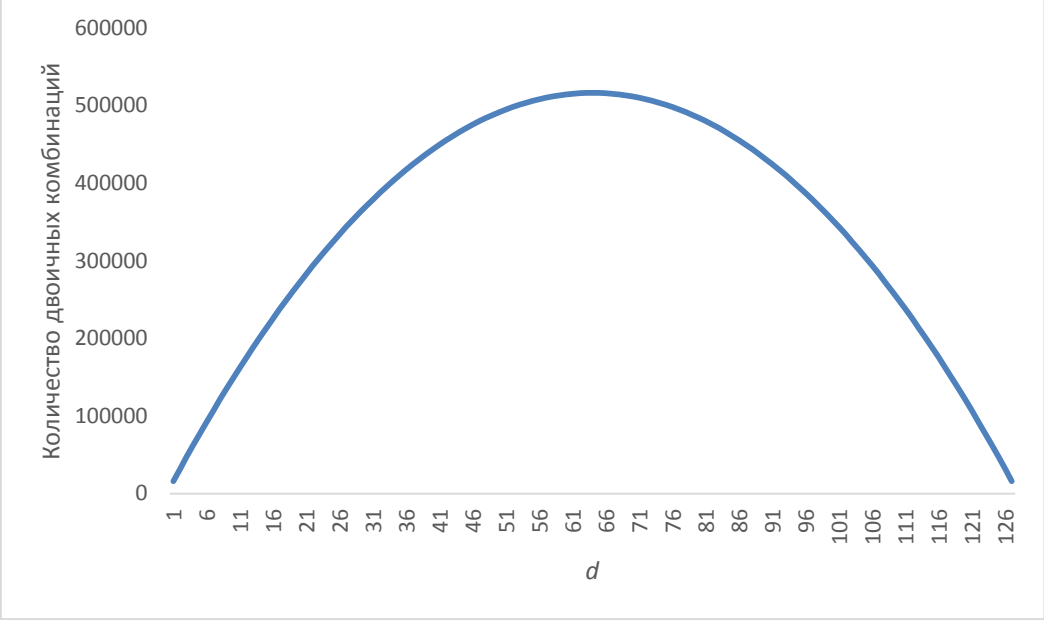


Рис. 5.2. $E(3,2^7)$ - *дополнительное количество двоичных комбинаций на всевозможных* $k = 3$ *из* $2^m = 128$ *разрядах*

Очевидно, что погрешность между экспериментальными значениями $ACD(CRT_k, CRT_l)$ и теоретическими, согласно следствию 5.6, должна умень-

шаться с ростом значения q. При $q = 2^m$ экспериментальные и теоретические значения должны быть равными, что подтверждается практическими результатами, приведенными на рис. 5.4. На указанном рисунке приведены усредненные значения отклонений экспериментальных данных от теоретических результатов в зависимости от величины q.

Как видно из приведенного рисунка, уже для q >100, экспериментальные результаты практически не отличаются от теоретических значений, что подтверждает правомерность использования результатов теоремы 5.6 для целей генерирования управляемых вероятностных тестов.

Как видно из приведенного рисунка, уже для q >100, экспериментальные результаты практически не отличаются от теоретических значений, что подтверждает правомерность использования результатов теоремы 5.5 для целей генерирования управляемых вероятностных тестов.

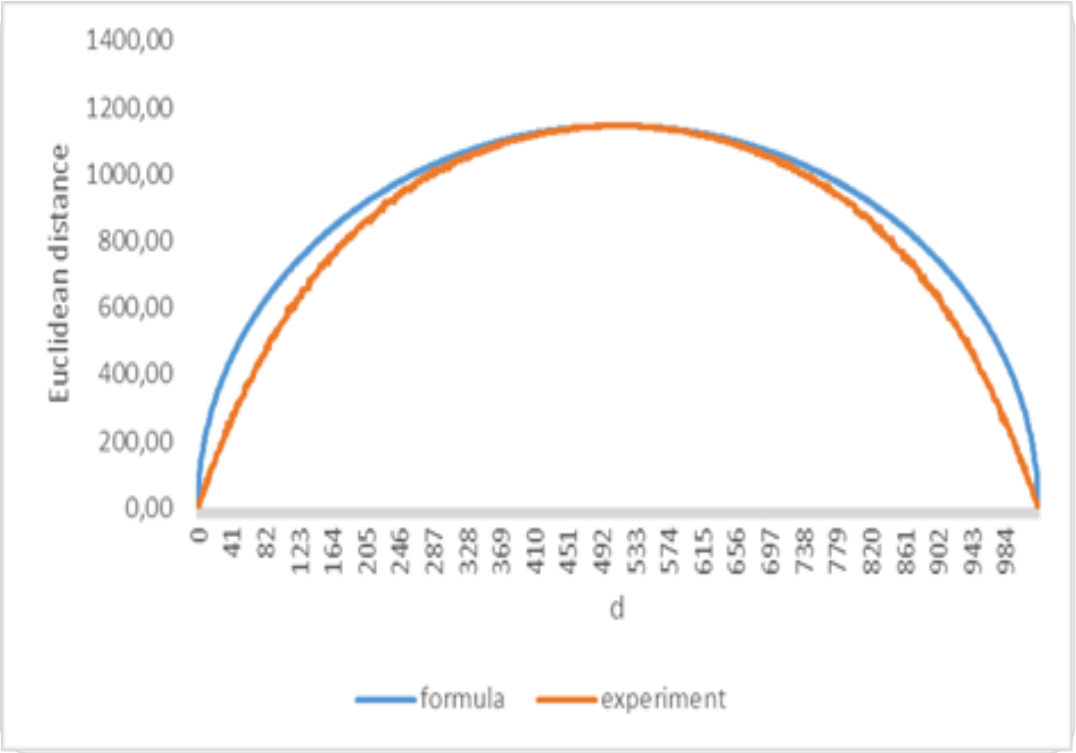


Рис. 5.3. *Среднее значение расстояния Евклида ACD*(CRT_k, CRT_l) (*experiment*) *для* m = 10 и q = 5 *и вычисленное в соответствии с теоремой* 5.6 (*formula*).

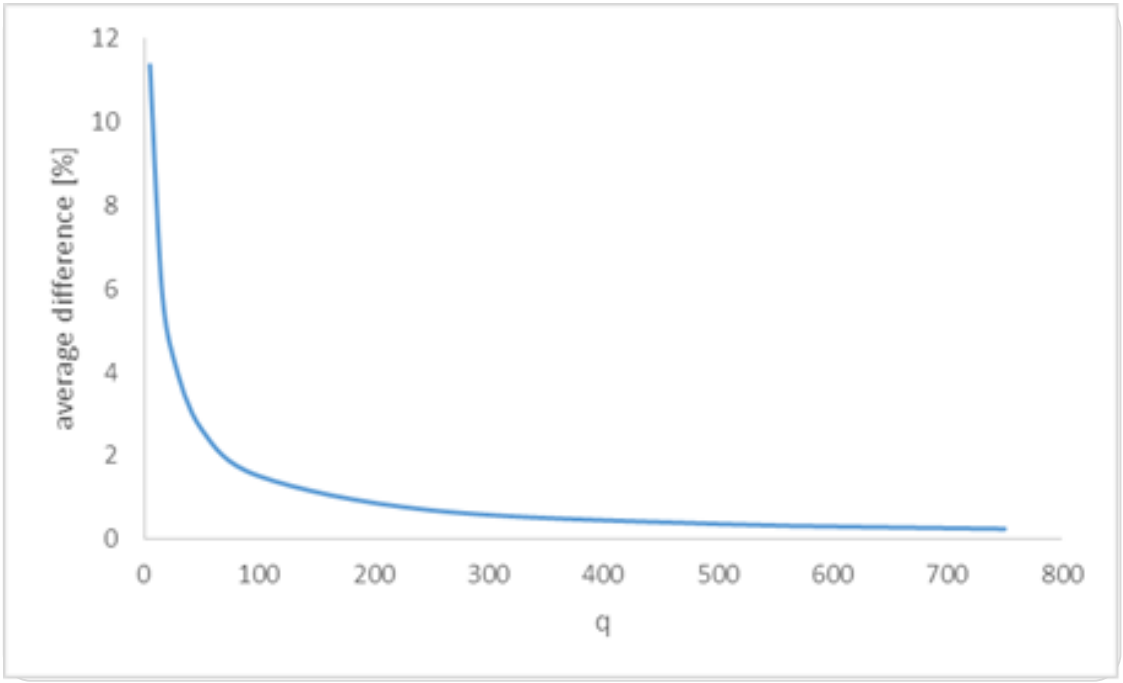


Рис. 5.4. *Среднее значение отклонения* (%) (*average difference*) *экспериментального значения ACD*(*CRTk*, *CRTl*) *от теоретической величины, полученной по формуле, для* $m = 10$.

Таким образом, в данной главе всесторонне рассмотрена концепция многократных управляемых вероятностных тестов, проведен анализ существующих решений и предложен формальный метод генерирования многократных тестов. Доказана эффективность применения расстояния Евклида для целей построения многократных тестов, которая подтверждается экспериментальными результатами для случая многократных тестов запоминающих устройств.

Глава 6. Квази-вероятностное тестирование

6.1. Квази-Монте-Карло метод

Метод *Монте-Карло* (*Monte Carlo method − MC*) является общим названием группы численных методов, основанных на получении большого числа реализаций вероятностного (случайного) процесса, который формируется таким образом, чтобы его вероятностные характеристики совпадали с искомыми величинами решаемой задачи [117, 293]. Метод Монте-Карло можно определить как метод моделирования случайных величин с целью вычисления характеристик их распределений. Данный метод используется для решения широкого круга задач в различных прикладных областях.

Возникновение идеи использования случайных явлений в области приближённых вычислений принято относить к 1777 году, когда появилась работа Бюффона об определении числа π с помощью случайных бросаний иглы на разграфлённую параллельными линиями бумагу [293]. Содержание предложенной идеи заключается в том, что экспериментально воспроизводится событие, вероятность которого выражается через число π, и приближённо оценивается эта вероятность. Отечественные работы по методу Монте-Карло появились в 1955-1956 годах. С того времени накопилась обширная библиография по методу Монте-Карло. Даже беглый просмотр названий работ позволяет сделать вывод о применимости метода Монте-Карло для решения прикладных задач из большого числа областей науки и техники.

Сущность метода Монте-Карло состоит в следующем. Требуется найти численное значение α некоторой изучаемой величины. Для этого формируют такую случайную величину x, математическое ожидание которой равно α, т. е. $M(x) = \alpha$. При этом пусть дисперсия случайной величины x, определяется как $D(x) = \beta^2$.

Далее рассматриваются n независимых случайных величин $x_1, x_2, x_3, \ldots, x_n$, распределение которых совпадает с распределением x. Если n достаточно велико, то, согласно центральной предельной теореме, распределение суммы величин $x_1, x_2, x_3, \ldots, x_n$ будет приблизительно нормальным с параметрами $M = n \times \alpha$ и $\sigma = \beta\sqrt{n}$.

Практическая реализация метода состоит в моделировании n испытаний, в результате которых получают n возможных значений x_i, $i \in \{1, 2, 2, \ldots, n\}$. Затем вычисляется их среднее арифметическое:

$$x^* = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Полученное значение x^* принимают в качестве оценки α^* (приближенного значения) искомого значения α.

Поскольку метод Монте-Карло требует проведения большого числа испытаний, его часто называют методом статистических испытаний. Теория этого метода указывает, как наиболее целесообразно выбрать случайную величину *x*, как найти её возможные значения. Простота метода Монте-Карло объясняет его популярность. Однако данный метод имеет большую вычислительную погрешность, которая, как правило, пропорциональна $\sqrt{b/n}$, где b некоторая постоянная величина, а n – число испытаний. Ясно, что добиться высокой точности на таком пути невозможно. Поэтому обычно говорят, что метод Монте-Карло особенно эффективен при решении тех задач, в которых результат нужен с небольшой точностью.

Рассмотренные в предыдущих главах методы вероятностного тестирования в своей основе используют модель черного ящика и направлены на формирование входных тестовых наборов как подмножества входных воздействий формируемых в большинстве случаев с использованием случайных воздействий. Вероятностное тестирование и все многообразие его модификаций можно описать в терминах численных *методов Монте-Карло*, основанных на получении большого числа реализаций стохастического (случайного) процесса. Основными недостатками методов вероятностного тестирования является их невысокая полнота покрытия неисправностей и большая длина тестовых последовательностей. Подобными недостатками характеризуется и метод Монте-Карло, для которого характерны значительная вычислительная погрешность и большая временная сложность.

Поэтому в качестве альтернативного решения для тестирования вычислительных систем в [345] предлагается использование идеи квази-вероятностного тестирования, основанного на применении квази-случайных последовательностей в качестве тестовых последовательностей, эффективно покрывающих пространство входных воздействий систем [35, 39, 81, 150, 180]. Аналогично, как и в методе *квази-Монте-Карло* (*КМК*) [209], квази-вероятностное тестирование [35, 345], очевидно, позволит достичь большей полноты покрытия при меньших длинах тестовых последовательностей. Основой реализации квази-вероятностного тестирования являются квази-случайные последовательности. Квази-случайные последовательности, также как и псевдослучайные последовательности относятся к множеству неслучайных последовательностей широко применяемых на практике в качестве реальной альтернативы случайным последовательностям. Дадим их определения [294, 301].

Определение 6.1. Последовательность неслучайных чисел называется *псевдослучайной последовательностью* (*pseudorandom sequence*) чисел, если она обладает всеми свойствами случайной последовательности.

Определение 6.2. Последовательность неслучайных чисел называется *квази-случайной* последовательностью (*quasi-random*), если ее можно использовать в реализации алгоритмов Монте-Карло вместо случайной последовательности.

Именно квази-случайные последовательности используются на практике для различных задач метода *КМК*, что позволяет достичь меньших вычислительных погрешностей и более быстрой сходимости [180, 292, 294]. Это достигается не столько свойством независимости, характерным для псевдослучайных последовательностей, сколько свойством равномерности. Такие последовательности в русскоязычной литературе называют согласно Соболю [180, 292, 294] $ЛП_\tau$-последовательностями, что интерпретируется как *Любой Последовательный* участок хорошо распределен (более равномерно по сравнению с псевдослучайными последовательностями). В англоязычной литературе такие последовательности называют последовательностями с малым дискрепансом (*low-discrepancy sequence*), а их разновидности – по именам авторов, выделяя последовательности Соболя [35, 39, 81, 150, 180, 292, 294].

а) *б*)

Рисунок 6.1 – *Точки в двухмерном пространстве*:
а) *квази-случайные*; *б*) *псевдослучайные*

Для визуальной демонстрации большей равномерности квази-случайной последовательности точек по сравнению с псевдослучайной последовательностью обычно рассматривается двухмерное пространство в виде единичного квадрата. Это пространство равномерно делится на под-квадраты. Так, например, квадрат на рис. 6.1 равномерно разбит на 64 под-квадрата, и на него нанесены 64 квази-случайные точки $ЛП_\tau$-последовательности (рис. 6.1, *а*) и 64 псевдослучайные точки (рис. 6.1, *б*). Из приведенных рисунков видно, что в каждый под-квадрат попало ровно по одной квази-случайной точке, в то время как для псевдослучайных точек равномерное заполнение под-квадратов не выполняется.

Следует отметить, что данное свойство $ЛП_\tau$-последовательностей выполняется для пространств произвольной размерности, а не только для двухмерного пространства [209, 214, 215]. Далее рассмотрим наиболее известные разновидности квази-случайных последовательностей.

6.2. Квази-случайные последовательности

Последовательности Корпута (*van der Corput*) являются простейшим примером квази-случайных последовательностей [150, 180]. Для произвольного целого $n \in \{1, 2, 3, \ldots, N\}$ значение элемента x_n последовательности Корпута может быть получено для любого базиса p представления числа n,

где p – простое целое число. Первоначально целое число n представляется в базисе p как:

$$n = \sum_{i=0}^{I} \alpha_i(n) p^i,$$

где $\alpha_i(n) \in \{0, 1, 2, \ldots, p-1\}$ является значением i-й цифры p-ичного представления числа n, а I – наименьшим целым значением i, для которого $\alpha_i(n) \neq 0$, а для всех $i > I$ выполняется равенство $\alpha_i(n) = 0$. Значение I вычисляется как:

$$I = \lfloor \log_p n \rfloor, \quad n = \overline{1, N}.$$

Затем для получения значения x_n цифры $\alpha_i(n)$ числа n транспонируются относительно запятой, разделяющей целую и дробную части p-ичного числа, таким образом, что первой после запятой оказывается младшая $\alpha_0(n)$ цифра p-ичного представления числа n. Следующей цифрой будет $\alpha_1(n)$ и т. д. Аналитически процедура транспонирования, в результате которой получается значение x_n, описывается формулой:

$$x_n = \sum_{i=0}^{I} \frac{\alpha_i(n)}{p^{i+1}}.$$

В качестве примера рассмотрим целое число $n = 11$, которое представим в базисе $p = 3$ как $11_{(10)} = 1\times3^2 + 0\times3^1 + 2\times3^0 = 102_{(3)}$ с $\alpha_0(11) = 2$, $\alpha_1(11) = 0$ и $\alpha_2(11) = 1$. Следует отметить, что $I = \lfloor \log_3 11 \rfloor = 2$ и соответственно $\alpha_i(n) = 0$ для $i > 2$. Выполнив процедуру транспонирования, получим значение элемента x_{11} последовательности Корпута $x_{11} = 2/3^1 + 0/3^2 + 1/3^3 = 19/27$, которое принадлежит интервалу [0, 1]. Квази-случайной последовательностью Корпута с основанием $p = 2$ и включающая $N = 7$ чисел, является последовательность x_n, состоящая из следующих элементов: $x_1 = 1/2_{(10)}=0{,}100_{(2)}$; $x_2 = 1/4_{(10)}=0{,}010_{(2)}$; $x_3 = 3/4_{(10)}=0{,}110_{(2)}$; $x_4 = 1/8_{(10)}=0{,}001_{(2)}$; $x_5 = 5/8_{(10)}=0{,}101_{(2)}$; $x_6 = 3/8_{(10)}=0{,}011_{(2)}$ и $x_7 = 7/8_{(10)}=0{,}111_{(2)}$.

Важным свойством последовательности Корпута является то, что после генерирования каждого множества из $2^k - 1$ элементов для приведенного примера $k = 1$, 2 и 3 последовательность максимально равномерно распределена, т. е. самый длинный интервал, который не содержит элементов из последовательности Корпута, является минимальным.

Использование произвольного базиса p предопределяет формирование иррациональных дробных чисел, что затрудняет применение подобных последовательностей для целей тестирования вычислительных систем при $p \neq 2$.

Последовательности Халтона (*Halton*) представляются последовательностью точек, задаваемых их координатами, в s-мерном ($s \geq 1$) пространстве

[81]. Они являются наиболее известными многомерными квази-случайными последовательностями. Данные последовательности служат основой для построения других разновидностей квази-случайных последовательностей [81]. Первая координата точек Халтона задается последовательностью Корпута с основанием $p = 2$, вторая координата определяется последовательностью Корпута с основанием 3. В общем случае координаты точек Халтона задаются последовательностями Корпута с использованием простых чисел p как оснований систем счисления [81].

В качестве примера рассмотрим последовательность точек Халтона в двухмерном пространстве ($s = 2$) для $p = 2$ и $p = 3$. Для построения данной последовательности необходимо использовать две последовательности Корпута для $p = 2$ и 3. В десятичной системе счисления эти последовательности иррациональных чисел принимают следующий вид: 1/2, 1/4, 3/4, 1/8, 5/8, 3/8, 7/8, ..., и 1/3, 2/3, 1/9, 4/9, 7/9, 2/9, 5/9, ... Тогда последовательность Халтона будет представляться последовательностью точек с координатами (1/2, 1/3), (1/4, 2/3), (3/4, 1/9), (1/8, 4/9), (5/8, 7/9), (3/8, 2/9), (7/8, 5/9), ... в двухмерном пространстве. Пример последовательности точек Халтона x_n для трехмерного случая приведен в таблице 6.1.

Таблица 6.1 – *Последовательность Халтона*

x_n	$p = 2$	$p = 3$	$p = 5$
n=1	1/2	1/3	1/5
n=2	1/4	2/3	2/5
n=3	3/4	1/9	3/5
n=4	1/8	4/9	4/5
n=5	5/8	7/9	1/25
n=6	3/8	2/9	6/25
n=7	7/8	5/9	11/25
n=8	1/16	8/9	16/25

Отмечается, что последовательности Халтона характеризуются низким качеством для размерностей s больше чем 14 [81]. На практике в силу корреляционных зависимостей последовательности Халтона используются для значений s не большее чем 6 [81].

Так же, как и в случае последовательностей Корпута, применение последовательностей Халтона для целей технической диагностики практически ограничивается только одномерными последовательностями, когда $s = 1$ и соответственно $p = 2$.

Последовательности Соболя (*Sobol*) используют двоичную систему счисления для формирования координат точек в s-мерном пространстве и в силу данного обстоятельства являются широко востребованными для современных приложений [292, 293, 294]. Последовательность точек Соболя в s-мерном пространстве строится на основании одномерной последовательности Корпута, когда первая координата точек формируется последовательно-

стью Корпута для $p = 2$, а остальные – путем процедуры перестановок [180]. При этом перестановки зависят от так называемых *направляющих чисел* (*direction numbers*) v_i^j, применяемых для всех измерений $j = \overline{1, s}$ s-мерного пространства. Направляющие числа $v_i^j = \frac{m_i^j}{2^i}, i = \overline{1, w}$, образуют последовательность дробных двоичных чисел с w двоичными бинарными значениями после запятой, где m_i^j представляет собой целое нечетное число, удовлетворяющее неравенству $0 < m_i^j < 2^i$. Для описания конкретной последовательности Соболя необходимо задать направляющие числа v_i^j для всех измерений s-мерного пространства.

Значение n-го элемента (точки) x_n последовательности Соболя определяется его координатами x_n^j для всех измерений s, которые вычисляются согласно выражению

$$x_n^j = \alpha_0(n)v_1^j \oplus \alpha_1(n)v_2^j \oplus ... \oplus \alpha_{w-1}(n)v_w^j, \quad j = \overline{1, s}.$$

Здесь $\alpha_i(n) \in \{0, 1\}$, $i = \overline{0, w-1}$, являются значениями цифр двоичного представления

$$\sum_{i=0}^{\lfloor \log_2 n \rfloor} \alpha_i(n)2^i$$

числа n, а символ $\oplus$ означает операцию поразрядного сложения по модулю два (XOR). Например, если $m_1^j = 1$, $m_2^j = 3$, $m_3^j = 7$ ($w = 3$) и соответственно $v_1^j = 0{,}100$, $v_2^j = 0{,}110$ и $v_3^j = 0{,}111$, можно получить значение произвольного элемента последовательности Соболя для $n \in \{1, 2, \ldots, 2^w - 1\} = \{1, 2, \ldots, 7\}$. Предположим, что $n = 7_{(10)}$, которое в двоичном представлении записывается в виде $111_{(2)}$, тогда значение j-й координаты x_7^j седьмого элемента x_7 последовательности Соболя вычисляется как

$$x_7^j = \alpha_0(7)v_1^j \oplus \alpha_1(7)v_2^j \oplus \alpha_2(7)v_3^j = 0{,}100 \oplus 0{,}110 \oplus 0{,}111 = 0{,}101.$$

Все элементы последовательности Соболя длиной $2^w - 1 = 2^3 - 1 = 7$ приведены в таблице 6.2.

Здесь для одномерного случая $v_i^1 = v_i$, а $x_n^1 = x_n$.

Таблица 6.2 – *Одномерная классическая последовательность Соболя*

$n_{(10)}$	$n_{(2)}= \alpha_2(n)\alpha_1(n)\alpha_0(n)$	x_n	$2^w \times x_n$
1	0 0 1	v_1	1 0 0
2	0 1 0	v_2	1 1 0
3	0 1 1	$v_1 \oplus v_2$	0 1 0
4	1 0 0	v_3	1 1 1
5	1 0 1	$v_1 \oplus v_3$	0 1 1
6	1 1 0	$v_2 \oplus v_3$	0 0 1
7	1 1 1	$v_1 \oplus v_2 \oplus v_3$	1 0 1

Последовательности Соболя, основанные на использовании двоичной системы счисления ($p = 2$), являются широко востребованной разновидностью квази-случайных последовательностей чисел в основном из-за удобства реализации на ЭВМ алгоритмов их генерирования. Основными недостатками классической последовательности Соболя являются заметная вычислительная сложность, зависящая от значения номера элемента (количества операций сложения по модулю два), а также ограниченное множество последовательностей, определяемое выбором направляющих чисел [35, 39, 81, 150, 180, 292, 294].

6.3. Модифицированные последовательности Соболя

Из таблицы 6.2 видно, что значение координаты n-го элемента x_n последовательности Соболя вычисляется как поразрядная сумма по модулю два до $w = \lfloor \log_2 n \rfloor$ операндов в зависимости от количества ненулевых компонентов двоичного представления $\alpha_{w-1}(n)\ \alpha_{w-2}(n)\ \ldots\ \alpha_1(n)\ \alpha_0(n)$ величины n. Количество операндов может быть снижено до одного при использовании кода Грея [168]. Известно, что двоичное число $n + 1$, закодированное в коде Грея, отличается от числа n только в одном бите. Представление числа n в коде Грея может быть получено согласно известному соотношению $n_g = g_{w-1}(n)\ g_{w-2}(n)\ \ldots\ g_1(n)\ g_0(n) = \alpha_{w-1}(n)\ \alpha_{w-2}(n)\ \ldots\ \alpha_1(n)\ \alpha_0(n) \oplus 0\alpha_{w-1}(n)\ \alpha_{w-2}(n)\ \ldots\ \alpha_1(n)$ [168]. Здесь индекс g числа n_g означает его представление в коде Грея.

В силу того, что $(n + 1)_g$ отличается от n_g только в одном бите, значение $x^j_{(n+1)g}$ будет отличаться от величины x^j_{ng} только значением одного направляющего числа v^j_i. Тогда значение j-й координаты $x^j_{(n+1)g}$ элемента $x_{(n+1)g}$ последовательности Соболя будет определяться как:

$$x^j_{(n+1)g} = x^j_{ng} \oplus v^j_i . \qquad (6.1)$$

Соотношение (6.1) является описанием экономичного способа Антонова и Салеева [242] для формирования последовательностей Соболя, приводимого во многих источниках, в том числе и в [150]. Процедура формирования последовательности Соболя для одномерного случая в соответствии с

(6.1) представлена в таблице 6.3. Здесь рассмотрен случай последовательности сгенерированной при условиях аналогичных последовательности, приведенной в таблице 6.2.

Таблица 6.3 – *Последовательность Соболя сформированная по способу Антонова-Салеева*

n	n_g	v_i	$2^w \times x_{ng}$
0 0 1	001⊕000=001	v_1	1 0 0
0 1 0	010⊕001=011	v_2	0 1 0
0 1 1	011⊕001=010	v_1	1 1 0
1 0 0	100⊕010=110	v_3	0 0 1
1 0 1	101⊕010=111	v_1	1 0 1
1 1 0	110⊕011=101	v_2	0 1 1
1 1 1	111⊕011=100	v_1	1 1 1

Значение индекса направляющего числа v_i^j в выражении (6.1) зависит от так называемой последовательности переключений T_q отраженного кода Грея [168]. Для классического отраженного кода Грея переключательная последовательность задается рекурсивной процедурой следующим образом. Первоначально задается $T_1 = 1$ и, если $q > 1$, $T_q = T_{q-1}, q, T_{q-1}$. Например, для $q = 4$ получим $T_4 = 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1$.

Применение соотношения (6.1) позволяет существенно снизить вычислительную сложность генерирования последовательностей Соболя, что и предопределило их широкое применение на практике [35, 39, 150].

Вторым недостатком последовательностей Соболя является ограниченность их количества, которое определяется наборами направляющих чисел. С целью расширения множества последовательностей Соболя введем формализацию представления модифицированных направляющих чисел в виде нижней треугольной матрицы (унитреугольной матрицы) с единичной главной диагональю [348].

Первоначально отметим, что, в силу ранее приведенных ограничений, направляющие числа для всех измерений $v_i = 0,\beta_{-1}(i)\ \beta_{-2}(i)\ \ldots\ \beta_{-w}(i)$, во всех случаях имеют определенные значения $\beta_{-i}(i) = 1$ и $\beta_{-j}(i) = 0$ для $j > i$, так же, как и произвольные значения $\beta_{-j}(i) \in \{0, 1\}$ для $j < i$. Это означает, что для всех возможных последовательностей Соболя и всех их координат $v_1 = 0{,}100\ldots0$, а соответственно $2^w \times v_1 = 1\ 0\ 0\ \ldots\ 0\ 0$ в силу того, что для m_1 существует только одно безальтернативное значение $m_1 = 1 < 2^1$. Так как m_2 есть нечетное целое, меньшее чем 2^2, оно может принимать два значения: 1 или 3. Соответственно $2^w \times v_2 = \beta_{-1}(2)\ 1\ 0\ \ldots\ 0\ 0$, где $\beta_{-1}(2) = 0$ для $m_2 = 1$ и $\beta_{-1}(2) = 1$ для $m_2 = 3$. Для m_3 имеем $2^w \times v_3 = \beta_{-1}(3)\beta_{-2}(3)\ 1\ 0\ \ldots\ 0\ 0$ и т. д.

Далее будем рассматривать случай одномерных последовательностей Соболя и обозначим значения $2^w \times v_i$ новой переменной μ_i, которую будем рассматривать как значения модифицированных направляющих чисел [348].

Отметим, что результаты, полученные для одномерного случая, легко обобщаются на многомерные последовательности Соболя.

Числа μ_i можно представить в виде нижней треугольной матрицы с единичной главной диагональю (таблица 6.4).

Таблица 6.4 – *Значения модифицированных направляющих чисел*

μ_i	$\beta_{-1}(i)$	$\beta_{-2}(i)$	$\beta_{-3}(i)$	…	$\beta_{-w+1}(i)$	$\beta_{-w}(i)$
μ_1	1	0	0	…	0	0
μ_2	$\beta_{-1}(2)$	1	0	…	0	0
μ_3	$\beta_{-1}(3)$	$\beta_{-2}(3)$	1	…	0	0
…	…	…	…	…	…	…
μ_{w-1}	$\beta_{-1}(w-1)$	$\beta_{-2}(w-1)$	$\beta_{-3}(w-1)$	…	1	0
μ_w	$\beta_{-1}(w)$	$\beta_{-2}(w)$	$\beta_{-3}(w)$	…	$\beta_{-w+1}(w)$	1

Согласно процедуре генерирования $x_{(n+1)g} = x_{ng} \oplus \mu_i$ последовательности Соболя (6.1) для получения очередного значения используется только одно направляющее число. Индекс i направляющего числа μ_i определяется последовательностью переключений отраженного кода Грея, в которой на каждой второй позиции использует индекс 1, на каждой четвертой индекс 2, на каждой восьмой 3 и т. д. Это следует из определения переключательной последовательности и видно из ранее приведенного примера для T_4 (см. таблицу 6.3) [168]. Таким образом, каждое второе значение последовательности Соболя получается с использованием μ_1, каждое четвертое с использованием μ_2 и т. д. Для μ_1 значение $\beta_{-1}(1) = 1$, что обеспечивает максимальную частоту изменения старшего бита кода элемента x_{ng} последовательности Соболя. Максимальная частота изменения следующего бита кода x_{ng} в два раза меньше и т. д. (см. таблицу 6.3).

Следует отметить, что наиболее важным элементом последовательности Соболя являются направляющие числа μ_i, $i = \overline{1, w}$, приведенные в таблице 6.4, которые определяют уникальные значения для каждой координаты точек в s-мерном пространстве.

Наиболее значимым свойством направляющих чисел, вытекающим из ограничений на эти числа и подтверждающимся видом введенной матрицы модифицированных порождающих чисел, является их линейная независимость [348].

Таблица 6.4 является обобщенной математической моделью одномерной последовательности Соболя, задаваемой w направляющими числами μ_i, $i = \overline{1, w}$. Действительно, набор значений $\beta_{-j}(i) \in \{0, 1\}$, $i, j \in \{1, 2, \ldots, w\}$ для $j < i$, определяет одну из 2^c, $c = (w^2 - w)/2$ последовательностей Соболя. Так, для $w = 3$, компоненты $\beta_{-1}(2)$, $\beta_{-1}(3)$ и $\beta_{-2}(3)$ могут принимать произвольные значения, определяя одну из возможных последовательностей: $ЛП_\tau(1)$, $ЛП_\tau(2)$, …, $ЛП_\tau(8)$ (см. таблицы 6.5 и 6.6).

Для μ_1 значение $\beta_{-1}(1) = 1$, что обеспечивает максимальную частоту изменения старшего бита кода элемента последовательности Соболя. Максимальная частота изменения следующего бита кода x_n в два раза меньше и так далее, как это показано на примерах, приведенных в таблицах 6.5 и 6.6.

Таблица 6.5 – *Последовательности Соболя* $ЛП_\tau(1)$, $ЛП_\tau(2)$, $ЛП_\tau(3)$, $ЛП_\tau(4)$ *для* $w = 3$

$w = 3$	$ЛП_\tau(1)$			$ЛП_\tau(2)$			$ЛП_\tau(3)$			$ЛП_\tau(4)$		
$\begin{vmatrix} 1 & 0 & 0 \\ \beta_{-1}(2) & 1 & 0 \\ \beta_{-1}(3) & \beta_{-2}(3) & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$			$\begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$			$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{vmatrix}$			$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix}$		
x_0	0	0	0	0	0	0	0	0	0	0	0	0
$x_1 = x_0 + \mu_1$	1	0	0	1	0	0	1	0	0	1	0	0
$x_2 = x_1 + \mu_2$	1	1	0	0	1	0	1	1	0	1	1	0
$x_3 = x_2 + \mu_1$	0	1	0	1	1	0	0	1	0	0	1	0
$x_4 = x_3 + \mu_3$	0	1	1	1	1	1	0	1	1	1	1	1
$x_5 = x_4 + \mu_1$	1	1	1	0	1	1	1	1	1	0	1	1
$x_6 = x_5 + \mu_2$	1	0	1	1	0	1	1	0	1	0	0	1
$x_7 = x_6 + \mu_1$	0	0	1	0	0	1	0	0	1	1	0	1

Таблица 6.6 – *Последовательности Соболя* $ЛП_\tau(5)$, $ЛП_\tau(6)$, $ЛП_\tau(7)$, $ЛП_\tau(8)$ *для* $w = 3$

$w = 3$	$ЛП_\tau(5)$			$ЛП_\tau(6)$			$ЛП_\tau(7)$			$ЛП_\tau(8)$		
$\begin{vmatrix} 1 & 0 & 0 \\ \beta_{-1}(2) & 1 & 0 \\ \beta_{-1}(3) & \beta_{-2}(3) & 1 \end{vmatrix}$	$\begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{vmatrix}$			$\begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix}$			$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix}$			$\begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix}$		
x_0	0	0	0	0	0	0	0	0	0	0	0	0
$x_1 = x_0 + \mu_1$	1	0	0	1	0	0	1	0	0	1	0	0
$x_2 = x_1 + \mu_2$	0	1	0	0	1	0	1	1	0	0	1	0
$x_3 = x_2 + \mu_1$	1	1	0	1	1	0	0	1	0	1	1	0
$x_4 = x_3 + \mu_3$	1	0	1	0	1	1	1	0	1	0	0	1
$x_5 = x_4 + \mu_1$	0	0	1	1	1	1	0	0	1	1	0	1
$x_6 = x_5 + \mu_2$	1	1	1	0	0	1	1	1	1	0	1	1
$x_7 = x_6 + \mu_1$	0	1	1	1	0	1	0	1	1	1	1	1

Для получения полного множества модифицированных направляющих чисел μ_i, $i = \overline{1, w}$ на основании $m < w$ исходных, так же, как и в оригинальном методе Соболя, будем использовать примитивные порождающие полиномы. В общем виде примитивный полином имеет вид $\varphi(x) = 1 \oplus \lambda_1 x^1 \oplus \lambda_2 x^2 \oplus \ldots \oplus \lambda_{m-1} x^{m-1} \oplus x^m$, где $m = \deg\varphi(x)$, а двоичные коэффициенты $\lambda_i \in \{0, 1\}, i = \overline{1, m-1}$, определяют конкретный вид полинома [301]. Подобный подход, когда на ос-

новании m исходных направляющих чисел и порождающего полинома $\varphi(x)$ генерируются остальные $w - m$ чисел, представлен в оригинальном методе Соболя и его классических модификациях, в том числе и в экономичном способе Антонова и Салеева.

Рассмотрим алгоритм формирования модифицированных направляющих чисел, представленных в виде нижней треугольной матрицы с единичной главной диагональю (см. таблицу 6.4).

При реализации каждой итерации значение предыдущих (ранее полученных) направляющих чисел будем модифицировать с целью формирования двоичных кодов, разрядность которых увеличивается от первоначальных m бит до w результирующих бит. Формально это достигается сдвигом направляющего числа на один разряд влево с одновременным приписыванием в младшем разряде двоичного нуля. Подобная модификация математически записывается как

$$\mu_j = \mu_j \times 2^1, \quad j = \overline{1,i}, \quad m \le i < w.$$

Значение μ_j в левой части равенства есть новое значение направляющего числа μ_j, полученное на основании его предыдущего значения. Само рекуррентное соотношение примет вид:

$$\mu_{i+1} = \lambda_1\mu_i \oplus \lambda_2\mu_{i-1} \oplus ... \oplus \lambda_{m-1}\mu_{i-m+2} \oplus \mu_{i-m+1} \oplus (\mu_{i-m+1} / 2^m). \qquad (6.2)$$

Значение $\mu_{i-m+1}/2^m$ представляет собой сдвинутую копию вправо на m позиций двоичного кода μ_{i-m+1}.

Например, для случая последовательности Соболя длиной $2^w - 1 = 2^5 - 1 = 31$, где $w = 5$, используем примитивный полином $\varphi(x) = 1 \oplus x^1 \oplus x^3$ степени $m = 3$, а для первых $m = 3$ направляющих чисел возьмем целые нечетные числа $m_1 = 1$, $m_2 = 3$ и $m_3 = 5$. Тогда v_1 =0,100, v_2 = 0,110, v_3 = 0,101 и соответственно $\mu_1 = v_1 2^3 = 100$, $\mu_2 = v_2 2^3 = 110$ и $\mu_3 = v_3 2^3 = 101$. Так как $w = 5$, остальные $(w - m)$ направляющие числа, в данном случае $5 - 3 = 2$ числа, генерируются с использованием рекуррентного соотношения (6.2) для заданных значений m и w

$$\mu_j = \mu_j \times 2^1, \quad j = \overline{1,i}, \quad 3 \le i < 5;$$

$$\mu_{i+1} = \mu_i \oplus \mu_{i-2} \oplus (\mu_{i-2} / 2^3).$$

Для $i = 3$ получим $\mu_1 = \mu_1 \times 2 = 1000$, $\mu_2 = \mu_2 \times 2 = 1100$ и $\mu_3 = \mu_3 \times 2 = 1010$, кроме того, $\mu_1/2^3 = 0\ 0\ 0\ 1$. Тогда $\mu_4 = \mu_3 \oplus \mu_1 \oplus (\mu_1/2^3) = 1\ 0\ 1\ 0 \oplus 1\ 0\ 0\ 0 \oplus 0\ 0\ 0\ 1 = 0\ 0\ 1\ 1$, и далее для $i = 4$ получим $\mu_5 = \mu_4 \oplus \mu_2 \oplus (\mu_2/2^3) = 0\ 0\ 1\ 1\ 0 \oplus 1\ 1\ 0\ 0\ 0 \oplus 0\ 0\ 0\ 1\ 1 = 1\ 1\ 1\ 0\ 1$.

Последовательность Соболя, соответствующая полученным направляющим числам μ_1 = 1 0 0 0 0, μ_2 = 1 1 0 0 0, μ_3 = 1 0 1 0 0, μ_4 = 0 0 1 1 0 и μ_5 = 1 1 1 0 1, приведена в таблице 6.7.

Таблица 6.7 – *Последовательность Соболя*

n	T_q	x_{nq}	n	T_q	x_{nq}	n	T_q	x_{nq}	n	T_q	x_{nq}
1	1	1 0 0 0 0	9	1	0 0 0 1 0	17	1	0 1 0 1 1	25	1	1 1 0 0 1
2	2	0 1 0 0 0	10	2	1 1 0 1 0	18	2	1 0 0 1 1	26	2	0 0 0 0 1
3	1	1 1 0 0 0	11	1	0 1 0 1 0	19	1	0 0 0 1 1	27	1	1 0 0 0 1
4	3	0 1 1 0 0	12	3	1 1 1 1 0	20	3	1 0 1 1 1	28	3	0 0 1 0 1
5	1	1 1 1 0 0	13	1	0 1 1 1 0	21	1	0 0 1 1 1	29	1	1 0 1 0 1
6	2	0 0 1 0 0	14	2	1 0 1 1 0	22	2	1 1 1 1 1	30	2	0 1 1 0 1
7	1	1 0 1 0 0	15	1	0 0 1 1 0	23	1	0 1 1 1 1	31	1	1 1 1 0 1
8	4	1 0 0 1 0	16	5	1 1 0 1 1	24	4	0 1 0 0 1			

Второй модификацией последовательностей Соболя является использование в качестве направляющих чисел w последовательных значений M-последовательности, формируемых в соответствии с примитивным порождающим полиномом степени w. В отличие от классического метода формирования направляющих чисел, в данном случае генерируется все множество w чисел, а не только $w - m$, на основании m исходных чисел.

Для соблюдения всех свойств последовательностей Соболя необходимо, чтобы μ_1 всегда равнялось w-разрядному коду 1 0 0 … 0 (см. таблицу 6.4). Для получения всего множества модифицированных направляющих чисел двоичный код 1 0 0 … 0 используем в качестве начального состояния генератора M-последовательности и сформируем $w - 1$ последующих состояний генератора, которые будем использовать как $\mu_2, \mu_3, \ldots, \mu_w$.

Например, для полинома $\varphi(x) = 1 \oplus x^1 \oplus x^4$ степени $m = 4$ направляющие числа принимают значения: μ_1 = 1 0 0 0, μ_2 = 1 1 0 0, μ_3 = 1 1 1 0 и μ_4 = 1 1 1 1. Изменив порождающий полином на $\varphi(x) = 1 \oplus x^3 \oplus x^4$, получим новое множество модифицированных направляющих чисел, а именно μ_1 = 1 0 0 0; μ_2 = 0 1 0 0; μ_3 = 0 0 1 0 и μ_4 = 1 0 0 1. Для двух приведенных примеров соответствующие треугольные матрицы V_1 и V_2 с единичной главной диагональю, построенные на основании модифицированных направляющих чисел, имеют вид

$$V_1 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{vmatrix}; \qquad V_2 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{vmatrix}.$$

Для каждого множества направляющих чисел, описываемых матрицами V_1 и V_2, может быть сформирована уникальная последовательность Соболя.

Очевидно, что количество множеств модифицированных порождающих чисел определяется количеством примитивных порождающих полиномов степени w. Для общего случая это количество вычисляется как $\Phi(2^w - 1)/w$, где Φ есть функция Эйлера от целого числа $2^w - 1$ [301].

Следующей модификацией последовательностей Соболя является использование для формирования произвольных значений $\beta_{-j}(i) \in \{0, 1\}$ для $j < i$ последовательных $(w^2 - w)/2$ бит произвольной псевдослучайной последовательности, в том числе и M-последовательности. Возможность формирования подобным образом модифицированных направляющих чисел μ_i следует из того факта, что равенство единице младшего бита двоичного представления целого числа свидетельствует о его нечетности. Это отражено, например, в таблице 6.4 в виде равенства $\beta_{-i}(i) = 1$, $i = \overline{1, w}$. Очевидно, что старшие биты нечетного целого числа могут принимать произвольные значения.

В случае модифицированных направляющих чисел, как отмечалось ранее, значение μ_1 всегда равняется w-разрядному коду 100…00. Соответственно $\mu_2 = \beta_{-1}(2)10\ldots00$, где $\beta_{-1}(2) \in \{0, 1\}$, $\mu_3 = \beta_{-1}(3)\ \beta_{-2}(3)\ 1\ 0\ldots0\ 0$, где $\beta_{-1}(3)$ и $\beta_{-2}(3) \in \{0, 1\}$, и т. д. Значение $\mu_w = \beta_{-1}(w)\ \beta_{-2}(w)\ \beta_{-3}(w)\ \ldots\ \beta_{-w+1}(w)1$, где $\beta_{-j}(w) \in \{0, 1\}$ для $j < w$. Таким образом, $\beta_{-j}(i) \in \{0, 1\}$ при $j < i$ для модифицированных направляющих чисел μ_i могут принимать произвольные двоичные значения. Количество возможных вариантов значений $\beta_{-j}(i) \in \{0, 1\}$ при $j < i$ определяется их числом $(w^2 - w)/2$ и вычисляется как $2^{(w^2-w)/2}$. Это подтверждается анализом таблицы 6.4 и матриц V_1 и V_2.

В качестве примера рассмотрим случай, когда $w = 4$. Тогда число значений $\beta_{-j}(i) \in \{0, 1\}$ при $j < i$, где $i = \overline{1, 4}$, равняется $(w^2 - w)/2 = (4^2 - 4)/2 = 6$. Это означает, что шесть элементов матрицы модифицированных направляющих чисел, находящихся под главной диагональю, могут принимать произвольные двоичные значения. Количество подобных матриц и соответственно множеств модифицированных направляющих чисел равняется $2^{(w^2-w)/2} = 2^6 = 64$. Две из указанных матриц V_1 и V_2 приводились ранее.

Максимально возможное количество уникальных множеств из w модифицированных направляющих чисел μ_i равняется $2^{(w^2-w)/2}$, а их элементы могут формироваться с использованием различных алгоритмов генерирования равновероятных двоичных цифр. В случае использования для этих целей M-последовательностей единственным ограничением на степень m порождающего полинома $\varphi(x)$ является равенство $\deg\varphi(x) = (w^2 - w)/2$. Выполнение данного равенства обеспечивает формирование различных вариантов значений $\beta_{-j}(i) \in \{0, 1\}$ при $j < i$ и соответственно всевозможных последовательностей Соболя для заданного значения w. Для случая, когда $w = 4$, для порождающего полинома $\varphi(x)$ необходимо, чтобы $\deg\varphi(x) = 6$, что позволит получить все возможные матрицы вида V_1 и V_2, которые будут отличаться значениями элементов, находящихся ниже главной диагонали.

6.4. Анализ свойств последовательностей Соболя

Математическая модель, описанная соотношением (6.1) и матрицей направляющих чисел в виде нижней треугольной матрицы с единичной диагональю (таблица 6.4), может быть расширена для случая последовательностей, относящихся не только к квази-случайным последовательностям. В общем случае, в качестве порождающей матрицы направляющих чисел V может быть использована любая квадратная матрица вида

$$V = \begin{vmatrix} \beta_{m-1}(0) & \beta_{m-2}(0) & \beta_{m-3}(0) & \dots & \beta_0(0) \\ \beta_{m-1}(1) & \beta_{m-2}(1) & \beta_{m-3}(1) & \dots & \beta_0(1) \\ \beta_{m-1}(2) & \beta_{m-2}(2) & \beta_{m-3}(2) & \dots & \beta_0(2) \\ \dots & \dots & \dots & \dots & \dots \\ \beta_{m-1}(m-1) & \beta_{m-2}(m-1) & \beta_{m-3}(m-1) & \dots & \beta_0(m-1) \end{vmatrix}, \qquad (6.3)$$

построенная из m линейно независимых двоичных векторов $v_i = \beta_{m-1}(i)\ \beta_{m-2}(i)\ \dots\ \beta_0(i)$, $i = \overline{0, m-1}$.

Для оценки свойств m-разрядной последовательности Соболя $A(n) = a_{m-1}\ a_{m-2}\ a_{m-3}\ \dots\ a_1\ a_0$, используемой в работе [229, 341] была введена метрика $M(j)$, $j \in \{0, 1, 2, \dots, m-1\}$, определяющая количество переключений (изменений) j-го разряда a_j кода последовательности $A(n)$. В общем случае, для произвольного значения j, величина данной метрики определяется согласно выражению:

$$M(j) = \sum_{i=0}^{m-1} \beta_j(i) \times 2^{m-1-i}. \qquad (6.4)$$

Для случая последовательностей Соболя, в силу наличия ограничений на значения направляющих чисел, выражение для данной метрики принимает вид [229, 341]:

$$M(j) = 2^j + \sum_{l=1}^{j} \beta_j(m-1-j+l) \times 2^{j-l}.$$

Из приведенного соотношения, например, следует, что младший разряд a_0 кода $A(n) = a_{m-1}\ a_{m-2}\ a_{m-3}\ \dots\ a_1\ a_0$ элементов последовательности Соболя изменит свое значение только один раз, так как, согласно (6.4), $M(0) = 1$. Величина $M(1) = 2^1 + \beta_1(m-1) \times 2^0$, в зависимости от значения $\beta_1(m-1)$, может принимать одно из двух значений, 2, если $\beta_1(m-1) = 0$, либо, 3 для случая, когда $\beta_1(m-1) = 1$. Это означает, что следующий разряд a_1 кода $A(n) = a_{m-1}\ a_{m-2}\ a_{m-3}\ \dots\ a_1\ a_0$ вне зависимости от длины последовательности Соболя изменит свое значение 2 либо 3 раза. В случае старшего разряда a_{m-1} имеем $M(m-1) = 2^{m-1} + \beta_{m-1}(1) \times 2^{m-2} + \beta_{m-1}(2) \times 2^{m-3} + \dots + \beta_{m-1}(m-1) \times 2^0$. Минимальное значение $M_{min}(m-1)$ количества переключений для старшего разряда равняется 2^{m-1}, в

случае, когда $\beta_{m-1}(1) = \beta_{m-1}(2) = \ldots = \beta_{m-1}(m-1) = 0$, а максимальное $M_{max}(m-1)$ равное $2^m - 1$ для $\beta_{m-1}(1) = \beta_{m-1}(2) = \ldots = \beta_{m-1}(m-1) = 1$ [341].

Для произвольного разряда $j \in \{0, 1, 2, \ldots, m-1\}$, имеем $M_{min}(j) = 2^j$, а $M_{max}(j) = 2^{j+1} - 1$, откуда следует, что всегда $M(j+1) - M(j) > 0$ для произвольных значений v_i, представляющих собой нижнюю треугольную матрицу с единичной диагональю. Таким образом, можно заключить, что количество переключений возрастает с ростом индекса j разряда кода $A(n) = a_{m-1}\, a_{m-2}\, a_{m-3} \ldots a_1\, a_0$ элементов последовательности Соболя. Приведенные оценки подтверждаются для частного случая последовательностей $ЛП_\tau(1)$, $ЛП_\tau(2)$, …, $ЛП_\tau(8)$ (см. таблицы 6.5 и 6.6).

В качестве меры различия последовательностей, в [335] предложено использование среднего расстояния Хэмминга $AHD[A(n), A(n+k)]$ между двумя последовательностями $A(n)$ и $A(n+k)$, которое для случая $k = 1$ вычисляется согласно выражению:

$$AHD[A(n), A(n+1)] = \frac{SHD[A(n), A(n+1)]}{2^m - 1}. \qquad (6.5)$$

Здесь $SHD[A(n), A(n+1)]$ представляет собой суммарное расстояние Хэмминга, которое равняется общему количеству переключений разрядов кода $A(n) = a_{m-1}\, a_{m-2}\, a_{m-3} \ldots a_1\, a_0$ при переходе к коду $A(n+1)$. Для случая последовательностей Соболя максимальное $AHD_{max}[A(n), A(n+1)]$ и минимальное $AHD_{min}[A(n), A(n+1)]$ значения данной характеристики определяются согласно (6.6):

$$\begin{aligned} AHD_{\max}[A(n), A(n+1)] &= \frac{\sum_{j=0}^{m-1} M_{\max}(j)}{2^m - 1} = \frac{\sum_{j=0}^{m-1} 2^{j+1} - 1}{2^m - 1} = 2 - \frac{m}{2^m - 1}; \\ AHD_{\min}[A(n), A(n+1)] &= \frac{\sum_{j=0}^{m-1} M_{\min}(j)}{2^m - 1} = \frac{\sum_{j=0}^{m-1} 2^j}{2^m - 1} = 1. \end{aligned} \qquad (6.6)$$

Конкретные значения элементов нижней треугольной матрицы (таблица 6.4) модифицированных направляющих чисел задают конкретный вид последовательности Соболя. Например, в случае, когда $\beta_{-j}(i) = 0$ для всех $j < i$ (см. таблицу 6.4), последовательность Соболя представляет собой одну из последовательностей кода Грея, например $ЛП_\tau(1)$, представленную в таблице 6.5. Для случая, когда $\beta_{-j}(i) = 1$ для всех $j < i$, последовательность Соболя (см. таблицу 6.6, $ЛП_\tau(8)$) представляет собой последовательность Корпута [345].

Приведенный анализ позволяет сформулировать следующее утверждение.

Утверждение 6.1. Модифицированная последовательность Соболя определяется видом порождающей матрицы V (6.3), представляющей собой

нижнюю треугольную матрицу (унитреугольную матрицу) с единичной главной диагональю.

Характерной особенностью последовательностей Соболя является возрастающая переключательная активность $M(j)$ разрядов кода $a_{m-1}\, a_{m-2}\, a_{m-3}\, \ldots\, a_1\, a_0$ при возрастании их индексов. Для подобных последовательностей всегда $M(j+1) - M(j) > 0$.

6.5. Неслучайные тестовые последовательности

Обобщенная математическая модель, представленная в предыдущем разделе, является расширением математической модели, используемой для модифицированных последовательностей Соболя [345, 229]. Основой данной модели является порождающая матрица V (6.3), которая и определяет основные свойства тестовых последовательностей и идентифицирует их подмножества. Также как и множество последовательностей Соболя, другие тестовые последовательности имеют детерминированную структуру и относятся к неслучайным тестовым последовательностям. Рассмотрим некоторые из них.

В случае пересчетных (счетчиковых) последовательностей, формируемых двоичными пересчетными схемами (счетчиками), наблюдается обратная зависимость для переключательной активности $M(j)$, по сравнению с последовательностями Соболя. А именно, частота изменений старшего разряда, кода $a_{m-1}\, a_{m-2}\, a_{m-3}\, \ldots\, a_1\, a_0$ является минимальной, а количество переключений в конкретном разряде уменьшается с ростом его индекса j.

Для формирования пересчетных последовательностей необходимо сформировать порождающую матрицу в соответствии с утверждением 6.2.

Утверждение 6.2. Пересчетная последовательность определяется видом порождающей матрицы V (6.3), представляющей собой нижнюю треугольную матрицу относительно побочной единичной диагонали.

Для произвольной пересчетной последовательности элементы матрицы V имеют определенные значения $\beta_i(i) = 1$, и $\beta_j(i) = 0$ для $j > i$, и произвольные значения $\beta_j(i) \in \{0, 1\}$ для $j < i$; $i, j \in \{0, 1, 2, \ldots, m-1\}$. Примеры подобных последовательностей $CS1$, $CS2$, …, $CS8$ приведены в таблице 6.8 для $m = 3$.

Таблица 6.8 – *Пересчетные последовательности* для $m = 3$

m=3	$CS1$	$CS2$	$CS3$	$CS4$	$CS5$	$CS6$	$CS7$	$CS8$
0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1
0 1 $\beta_0(1)$	0 1 0	0 1 1	0 1 0	0 1 0	0 1 1	0 1 1	0 1 0	0 1 1
1 $\beta_1(2)$ $\beta_0(2)$	1 0 0	1 0 0	1 1 1	1 0 1	1 1 0	1 0 1	1 1 0	1 1 1
$A(0)$	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
$A(1)=A(0)+v_0$	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1
$A(2)=A(1)+v_1$	0 1 1	0 1 0	0 1 1	0 1 1	0 1 0	0 1 0	0 1 1	0 1 0
$A(3)=A(2)+v_0$	0 1 0	0 1 1	0 1 0	0 1 0	0 1 1	0 1 1	0 1 0	0 1 1
$A(4)=A(3)+v_2$	1 1 0	1 1 1	1 0 1	1 1 1	1 0 1	1 1 0	1 0 0	1 0 0
$A(5)=A(4)+v_0$	1 1 1	1 1 0	1 0 0	1 1 0	1 0 0	1 1 1	1 0 1	1 0 1
$A(6)=A(5)+v_1$	1 0 1	1 0 1	1 1 0	1 0 0	1 1 1	1 0 0	1 1 1	1 1 0
$A(7)=A(6)+v_0$	1 0 0	1 0 0	1 1 1	1 0 1	1 1 0	1 0 1	1 1 0	1 1 1

Общее количество пересчетных последовательностей так же, как и последовательностей Соболя формируемых согласно (6.2), определяется величиной $q = 2^{(m^2-m)/2}$.

Для оценки свойств пересчетной тестовой последовательности используем метрику $M(j)$ (6.4), которая для данного случая принимает вид:

$$M(j) = 2^{m-1-j} + \sum_{l=1}^{m-1-j} \beta_j (j+l) \times 2^{m-1-j-l}.$$

Полученные значения метрики $M(j)$, позволяют оценить среднюю переключательную активность для генерируемой последовательности $A(n)$, которая определяется средним расстоянием Хэмминга. Так максимально возможная переключательная активность для последовательности Соболя и пересчетной последовательности достигается при всех единичных значениях $\beta_j(i)$ для $j < i$, участвующих в соотношениях для $M(j)$. Эта величина так же, как и минимальная переключательная активность, аналогично последовательностям Соболя, определяется соотношениями $M_{max}(j) = 2^{j+1} - 1$ и $M_{min}(j) = 2^j$. Соответственно, среднее расстояние Хэмминга $AHD[A(n), A(n+1)]$ между двумя последовательными элементами пересчетной последовательности вычисляется согласно соотношению (6.6).

Анализ матрицы (3) позволяет идентифицировать еще два множества неслучайных тестовых последовательностей, для которых максимально возможная переключательная активность принимает заметно большее значение, чем для оригинальных модифицированных последовательностей Соболя и пересчетных последовательностей. Эти последовательности назовем как модифицированные последовательности Соболя и пересчетные последовательности с максимальной переключательной активностью, которые соответствуют двум следующим утверждениям.

Утверждение 6.3. Порождающая матрица V (6.3) модифицированной последовательности Соболя с максимальной переключательной активностью представляет собой верхнюю треугольную матрицу с единичной побочной диагональю.

Утверждение 6.4. Порождающая матрица V (6.3) пересчетной последовательности с максимальной переключательной активностью представляет собой верхнюю треугольную матрицу относительно единичной главной диагонали.

Примеры последовательностей SS_m и CS_m из двух указанных множеств приведены в таблице 6.9. Для модифицированной последовательности Соболя SS_m с максимальной переключательной активностью значение $M(j)$ (6.4) вычисляется согласно выражению:

$$M(j) = 2^{m-1-j} + \sum_{l=1}^{j} \beta_j (j-l) \times 2^{m-1-j+l}.$$

Аналогичное выражение справедливо и для пересчетной последовательности CS_m с максимальной переключательной активностью:

$$M(j) = 2^j + \sum_{l=1}^{m-1-j} \beta_j (m-1-j-l) \times 2^{j+l}.$$

В обоих случаях, как в случае модифицированной последовательности Соболя с максимальной активностью (Утверждение 6.3), так и пересчетной последовательности с максимальной переключательной активностью (утверждение 6.4), используя метрику $M(j)$ можно получить аналогичные соотношения для среднего расстояния Хэмминга $AHD[A(n), A(n+1)]$, как и приведенные ранее (6.6).

В этом случае $AHD_{min}[A(n), A(n+1)]$ также равняется 1, что соответствует последовательностям кода Грея $SS_m4 = CS1$ и $CS_m4 = ЛП_\tau(1)$ (см. таблицы 6.8 и 6.5). В тоже время максимальная переключательная активность для обеих последовательностей SS_m и CS_m определяется согласно соотношению (6.7):

$$AHD_{\max}[A(n), A(n+1)] = m - 1 + \frac{m}{2^m - 1} \qquad (6.7)$$

Из последнего выражения видно, что максимальное среднее расстояние Хэмминга (6.7) значительно больше аналогичной величины для оригинальных модифицированных последовательностей Соболя и пересчетных последовательностей (6.6) практически на величину, равную $m - 3$.

Общее количество модифицированных последовательностей Соболя и пересчетных последовательностей с максимальной переключательной активностью определяется величиной $q = 2^{(m^2-m)/2}$.

Таблица 6.9 – *Модифицированные квазислучайные и пересчетные последовательности для* m=3

m=3	SS_m1	SS_m2	SS_m3	SS_m4	m=3	CS_m1	CS_m2	CS_m3	CS_m4
$\beta_2(0)\beta_1(0)$ 1	1 1 1	1 1 1	1 0 1	0 0 1	1 $\beta_1(0)$ $\beta_0(0)$	1 1 1	1 1 1	1 0 1	1 0 0
$\beta_2(1)$ 1 0	1 1 0	0 1 0	1 1 0	0 1 0	0 1 $\beta_0(1)$	0 1 1	0 1 0	0 1 1	0 1 0
1 0 0	1 0 0	1 0 0	1 0 0	1 0 0	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1
$A(0)$	0 0 0	0 0 0	0 0 0	0 0 0	$A(0)$	0 0 0	0 0 0	0 0 0	0 0 0
$A(1)=A(0)+v_0$	1 1 1	1 1 1	1 0 1	0 0 1	$A(1)=A(0)+v_0$	1 1 1	1 1 1	1 0 1	1 0 0
$A(2)=A(1)+v_1$	0 0 1	1 0 1	0 1 1	0 1 1	$A(2)=A(1)+v_1$	1 0 0	1 0 1	1 1 0	1 1 0
$A(3)=A(2)+v_0$	1 1 0	0 1 0	1 1 0	0 1 0	$A(3)=A(2)+v_0$	0 1 1	0 1 0	0 1 1	0 1 0
$A(4)=A(3)+v_2$	0 1 0	1 1 0	0 1 0	1 1 0	$A(4)=A(3)+v_2$	0 1 0	0 1 1	0 1 0	0 1 1
$A(5)=A(4)+v_0$	1 0 1	0 0 1	1 1 1	1 1 1	$A(5)=A(4)+v_0$	1 0 1	1 0 0	1 1 1	1 1 1
$A(6)=A(5)+v_1$	0 1 1	0 1 1	0 0 1	1 0 1	$A(6)=A(5)+v_1$	1 1 0	1 1 0	1 0 0	1 0 1
$A(7)=A(6)+v_0$	1 0 0	1 0 0	1 0 0	1 0 0	$A(7)=A(6)+v_0$	0 0 1	0 0 1	0 0 1	0 0 1

Обобщенная математическая модель формирования неслучайных тестовых последовательностей (6.3) позволяет также формировать последовательности с минимальной переключательной активностью.

К данному множеству относятся упоминаемые ранее последовательности кода Грея, пример которых присутствует во всех ранее рассмотренных множествах тестовых последовательностей. Действительно, $ЛП_{\tau}(1)$ принадлежит множеству модифицированных последовательностей Соболя, SS_m4 относится к множеству модифицированных последовательностей Соболя с максимальной переключательной активностью, $CS1 = SS_m4$ представляет собой пересчетную последовательность, $CS_m4 = SS1$ соответствует пересчетной последовательности с максимальной переключательной активностью. Каждой из указанных последовательностей присущи свойства множеств последовательностей, к которым они принадлежат.

В общем случае последовательность с минимальной переключательной активностью (минимальным расстоянием Хэмминга), формируемая согласно (6.1), обеспечивается матрицей V (6.3) с минимальным количеством единичных значений. Для произвольного случая подобная матрица строиться в соответствии с утверждением 6.5.

Утверждение 6.5. Порождающая матрица V (6.3) для последовательности с минимальной переключательной активностью состоит из отличающихся друг от друга строк, каждая их которых содержит по одной единице.

Согласно приведенному утверждению подобная порождающая матрица характеризуется тем, что она представляет собой множество отличающихся друг от друга столбцов содержащих по одной единице. Из этого следует, что каждый столбец обеспечивает одно из возможных значений (2^0, 2^1, 2^2, …, 2^{m-1}) переключательной активности, В силу этого, последовательность столбцов в матрице, удовлетворяющей утверждению 6.5, не влияет на среднее значение переключательной активности. Для последовательностей $A(n)$, формируемых с использованием подобных порождающих матриц V (6.3), $AHD[A(n), A(n+1)] = 1$, а общее их количество равняется $m!$

С целью получения последовательностей тестовых наборов с максимальным расстоянием Хэмминга между соседними векторами $A(n)$ и $A(n+1)$, за основу может быть взята последовательность переключений T_{m-1} отраженного кода Грея. В этом случае последовательность переключений T_{m-1} имеет инверсную интерпретацию, а именно текущий элемент последовательности T_{m-1} определяет индекс неизменяемого разряда $A(n)$ при переходе к следующему элементу $A(n+1)$. В этом случае все разряды, кроме одного, при переходе от $A(n)$ к $A(n+1)$ будут изменяться. Соответственно, расстояние Хэмминга между двумя последовательными элементами будет равняться $m-1$. Такие последовательности получили название последовательностей анти-Грея [233].

Для обеспечения равенства расстояния Хэмминга величине $m-1$, в рамках выбранной математической модели генерирования последовательностей, очевидным является использование матрицы направляющих чисел (6.3)

инверсной по отношению к случаю, соответствующему утверждению 6.5. Отметим, что утверждение 6.5 позволяет генерировать последовательности с минимальным расстоянием Хэмминга равным 1. Рассмотрим несколько примеров использования матриц (6.3) инверсных по отношению к аналогичным матрицам для последовательностей кода Грея.

Как видно из приведенной таблицы 6.10, сформированы всевозможные комбинации двоичных наборов из двух ($m = 2$) и четырех ($m = 4$) бит, и таким образом, получены последовательности анти-Грея с расстоянием Хэмминга равным $m - 1$. В тоже время, для $m = 3$ данный подход оказался неработоспособным в силу линейной зависимости между столбцами порождающей матрицы V для нечетных значений m [233].

Таблица 6.10 – *Примеры последовательностей с максимальной переключательной активностью*

T_{m-1}	m=2	m=3	m=4	T_{m-1}	m=4
	$V=\begin{matrix}10\\01\end{matrix}$	$V=\begin{matrix}1&1&0\\1&0&1\\0&1&1\end{matrix}$	$V=\begin{matrix}1&1&1&0\\1&1&0&1\\1&0&1&1\\0&1&1&1\end{matrix}$		$V=\begin{matrix}1&1&1&0\\1&1&0&1\\1&0&1&1\\0&1&1&1\end{matrix}$
	0 0	**0 0 0**	0 0 0 0	3	1 1 0 0
0	1 0	**1 1 0**	1 1 1 0	0	0 0 1 0
1	1 1	**0 1 1**	0 0 1 1	1	1 1 1 1
0	0 1	1 0 1	1 1 0 1	0	0 0 0 1
2		1 1 0	0 1 1 0	2	1 0 1 0
0		**0 0 0**	1 0 0 0	0	0 1 0 0
1		**1 1 0**	0 1 0 1	1	1 0 0 1
0		**0 1 1**	1 0 1 1	0	0 1 1 1

Тестовая последовательность с переключательной активностью равной $m - 1$, для четных значений m, формируемая согласно соотношению (6.2), обеспечивается матрицей V (6.3), удовлетворяющей утверждению 6.6.

Утверждение 6.6. Порождающая матрица V (6.3), состоящая из $m \times m$ элементов, для последовательности с переключательной активностью равной $m - 1$, для четных значений m, состоит из m отличающихся друг от друга строк, каждая их которых содержит $m - 1$ единиц.

Для последовательностей, формируемых с использованием подобных порождающих матриц V (3), $AHD[A(n), A(n + 1)] = m - 1$. Подобные последовательности формируются только для четных значений m, а общее их количество равняется $m!$

В качестве операции, позволяющей достичь максимальной переключательной активности, при генерировании тестовых последовательностей, весьма эффективным является использование операции инвертирования всех разрядов предыдущего элемента $A(n)$ для получения последующего значения $A(n + 1)$. В этом случае расстояние Хэмминга равняется предельно возможному значению равному m. В терминах математической модели, используе-

мой в данной статье, инвертирование всех разрядов кода элемента последовательности эквивалентно применению одного из направляющих чисел v_i представляющего собой единичный вектор, т.е. v_i = 1 1 1 … 1. Очевидно, что для достижения максимального среднего расстояния Хэмминга необходимо чтобы таким направляющим числом было число v_0. Последующее инвертирование тестового набора $A(n + 1)$ для получения $A(n + 2)$ приведет к повторению исходного набора, т.е. $A(n + 2) = A(n)$. Для целей обеспечения предельно возможной средней переключательной активности необходимо перемежающее получение расстояний Хэмминга равных m и $m – 1$ для последовательных пар элементов $A(n)$, $A(n + 1)$ и $A(n + 1)$, $A(n + 2)$ [233]. Обобщая приведенные рассуждения, можно сформулировать следующее утверждение.

Утверждение 6.7. Порождающая матрица V (6.3), состоящая из $m \times m$ элементов, для последовательности с предельной максимальной переключательной активностью состоит из одного единичного столбца и $m – 1$, отличающихся друг от друга столбцов, содержащих по одному нулевому значению в каждой из $m – 1$ строк, кроме первой.

Примеры последовательностей из множества, определяемого утверждением 6.7, для m = 3 приведены в таблице 6.11.

Таблица 6.11 – *Последовательности с предельной максимальной переключательной активностью для* m = 3

m=3	*LS*1	*LS*2	*LS*3	*LS*4	*LS*5	*LS*6
V	1 1 1 0 1 1 1 0 1	1 1 1 1 0 1 0 1 1	1 1 1 0 1 1 1 1 0	1 1 1 1 1 0 0 1 1	1 1 1 1 1 0 1 0 1	1 1 1 1 0 1 1 1 0
$A(0)$	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
$A(1)=A(0)+v_0$	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1
$A(2)=A(1)+v_1$	1 0 0	0 1 0	1 0 0	0 0 1	0 0 1	0 1 0
$A(3)=A(2)+v_0$	0 1 1	1 0 1	0 1 1	1 1 0	1 1 0	1 0 1
$A(4)=A(3)+v_2$	1 1 0	1 1 0	1 0 1	1 0 1	0 1 1	0 1 1
$A(5)=A(4)+v_0$	0 0 1	0 0 1	0 1 0	0 1 0	1 0 0	1 0 0
$A(6)=A(5)+v_1$	0 1 0	1 0 0	0 0 1	1 0 0	0 1 0	0 0 1
$A(7)=A(6)+v_0$	1 0 1	0 1 1	1 1 0	0 1 1	1 0 1	1 1 0

Переключательная активность рассмотренной последовательности состоит из суммы переключательных активностей разрядов определяемых утверждением 6.7. Для любой подобной матрицы эта сумма включает слагаемые $(2^m – 1) – 0$, $(2^m – 1) – 1$, $(2^m – 1) – 2^1$, $(2^m – 1) – 2^2$, …, $(2^m – 1) – 2^{m–2}$. Для матриц V отвечающих данным требованиям получим.

$$AHD_{\max}[A(n), A(n+1)] = m - \frac{1}{2} + \frac{1}{2(2^m - 1)} \qquad (6.8)$$

Очевидно, что данное значение $AHD_{max}[A(n – 1), A(n)]$ (6.8) является максимально достижимым [233].

Таким образом, в данной главе рассмотрена модель генерирования адресных последовательностей, как результат обобщения экономичного способа Антонова и Салеева, используемого для формирования последовательностей Соболя. Применение, предложенной в данной главе, модели позволяет формировать широкий спектр тестовых последовательностей. Данная модель формализует процедуру генерирования различных тестов от простейших квази-случайных тестовых последовательностей до последовательностей с различными свойствами необходимыми для тестирования современных вычислительных систем.

Глава 7. Псевдослучайные тестовые последовательности

7.1. Псевдослучайные последовательности

Применение вероятностного тестирования характеризуется принципиальной невозможностью, либо сложностью повторного формирования ранее сгенерированных случайных тестовых последовательностей для воспроизведения тестового эксперимента. Поэтому, даже в рамках многообразных методов вероятностного тестирования, в качестве тестовых последовательностей используются псевдослучайные последовательности. Подобные последовательности по своей сути являются детерминированными последовательностями, которые характеризуются свойствами случайности близкими к вероятностным характеристикам истинно случайных последовательностей. Наиболее часто при формировании псевдослучайных последовательностей, применяемых для различных целей, используются два метода. Первый из них, лежащий в основе большинства программных генераторов псевдослучайных чисел, описывается следующим рекуррентным соотношением [301, 114]:

$$X_k = AX_{k-1} + B \mod M, \quad k = 1, 2, 3, \ldots, \tag{7.1}$$

где A, B, M - постоянные целые числа; $X_0 > 0$; $A > 0$; $B \geq 0$; $M > X_0$; $M > A$; $M > B$.

Данный метод получил название *мультипликативного конгруэнтного* (при $B = 0$) или *смешанного конгруэнтного* (при $B > 0$), а формируемая в соответствии с ним последовательность – линейной конгруэнтной [301, 114]. При выборе величины $M = r^m$, где r является основанием системы счисления, в которой определяются значения m-разрядных псевдослучайных чисел X_k, соотношение (7.1) преобразуется к следующему виду.

$$X_k = AX_{k-1} + B \mod r^m, \quad k = 1, 2, 3, \ldots$$

Отметим, что не всякий набор значений X_0, A, B, r и m приводит к формированию последовательностей, близких по своим свойствам к равномерным случайным [301, 114]. Например, при $X_0 = 110$, $A = 011$, $B = 001$, $r = 2$ и $m = 3$ формируемая числовая последовательность имеет следующий вид: $X_0 = 110$, $X_1 = 011$, $X_2 = 010$, $X_3 = 111$, $X_4 = 110$, $X_5 = 011, \ldots$. Как видно, значения младшей цифры приведенной последовательности псевдослучайных чисел $X_k = X_0, X_1, X_2, \ldots$ имеют период повторения, равный двум. Период повторения старшей цифры и цифры, формируемой во втором разряде, равен соответственно 4 и 1. Для общего случая в [301] показано, что период L_λ повторения цифры λ-го $\lambda \in \{1, 2, 3, \ldots, m\}$ разряда псевдослучайных чисел X_k последовательности (7.1) при $B = 0$ оценивается соотношением:

$$\max L_\lambda = r^{\lambda-1}(r-1), \tag{7.2}$$

а при $B \neq 0$ соотношением:

$$\max L_\lambda = r^{2\lambda-1}(r-1), \tag{7.3}$$

Так, для двоичной системы счисления ($r = 2$) на основании (7.2) получаем, что период L_1 младшего разряда равен 1. Отсюда следует, что младший разряд последовательности чисел X_k остается неизменным и равен 1. Второй разряд имеет период $L_2 \leq 2$, что означает либо постоянство его значения, либо изменение в каждом такте на противоположное.

Использование соотношения (7.1) при $B \neq 0$ позволяет несколько увеличить период повторения цифр в разрядах чисел, что вытекает из выражения (7.3). Однако наличие указанной периодичности принципиально не устраняется, что влияет на качество псевдослучайной последовательности чисел [327].

Для уменьшения влияния периодичности значений разрядов псевдослучайных чисел (7.1) на качество их последовательностей, как правило, используются значения $M \neq 2^r$. Сочетание $M \neq 2^r$ и специальных значений A позволяет заметно улучшить качество псевдослучайных чисел в большей части для устранения закономерности уменьшения периода с уменьшением разряда числа [114]. Так для *Pentium* PC часто используемым значением $M = 2^{31} - 1 = 2147483647$, а в качестве множителя A величины 16807,630360016, 1078318381, 1203248318, 397204094, 2037812808, 1323257245, 764261123, 112817, … .

Применительно к проблеме тестирования современных вычислительных систем периодичность может заметно снизить полноту контроля. Так, например, формирование по определенному входу цифрового устройства системы неизменяющегося логического сигнала не позволяет установить даже константную неисправность данного входного полюса, соответствующую значению формируемого сигнала. Кроме того, рассматриваемый метод формирования псевдослучайных чисел и особенно его модификации отличается сложностью практической реализации, связанной с необходимостью выполнения операции умножения. Так как приведенный метод обладает отмеченными недостатками, наиболее широко применяется второй способ формирования псевдослучайных последовательностей, основанный на использовании соотношения [301]:

$$b_k = \sum_{i=1}^{m} \oplus \alpha_i b_{k-i}, \quad k = 0, 1, 2, \ldots, \tag{7.4}$$

где k представляет собой индекс формируемого двоичного символа $b_k \in \{0, 1\}$ псевдослучайной последовательности, а значения $\alpha_i \in \{0, 1\}$ являются

постоянными коэффициентами. Обозначение вида $\sum_{i=1}^{m}{}^{\oplus}$ определяет операцию суммирования по модулю два m логических переменных. При соответствующем выборе коэффициентов α_i на основании характеристического полинома $\varphi(x) = 1 \oplus \alpha_1 x^1 \oplus \alpha_2 x^2 \oplus \ldots \oplus \alpha_{m-1} x^{m-1} \oplus \alpha_m x^m$, который должен быть примитивным, последовательность бит b_k имеет максимальную длину, равную $2^m - 1$. Такая последовательность называется *M*-последовательностью (*M-sequence*) [18].

7.2. Генераторы *M*-последовательностей

Генераторы *M*-последовательностей являются одними из наиболее часто используемых типов генераторов псевдослучайных последовательностей, применяемых в различных приложениях. Для их реализации используется сдвиговый регистр с линейной обратной связью (*Linear Feedback Shift Register – LFSR*) [301]. Одной из основных причин широкого применения генераторов *M*-последовательностей является сравнительно простая их реализация как программная, так и аппаратная. Кроме того, отмечаются хорошие статистические свойства *M*-последовательностей, практически не отличающиеся от свойств случайных последовательностей. В то же время генераторы *M*-последовательностей, подобно, как и любые другие генераторы псевдослучайных последовательностей, обеспечивают воспроизводимость выходных последовательностей. Подобные генераторы (далее их будем обозначать как *LFSR*) строятся на основании примитивных порождающих полиномов $\varphi(x)$. Так, согласно примитивному порождающему полиному $\varphi(x) = 1 \oplus x \oplus x^4$, структурная схема *LFSR* может быть представлена в виде, показанном на рис. 7.1.

Схема LFSR, приведенного на рис.7.1, состоит из четырехразрядного регистра сдвига на один разряд вправо и двухвходового сумматора по модулю два, включенного в цепь обратной связи.

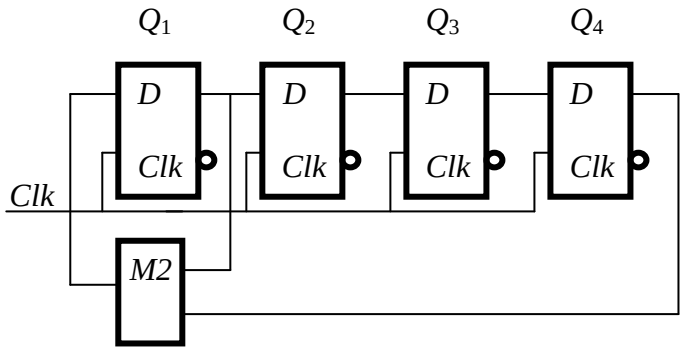


Рисунок 7.1 – *Функциональная схема LFSR*

Аналитически функционирование приведенного генератора описывается системой уравнений:

$$Q_1(k+1) = Q_1(k) \oplus Q_4(k);$$
$$Q_2(k+1) = Q_1(k);$$
$$Q_3(k+1) = Q_2(k);$$
$$Q_4(k+1) = Q_3(k).$$

Последовательные состояния генератора приведены в таблице 7.1.

Таблица 7.1. *Состояния генератора LFSR*

Clk	$Q_1\,Q_2\,Q_3\,Q_4$	*Clk*	$Q_1\,Q_2\,Q_3\,Q_4$
0	1 0 0 0	8	1 1 0 1
1	1 1 0 0	9	0 1 1 0
2	1 1 1 0	10	0 0 1 1
3	1 1 1 1	11	1 0 0 1
4	0 1 1 1	12	0 1 0 0
5	1 0 1 1	13	0 0 1 0
6	0 1 0 1	14	0 0 0 1
7	1 0 1 0	15	1 0 0 0

Как видно из приведенной таблицы 7.1, используя начальное состояние 1000, *LFSR* при подаче тактовых импульсов *Clk* генерирует всевозможные четырехразрядные двоичные коды, кроме нулевого кода (0 0 0 0). Последовательность генерируемых кодов имеет случайный характер, и её свойства не зависят от ненулевого начального состояния.

Для любой разрядности двоичных кодов m существуют примитивные порождающие полиномы $\varphi(x)$. Полиномы с минимальным количеством ненулевых коэффициентов для разрядностей от 1 до 28 приведены в таблице 7.2.

Таблица 7.2. *Примитивные порождающие полиномы*

$m = deg\,\varphi(x)$	$\varphi(x)$	$m = deg\,\varphi(x)$	$\varphi(x)$
1	$1 \oplus x$	15	$1 \oplus x \oplus x^{15}$
2	$1 \oplus x \oplus x^2$	16	$1 \oplus x^2 \oplus x^3 \oplus x^5 \oplus x^{16}$
3	$1 \oplus x \oplus x^3$	17	$1 \oplus x^3 \oplus x^{17}$
4	$1 \oplus x \oplus x^4$	18	$1 \oplus x^7 \oplus x^{18}$
5	$1 \oplus x^2 \oplus x^5$	19	$1 \oplus x \oplus x^2 \oplus x^5 \oplus x^{19}$
6	$1 \oplus x \oplus x^6$	20	$1 \oplus x^3 \oplus x^{20}$
7	$1 \oplus x \oplus x^7$	21	$1 \oplus x^2 \oplus x^{21}$
8	$1 \oplus x \oplus x^5 \oplus x^6 \oplus x^8$	22	$1 \oplus x \oplus x^{22}$
9	$1 \oplus x^4 \oplus x^9$	23	$1 \oplus x^5 \oplus x^{23}$
10	$1 \oplus x^3 \oplus x^{10}$	24	$1 \oplus x^3 \oplus x^4 \oplus x^9 \oplus x^{24}$
11	$1 \oplus x^2 \oplus x^{11}$	25	$1 \oplus x^3 \oplus x^{25}$
12	$1 \oplus x^3 \oplus x^4 \oplus x^7 \oplus x^{12}$	26	$1 \oplus x \oplus x^2 \oplus x^6 \oplus x^{26}$
13	$1 \oplus x \oplus x^3 \oplus x^4 \oplus x^{13}$	27	$1 \oplus x \oplus x^2 \oplus x^5 \oplus x^{27}$
14	$1 \oplus x \oplus x^{11} \oplus x^{12} \oplus x^{14}$	28	$1 \oplus x^3 \oplus x^{28}$

Для произвольной степени $m = deg\varphi(x)$ порождающего полинома $\varphi(x) = a_0x^0 \oplus a_1x^1 \oplus a_2x^2 \oplus \ldots \oplus a_{m-1}x^{m-1} \oplus a_mx^m$; $a_m = a_0 = 1$; $a_i \in \{0, 1\}$ существуют две альтернативные структуры реализации *LFSR*, а именно, с внешними и внутренними сумматорами по модулю два. На рис. 7.2 приведена структура *LFSR* с внешними сумматорами по модулю два. Подобная реализация *LFSR* характеризуется своей регулярной структурой включающей регистр сдвига и комбинационную схему, содержащую сумматоры по модулю два.

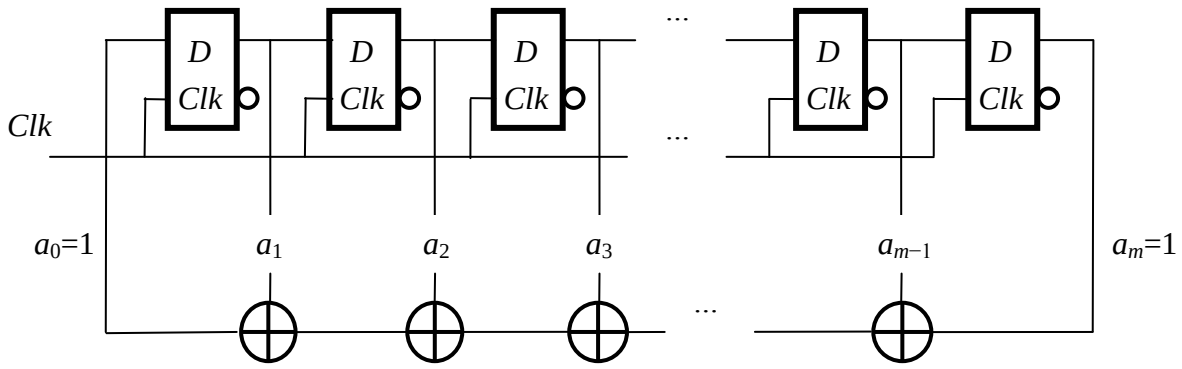


Рисунок 7.2 – *Структурная схема LFSR с внешними сумматорами по модулю два*

На рис.7.3 приведена структура *LFSR* с внутренними сумматорами по модулю два. Альтернативная реализация *LFSR* описывается тем же примитивным порождающим полиномом, что и предыдущая структура. В данном случае достигается максимальное быстродействие по сравнению с предыдущей схемой *LFSR*.

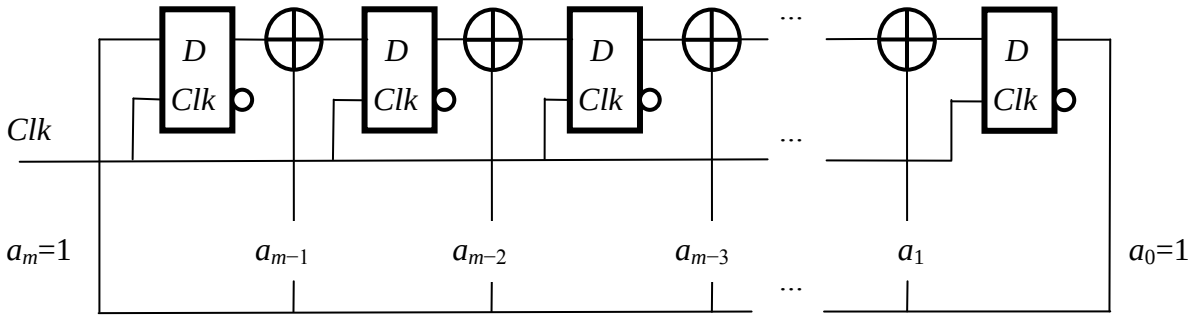
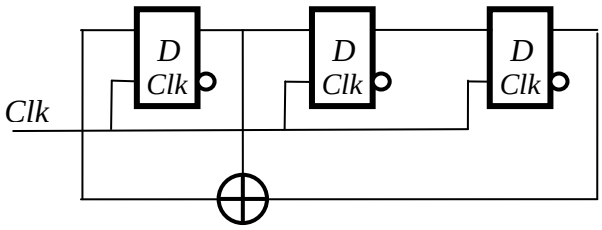


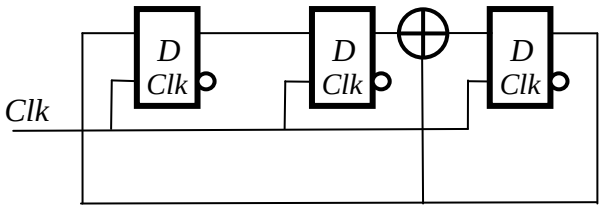
Рисунок 7.3 – *Структурная схема LFSR с внутренними сумматорами по модулю два*

Пример 7.1. В качестве примера *LFSR* с внешними и внутренними сумматорами рассмотрим случай порождающего полинома $\varphi(x) = 1 \oplus x \oplus x^3$,

для которого ниже (рис. 7.4) приведены соответствующие структуры для случая реализаций с внешними и внутренними сумматорами по модулю два.



а)



б)

Рисунок 7.4 – *Структурные схемы LFSR для порождающего полинома* $\varphi(x) = 1 + x + x^3$ *с внешними а*) *и внутренними б*) *сумматорами по модулю два*

Функционирование *LFSR* описывается математическим соотношением, представленным системой уравнений:

$$\begin{aligned} b_1(k+1) &= \sum_{i=1}^{m} \oplus \alpha_i b_i(k); \\ b_j(k+1) &= b_{j-1}(k);\ j = \overline{2, m}, \quad k = 0, 1, 2, \ldots . \end{aligned} \tag{7.5}$$

Матричное описание *LFSR* имеет вид.

$$\begin{vmatrix} b_1(k+1) \\ b_2(k+1) \\ b_3(k+1) \\ \ldots \\ b_m(k+1) \end{vmatrix} = \begin{vmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \ldots & \alpha_{m-1} & \alpha_m \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{vmatrix} \times \begin{vmatrix} b_1(k) \\ b_2(k) \\ b_3(k) \\ \ldots \\ b_m(k) \end{vmatrix}$$

В векторной форме будем иметь:

$$B(k+1) = V \times B(k),$$

где V представляет собой порождающую (генерирующую) матрицу, а $B(k)$ – вектор-столбец текущего, а $B(k + 1)$ – последующего состояния *LFSR*. Основой построения любого *LFSR* является примитивный порождающий полином

$\varphi(x) = a_0x^0 \oplus a_1x^1 \oplus a_2x^2 \oplus ... \oplus a_{m-1}x^{m-1} \oplus a_mx^m$, который и определяет его структуру, в зависимости от его старшей степени m и коэффициентов $a_i \in \{0, 1\}$.

7.3. Свойства *M*-последовательностей

Псевдослучайные последовательности (*M*-последовательности) характеризуются следующими свойствами.

1. Существует $\psi(2^m - 1)/m$ примитивных полиномов для заданного значения его степени m, где ψ есть функция Эйлера. Так, например, для $m = 3$ их количество вычисляется как $\psi(2^m - 1)/m = \psi(2^3 - 1)/3 = 6/3 = 2$. Таким образом, для $m = 3$ существует два примитивных полинома $\varphi(x) = 1 \oplus x \oplus x^3$ и $\varphi(x) = 1 \oplus x^2 \oplus x^3$. Следует отметить, что функция Эйлера принимает большие значения даже для небольших значений m и является немонотонно, но возрастающей функцией.

2. Для заданного полинома $\varphi(x)$ всегда существует инверсный полином $\varphi(x)^{-1}$, который может быть получен согласно следующему отношению:

$$\varphi(x)^{-1} = x^m \varphi(x^{-1}).$$

Пример 7.2. Для порождающего полинома $\varphi(x) = 1 \oplus x^2 \oplus x^5$ инверсный полином имеет вид: $\varphi(x)^{-1} = x^5\varphi(x^{-1}) = x^5 (1 \oplus x^{-2} \oplus x^{-5}) = 1 \oplus x^3 \oplus x^5$.

3. Период *M*-последовательности зависит от степени m примитивного порождающего полинома и равняется $L = 2^m - 1$.

Отметим что *LFSR*, построенный на базе примитивного порождающего полинома $\varphi(x)$ степени m, генерирует всевозможные m-разрядные двоичные коды, кроме кода, состоящего из всех нулей.

4. Для заданного полинома $\varphi(x)$ существует L различных *M*-последовательностей, каждая из которых отличается фазовым сдвигом.

Пример 7.3. Для порождающего полинома $\varphi(x) = 1 \oplus x \oplus x^4$ существует 15 *M*-последовательностей, приведенных в таблице 7.3.

5. *M*-последовательность представляет собой псевдослучайную последовательность, в которой вероятность появления нулей и единиц определяется соотношениями:

$$p(b_k = 0) = \frac{2^{m-1} - 1}{2^m - 1} = \frac{1}{2} - \frac{1}{2^{m+1} - 2};$$

$$p(b_k = 1) = \frac{2^{m-1}}{2^m - 1} = \frac{1}{2} + \frac{1}{2^{m+1} - 2}.$$

Как видно из приведенных соотношений, вероятность появления нуля и единицы уже для $m > 10$ практически равняется 0,5.

Таблица 7.3 – *M-последовательности* ($\varphi(x) = 1 \oplus x \oplus x^4$)

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
1	0	0	0	1	0	0	1	1	0	1	0	1	1	1
1	1	0	0	0	1	0	0	1	1	0	1	0	1	1
1	1	1	0	0	0	1	0	0	1	1	0	1	0	1
1	1	1	1	0	0	0	1	0	0	1	1	0	1	0
0	1	1	1	1	0	0	0	1	0	0	1	1	0	1
1	0	1	1	1	1	0	0	0	1	0	0	1	1	0
0	1	0	1	1	1	1	0	0	0	1	0	0	1	1
1	0	1	0	1	1	1	1	0	0	0	1	0	0	1
1	1	0	1	0	1	1	1	1	0	0	0	1	0	0
0	1	1	0	1	0	1	1	1	1	0	0	0	1	0
0	0	1	1	0	1	0	1	1	1	1	0	0	0	1
1	0	0	1	1	0	1	0	1	1	1	1	0	0	0
0	1	0	0	1	1	0	1	0	1	1	1	1	0	0
0	0	1	0	0	1	1	0	1	0	1	1	1	1	0
0	0	0	1	0	0	1	1	0	1	0	1	1	1	1

6. В псевдослучайной *M*-последовательности серии из одного символа (единицы либо нуля) встречаются 2^{m-2} раз, из двух единиц или нулей 2^{m-3} и т. д. Серии из $m - 1$ нулей и m единиц встречаются лишь по одному разу. Сравнивая выражения для оценки вероятности появления серий из одинаковых символов для случайных последовательностей с соответствующей вероятностью для *M*-последовательностей, можно убедиться в их практической эквивалентности.

7. Функция автокорреляции *M*-последовательности максимально близка к автокорреляционной функции идеальной случайной последовательности. Действительно, оригинальная *M*-последовательность является идентичной в $2^{m-1}-1$ позициях со сдвинутой своей копией на любое число тактов, и будет отличаться в 2^{m-1} позициях. Так, например, последовательность 000111101011001, генерируемая в соответствии с полиномом $\varphi(x) = 1 \oplus x \oplus x^4$, и ее сдвинутая на две позиции копия 011110101100100 совпадают на семи позициях и имеют различное значение на восьми позициях.

Математически автокорреляционная функция определяется выражением [301]:

$$R_b(\tau) = \begin{cases} 1 & \text{при} \quad \tau = 0 \bmod L, \\ -\dfrac{1}{L} & \text{при} \quad \tau \neq 0 \bmod L. \end{cases}$$

8. Свойство *сдвига и сложения*. Для любого s ($1 \le s < L$) существует $r \neq s$ ($1 \le r < L$) такое что $\{b_k\} \oplus \{b_{k-s}\} = \{b_{k-r}\}$, где операция вычитания выполняется по модулю $L = 2^m - 1$. Для случая *M*-последовательностей, описываемых полиномом $\varphi(x) = 1 \oplus x \oplus x^4$ (см. таблицу 7.3), при значениях $k = 4$ и $s = 2$ результатам поразрядного сложения по модулю два $\{b_4\}$ и $\{b_{4-2}\} = \{b_2\}$

будет $\{b_{4-9}\} = \{b_{10}\}$. Соответственно $r = 9$, как это видно из результата поразрядного сложения исходных последовательностей:

$\{b_4\}$ 000111101011001
$\{b_{4-2}\}$ 011110101100100
$\{b_{4-9}\}$ 011001000111101

9. Среди L M-последовательностей, полученных на основании примитивного полинома $\varphi(x)$, существует единственная последовательность, для которой выполняется равенство $b_k = b_{2k}$, $k = 0, 1, 2, \ldots$. Такая M-последовательность называется *характеристикой* (*characteristic M-sequence*) и получается следующим образом. Для заданного примитивного порождающего полинома $\varphi(x)$ строится система линейных уравнений $b_i = b_{2i}$, $i = 0, 1, 2, \ldots, m-1$. Ненулевое решение приведенной системы уравнений и есть искомая характеристическая последовательность.

Пример 7.4. В качестве примера рассмотрим процедуру определения характеристической последовательности для полинома $\varphi(x) = 1 \oplus x \oplus x^4$. Для этого случая система линейных уравнений имеет форму:

$$b_0 = b_0;$$
$$b_1 = b_2;$$
$$b_2 = b_4 = b_0 \oplus b_3;$$
$$b_3 = b_6 = b_2 \oplus b_5 = b_2 \oplus b_1 \oplus b_4 = b_2 \oplus b_1 \oplus b_0 \oplus b_3.$$

Единственное ненулевое решение $b_0\, b_1\, b_2\, b_3 = 0\ 1\ 1\ 1$, удовлетворяющее приведенной системе, и есть искомая характеристическая M-последовательность. В таблице 7.3 эта последовательность приведена под номером 2.

10. Свойство *децимаций* (*decimation*). Децимация M-последовательности $\{b_i\}$ по индексу q, ($q = 1, 2, 3, \ldots$) означает порождение другой последовательности $\{b_j\}$ в результате выборки q-тых элементов исходной M-последовательности $\{b_i\}$, что определяется равенством $b_j = b_{qi}$. Если наибольший общий делитель периода L M-последовательности и индекса q равен единице, то есть $(L, q) = 1$, то период $\{b_j\}$ будет равен $L = 2^m - 1$, и децимация называется нормальной.

Пример 7.5. В качестве примера рассмотрим нормальную децимацию M-последовательности $\{b_5\}$ с индексом 5, которая соответствует порождающему полиному $\varphi(x) = 1 \oplus x \oplus x^4$ и приведена в таблице 7.3. В качестве индекса q децимации используем $q = 2$. Так как для данного случая $(L, q) = (15, 2) = 1$, в результате децимации будет получена M-последовательность $\{b_{11}\}$, описываемая тем же порождающим полиномом (см. таблицу 7.3).

Таблица 7.4 – *Пример децимации М-последовательности*

i	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
$\{b_5\}$, b_i =	1	0	0	0	1	1	1	1	0	1	0	1	1	0	0
$\{b_3\}$, $b_j = b_{2i}$ =	1	0	1	1	0	0	1	0	0	0	1	1	1	1	0

Как видно из предыдущего примера, в результате децимации получили *M*-последовательность, которая описывается тем же порождающим полиномом. Результатом децимации может быть любая *M*-последовательность, соответствующая любому примитивному порождающему полиному той же степени *m.*

7.4. Генераторы нелинейных *M*-последовательностей

В общем случае генераторы нелинейных последовательностей строятся с использованием нелинейных зависимостей для получения очередного элемента последовательности. Чаще всего для этих целей используются регистры сдвига с нелинейной обратной связью. Структура подобного генератора включает в себе два основных блока, а именно: сдвиговой регистр на один разряд и комбинационную схему. Комбинационная схема необходима для реализации нелинейная функции, которая описывает обратную связь генератора. Аналогично регистрам сдвига с линейной обратной связью (*LFSR*) подобные устройства называются *регистрами сдвига с нелинейной обратной связью* (*nonlinear feedback shift register – NFSR*). Общий вид подобного устройства приведен на рисунке 7.5.

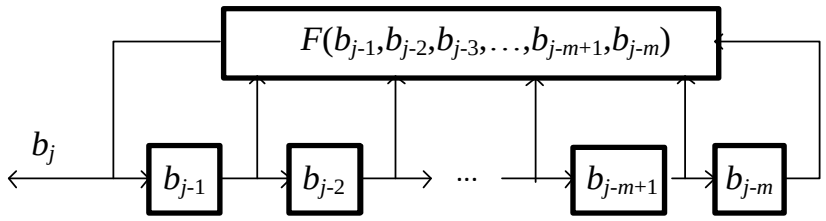


Рисунок 7.5 – *Регистр сдвига с нелинейной обратной связью*

Хорошо изученной разновидностью подобных генераторов являются генераторы *последовательностей де Брейна* (*de Bruijn sequences*), характерной чертой для которых является генерирование всевозможных 2^m двоичных комбинаций, где *m* является разрядностью регистра сдвига, на котором формируется последовательность де Брейна. Существует процедура преобразования *LFSR* в *NFSR*, генерирующего последовательность де Брейна. В этом случае нелинейная функция $F(b_{j-1}, b_{j-2}, b_{j-3}, \ldots, b_{j-m+1},b_{j-m})$ образуется как сумма по модулю два функции $f(b_{j-1}, b_{j-2}, b_{j-3}, \ldots, b_{j-m+1},b_{j-m})$, описывающей линейную обратную связь *LFSR*, с нелинейной функцией $g(b_{j-1}, b_{j-2}, b_{j-3}, \ldots, b_{j-m+1},b_{j-m}) = \overline{b_{j-1}}\ \overline{b_{j-2}}\ \overline{b_{j-3}}\ \ldots\ \overline{b_{j-m+2}}\ \overline{b_{j-m+1}}$.

Вид функции $g(b_{j-1}, b_{j-2}, b_{j-3}, \ldots, b_{j-m+1}, b_{j-m}) = \overline{b_{j-1}}\ \overline{b_{j-2}}\ \overline{b_{j-3}}\ \ldots\ \overline{b_{j-m+2}}\ \overline{b_{j-m+1}}$ свидетельствует о том, что данная нелинейная функция принимает единичное значение только на двух наборах переменных $b_{j-1}, b_{j-2}, b_{j-3}, \ldots, b_{j-m+1}, b_{j-m}$ (двух состояниях регистра сдвига), а именно: на наборе 000…01 и 000…00. Отметим, что в последовательности кодов, формируемых *LFSR*, присутствует только код 000…01, и нет больше кодов, у которых все первые $m-1$ бита принимают нулевое значение. Кроме того, независимо от вида конкретного примитивного порождающего полинома, функция $f(b_{j-1}, b_{j-2}, b_{j-3}, \ldots, b_{j-m+1}, b_{j-m})$, описывающая линейную обратную связь *LFSR*, всегда принимает единичное значение на двоичном наборе 000…01.

Таким образом, при появлении на регистре сдвига набора 000…01 обе функции g и f принимают единичное значение, а F, соответственно, нулевое, в силу чего на регистре сдвига будет получен код 000…00, который будет являться следующим набором аргументов для указанных функций g, f и F. В следующем цикле g будет равняться единице, а f – нулю. Тогда $F = 1$, и на регистре сдвига будет получен код 100…00.

Пример 7.6. Простейшим примером генератора последовательности де Брейна является генератор, представленный на рис. 7.6.

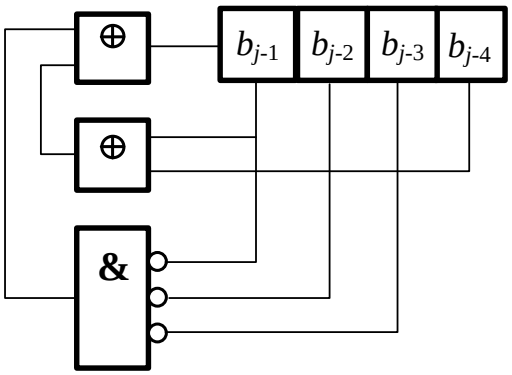


Рис.7.6. *Генератор последовательности де Брейна*

Данный генератор синтезирован на базе *LFSR*, соответствующего порождающему полиному $\varphi(x) = 1 \oplus x \oplus x^4$, линейная обратная связь которого описывается функцией $f(b_{j-1}, b_{j-2}, b_{j-3}, b_{j-4}) = b_{j-1} \oplus b_{j-4}$. Нелинейная функция $F(b_{j-1}, b_{j-2}, b_{j-3}, \ldots, b_{j-m+1}, b_{j-m})$ для данного примера принимает вид: $F(b_{j-1}, b_{j-2}, b_{j-3}, \ldots, b_{j-m+1}, b_{j-m}) = b_{j-1} \oplus b_{j-4} \oplus (\overline{b_{j-1}}\ \overline{b_{j-2}}\ \overline{b_{j-3}})$. Диаграмма состояний генератора приведена в таблице 7.5.

Последовательность де Брейна, как видно из приведенного примера, может быть получена путем добавления нулевого символа к последовательности из 2^m-1 бит, таким образом, чтобы на периоде ее повторения образовалась серия из m нулей. Такая последовательность соответственно имеет период, равный 2^m , и характеризуется наличием на интервале повторения всевозможных кодов длиной m, каждый из которых встречается только один раз.

Последнее свойство и определило интерес к использованию последовательностей де Брейна для различных приложений, предназначенных для тестирования современных вычислительных систем.

Таблица 7.5 – *Диаграмма состояний генератора последовательности де Брейна*

j	b_{j-1}	b_{j-2}	b_{j-3}	b_{j-4}	j	b_{j-1}	b_{j-2}	b_{j-3}	b_{j-4}
1	1	0	0	0	*9*	1	1	0	1
2	1	1	0	0	*10*	0	1	1	0
3	1	1	1	0	*11*	0	0	1	1
4	1	1	1	1	*12*	1	0	0	1
5	0	1	1	1	*13*	0	1	0	0
6	1	0	1	1	*14*	0	0	1	0
7	0	1	0	1	*15*	0	0	0	1
8	1	0	1	0	*16*	0	0	0	0

Частным случаем последовательностей де Брейна являются так называемые *последовательности Форда* (*Ford sequences*), которые описываются более простым алгоритмом формирования. Последний носит рекуррентный характер и заключается в формировании очередного m-разрядного кода $X_k = (b_2, b_3\ b_4, \ldots, b_{m+1})$ на основе предыдущего кода $X_{k-1} = (b_1, b_2, b_3, \ldots, b_m)$, где значение $b_{m+1} \in \{0, 1\}$ определяется следующим образом:

1. Формируется код вида $X^*_{k-1} = (b_2, b_3, b_4, \ldots, b_m, 1)$.

2. Строятся всевозможные циклические сдвиги кода X^*_{k-1}. Среди них выбирается код $X^{**}_{k-1} = (b_i, b_{i+1}, \ldots, b_m, 1, b_2, \ldots, b_{i-1})$, представляющий собой наибольшее m-разрядное число.

3. Если $b_2 = \ldots = b_{i-1} = 0$, то значение b_{m+1} равняется $b_1 \oplus 1$, в противном случае b_{m+1} равно b_1. Отметим, что начальным кодом X_0 для формирования последовательности Форда может быть любой код, включая $X_0 = (0, 0, 0, \ldots, 0)$.

В качестве примера в таблице 7.6 приведены результаты формирования последовательности Форда для $m = 4$.

Весьма близкими по свойствам к M-последовательностям и находящими наряду с ними широкое практическое применение являются *последовательности Голда и Касами.* Последовательность Голда образуется путем суммирования по модулю 2 двух M-последовательностей, порождаемых отличными примитивными полиномами $\varphi'(x)$ и $\varphi''(x)$ одной и той же степени m. Период данной последовательности равен $2^m - 1$, а порождающий ее полином представляет собой произведение $\varphi'(x)$ на $\varphi''(x)$.

Более широким классом последовательностей, включающим в качестве подмножества последовательности Голда, является множество последовательностей Касами. Последние могут быть получены в результате двоичного сложения трех M-последовательностей $\{a_i\}$, $\{b_i\}$ и $\{c_i\}$, порождаемых полиномами $\varphi'(x)$ и $\varphi''(x)$ степени m и $\varphi^*(x)$ степени $m/2$ соответственно,

где m – четно. Наиболее важное качество последовательностей Касами, а также входящих в них подмножеств последовательностей, является их высокие корреляционные свойства.

Таблица 7.6 – *Диаграмма состояний генератора последовательности Форда*

k	$X_k=(b_2,b_3,b_4,\ldots,b_{m+1})$	$X^{**}_{k-1}=(b_i,b_{i+1},\ldots,b_m,1,b_2,\ldots,b_{i-1})$	b_{m+1}
0	0 0 0 0	1 0 0 0	$b_1\oplus1=0\oplus1=1$
1	0 0 0 1	1 1 0 0	$b_1\oplus1=0\oplus1=1$
2	0 0 1 1	1 1 1 0	$b_1\oplus1=0\oplus1=1$
3	0 1 1 1	1 1 1 1	$b_1\oplus1=0\oplus1=1$
4	1 1 1 1	1 1 1 1	$b_1\oplus1=1\oplus1=0$
5	1 1 1 0	1 1 1 0	$b_1=1$
6	1 1 0 1	1 1 1 0	$b_1=1$
7	1 0 1 1	1 1 1 0	$b_1\oplus1=1\oplus1=0$
8	0 1 1 0	1 1 1 0	$b_1=0$
9	1 1 0 0	1 1 0 0	$b_1=1$
10	1 0 0 1	1 1 0 0	$b_1\oplus1=1\oplus1=0$
11	0 0 1 0	1 0 1 0	$b_1\oplus1=0\oplus1=1$
12	0 1 0 1	1 1 1 0	$b_1=0$
13	1 0 1 0	1 0 1 0	$b_1\oplus1=1\oplus1=0$
14	0 1 0 0	1 1 0 0	$b_1=1$
15	1 0 0 0	1 0 0 0	$b_1\oplus1=1\oplus1=0$
16	0 0 0 0	1 0 0 0	

К псевдослучайным последовательностям могут быть также отнесены многоуровневые последовательности максимальной длины. В общем случае r-уровневая последовательность имеет период $L = r^m-1$, где m представляет собой степень примитивного полинома $\varphi(x)$ над полем $GF(r)$. Таблицы примитивных многочленов для различных значений r представлены в работах [280, 287]. Аппаратурная реализация генератора r-уровневой M-последовательности состоит из r-разрядного регистра сдвига, ячейки которого содержат элементы поля $GF(r)$, охваченного обратной связью в соответствии с порождающим полиномом $\varphi(x)$.

На рис. 7.7 показана структурная схема генератора трехуровневой M-последовательности. Элементами ее являются −1, 0 и +1, а значение очередного символа b_i определяется рекуррентным соотношением:

$$b_i = -1\times b_{i-1} + 1\times b_{i-2} \quad GF(3). \tag{7.6}$$

Таблица истинности используемых в выражении (7.6) операций сложения и умножения в поле $GF(3)$ представлена ниже (таблица 7.7).

В трехуровневой M-последовательности содержится на периоде повторения по 3^{m-1} символов +1 и −1, а также $3^{m-1}-1$ символов 0. Каждое из возможных m-разрядных сочетаний элементов +1, −1 и 0, исключая код, состо-

ящий только из нулей, встречается на периоде лишь один раз. Автокорреляционная функция последовательности определяется выражением [68]:

$$K(\tau)=\frac{1}{3^m-1}\sum_{i=0}^{3^m-1}b_i b_{i-\tau}=\begin{cases}\dfrac{2\times 3^{m-1}}{3^m-1} & \text{при} \quad \tau=0, L, 2L, \dots,\\ -\dfrac{2\times 3^{m-1}}{3^m-1} & \text{при} \quad \tau=\dfrac{L}{2},\dfrac{3L}{2},\dfrac{5L}{2},\dots,\\ 0 & \text{для остальных значений } \tau,\end{cases}$$

где $L = 3^m-1$.

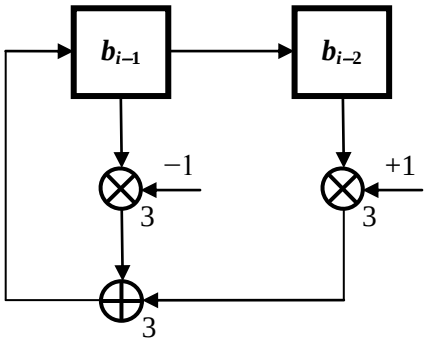


Рисунок 7.7 – *Генератор трехуровневой М-последовательности*

Более детально свойства трехуровневых *М*-последовательностей изложены в [68].

Таблица 7.7 – *Таблица истинности операций* $S=a\oplus_3 b$ и $Q=a\otimes_3 b$

a	*b*	*S*	*Q*
+1	+1	−1	+1
+1	0	+1	0
+1	−1	0	−1
0	+1	+1	0
0	0	0	0
0	−1	−1	0
−1	+1	0	−1
−1	0	−1	0
−1	−1	+1	+1

Необходимо отметить, что, хотя использование многоуровневых *М*-последовательностей в задачах идентификации и тестирования вычислительных систем обещает ряд преимуществ, они не получили широкого применения. Главной причиной является техническая сложность их реализации.

Весьма широкое практическое применение для различных целей находят в настоящее время комбинированные числовые последовательности,

формируемые из дискретных отсчетов значений случайного и псевдослучайного типов и сочетающие, таким образом, их основные положительные качества [282, 297, 298, 300].

7.5. Комбинированные псевдослучайные тестовые последовательности

В последнее время большое внимание уделяется вопросам создания методов и средств проверки функционирования сложных вычислительных систем, имитационного моделирования, построению систем связи с использованием шумоподобных сигналов и т. д. Решение вышеназванных задач предопределяет необходимость создания быстродействующих высокоэкономичных и надежных генераторов равномерно распределенных псевдослучайных тестовых воздействий [208, 248, 249, 301, 302, 306, 308]. При этом все более возрастающую роль приобретают требования к качеству псевдослучайных последовательностей, особенно при построении автоматизированных систем тестового диагностирования вычислительных систем, основанных на вероятностных принципах, которые, как отмечалось ранее, характеризуются рядом существенных преимуществ по сравнению с традиционными методами тестового диагностирования.

Методы вероятностного тестирования предполагают наличие генераторов псевдослучайных тестовых последовательностей, которые в подавляющем большинстве случаев строятся с использованием *M*-последовательностей либо их модификаций. Несомненно, генераторы псевдослучайных тестовых воздействий, рассмотренные в [86, 207, 208, 301, 302], отличаются рядом весьма существенных достоинств по сравнению с другими известными устройствами подобного типа. В тоже время использование таких устройств, в частности, при построении современных автоматизированных систем тестового диагностирования вычислительных систем, накладывает ограничения принципиального характера на вид тестируемого объекта. Причина возникновения ограничений на вид контролируемых цифровых устройств вычислительных систем заключается в детерминированной структуре псевдослучайных последовательностей. Для любого генератора псевдослучайных последовательностей можно утверждать, что после определенного кода X_k на его выходе всегда будет следовать X_{k+1} заранее однозначно предопределенное. В тоже время в физических генераторах случайных тестовых последовательностей X_{k+1} может принимать любое значение из множества возможных. Влияние детерминированной структуры псевдослучайных тестовых последовательностей на их качественные показатели поясним на примере тестирования фрагмента цифрового устройства, приведенного на рис. 7.8, вероятностными методами тестирования.

На приведенном рисунке представлен фрагмент цифрового устройства, состоящего из элементов задержки на один такт *1* и *2* (D триггеры), элемента 2И *3,* элемента 2ИЛИ *4* и двух двухвходовых сумматоров по модулю два с инверсными выходами *5*, *6*. Предположим, что на входы 1 и 2 цифрового

устройства подключены выходы $b_{j-i}(k)$ и $b_{m-i}(k)$-го разрядов i = 0, 1, ..., j−1 генератора тестовых сигналов [230,231], последовательность состояний которых описывается следующей системой логических уравнений:

$$\begin{aligned} b_{m-i}(k+1) &= b_{j-i}(k) \oplus b_{m-i}(k), \quad i = 0, 1, ..., j-1; \\ b_{m-i}(k+1) &= b_{2j-i}(k) \oplus b_{m+j-i}(k) \oplus b_{m-i}(k), \quad i = j, j+1, ..., m-1, \end{aligned} \tag{7.7}$$

где $b_{m-i}(k + 1) \in \{0, 1\}$ является содержимым $m - i$ -го разряда генератора тестов в $k + 1$ такт его работы; m представляет собой разрядность генератора; значение j определяется видом порождающего характеристического полинома $\varphi(x) = 1 \oplus x^j \oplus x^m$, зависит от величины m, причем необходимо, чтобы $j > m/2$ [326]. Учитывая то, что элементы задержки *1* и *2* задерживают информацию на один такт, в некоторый (k + 1)-ый такт тестирования цифрового устройства на выходах элементов *1* и *2* будут значения $b_{j-i}(k)$ и $b_{m-i}(k)$, соответственно.

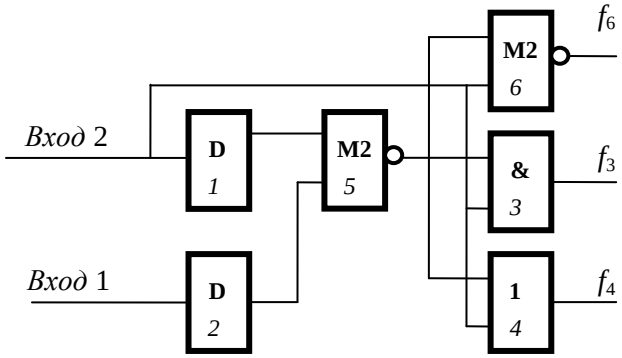


Рисунок 7.8 – *Фрагмент цифрового устройства*

С учетом (7.7) значение переключательной функции на выходе элемента *5* будет определяться как $\overline{b_{j-i}(k) \oplus b_{m-i}(k)}$, что в соответствии с (7.7) равняется $\overline{b_{m-i}(k+1)}$. В тоже время значение переменной на входе *2* определятся $b_{m-i}(k + 1)$. Таким образом, для переключательных функций на выходах элементов *3*, *4*, *6* можно записать

$$\begin{aligned} f_3 &= b_{m-i}(k+1)\overline{b_{m-i}(k+1)} = 0; \\ f_4 &= b_{m-i}(k+1) + \overline{b_{m-i}(k+1)} = 1; \\ f_6 &= \overline{b_{m-i}(k+1) \oplus \overline{b_{m-i}(k+1)}} = 0. \end{aligned}$$

Постоянные значения переключательных функций f_3, f_4 и f_6 говорят о том, что неисправности вида ≡ 1 по выходам элементов *3* и *6*, а также ≡ 0 на выходе элемента *4* при сколь угодно больших статистиках входных тестовых последовательностей не обнаруживаются. При использовании, например, сигнатурного анализа значения контрольных сумм (сигнатур) для исправного

цифрового устройства (рис. 7.8) и его модификаций с указанными неисправностями будут идентичны, что будет свидетельствовать о исправности неисправного цифрового устройства.

Причина появления подобных пассивных цепей в тестируемых цифровых устройствах обусловлена детерминированной природой входных псевдослучайных тестовых сигналов [230]. Введение большого количества промежуточных контрольных точек, использование синхронных элементов памяти, пере коммутация выходных каналов генератора несколько снижает значение P вероятности появления пассивных цепей в контролируемых схемах, однако принципиально $P \neq 0$ [301, 302, 325].

Подобным образом можно привести большое множество примеров, показывающих ограниченность применения псевдослучайных сигналов в силу их детерминированности. В частности, как показано в [280] по тем же причинам псевдослучайные последовательности нельзя использовать для решения серьезных задач криптографии.

Стремление улучшить качество тестовых последовательностей приводит к необходимости использования физических принципов при построении генераторов исходных равномерно распределенных чисел (двоичных наборов). Однако необходимость обеспечения максимального быстродействия и требуемой разрядности, достигающей 180 разрядов [244], приводит к существенному усложнению аппаратурной реализации подобных устройств.

Наиболее экономично многоразрядные тестовые последовательности равномерно распределенных наборов могут быть получены при помощи комбинированного подхода использующего псевдослучайные и случайные последовательности одновременно [253, 270]. Преимущества данного подхода заключаются в том, что природа выходных последовательностей максимально приближена к истинно случайным, при этом аппаратурные затраты практически не отличаются от затрат на построение генераторов псевдослучайных последовательностей. В тоже время невозможность получения на выходе устройства, описанного в [253, 270] m-разрядного псевдослучайного числа $X_k = 000 \ldots 0$, приводит к искажению равномерного закона распределения. Однако более существенным недостатком подхода, описанного в [270], является ограниченность его функциональных возможностей. При практической реализации генератора тестовых последовательностей, подобно как в [270], оказывается возможным построение генератора только для узкого класса примитивных полиномов, имеющих вид $\varphi(x) = 1 \oplus x^j \oplus x^m$.

Комбинированными генераторами псевдослучайных тестовых последовательностей называются устройства, включающие в свой состав как физические датчики случайных последовательностей, так и генераторы псевдослучайных последовательностей, сочетающие в себе таким образом преимущества обоих видов генерирования [228, 253, 255, 297, 298, 270,]. Формируемые такими устройствами тестовые последовательности характеризуются отсутствием периодичности, возможностью получения нулевого тестового набора, появлением любого из генерируемых тестовых наборов повторно че-

рез случайное число тактов, равновероятностью двоичных символов и низким уровнем авто- и взаимной корреляции между наборами.

Рассмотрим методику синтеза генератора комбинированных псевдослучайных тестовых последовательностей, отличающегося рядом весьма значительных достоинств. Принцип действия подобного генератора основан на рандомизации последовательности де Брейна [92].

Если все 2^m двоичные комбинации входят в циклическую последовательность m разрядных тестовых наборов, то такие последовательности, как отмечалось ранее, называются последовательностями де Брейна или циклами де Брейна. Очевидно, что использование подобных последовательностей в силу плохих корреляционных свойств в чистом виде нецелесообразно. В тоже время введение рандомизации таких последовательностей позволит получить абсолютную равномерность и независимость [92, 301]. Введение нулевой комбинации в M-последовательность позволит получить подобный цикл, содержащий всевозможные m-разрядные двоичные комбинации. Функционирование устройства, генерирующего на выходе последовательность де Брейна, с периодом 2^m, описывается выражением:

$$b_i(k+1)=\begin{cases}\sum\limits_{n=1}^{m}{}^{\oplus}\,\alpha_n b_n(k)\oplus\overline{b_1(k)+b_2(k)+\ldots+b_{m-1}(k)}; & i=1,\\ b_{i-1}(k); \quad i=\overline{2,m}, & k=0,1,2,\ldots.\end{cases} \tag{7.8}$$

Постоянные коэффициенты $\alpha_n\in\{0,\ 1\}$ определяются примитивным порождающим полиномом $\varphi(x)$.

Используя методику, приведенную в [326], можно показать зависимость состояний m-го разряда генератора от значений других разрядов и количества выполненных тактов. Так, для $n < m$:

$$b_m(k+n)=b_{m-1}(k+n-1)=\ldots=b_{m-i}(k+n-i)=\ldots=b_{m-n}(k).$$

Тогда через $n = m - 1$ тактов, согласно (7.8):

$$b_m(k+m)=b_1(k+1)=\sum_{n=1}^{m}{}^{\oplus}\,\alpha_n b_n(k)\oplus\overline{b_1(k)+b_2(k)+\ldots+b_{m-1}(k)}. \tag{7.9}$$

Подобным образом легко показать, что:

$$\begin{aligned}b_{m-1}(k+m)&=\alpha_1 b_1(k+1)\oplus\sum_{n=1}^{m-1}{}^{\oplus}\,\alpha_{n+1}b_n(k)\oplus\overline{b_1(k+1)+b_2(k+1)+\ldots+b_{m-1}(k+1)}=\\ &\alpha_1 b_m(k+m)\oplus\sum_{n=1}^{m-1}{}^{\oplus}\,\alpha_{n+1}b_n(k)\oplus\overline{b_m(k+m)+b_1(k)+\ldots+b_{m-2}(k)}.\end{aligned} \tag{7.10}$$

Для $b_{m-2}(k+m)$ получим:

$$b_{m-2}(k+m)=\alpha_1 b_{m-1}(k+m)\oplus\alpha_2 b_m(k+m)\oplus\sum_{n=1}^{m-2}{}^{\oplus}\,\alpha_{n+2}b_n(k)\oplus$$
$$\oplus\,\overline{b_m(k+m)+b_{m-1}(k+m)+b_1(k)+\ldots+b_{m-3}(k)}. \qquad (7.11)$$

Обобщая выражения (7.9), (7.10) и (7.11), можно записать:

$$b_{m-i}(k+m)=\sum_{n=1}^{i}{}^{\oplus}\,\alpha_{i+1-n}b_{m+1-n}(k+m)\oplus\sum_{n=1}^{m-i}{}^{\oplus}\,\alpha_{n+i}b_n(k)\oplus$$
$$\oplus\,\overline{b_m(k+m)+\ldots+b_{m+1-i}(k+m)+b_1(k)+\ldots+b_{m-i-1}(k)};\quad i=\overline{1,m-2}.$$

При $i = m - 1$ для $b_{m-i}(k+m)$ получим:

$$b_{m-m+1}(k+m)=b_1(k+m)=\sum_{n=1}^{m-1}{}^{\oplus}\,\alpha_{m-n}b_{m+1-n}(k+m)\oplus\alpha_m b_1(k)\oplus$$
$$\oplus\,\overline{b_m(k+m)+b_{m-1}(k+m)+b_{m-2}(k+m)+\ldots+b_2(k+m)}.$$

Окончательно система уравнений, описывающая формирование m-разрядного кода последовательности де Брейна, имеет вид:

$$b_{m-i}(k+m)=\begin{cases}\sum_{n=1}^{m}{}^{\oplus}\,\alpha_n b_n(k)\oplus\overline{b_1(k)+\ldots+b_{m-1}(k)}, & i=0;\\ \sum_{n=1}^{i}{}^{\oplus}\,\alpha_{i+1-n}b_{m+1-n}(k+m)\oplus\sum_{n=1}^{m-i}{}^{\oplus}\,\alpha_{n+i}b_n(k)\oplus & \\ \overline{b_m(k+m)+\ldots+b_{m+1-i}(k+m)+b_1(k)+\ldots+b_{m-i-1}(k)}, & i=\overline{1,m-2};\\ \sum_{n=1}^{m-1}{}^{\oplus}\,\alpha_{m-n}b_{m+1-n}(k+m)\oplus\alpha_m b_1(k)\oplus\overline{b_m(k+m)+\ldots+b_2(k+m)}, & i=\overline{m-1}.\end{cases} \qquad (7.12)$$

Реализуя систему уравнений (7.12) и дополняя устройство физическим генератором равновероятных двоичных цифр (ГРДЦ), получим комбинированный генератор тестовых воздействий, структурная схема которого изображена на рис. 7.9.

Блок *1* представляет собой статический регистр, на который записывается очередной тестовый набор, при подаче тактового импульса на управляющие входы триггеров регистра. Комбинационные схемы КС1 *3* и КС2 *2* построены согласно системе уравнений (7.12) и предназначены для формирования последовательных m-разрядных сегментов $\{b_{m-i}(k+m)\}$ и $\{b_{m-i}(k+2m)\}$, $i=\overline{0,m-1}$, $k=0,1,2,\ldots$. При поступлении тактового импульса в зависимости от значения равновероятной двоичной цифры, полученной на выходе блока *7* m-разрядный набор $\{b_{m-i}(k+m)\}$ или $\{b_{m-i}(k+2m)\}$ через m элементов И блока *4* или *5* и m элементов ИЛИ блока *6* записываются в регистр *1*. В тоже время, этот набор, формируемый на выходе устройства, представляет собой очередной тестовый набор X_k. Блок 7 является генератором последовательности

случайных равновероятных двоичных цифр $\{a(k)\}$, $k = 0, 1, 2, \ldots$ и их инверсных значений.

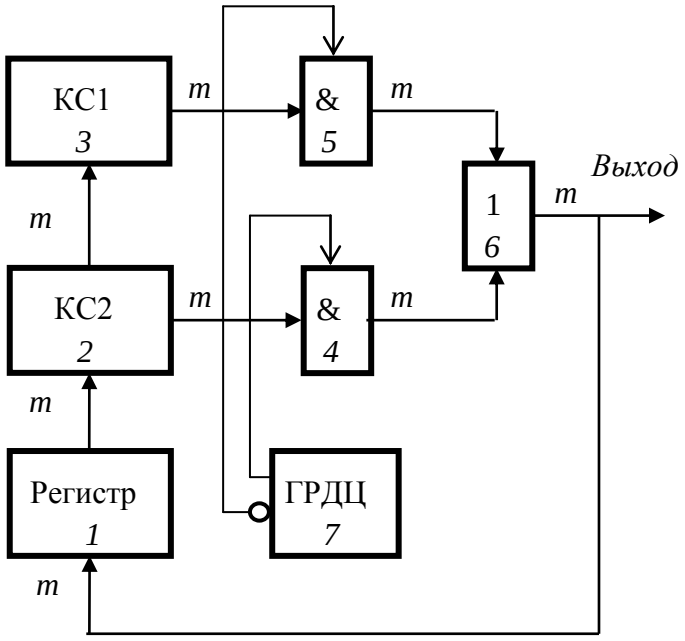


Рисунок 7.9 – *Комбинированный генератор тестовых последовательностей*

Рассмотрим свойства последовательности $\{X_k\}$, формируемой комбинированным генератором тестовых последовательностей [92, 323].

1. Период L последовательности $\{X_k\}$ равен бесконечности, так как порядком следования сегментов $\{b_{m-i}(k+m)\}$ и $\{b_{m-i}(k+2m)\}$, $i=\overline{0,m-1}$, $k=0,1,2,...$ управляет генератор случайных равновероятных двоичных цифр *7*, построенный на физических принципах и произвольно определяющий чередование кодов последовательности де Брейна.

2. Вероятностное описание последовательностей $\{X_k\}$, представляющее собой циклическое случайное блуждание, достигается заданием начальных вероятностей и матрицы одношаговых вероятностей бистохастического циркулянта. Для некоррелированной последовательности $\{a(k)\}$, $k = 0, 1, 2, \ldots$ бистохастический циркулянт $G(p, q) = \{g_{i,c}\}$, $q = 1 - p$; $i,c=\overline{1,2^m}$, имеет вид [243]:

$$G(p,q)=\begin{vmatrix} 0 & p & q & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & p & q & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & p & q \\ q & 0 & 0 & 0 & \cdots & 0 & 0 & p \\ p & q & 0 & 0 & \cdots & 0 & 0 & 0 \end{vmatrix}. \qquad (7.13)$$

Элементы бесконечного множества тестовых наборов $\{X_k\}$ представляют собой m-разрядные коды последовательности де Брейна $\{\eta_j\} = \{b_j, b_{j+1}, b_{j+2}, \ldots, b_{j+m-1}\}$, $j = \overline{0, 2^m - 1}$. Известно, что для бистахостического циркулянта финальным вектором вероятностей нахождения генератора комбинированных псевдослучайных тестовых последовательностей в η_j-ом состоянии является вектор $1/2^m, 1/2^m, \ldots, 1/2^m$ [243]. Иными словами:

$$\lim_{k\to\infty} P(X_k = \eta_j) = 1/2^m, 1/2^m, \ldots 1/2^m, \quad j = \overline{0, 2^m - 1}.$$

Скорость сходимости $P(X_k = \eta_j)$ к величине $1/2^m$ определяется функционалом [276]:

$$\chi(G^k(p,q)) = \max_c (\max_{i,l} |b_{i,c} - b_{l,c}|),$$

который для стохастической $2^m \times 2^m$ матрицы $G(p, q)$ (7.13) имеет вид:

$$\chi(G^k(p,q)) = (|p-q|^{\lfloor k/m \rfloor} \gamma_c |p-q|^{\lfloor k/m \rfloor}),$$

где $\gamma \le 1$; $\lfloor k/m \rfloor$- целая часть числа k/m.

Таким образом, появление на выходе устройства любого m-разрядного тестового набора равновероятно, так как значения последовательности $\{X_k\}$ представляют собой всевозможные m-разрядные сегменты цикла де Брейна. Для малых значений k равномерность распределения последовательности $\{X_k\}$ будет определяться равномерностью распределения последовательности $\{\eta_j\} = \{b_j, b_{j+1}, b_{j+2}, \ldots, b_{j+m-1}\}$, $j = \overline{0, 2^m - 1}$.

3. Математическое ожидание последовательности $\{X_k\}$ равно:

$$M[X_k] = \sum_{n=1}^{2^m} \eta_n P(X_k = \eta_n) = \frac{1}{2}(2^m - 1).$$

4. Дисперсия $\{X_k\}$ определяется как:

$$D[X_k] = M[X_k^2] - M^2[X_k] = \frac{1}{12}(2^{2m} - 1).$$

5. Выражение для автокорреляционной функции имеет вид:

$$K_X(\tau) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} X_k^0 X_{k+\tau}^0, \qquad (7.14)$$

где X_k^0 – центрированная случайная величина, каждому значению которой при конкретном *k* соответствует:

$$\eta_{j(k)}^0 :\ X_k^0 = \eta_{j(k)}^0, \quad j(k) \in \{0,1,...,2^m - 1\}, k = 0,1,\ldots.$$

Очевидно, что величине $X_{k+\tau}^0$ будет соответствовать $\eta_{j(k)+\tau+l\,(\mathrm{mod}\,2^m)}^0$, $l \in \{0,1,...,\tau\}$. Тогда выражение (7.14) запишется в виде:

$$K_X(\tau) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} \eta_{j(k)}^0 \left[\sum_{l=0}^{\tau} C_\tau^l p^{\tau-l} q^l \eta_{j(k)+\tau+l(\mathrm{mod}\,2^m)}^0 \right].$$

Изменив порядок суммирования в последнем выражении и преобразовав, получим:

$$K_X(\tau) = \sum_{l=0}^{\tau} C_\tau^l p^{\tau-l} q^l \lim_{n\to\infty} \frac{1}{n} \sum_{\substack{k=0 \\ j(k)<2^m}}^{n-1} \eta_{j(k)}^0 \eta_{j(k)+\tau+l(\mathrm{mod}\,2^m)}^0.$$

Выражение:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{\substack{k=0 \\ j(k)<2^m}}^{n-1} \eta_{j(k)}^0 \eta_{j(k)+\tau+l(\mathrm{mod}\,2^m)}^0,$$

можно оценить выборочной автокорреляционной функцией *M*-последовательности $K_\eta(\tau + l)$, которая при $1 \le \tau + l \le 2^m/m$ имеет математическое ожидание $M[K_\eta(\tau + l)] = 0$ и дисперсию $D[K_\eta(\tau + l)] \cong 1/n$. Таким образом:

$$K_X(\tau) = \sum_{l=0}^{\tau} C_\tau^l p^{\tau-l} q^l K_\eta(\tau + l).$$

Используя формулу Стирлинга, можно показать, что:

$$C_\tau^l = \frac{\sqrt{\tau} \times \tau^\tau}{\sqrt{2\pi} \times l^{l+1/2} \times (\tau - l)^{\tau-l+1.2}},$$

причем величина C_τ^l, $l \in \{0,1,...,\tau\}$ оценивается, как:

$$\max C_\tau^l = C_\tau^{\tau/2}, \quad \text{где} \quad C_\tau^{\tau/2} = \sqrt{2/\pi} \times 2^\tau / \sqrt{\tau}.$$

Произведение $p^{\tau-l}q^l$ можно оценить $q^\tau = (1/2+\delta)^\tau$, где $\delta = |q - p|/2$.

Окончательно для $K_X(\tau)$ получим:

$$K_X(\tau) \le \sum_{l=0}^{\tau} \sqrt{2/\pi}\,\frac{2^{\tau}}{\sqrt{\tau}}(1/2+\delta)^{\tau} K_{\eta}(\tau+l). \qquad (7.15)$$

Из выражения (7.15) следует, что для $\tau < 2^{m-}1/m$ значение $K_X(\tau) \cong 0$, так как $K_\eta(\tau+l) \cong 0$. Для $\tau > 2^{m-}1/m$, предполагая, что $\delta \to 0$ и $K_\eta(\tau+l)$ имеет нулевое значение во всей области определения, окончательно получим, что $K_X(\tau) \cong 0$.

Свойства последовательности $\{X_k\}$ максимально близки к свойствам последовательности случайных чисел, полученных с использованием идеального генератора построенного на физических принципах. В тоже время, представленный на рис.7.9 комбинированный генератор тестовых последовательностей отличается стабильностью характеристик, быстродействием и сравнительно невысокой аппаратурной сложностью, что присуще устройствам для формирования псевдослучайных тестовых последовательностей.

7.6. Свойства псевдослучайных тестовых последовательностей

Использование метода (7.4) открывает широкие возможности по формированию не только бинарных *M*-последовательностей, символы которых принимают лишь два значения, но и последовательностей многоразрядных псевдослучайных тестовых последовательностей. В качестве последних могут рассматриваться двоичные коды, описывающие состояния регистра сдвига генератора *M*-последовательности в различные моменты времени. Поскольку период генерируемой *M*-последовательности достигает значения $L = 2^m - 1$, где $m = \deg \varphi(x)$, *m*-разрядные коды могут быть названы псевдослучайными числовыми последовательностями максимальной длины. Рассмотрим их основные свойства [189, 300].

Пусть последовательность $X_0, X_1, \ldots, X_k, \ldots, X_{n-1}$ представляет собой *r*-разрядные тестовые наборы $r \in \{1, 2, \ldots, m\}$, лежащие в интервале (0, 1) и сформированные из *r* последовательных символов *M*-последовательности. Таким образом, что $X_k = 0, b_{k+l-1}\, b_{k+l-2} \ldots b_{k+l-r}$, где b_{k+l-1} представляет собой символ *M*-последовательности, а *l* – случайное число из интервала $(0, 2^m - 1)$. Тогда усредненные по всем начальным значениям X_0, определяемым значениями *l*, статистические характеристики последовательности имеют следующий вид.

Частота ω попадания X_k в интервал единичного отрезка, для которого зафиксированы первые *d* разрядов в двоичном представлении, то есть длиной 2^{-d}, имеет среднее значение [300]:

$$M[\omega] = 2^{-d}\left(1 + \frac{1}{2^m - 1}\right) - \beta\frac{1}{2^m - 1} \cong 2^{-d}$$

для любого числа кодов X_k где $\beta = 1$, если все *d* разрядов, задающих интервал, равны нулю и $\beta = 0$ в остальных случаях.

Дисперсия ω ограничена величиной:

$$D[\omega] < \frac{1}{4} \times \left(\frac{1}{n} + \frac{2}{2^m - 1} \right) \times \left(1 + \frac{1}{2^m - 1} \right) \cong \frac{1}{4^n}.$$

Частота попадания кодов (X_{k+1}, X_{k+2}, …, X_{k+g}) в интервал из единичного g-мерного куба, для которого фиксированы первые d_i разрядов представления X_{k+i}, т. е. в интервале объемом $2^{-d_1} \times 2^{-d_2} \times ... \times 2^{-d_g}$, имеет среднее значение:

$$M'[\omega] = 2^{-(d_1 + d_2 + ... + d_g)} \left(1 + \frac{1}{2^m - 1} \right) - \beta \frac{1}{2^m - 1} \cong 2^{-(d_1 + d_2 + ... + d_g)}$$

для любого числа n кодов $\{X_{k+1}, X_{k+2}, \ldots, X_{k+g}\}$, где $\beta = 1$, если все $(d_1 + d_2 + \ldots + d_g)$ разрядов, задающих интервал, равны нулю и $\beta = 0$ в остальных случаях.

Дисперсия ω ограничена величиной:

$$D'[\omega] < \frac{1}{4} \times \left(\frac{1}{n} + \frac{2}{2^m - 1} \right) \times \left(1 + \frac{1}{2^m - 1} \right) \cong \frac{1}{4^n}.$$

Для центрированной последовательности тестовых наборов $X_1^0, X_2^0, ..., X_k^0$, изменяющихся в интервале (−1, +1) и полученных в соответствии с соотношением $X_k^0 = 1 - 2X_k - 2^{-r}$, статистические характеристики имеют вид:

$$M[X_k^0] = \frac{1 - 2^{-r}}{2^m - 1} \cong 0,$$

$$D[X_k^0] = \frac{1}{3} + \left[\frac{1}{3} \frac{1 - 2^{-2r}}{2^m - 1} - \frac{1 - 2^{-r}}{2^m - 1} - \left(\frac{1 - 2^{-r}}{2^m - 1} \right)^2 \right] \cong \frac{1}{3}.$$

Для анализа автокорреляционной функции псевдослучайных тестовых последовательностей, формируемых из M-последовательности $\{b_i\}$, $i = 0, 1, 2, \ldots$, преобразуем последнюю в последовательность $\{c_i\}$, в соответствии с соотношением $c_i = 1 - 2b_i$. Тогда значения тестовых наборов псевдослучайной последовательности будут определяться из соотношения:

$$X_k = \sum_{n=0}^{r-1} 2^n c_{k-n}. \qquad (7.16)$$

Периодическая автокорреляционная функция последовательности X_0, X_1, X_2, …, X_k, …, X_{n-1} определяется следующим образом [192]:

$$K_X(\tau) = \frac{1}{L} \sum_{k=0}^{L-1} X_k X_{k+\tau}. \qquad (7.17)$$

Подставив выражение (7.16) в (7.17), получим:

$$K_X(\tau) = \sum_{n=0}^{r-1}\sum_{\gamma=0}^{r-1} 2^{n+\gamma} K_b(\tau + n - \gamma). \tag{7.18}$$

Поскольку функция $K_b(\tau)$ (см. раздел 7.3, свойство 7) четная и периодическая, то $K_X(\tau)$ также будет четной и периодической. Следовательно, для определения $K_X(\tau)$ достаточно найти ее значения при $0 \le \tau \le L/2$. Учитывая, что на основании свойства 7, описанного в разделе 7.3, функция $K_b(\tau + n - \gamma)$ для $0 \le \tau \le r-1$ принимает $r-\tau$ единичных значений выражение (7.18) для $0 \le \tau \le r-1$ преобразуется к виду:

$$K_X(\tau) = \frac{L+1}{L}\sum_{n=0}^{r-1-\tau} 2^{2n+\tau} - \frac{1}{L}\sum_{n=0}^{r-1}\sum_{\gamma=0}^{r-1} 2^{n+\gamma}, \quad \tau = \overline{0, r-1}.$$

Значения сумм в последнем выражении вычисляется по формулам для геометрических прогрессий. В результате выражение для автокорреляционной функции примет вид:

$$K_X(\tau) = \begin{cases} \dfrac{L+1}{L}\dfrac{2^{2r-\tau} - 2^r}{2^2 - 1} - \dfrac{1}{L}\dfrac{2^r - 1}{2 - 1}, & \tau = \overline{0, r-1}, \\ -\dfrac{1}{L}\left(\dfrac{2^r - 1}{2 - 1}\right)^2, & \tau = \overline{r, (L-1)/2}. \end{cases} \tag{7.19}$$

Тогда нормированная автокорреляционная функция запишется как:

$$R_X(\tau) = \frac{K_X(\tau)}{K_X(0)} = \begin{cases} \dfrac{2^{2r-\tau} - 2^r - B}{2^{2r} - 1 - B}, & \tau = \overline{0, r-1}, \\ -\dfrac{B}{2^{2r} - 1 - B}, & \tau = \overline{r, (L-1)/2}, \end{cases}$$

где $B = 3(2^r - 1)^2 / 2^m; \; m = \log_2(L+1).$

При $r = m$ значение $R_X(r)$ определяется согласно выражению:

$$R_X(r) = -\frac{3(2^r - 1)}{2^r(2^r + 1) - 3(2^r - 1)}, \quad r = m > 2.$$

Отсюда для достаточно больших $r = m$ получим:

$$R_X(r) = -\frac{3}{2^r} \cong 0.$$

Приведенные результаты свидетельствуют о том, что псевдослучайные тестовые последовательности, формируемые на основе *M*-последовательностей, отличаются хорошими статистическими свойствами, причем особенно важной для практических применений является обеспечиваемая, в данном случае, равномерность многомерных законов распределения. Последняя является следствием свойств *M*-последовательности и основана на следующем утверждении [189, 192, 297].

Утверждение 7.1. Последовательность *r*-разрядных псевдослучайных тестовых наборов называется асимптотически случайной, если для каждого $g = \overline{1,m}$, она *g* равномерна с точностью до *t* разрядов, где $t = \min(r, \lfloor m/g \rfloor)$.

Проверка последовательности на соответствие критерию асимптотической случайности базируется на анализе линейной зависимости между *gt* символами, образованными *t* старшими разрядов *g* последовательных наборов.

Предположим, что псевдослучайные тестовые наборы X_k, X_{k+1}, …, X_{k+g-1} представляют собой непересекающиеся выборки по *r* смежных символов *M*-последовательности. Из того, что последовательность состояний регистра сдвига, на котором генерируется М-последовательность, имеет период 2^m-1, следует, что содержимое всех *m* разрядов регистра сдвига представляет собой линейно независимые символы. Поэтому любые $g \le m/r$-разрядные псевдослучайные наборы также будут независимыми, и, следовательно, величина $t = \lfloor m/r \rfloor$ есть предел *g* равномерности.

Таким образом, выбирая соотношение между *m* и *r*, можно обеспечить любую заданную мерность равномерного распределения тестовых наборов. Так, в работе [192] приведен пример формирования на основе *M*-последовательности, порождаемой полиномом $\varphi(x) = 1 \oplus x^{502} \oplus x^{607}$, последовательности 15-разрядных псевдослучайных наборов, для которых обеспечивается 40-мерный закон равномерного распределения.

Увеличение мерности закона равномерного распределения псевдослучайных наборов повышает величину $m = \deg\varphi(x)$, поэтому тестовые наборы будут описываться апериодическими статистическими характеристиками, которые в значительной степени зависят от начального состояния регистра генератора *M*-последовательности. В работе [266] исследованы свойства трех видов начального состояния регистра сдвига:

1. *m*-разрядный набор 100…0;

2. *m*-разрядный набор, сформированный из последовательности идеальной случайной последовательности;

3. *m*-разрядный набор, представляющий собой *M*-последовательность для которой степень порождающего полинома $\psi(x)$ определяется из соотношения $\deg \psi(x) = \lceil \log_2 \deg \varphi(x) \rceil$.

Показано, что для задания начального состояния регистра генератора *M*-последовательности с порождающим полиномом высокого порядка лучше всего подходит второй вариант, т. е. использование идеальной случайной по-

следовательности. В то же время третий вариант более предпочтителен, чем первый, широко применяемый для *M*-последовательностей с порождающими полиномами низкого порядка [266].

В литературе приводятся сведения об экспериментальных исследованиях характеристик псевдослучайных тестовых наборов, формируемых из *M*-последовательностей [122, 300], которые полностью согласуются с полученными выше теоритическими результатами.

Глава 8. Псевдо-исчерпывающее тестирование

8.1. Сущность исчерпывающего и псевдо-исчерпывающего тестирования

Как отмечалось ранее, одной из самых распространённых технологий тестирования вычислительных систем является технология черного ящика, которая предопределяет применение тестовых последовательностей большой длины, таких как вероятностные тестовые последовательности и их модификации. В пределе вероятностные тесты большой длины аппроксимируют, либо повторяют, так называемые *исчерпывающие тесты* (*exhaustive tests – ET*) [16, 289, 302, 321]. Название подобных тестов свидетельствует об исчерпывающем формировании входных тестовых наборов, определяемых как объектом тестирования, так и внешней средой его применения.

Для случая исчерпывающих тестов, используемых для тестового диагностирования цифровых комбинационных устройств с N входами и пространством входных наборов, состоящим из 2^N двоичных наборов (векторов) справедливо следующее определение.

Определение 8.1. *Исчерпывающий тест* (*ET*) для комбинационных цифровых устройств с N входами представляет собой множество из 2^N всевозможных тестовых наборов, где N является размером набора в битах.

Расширение понятия исчерпывающего теста для более сложных цифровых устройств и вычислительных систем в целом предполагает генерирование всевозможных входных тестовых воздействий для каждого состояния устройства или системы, что является чрезвычайно сложной задачей.

Несомненным достоинством исчерпывающего тестирования является его максимальная эффективность, которую невозможно превзойти в рамках любых других известных подходов и методов тестирования вычислительных систем [14, 105, 121, 186, 187, 264, 275, 330]. Однако, в силу нереально высокой временной сложности подобных тестов, определяемой как большим числом состояний современных систем, так и множеством возможных входных данных, в настоящее время используются их различные аппроксимации, среди которых выделяются *псевдо-исчерпывающие тесты* (*pseudo-exhaustive tests – PET*) [14, 105, 121, 186, 187, 264, 275, 330]. Подобные тесты характеризуются весьма важным свойством, которое в отличие от тестов, построенных на основании ортогональных массивов вместо ограничения *точно один раз* (*exactly once*) используют ограничение *не менее одного раза* (*at least once*) [45, 88, 186]. В результате было сформулировано понятие так называемых *покрывающих массивов* (*covering arrays*), использование которых гарантировало, что проверяемые параметры тестируемого объекта будут проверены не реже одного раза [45, 88, 186]. По своей сути псевдо-исчерпывающие тесты и покрывающие массивы являются тождественными понятиями, они и представляют собой реальную альтернативу исчерпывающим тестам, а для двоичного случая соответствуют следующему определению.

Определение 8.2. *Псевдо-исчерпывающий тест* (*PET*) для объектов тестирования с N входами представляет собой множество тестовых наборов $T(N, k)$, обеспечивающих всевозможные 2^k двоичные комбинации на любых k из N входах.

Примером псевдо-исчерпывающего теста $T(N, k)$ может быть тест $T(6, 2) = \{000000, 000011, 011100, 101101, 110110, 111011\}$, приведенный в [105]. Как видно, для любых трех разрядов теста $T(6, 2)$ формируются все возможные двоичные комбинации, хотя бы по одному разу. Сложность этого теста (количество тестовых наборов) $O(T(N, k)) = O(T(6, 2)) = 6$, что заметно меньше, чем сложность исчерпывающего теста, которая для $N = 6$ равняется $2^N = 2^6 = 64$. Для общего случая сложность $O(T(N, k))$ псевдо-исчерпывающего теста $T(N, k)$ оценивается неравенством $2^k \le O(T(N, k)) \le 2^N$ [348]. В таблице 8.1 приведены примеры псевдо-исчерпывающих тестов $T(N, k)$ для двоичного случая [304, 305].

Как видно из приведенных примеров, на любых k из N разрядах тестовых наборов тестов $T(N, k)$ обеспечиваются все 2^k возможные двоичные комбинации.

Таблица 8.1 – *Псевдо-исчерпывающие тесты*

$T(3, 2)$	$T(4, 2)$	$T(4, 3)$	$T(5, 2)$	$T(5, 3)$
0 0 0	0 0 0 0	0 0 0 0	1 1 1 1 1	1 0 0 0 0
0 1 1	0 1 1 1	0 0 1 1	1 0 0 0 0	0 1 0 0 0
1 0 1	1 0 1 1	0 1 1 0	0 1 0 0 0	0 0 1 0 0
1 1 0	1 1 0 1	0 1 0 1	0 0 1 0 0	0 0 0 1 0
	1 1 1 0	1 1 0 0	0 0 0 1 0	0 0 0 0 1
		1 1 1 1	0 0 0 0 1	0 1 1 1 1
		1 0 1 0		1 0 1 1 1
		1 0 0 1		1 1 0 1 1
				1 1 1 0 1
				1 1 1 1 0

Псевдо-исчерпывающие тесты строятся не только для двоичного случая [92, 186, 187, 208, 301]. Примером недвоичного псевдо-исчерпывающего теста может быть псевдо-исчерпывающий тест для четверичных данных (8.1).

Обозначение вида $T(3, 2, 4)$ отличается от обозначения $T(N, k)$ наличием третьего аргумента, определяющего основание системы счисления, отличной от двоичной системы счисления. Достаточно часто псевдо-исчерпывающие тесты обозначаются одним параметром, а именно значением k и называются k-псевдо-исчерпывающими тестами [52].

$$T(3,2,4) = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 1 & 3 \\ 0 & 2 & 2 \\ 0 & 3 & 1 \\ 1 & 0 & 3 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & 3 & 0 \\ 2 & 0 & 2 \\ 2 & 1 & 1 \\ 2 & 2 & 0 \\ 2 & 3 & 3 \\ 3 & 0 & 1 \\ 3 & 1 & 0 \\ 3 & 2 & 3 \\ 3 & 3 & 2 \end{vmatrix}. \qquad (8.1)$$

Сложность получения конструктивных алгоритмов генерирования псевдо-исчерпывающих тестов предопределила поиск границ сложности в рамках, которых возможно получение подобных решений. Так для $k = 3$ данная оценка в виде неравенства (8.2) была получена в [23]:

$$O(T(N,3)) \le 7{,}5 \log_2 N. \qquad (8.2)$$

Тогда сложность реализации псевдо-исчерпывающих тестов при $k = 3$ даже для больших значений N становиться реальной задачей при тестировании современных вычислительных машин. Действительно количество тестовых наборов достаточно мало чему свидетельствуют значения верхней границы сложности (8.2), приведенные в таблице 8.2 для ряда величин N.

Таблица 8.2 – *Максимальное количество тестовых наборов для теста* $T(N, 3)$

N	2^2	2^4	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
$O(T(N, k))$	8	17	32	51	60	84	101	108	129	132

С увеличением k количество тестовых наборов заметно возрастает, что следует из оценки мощности $T(N, k)$, полученной для фиксированных значений k и больших N [23, 46, 47]:

$$2^{k-2} \times \log_2 N \le O(T(N,k-1)) \le 2^{k-1} \times (k-1) \times (\log_2 e)^{-1} \log_2 N. \qquad (8.3)$$

Конкретные значения $O(T(N, k))$ для фиксированных величин k и N приведены в таблице 8.3 [23, 46, 47].

Таблица 8.3 – *Значения* $O(T(N, k))$

N	k						
	2	3	4	5	6	7	8
4	5	8	16				
5	6	10	16	32			
6	6	12	21	32	64		
7	6	12	24	42	64	128	
8	6	12	24	56	85	128	256
9	6	12	24	72	120	170	256
10	6	12	24	90	165	240	341
11	7	12	24	101	213	330	496
12	7	16	24	101	261	440	715
13	7	16	38	104	309	572	1001
14	7	16	52	118	357	728	1365

Весьма важным отличием псевдо-исчерпывающих тестов по сравнению с исчерпывающими тестами является их существенно меньшая сложность при такой же или незначительно меньшей по сравнению с исчерпывающими тестами эффективностью [47].

Основную идею псевдо-исчерпывающих тестов можно показать на двух примерах для случаев параллельного и последовательностного формирования тестовых наборов. Первый случай характеризуется параллельным (одновременным) формированием всех бит тестового набора. В англоязычной литературе этот подход имеет название (*test-per-clock*), что в случае цифровых устройств означает генерирование очередного тестового набора в результате подачи одного импульса синхронизации [258, 262, 265, 281, 304, 305].

Пример 8.1. Для комбинационной схемы с тремя входами и двумя выходами, приведенной на рис. 8.1, исчерпывающий тест должен включать все возможные комбинации из трех бит.

Соответственно, его сложность будет равняться $2^N = 2^3 = 8$. Исчерпывающий тест позволит достичь 100% полноты покрытия в классе логических неисправностей, приводящих к изменению логики комбинационной схемы, так как будет проверено соответствие схемы ее таблице истинности. В тоже время тест, приведенный в таблице 8.4, также позволяет достичь 100% полноты покрытия.

Как видно из приведенной таблицы 8.4, для каждого из двух выходов $f(x_1, x_2)$ и $f(x_2, x_3)$ формируются всевозможные двоичные комбинации, количество которых равняется четырем. Таким образом, для каждой из двух подсхем, определяемых выходами схемы, генерируется исчерпывающий тест, а для схемы целиком так называемый псевдо-исчерпывающий тест, сложность которого существенно меньше сложности исчерпывающего теста.

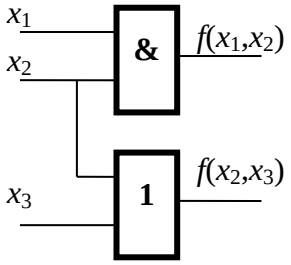


Рисунок 8.1 – *Комбинационная схема*

Таблица 8.4 – *Псевдо-исчерпывающий тест для случая test-per-clock структур*

T_i	x_1	x_2	x_3
T_1	0	0	0
T_2	0	1	0
T_3	1	0	1
T_4	1	1	1

Второй случай реализации псевдо-исчерпывающего тестирования широко распространен в схемах самотестирования вычислительных систем, основанных на сканировании пути (*scan path*) и имеющий англоязычное название *(test-per-scan*) [262, 304, 305]. Для подобных структур тестовые наборы формируются последовательно, путем реализации операции сдвига на цепи сканирования, которая в простейшем случае представляет собой регистр сдвига на один разряд. Таким образом, тестируемая комбинационная схема, по сути, представляет собой схему, выходы которой подключены к регистру сдвига. В процессе подачи тестового набора, состоящего из N двоичных бит, в худшем случае, необходимо сгенерировать N синхронизирующих импульсов для выполнения N микроопераций сдвига. Такое количества сдвигов необходимо для записи в последовательном коде на N-разрядный регистр N бит очередного тестового набора. Так, для рассмотренного выше примера необходимо выполнить $N = 3$ такта синхронизации для записи каждого из четырех тестовых наборов (см. таблицу 8.4). Отметим, что в случае произвольного подключения входов схемы к регистру сдвига, когда его разрядность больше количества входов схемы, время записи тестовых наборов растет пропорционально длине регистра сдвига.

Применение различных приемов зачастую позволяет заметно снизить сложность тестовой процедуры для архитектур, реализующих *test-per-scan* структуры [262, 304, 305, 321]. Например, для комбинационной схемы, приведенной на рис 8.1, псевдо-исчерпывающий тест может быть реализован за 7 тактов, как это видно из примера 8.2.

Пример 8.2. Для комбинационной схемы с тремя входами и двумя выходами, приведенной на рис 8.1, используя тест, представленный в таблице

8.5, достигается исчерпывающее тестирование обеих подсхем, каждая из которых определяется одним из выходов исходной схемы.

Для подачи первого тестового набора на вход регистра сдвига (вход цепи сканирования) последовательно подаются три нулевых значения, после чего на входах x_1, x_2 и x_3 будет сформирован первый тестовый набор. Для подачи второго тестового набора и последующих наборов необходимо выполнить по одной операции сдвига. Таким образом, на входах x_1, x_2 и x_2, x_3 будут сформированы всевозможные комбинации из двух бит, что обеспечивает псевдо-исчерпывающее тестирование приведенной схемы. Отметим, что, и в данном случае тест, приведенный в таблице 8.5, является не единственно возможным решением.

Таблица 8.5 – *Псевдо-исчерпывающий тест для случая test-per-scan структур*

T_i	x_1	x_2	x_3
T_1	0	0	0
T_2	1	0	0
T_3	1	1	0
T_4	0	1	1
T_5	0	0	1

Техническая реализация данного псевдо-исчерпывающего теста по методу *test-per-scan* структуры приведена на рис. 8.2.

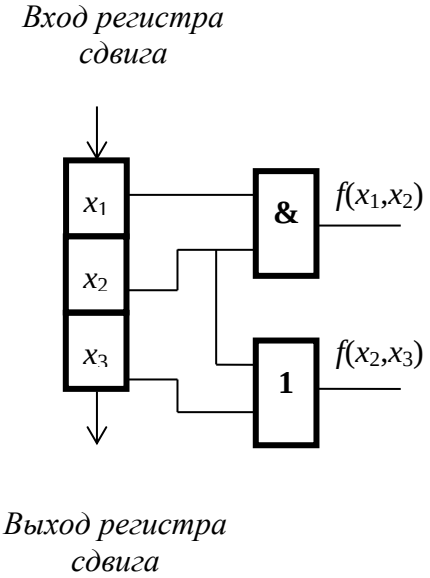


Рисунок 8.2 – *Комбинационная схема, реализующая test-per-scan структуру*

Третий пример псевдо-исчерпывающих тестов относится к запоминающим устройствам для случая обнаружения сложных кодочувствительных неисправностей. Как было показано ранее на рис. 1.7, в каждой из кодочувствительных неисправностей в явном виде выделяется базовый запоминающий элемент *b* и соседние запоминающие элементы *n*. Соседние запоминаю-

щие ячейки (ЗУ) являются физическими соседями по отношению к базовому запоминающему элементу. Количество k ячеек, участвующих в кодочувствительной неисправности и их местоположение может быть произвольным. Поэтому на практике обычно используют модели кодочувствительных неисправностей с небольшим числом k, равным 3, 5 и 9, конфигурации и обозначение которых приведены на рис. 1.7. Для обнаружения различных модификаций подобных неисправностей в соседних ячейках необходимо сформировать всевозможные 2^{k-1} двоичные комбинации, которые в совокупности по своей сути представляют псевдо-исчерпывающий тест для запоминающего устройства [235, 303, 331, 332].

8.2. Псевдо-исчерпывающие тесты на базе кодов Рида-Соломона

В качестве систематического подхода для формирования псевдо-исчерпывающих тестовых наборов $T(N, k, w)$ для слово-ориентированных данных $q = 2^w$ размерностью w бит в работах [123, 303] было предложено применение *кодов Рида-Соломона* (*Reed-Solomon code*). Отметим, что для $w = 0$ тест $T(N, k, w)$ представляет собой двоичный случай псевдо-исчерпывающего теста $T(N, k)$.

Расширенный $(q, q-k, k+1)$ *код Рида-Соломона* (*RS* код) над полем Галуа $GF(2^w)$ определяется следующей проверочной матрицей [123, 159]:

$$H = \begin{vmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & \alpha^1 & \alpha^2 & \dots & \alpha^{q-2} \\ 0 & 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(q-2)} \\ 0 & 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(q-2)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \dots & \alpha^{(k-1)(q-2)} \end{vmatrix}, \qquad (8.4)$$

где α является примитивным элементом над $GF(2^w)$ ($\alpha^i \neq \alpha^j$ для $i \neq j \in \{0, 1, \dots, q-2\}$). Так как любые k столбцов матрицы H являются линейно независимыми над полем $GF(2^w)$, то строки матрицы могут служить как тестовые наборы псевдо-исчерпывающего теста $T(2^w, k, w)$ [123]. Сложность данного теста определяется как $O(T(2^w, k, w)) = q^k = 2^{wk}$.

Пример 8.3. Предположим $q = 2^w = 4$ и $GF(2^2) = (0, 1, \alpha, \alpha^2)$, где α представляет собой корень порождающего полинома $\varphi(x) = x^2 + x + 1$ ($\alpha^3 = 1$). В поле Галуа, описываемом приведенным полиномом, определены операции сложения и умножения в соответствии со следующими таблицами [123].

Таблица 8.6 – *Операция сложения* (+)

+	0	1	α	α^2
0	0	1	α	α^2
1	1	0	α^2	α
α	α	α^2	0	1
α^2	α^2	α	1	0

Таблица 8.7 – *Операция умножения* (×)

×	0	1	α	α^2
0	0	0	0	0
1	0	1	α	α^2
α	0	α	α^2	1
α^2	0	α^2	1	α

Здесь каждое четверичное ($q = 2^w = 4$) значение кодируется двумя битами 0 = 00, 1 = 01, α =10 и α^2 = 11.

Таким образом, для построения оптимального множества псевдо-исчерпывающих тестовых наборов может быть использована проверочная матрица расширенного *RS* кода (8.3) над полем $GF(2^2)$ следующего вида:

$$H = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 \end{vmatrix}.$$

Тогда любой тестовый набор $T_i = t_{i,0}, t_{i,1}, t_{i,2}, t_{i,3}$ может быть получен как:

$$(v_0, v_1) \times \begin{vmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 \end{vmatrix} = (v_0,\ v_0 + v_1,\ v_0 + \alpha v_1, v_0 + \alpha^2 v_1),$$

$$v_0, v_1 \in GF(2^2).$$

Один из возможных тестовых наборов принимает вид:

$$(\alpha, \alpha^2) \times \begin{vmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 \end{vmatrix} = (\alpha,\ \alpha + \alpha^2,\ \alpha + \alpha^3, \alpha + \alpha^4) = (\alpha, 1, \alpha^2, 0).$$

В результате умножения всевозможных векторов $V = (v_0, v_1)$ на матрицу H могут быть получены всевозможные псевдо-исчерпывающие тестовые наборы теста $T(N, k) = T(4, 2)$ для четверичных данных, приведенные в таблице 8.8. Как видно из указанной таблицы, для любых $k = 2$ четверичных слов ($q = 2^2$) формируются $q^k = (2^2)^2 = 16$ комбинаций различных их значений.

Таблица 8.8 – *Псевдо-исчерпывающие тестовые наборы* $T_i = t_{i,0}, t_{i,1}, t_{i,2}, t_{i,3}$, где 0 = 00, 1 = 01, α = 10 и α^2 = 11.

v_0	v_1	$t_{i,0}$	$t_{i,1}$	$t_{i,2}$	$t_{i,3}$
0	0	0	0	0	0
1	0	1	1	1	1
α	0	α	α	α	α
α^2	0	α^2	α^2	α^2	α^2
0	1	0	1	α	α^2

1	1	1	0	α^2	α
α	1	α	α^2	0	1
α^2	1	α^2	α	1	0
0	α	0	α	α^2	1
1	α	1	α^2	α	0
α	α	α	0	1	α^2
α^2	α	α^2	1	0	α
0	α^2	0	α^2	1	α
1	α^2	1	α	0	α^2
α	α^2	α	1	α^2	0
α^2	α^2	α^2	0	α	1

Следующая теорема является основой для построения псевдо-исчерпывающих тестов для различных k и $N = q - 1$ [105, 107].

Теорема 8.1. Для $q = 2^w$, примитивного элемента α над $GF(q)$ ($\alpha^l \neq \alpha^j$; $l \neq j$; $l, j = 0, 1, 2, \ldots, q-2$), примитивного элемента β над $GF(q^k)$ ($\beta^l \neq \beta^j$; $l \neq j$; $l, j = 0, 1, 2, \ldots, q^k-2$), и матриц:

$$H = \begin{vmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \alpha^1 & \alpha^2 & \ldots & \alpha^{q-2} \\ 1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(q-2)} \\ 1 & \alpha^3 & \alpha^6 & \ldots & \alpha^{3(q-2)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \ldots & \alpha^{(k-1)(q-2)} \end{vmatrix}$$

обозначив:

$$\beta^{i-1}=(\alpha^{j0},\alpha^{j1},\ldots,\alpha^{j(k-1)})\in GF(q^k),$$

и $T_0 = (0, 0, \ldots, 0)$, $T_i=(t_{i0}, t_{i1}, \ldots, t_{i(q-1)}) = (\alpha^{j0},\alpha^{j1}, \ldots, \alpha^{j(k-1)})\times H$, где $t_{ij}\in GF(q)$, $i = 1, 2, \ldots, q^k$ получим, что:

1. Для любых $j_0 \le j_1 \le \ldots \le j_{k-1}$ и любых $A_{j0}, A_{j1}, \ldots, A_{j(k-1)}\in GF(q)$, существует такое $i\in(0, 1, \ldots, q^k-1)$, для которого выполняется система равенств

$$t_{i(j0)}= A_{j0}, t_{i(j1)}= A_{j1},\ldots, t_{i(j(k-1))} = A_{j(k-1)}.$$

2. Для любых $s\in(0, 1, \ldots, q-2)$, $j_0 \le j_1 \le \ldots \le j_{k-3}$, $s\notin(j_0, j_1, \ldots, j_{k-3})$ и для любых $A_{j0}, A_{j1}, \ldots, A_{j(k-3)}, A_s, A_s'\in GF(q)$, исключая случай, когда $A_{j0} = A_{j1}= \ldots = A_{j(k-3)} = A_s = A_s' = 0$, существует такое $i \in (0, 1, 2, \ldots, q^k)$, для которого выполняется система равенств:

$$t_{i(j0)}= A_{j0}, t_{i(j1)}= A_{j1},\ldots, t_{i(j(k-3))} = A_{j(k-3)}, t_{i(s)}= A_s, t_{i(js)}= A_s' \qquad (8.5)$$

Доказательство приведенной теоремы 8.1 приведено в [303, 304], где также приводятся следующие следствия из этой теоремы.

Следствие 8.1. Теорема 8.1 справедливы для более общего случая, когда любое подмножество $J = (j_0, j_1, \ldots, j_{k-3})$ в $t_{i(j)}$ (8.5) может быть заменено на $t_{(i+1)j}$, $j \in J$.

Следствие 8.2. Теорема 8.1 и следствие 8.1 справедлива для $k = 2$ и $N = q + 1$, когда используется проверочная матрица (8.6) (q+1, q+1−k, k) MDS [123] вместо проверочной матрицы H, представленной в теореме 8.1.

$$H = \begin{vmatrix} 1 & 0 & 1 & 1 & 1 & \ldots & 1 \\ 0 & 0 & 1 & \alpha & \alpha^2 & \ldots & \alpha^{q-2} \\ 0 & 0 & 1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(q-2)} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 1 & 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \ldots & \alpha^{(k-1)(q-2)} \end{vmatrix} \quad (8.6)$$

Следствие 8.3. Теорема 8.1 справедлива для любого k и $N = q$, когда проверочная матрица (8.7) представляет (q, $q-k$, k) MDS код [123]:

$$H = \begin{vmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 0 & 1 & \alpha^1 & \alpha^2 & \ldots & \alpha^{q-2} \\ 0 & 1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(q-2)} \\ 0 & 1 & \alpha^3 & \alpha^6 & \ldots & \alpha^{3(q-2)} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \ldots & \alpha^{(k-1)(q-2)} \end{vmatrix}. \quad (8.7)$$

Применяя теорему 8.1 и ее следствия 8.1, 8.2 и 8.3, оказывается возможным построение псевдо-исчерпывающих тестов $T(N, k)$ для эффективного тестирования запоминающих устройств [19, 304, 305]. Данный подход построения псевдо-исчерпывающих тестов целесообразно применять для вычислительных систем, реализующих архитектуру *test-per-clock*.

8.3. Псевдо-исчерпывающее тестирование с применением циклических кодов

В качестве основы методологии использования циклических кодов для построения псевдо-исчерпывающих тестов используется следующая теорема, которую приведем без доказательства [19, 304, 305].

Теорема 8.2. Циклический код длиной N, генерируемый на базе порождающего полинома $\varphi(x)$ и имеющий минимальное кодовое расстояние k + 1, позволяет формировать всевозможные двоичные комбинации на произвольных k разрядах регистра сдвига длиной $w \le N$.

Таким образом, условия теоремы 8.2 определяет псевдо-исчерпывающий тест $T(N, k)$, формируемый на базе циклического кода. Согласно данной теореме, M-последовательность, описываемая примитивным полиномом и представляющая собой циклический код с кодовым расстоянием k + 1, равным 3, позволяет обеспечивать всевозможные двоичные комбинации на любых двух разрядах регистра.

Как было показано в [19, 304, 305] этот метод применим для комбинационных схем, имеющих N входов и r выходов, причем каждый выход схемы

зависит не более чем от $k < N$ входов. Обозначим подобные комбинационные схемы через $(N, k)КС$.

Задача синтеза псевдо-исчерпывающего теста, основанного на регистре сдвига, состоит в поиске непримитивного полинома степени N, описывающего структуру генератора теста $T(N, k)$ и начального тестового набора. Методика синтеза подобного генератора основывается на следующей теореме [305].

Теорема 8.3. Для произвольной $(N, k)КС$, где $N > k > 0$, можно найти значение целого положительного v, удовлетворяющего неравенству:

$$k \le \left\lfloor \frac{v}{N-v+1} \right\rfloor + \left\lceil \frac{v}{N-v+1} \right\rceil, \qquad (8.8)$$

для которого генератор циклической последовательности, описываемый полиномом:

$$\varphi(x) = g(x)\psi(x) = (1 \oplus x \oplus x^2 \oplus \ldots \oplus x^{N-v})\psi(x), \qquad (8.9)$$

где $\psi(x)$ – примитивный полином степени v, а начальное состояние генератора $S_0(x)$ нацело делиться на полином $g(x)$, обеспечивает всевозможные двоичные комбинации на любых k из N входах $(N, k)КС$.

С помощью теоремы 8.3 можно сформулировать алгоритм синтеза генераторов псевдо-исчерпывающих $T(N, k)$ тестов для $(N, k)КС$ [305]. Этот алгоритм включает следующие этапы [305].

Алгоритм 8.1. Алгоритм синтеза псевдо-исчерпывающих $T(N, k)$ тестов.

1. Для $(N, k)КС$ определяется максимальное количество входов k комбинационной N-входовой схемы, от которого зависит один из r ее выходов. Для этого используется метод обратного хода [302].

2. Вычисляется минимальное целое положительное v, удовлетворяющее неравенству (8.8).

3. Строиться генератор циклической последовательности, состоящий из сдвигового регистра с внутренними сумматорами по модулю два, в соответствии с полиномом (8.9).

4. В качестве начального значения генератора выбирается код, описываемый полиномом $S_0(x)$, таким, что $S_0(x)$ нацело делиться на полином $g(x)$.

5. Сложность псевдо-исчерпывающего теста $T(N, k)$ для $(N, k)КС$ вычисляется, как $O(T(N, k)) = 2^v - 1$ для $v > k$ или $O(T(N, k)) = 2^v$ для $v = k$.

Пример 8.4. В качестве примера рассмотрим синтез генератора псевдо-исчерпывающего теста $T(N, k) = T(4, 2)$ для схемы $(4, 2)КС$. Используя алгоритм 8.1, последовательно выполним все его этапы.

1. Из условия задачи синтеза следует, что любой выход комбинационной схемы $(4, 2)КС$ зависит не более чем от двух его входов ($k = 2$).

2. Значение v определяется из неравенства:

$$2 \le \left\lfloor \frac{v}{N-v+1} \right\rfloor + \left\lceil \frac{v}{N-v+1} \right\rceil = \left\lfloor \frac{v}{5-v} \right\rfloor + \left\lceil \frac{v}{5-v} \right\rceil .$$

Получим, что минимальное v, удовлетворяющее последнему неравенству, равняется трем (v = 3).

3. Порождающий полином $\varphi(x)$ генератора теста $T(4, 2)$ в соответствии с выражением (8.9) имеет вид:

$$\varphi(x) = (1 \oplus x)(1 \oplus x \oplus x^3) = 1 \oplus x^2 \oplus x^3 \oplus x^4, \qquad (8.10)$$

где $\psi(x) = (1 \oplus x \oplus x^3)$. Очевидно, что в качестве $\psi(x)$ может быть выбран и другой примитивный полином той же степени, например $\psi(x) = (1 \oplus x^2 \oplus x^3)$. На основании полинома строиться функциональная схема генератора теста, которая представлена на рис. 8.3.

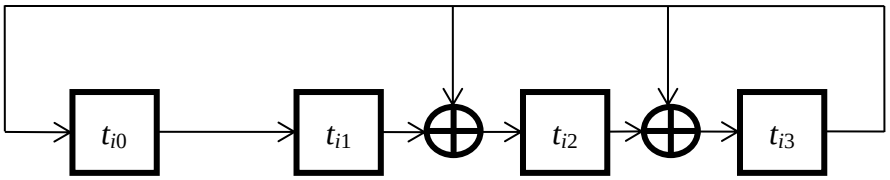


Рисунок 8.3 – *Генератор псевдо-исчерпывающего теста* $T(4, 2)$

4. В качестве начального кода может быть выбран любой из кодов 1100, 0110, 0011, 1010, 0101, 1001 и 1111, так как полиномы $S_0(x)$ соответствующие им, нацело делятся на полином $g(x) = 1 \oplus x$. Предположим, что начальный код равен 1100, тогда временная диаграмма формирования тестовых наборов $T_i = t_{i,0}, t_{i,1}, t_{i,2}, t_{i,3}$ псевдо-исчерпывающего теста $T(4, 2)$, будет иметь следующий вид (см. таблицу 8.9).

Таблица 8.9 – *Псевдо-исчерпывающие тестовые наборы*

i	$t_{i,0}$	$t_{i,1}$	$t_{i,2}$	$t_{i,3}$
0	1	1	0	0
1	0	1	1	0
2	0	0	1	1
3	1	0	1	0
4	0	1	0	1
5	1	0	0	1
6	1	1	1	1

5. Сложность теста $O(T(N, k)) = O(T(4, 2))$ равна $2^v - 1 = 2^3 - 1 = 7$.

Пример 8.5. В качестве второго примера рассмотрим (10, 5)*КС*, для которой в соответствии с неравенством (8.8) получим, что v = 8. Тогда один из возможных порождающих полиномов $\varphi(x)$ будет иметь вид:

$$\varphi(x) = (1 \oplus x \oplus x^2)(1 \oplus x \oplus x^3 \oplus x^4 \oplus x^8) =$$
$$= 1 \oplus x \oplus x^4 \oplus x^6 \oplus x^8 \oplus x^9 \oplus x^{10}.$$

Соответствующая ему структурная схема приведена на рис. 8.4.

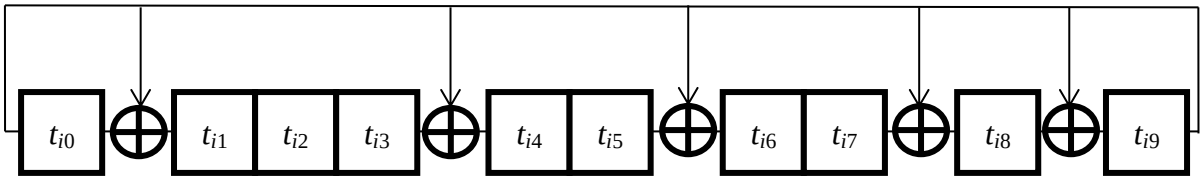


Рисунок 8.4 – *Генератор псевдо-исчерпывающего теста* $T(10, 5)$

В качестве начального кода можно использовать код 1110000000. Так как соответствующий ему полином $S_0(x)$ нацело делится на полином $g(x) = 1 \oplus x \oplus x^2$. Сложность тестовой последовательности $T(10, 5)$, обеспечивающей всевозможные двоичные комбинации на любых 5 из 10 входах комбинационной схемы $(10, 5)KC$, определяется как $O(T(10, 4)) = 2^8 - 1 = 255$.

Основным достоинством рассмотренного метода синтеза псевдо-исчерпывающих тестов является возможность его применения для вычислительных систем, реализованных в соответствии с широко применяемой в настоящее время структурой *test-per-scan* [163, 262, 304, 305].

8.4. Псевдо-исчерпывающие тесты на основе векторов заданного веса

Данный метод синтеза псевдо-исчерпывающих тестов так же, как и два предыдущих подхода направлен на построение теста, обеспечивающего исчерпывающие тесты для любых k из N разрядов. Предполагая, что тестовые наборы представляют собой двоичные вектора, которые формируются на входном регистре, подобные тесты применимы как для *test-per-clock* архитектур, так и для *test-per-scan*. Одним из решений задачи синтеза, так называемых универсальных псевдо-исчерпывающих тестов, является метод, основанный на применении векторов заданного веса [305]. Традиционно, под *весом* (w) понимается количество единичных компонент в векторе.

Рассматриваемый метод основан на применении следующей теоремы [305, 186].

Теорема 8.4. Последовательность двоичных тестовых наборов T_i псевдо-исчерпывающего теста $T(N, k)$, размерности N позволяет обеспечить всевозможные двоичные комбинации на любых $k < N$, из N произвольных разрядов, если она содержит все N-мерные вектора веса $w \le N$, такого, что $w = c$

mod (N – k + 1) для постоянного c, удовлетворяющего неравенству $0 \le c \le N - k$.

Важным следствием приведенной теоремы является существование $N - k + 1$ решений задачи построения псевдо-исчерпывающего теста. Рассмотрим некоторые из них для конкретных значений N и k.

Пример 8.6. Предположим, что $N = 5$, а $k = 2$ и соответственно $N - k + 1 = 4$. Согласно теореме 8.4, значение c изменяется в пределах от 0 до $N - k = 5 - 2 = 3$. Это означает, что существует 4 решения задачи построения псевдо-исчерпывающего теста $T(5, 2)$, которые определяются значениями 0, 1, 2, и 3 константы c. Например, для $c = 0$ согласно теореме 8.4 тест $T(5, 2)$ должен содержать вектор, вес w которых удовлетворяет следующему линейному сравнению

$$w = c \bmod (N - k + 1) = 0 \bmod 4.$$

Решением приведенного сравнения являются $w = 0$ и $w = 4$, что свидетельствует о необходимости использования в тесте $T(5, 2)$ векторов веса 0 и векторов, имеющих вес 4. В результате псевдо-исчерпывающий тест имеет следующий вид:

$$T_0(5,2) = \left| 0 \quad 0 \quad 0 \quad 0 \quad 0 \right| \cup \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix}.$$

Нижний индекс в обозначении вида $T_0(5, 2)$ представляет собой величину константы c. Для другого значения c, например, равного 2, псевдо-исчерпывающий тест имеет следующий вид:

$$T_2(5,2) = \begin{vmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{vmatrix}.$$

Сравнивая сложность двух приведенных тестов, можно отметить, что $O(T_0(5, 2)) = 6 < O(T_2(5, 2)) = 10$. Продолжая исследование данного примера, можно построить псевдо-исчерпывающие тесты $T_1(5, 2)$ и $T_3(5, 2)$ для $c = 1$ и $c = 3$. В результате получим, что минимальное количество тестовых наборов, равное 6, будут содержать тесты $T_0(5, 2)$ и $T_3(5, 2)$, а совокупность тестов $T_0(5, 2)$, $T_1(5, 2)$, $T_2(5, 2)$ и $T_3(5, 2)$ является непересекающимися подмножествами всего множества двоичных 5-разрядных векторов.

Последнее утверждение, справедливое для общего случая, позволяет оценить минимальную сложность псевдо-исчерпывающего теста, который можно построить на базе векторов заданного веса.

Принимая во внимание, что минимальная сложность $O_{min}(T_c(N, k))$ псевдо-исчерпывающего теста ($T_c(N, k)$) не превышает среднего количества тестовых наборов для всех $c = N - k + 1$ решений, можно показать, что

$$O_{min}(T_c(N,k)) \le \left\lfloor \frac{2^N}{N-k+1} \right\rfloor \qquad (8.11)$$

Приведенное выражение представляет собой верхнюю оценку минимального количества тестовых наборов псевдо-исчерпывающего теста $T_c(N, k)$. Уточнения оценки (8.11) могут быть получены только для частных случаев соотношений N и k. Так для $k < N/2$ псевдо-исчерпывающий тест минимальной сложности, согласно теореме 8.4, будет получен, когда $w = \lfloor k/2 \rfloor$ и $w = \lfloor k/2 \rfloor + N - k + 1$. Тогда:

$$O_{min}(T_c(N,k)) \le \binom{N}{\lfloor k/2 \rfloor} + \binom{N}{\lfloor k/2 \rfloor + N - k + 1}.$$

Зависимость минимальной сложности $O_{min}(T_c(N, k))$ и ее оценки (8.11) псевдо-исчерпывающего теста ($T_c(N, k)$) показывает невозможность применения данного метода для больших значений N и k. В тоже время для приемлемых величин сложности теста и, соответственно, небольших значений N и k тестовые последовательности $T_c(N, k)$ легко формируются с помощью сдвиговых регистров, что позволяет применять подобные теста как для архитектур вычислительных систем типа *test-per-clock*, так и для архитектур типа *test-per-scan*.

Более подробно рассмотрим случай архитектур типа *test-per-scan*, для которых каждый тестовый набор побитно записывается на регистр сдвига как результат выполнения N микроопераций сдвига, количество которых равняется разрядности тестовых наборов псевдо-исчерпывающего теста $T_c(N, k)$. Для сложности (количества тестовых наборов) $O(T_c(N, k))$ теста равной n, общее количество микроопераций сдвига равняется $N \times n$. Существенное уменьшение количества сдвигов для реализации псевдо-исчерпывающего теста можно достичь, используя циклический регистр сдвига с сумматором по модулю два на его входе [305]. Применение данной схемы (см. рис. 8.5), по

сути, означает сжатие псевдо-исчерпывающего теста $T_c(N, k)$, состоящего из $N \times n$ двоичных бит до бинарной однобитной последовательности a_i, которая подается на вход сумматора по модулю два.

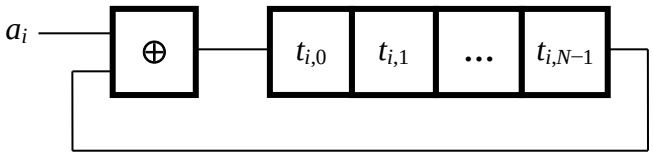


Рисунок 8.5 – *Циклический сдвиговый регистр*

Входная последовательность a_i сдвигового регистра определяет правила формирования тестовых наборов, и в пределе может содержать $N \times n$ бит. Минимальные оценки сложности последовательности a_i приведены в [305]. В качестве примера рассмотрим псевдо-исчерпывающий тест $T_3(5, 2)$.

Пример 8.7. Предположим, что $N = 5$, а $k = 2$ и соответственно $N - k + 1 = 4$. Согласно теореме 8.4 параметр c может принимать значение 3. Соответствующий этому значению c тест имеет вид

$$T_3(5,2) = \begin{vmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{vmatrix}.$$

В таблице 8.10 приведен один из возможных вариантов последовательности a_i и состояния регистра сдвига, формирующего тестовые наборы псевдо-исчерпывающего теста $T_3(5, 2)$. Исходное состояние регистра сдвига (рис. 8.5) принимается равным нулевому состоянию всех его разрядов. Далее в течение первых трех тактов формируется первый тестовый набор 11100 для этого $a_1 = 1$, $a_2 = 1$ и $a_3 = 1$. Порядок формирования последующих тестовых наборов теста определяется символами a_i. В результате 10 тестовых наборов теста $T_3(5, 2)$ формируются в результате выполнения только 13 микроопераций сдвига. Отметим, что максимальное количество сдвигов для данного теста равняется $N \times n = 5 \times 10 = 50$. В результате при генерировании 10 наборов псевдо-исчерпывающего теста $T_3(5, 2)$ только 3 являются избыточными.

Поиск оптимального вида последовательности a_i является довольно трудоемкой задачей, однако ее решение позволяет существенно уменьшить сложность реализации псевдо-исчерпывающего теста, основанного на применении векторов заданного веса. Кроме того, необходимо отметить наличие дополнительного запоминающего устройства для хранения последовательности a_i.

Таблица 8.10 – *Временная диаграмма состояний циклического регистра*

i	a_i	$t_{i,0}$	$t_{i,1}$	$t_{i,2}$	$t_{i,3}$	$t_{i,4}$	$T_3(5, 2)$
0		0	0	0	0	0	
1	1	1	0	0	0	0	×
2	1	1	1	0	0	0	×
3	1	1	1	1	0	0	1 1 1 0 0
4	0	0	1	1	1	0	0 1 1 1 0
5	0	0	0	1	1	1	0 0 1 1 1
6	0	1	0	0	1	1	1 0 0 1 1
7	0	1	1	0	0	1	1 1 0 0 1
8	1	0	1	1	0	0	×
9	1	1	0	1	1	0	1 0 1 1 0
10	0	0	1	0	1	1	0 1 0 1 1
11	0	1	0	1	0	1	1 0 1 0 1
12	0	1	1	0	1	0	1 1 0 1 0
13	0	0	1	1	0	0	0 1 1 0 1

Несомненным достоинством рассмотренного метода формирования псевдо-исчерпывающих тестов является его применимость как для архитектур вычислительных систем типа *test-per-clock*, так и для архитектур типа *test-per-scan*.

8.5. Псевдо-исчерпывающие циклические последовательности

Основная идея использования псевдо-исчерпывающих циклических последовательностей заключается в аппроксимации псевдо-исчерпывающих тестов $T(N, k)$ циклическими последовательностями, не требующими для своей реализации заметных дополнительных аппаратурных затрат [105]. Вторым, несомненно, тоже очень важным достоинством подобных последовательностей является возможность реализации неразрушающего тестирования [105]. Очевидно, что для обеспечения восстановления информации тестовые наборы должны обладать таким свойством, как цикличность. В качестве методов генерирования циклических тестовых наборов может быть использован ряд известных алгоритмов. К их числу можно отнести алгоритмы, основанные на формировании сдвиговых последовательностей, псевдослучайных последовательностей (M-последовательностей), пересчетных последовательностей, последовательностей Джонсона и др. Определим для некоторых разновидностей циклических последовательностей их аналитические выраже-

ния, которые позволяли бы вычислять состояние символов в любой момент времени, а также длину их периода.

Сдвиговая последовательность

Сдвиговую последовательность (*shift sequence – SS*) можно получить с помощью *N*-разрядного сдвигового регистра с обратной положительной связью, который в англоязычной литературе имеет название *кольцевой счетчик* (*ring counter*) [305]. Состояния разрядов такого счетчика можно представить в виде *N*-мерных векторов $A(k) = a_0(k)\ a_1(k)\ a_2(k)\ \ldots\ a_{N-1}(k)$, $k = 0, 1, 2, \ldots$, для которых будет выполняться соотношение (8.12).

Период сдвиговой последовательности (Q_{SS}) зависит от начального состояния символов последовательности $A(0) = a_0(0)a_1(0)a_2(0)...a_{N-1}(0)$. Так, например, для $N = 4$ все множество состояний делиться на несколько *орбит* (*orbit*), имеющих различную длину, другими словами различный период Q_{SS}.

$$\begin{vmatrix} a_0(k+1) \\ a_1(k+1) \\ a_2(k+1) \\ \ldots \\ a_{N-1}(k+1) \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0 & \ldots & 0 & 1 \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{vmatrix} \times \begin{vmatrix} a_0(k) \\ a_1(k) \\ a_2(k) \\ \ldots \\ a_{N-1}(k) \end{vmatrix} \qquad (8.12)$$

Действительно, в соответствии с (8.12) получим 6 орбит, представленных в таблице 8.11.

Таблица 8.11 – *Орбиты сдвиговой последовательности SS для N = 4*

Орбиты SS					
0 0 0 0	0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0	0 0 1 1 0 1 1 0 0 0 1 1 1 0 0 1	0 1 0 1 1 0 1 0	0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0	1 1 1 1

Очевидно, что максимальный период Q_{SS} равняется величине N кроме того, данная величина влияет на число классов орбит с различными периодами. Так для $N = 12$ будем иметь орбиты с периодом $Q_{SS} = 1$ (0 0 0 0 0 0 0 0 0 0 0 0, 1 1 1 1 1 1 1 1 1 1 1 1), с периодом $Q_{SS} = 2$ (0 1 0 1 0 1 0 1 0 1 0 1), с периодом $Q_{SS} = 3$ (0 1 1 0 1 1 0 1 1 0 1 1 0), с периодом $Q_{SS} = 4$ (0 1 1 1 0 1 1 1 0 1 1 1), с периодом $Q_{SS} = 6$ (0 1 1 1 1 1 0 1 1 1 1 1) и с периодом $Q_{SS} = 12$ (0 0 0 0 0 0 0 0 0 0 0 1). Число классов орбит определяется числом делителей величины *N*. Так для $N = 4$ имеется три делителя 1, 2 и 4, и соответственно, три класса орбит (см. таблицу 8.11), а для $N = 12$ – шесть делителей 1, 2, 3, 4, 6, 12, что обеспечивает шесть классов орбит. Количество орбит определенного класса зависит только от делителя d числа N, который порождает данный класс и для $d > 1$ их количество не менее чем $d - 1$, а для $d = 1$ всегда имеем две орбиты 0 0 0 … 0 и 1 1 1 … 1.

Для N-разрядного сдвигового регистра, с любым начальным состоянием, отличным от состояний 0 0 0 ... 0 и 1 1 1 ... 1, $Q_{SS} = N$ только в том случае, когда N представляет собой простое число. Для простых N все пространство, состоящее из $2^N - 2$ элементов, делится на r орбит равной длины:

$$r = \frac{2^N - 2}{N}. \tag{8.13}$$

Приведенный анализ сдвиговых последовательностей показывает необходимость деления входного тестового набора на блоки, размерность N которых представляется простым числом, либо числом, имеющим минимальное количество делителей.

Последовательность Джонсона

Генерирование *последовательности Джонсона* (*Johnson sequence – JS*) можно производить на сдвиговом регистре с обратной отрицательной связью от старшего разряда регистра к младшему, что соответствует так называемому счетчику Джонсона, который в англоязычной литературе имеет альтернативное название *twisted ring counter* [19]. Состояния разрядов счетчика Джонсона (8.14) представляются следующим соотношением:

$$\begin{vmatrix} a_0(k+1) \\ a_1(k+1) \\ a_2(k+1) \\ \dots \\ a_{N-1}(k+1) \end{vmatrix} = \begin{vmatrix} 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & 0 \end{vmatrix} \times \begin{vmatrix} a_0(k) \\ a_1(k) \\ a_2(k) \\ \dots \\ a_{N-1}(k) \end{vmatrix} \oplus \begin{vmatrix} 1 \\ 0 \\ 0 \\ \dots \\ 0 \end{vmatrix}. \tag{8.14}$$

Для N-разрядного счетчика Джонсона с любым начальным состоянием период формируемой им последовательности определяется как $Q_{JS} = 2N$ только в том случае, когда: $N = 2^k$, $k = 0, 1, 2, \dots$. При этом все бинарное пространство, состоящее из 2^N элементов, будет делиться на r орбит равной длины, где r вычисляется как:

$$r = \frac{2^N}{2 \times N} = 2^{N-1-\log_2 N}. \tag{8.15}$$

Учитывая то, что, как правило, данные, используемые в современных вычислительных системах, являются байт ориентированными, условие $N = 2^k$ легко выполнимо. Кроме того, следует отметить, что последовательность Джонсона имеет период $Q_{JS} = 2N$ и в силу этого является одним из перспективных кандидатов для реализации псевдо-исчерпывающих неразрушающих циклических тестов.

Псевдослучайная последовательность

В качестве генератора *псевдослучайной* последовательности (*pseuorandom sequence – PS*) в подавляющем большинстве случаев в диагностике используются генераторы *M*-последовательностей, подробно представленные в главе 7 [207, 208, 301, 302]. Подобный генератор строиться на базе *N*-разрядного сдвигового регистра с линейной обратной связью, который описывается примитивным порождающим полиномом $\varphi(x) = 1 \oplus \beta_1 x^1 \oplus \beta_2 x^2 \oplus \beta_3 x^3 \ldots \oplus \beta_N x^N$ [207, 208, 301, 302]. Состояния разрядов такого сдвигового регистра представляются следующим соотношением:

$$\begin{vmatrix} a_0(k+1) \\ a_1(k+1) \\ a_2(k+1) \\ \ldots \\ a_{N-1}(k+1) \end{vmatrix} = \begin{vmatrix} \beta_1 & \beta_2 & \beta_2 & \ldots & \beta_{N-1} & \beta_N \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{vmatrix} \times \begin{vmatrix} a_0(k) \\ a_1(k) \\ a_2(k) \\ \ldots \\ a_{N-1}(k) \end{vmatrix}, \qquad (8.16)$$

где $\beta_i \in \{0, 1\}$ – постоянные коэффициенты порождающего полинома $\varphi(x)$. Вся последовательность *N*-разрядных двоичных кодов для любого примитивного порождающего полинома делиться на две орбиты, одна из которых включает только код 000…0, а вторая – все остальные ненулевые коды, в этом случае период $Q_{PS} = 2^N - 1$. Более подробную информацию о псевдослучайных последовательностях, генерируемых на базе примитивных порождающих полиномов можно найти в главе 7 и следующих литературных источниках [207, 208, 301, 302].

Несомненный интерес приставляют собой так называемые почти псевдослучайные последовательности, формируемые на базе непримитивных порождающих полиномов $\varphi(x)$ [52]. В этом случае количество орбит всегда больше двух. Например, используя полином $\varphi(x) = 1 \oplus x^2 \oplus x^3 \oplus x^4$, в соответствии с (8.16) получим 4 орбиты, представленные в таблице 8.12.

Таблица 8.12 – *Орбиты псевдослучайной последовательности PS для N = 4*

Орбиты PS			
0 0 0 0	1 0 0 0 0 1 0 0 1 0 1 0 1 1 0 1 0 1 1 0 0 0 1 1 0 0 0 1	1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0	1 1 1 1

Очевидно, что вид полинома и начальное значение определяют локальный цикл, формируемый на базе конкретного полинома. Количество циклов и их длины также однозначно зависят от порождающего полинома.

Глава 9. Псевдо-исчерпывающие *M*-последовательности

9.1. Сущность псевдо-исчерпывающих *M*-последовательностей

Как отмечалось в предыдущем разделе, радикальным подходом для тестирования современных вычислительных систем является применение различных методов формирования псевдо-исчерпывающих тестовых последовательностей, как одной из разновидностей исчерпывающего тестирования. Исчерпывающее и псевдо-исчерпывающее тестирования широко применяются для построения самотестируемых вычислительных систем, которые, как правило, реализуются по одному из методов контролепригодного синтеза. В подавляющем большинстве случаев современные вычислительные системы, и в первую очередь их аппаратная часть, построены по структуре типа *test-per-scan*, которая является результатом применения одного из методов проектирования, основанного на применении идеологии *сканирования пути* (*scan path design*) [132, 198, 199, 207, 208, 302, 304, 319]. Основная идея этих методов заключается в том, что в режиме тестирования запоминающие элементы последовательностного цифрового устройства объединяются в *цепь сканирования* (*scan path*), к которой подключаются комбинационные схемы. По сути, цепь сканирования представляет собой регистр сдвига с последовательным входом для записи тестовых воздействий и последовательным выходом для снятия реакций на тестовые воздействия. В зависимости от организации запоминающих элементов в цепи сканирования различают такие методологии проектирования, как: *full scan path*, *partial scan path*, *random access scan path*, *multiple scan path*, *hierarchical scan path*, *reconfigurable scan path*, *virtual scan chains* и другие модификации методологии сканирования пути [154, 198, 199, 321]. Достаточно известными и хорошо апробированными на практике являются такие методы проектирования, как: *Stanford scan path*, *Illinois scan architecture*, LSSD, S^3, STUMPS, и повсеместно применяемый при проектировании вычислительных систем стандарт IEEE 1149.1 (JTAG) и его новейшие версии [154, 198, 199, 262, 263, 321].

В результате применения одного из методов проектирования последовательностного цифрового устройства по идеологии сканирования пути, в режиме тестирования оно будет состоять из единого регистра сдвига и множества комбинационных подсхем, представляющих собой комбинационную часть устройства. В общем случае структура устройства в режиме тестирования может состоять из более сложного сочетания сдвиговых регистров и комбинационных схем, однако всегда можно выделить структуру, приведенную на рис. 9.1.

В режиме тестирования устройство будет состоять из регистра сдвига, состоящего из w разрядов и комбинационной схемы G, включающей r подсхем G_j, $j = \overline{1,r}$, где j-я подсхема имеет n_j входов и v_j входов, как правило, $v_j = 1$. Обозначим через n количество входов схемы G, а через $v = r$ – число ее выходов.

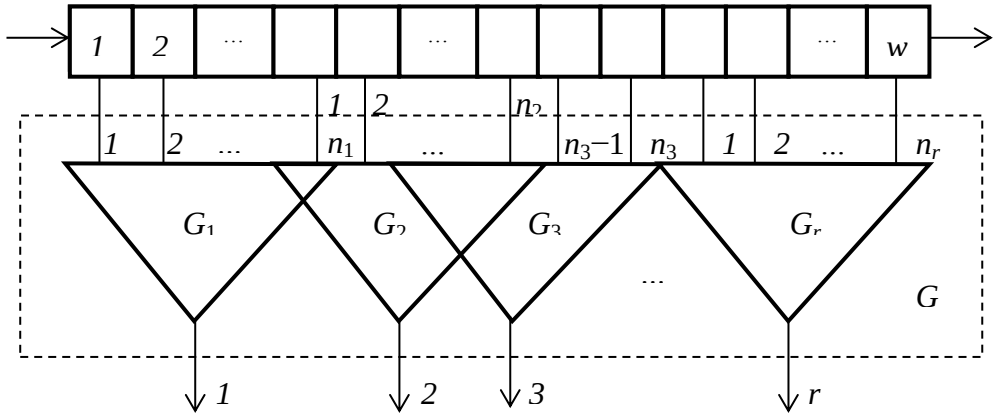


Рисунок 9.1 – *Структура устройства в режиме сканирования пути*

Задача тестирования устройства, приведенного на рис. 9.1, состоит в тестировании комбинационной части G, включающей r одно-выходных подсхем G_j. Несмотря на то, что структура последовательностной схемы в режиме тестирования состоит из более сложного подключения регистров сдвига и комбинационных подсхем, всегда решается задача формирования тестовой последовательности, последовательно подаваемой на регистр сдвига, к которому подключена тестируемая комбинационная схема. Одним из радикальных решений является генерирование псевдо-исчерпывающих тестовых последовательностей для каждой из подсхем комбинационной части тестируемого устройства.

Для объекта тестирования, структура которого представлена на рис. 9.1, эта задача заключается в формировании тестовой псевдо-исчерпывающей последовательности $t_i \in \{0, 1\}$, $i = \overline{1,l}$, удовлетворяющей следующим условиям.

1. Последовательность бинарных тестовых значений $t_i \in \{0, 1\}$, $i = \overline{1,l}$, подаваемых на вход регистра сдвига длиной w, обеспечивает формирование всевозможных двоичных наборов для всех подсхем G_j, $j = \overline{1,r}$, схемы G.

2. Длина l формируемой последовательности должна принимать минимально возможное значение, как правило, меньшее 2^w, что позволит реализовать процедуру тестирования в реальном масштабе времени.

Для решения подобной задачи формирования последовательности t_i, $i = \overline{1,l}$ используются различные методы и подходы [207, 302]. При этом основным критерием эффективности их применения является сложность построения генератора псевдо-исчерпывающей последовательности.

Применение M-последовательности в качестве генератора тестовой последовательности t_i, $i = \overline{1,l}$ является одним из наиболее эффективных решений. Аппаратурные затраты в этом случае будут состоять из m разрядов регистра сдвига, где $m = \deg \varphi(x)$ – степень порождающего полинома $\varphi(x)$, удовлетворяющая условию:

$$m = \deg \varphi(x) > w, \qquad (9.1)$$

и незначительного количества сумматоров по модулю два (см. главу 7). В этом случае обеспечивается реализация исчерпывающего теста для всей комбинационной схемы G. Длина же l тестовой последовательности t_i, $i=\overline{1,l}$, для реальных значений w = 100-300 [14] принимает очень большую величину, что не позволяет реализовать исчерпывающее тестирование в реальном масштабе времени. Выражение (9.1) дает лишь верхнюю оценку степени порождающего полинома $\varphi(x)$, применение которого позволяет реализовать исчерпывающее тестирование. Нижнюю оценку можно вычислить из следующего неравенства:

$$\deg \varphi(x) > \max_j n_j . \qquad (9.2)$$

Отметим, что неравенство (9.2) является необходимым, но не достаточным условием, которому должен удовлетворять порождающий полином $\varphi(x)$ для реализации псевдо-исчерпывающего теста схемы G. Действительно, для комбинационной схемы G (рис. 9.2), состоящей из одной подсхемы G_1 с n_1 = 3 входами, нельзя обеспечить всевозможные входные комбинации при использовании полинома $\varphi(x) = 1 \oplus x^3 \oplus x^4$, для которого $\deg \varphi(x) = 4 > n_1 = 3$. На входах *1*, *2* и *3* схемы G_1 будут отсутствовать входные комбинации 0 0 1, 0 1 0, 1 0 0 и 1 1 1, что определяется видом выбранного полинома $\varphi(x)$. На основании свойства сдвига и сложения M-последовательности, порождаемой полиномом $\varphi(x) = 1 \oplus x^3 \oplus x^4$, можно показать, что для значений $a_1(i)$, $a_4(i)$, и $a_5(i)$ разрядов *1*, *4* и *5* регистра сдвига выполняется соотношение $t_{i-1} = t_{i-4} \oplus t_{i-5}$, $i = 0, 1, 2, \ldots$. Это приводит к отсутствию ряда входных комбинаций и, как следствие, невозможности обнаружения некоторых неисправностей, например ≡0 по выходу элемента 2И.

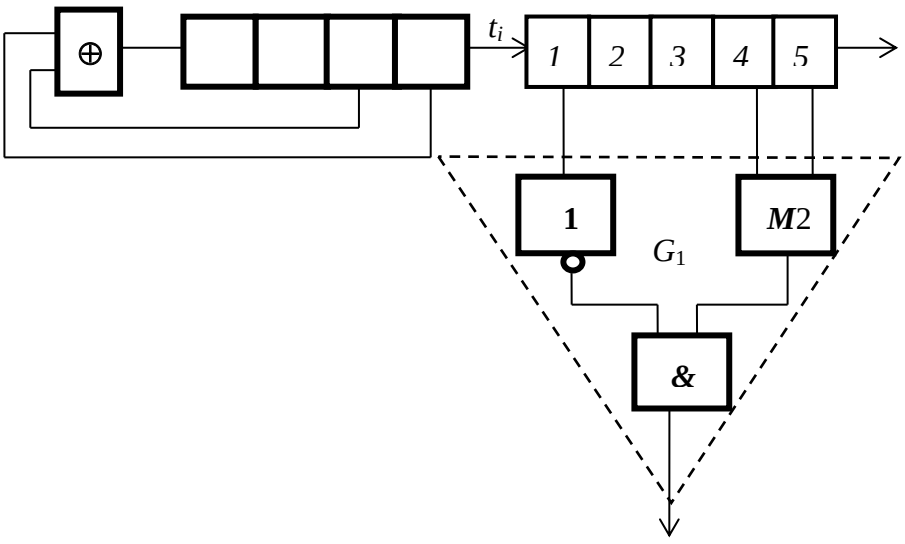


Рисунок 9.2 – *Пример реализации тестирования цифровой схемы*

Возникает задача нахождения порождающего полинома $\varphi(x)$, формирующего псевдо-исчерпывающую M-последовательность t_i, $i=\overline{1,2^m}$, которая удовлетворяет следующим условиям:

1. При генерировании M-последовательности t_i, $i=\overline{1,2^m}$, описываемой полиномом $\varphi(x)$, на разрядах регистра сдвига (см. рис. 9.1), номера $Q_c^j=\overline{1,n_j}$, которых определяются топологией подключения входов j-й подсхемы G_j к регистру, формируются всевозможные n_j-разрядные двоичные комбинации. Это условие должно выполняться для каждой подсхемы G_j.

2. Старшая степень m порождающего полинома $\varphi(x)$ принимает минимально возможное значение, определяемое неравенством:

$$\max_j n_j < m \le w+1,$$

или с учетом возможности формирования нулевого n_j-разрядного кода, в результате начальной установки регистра $P1$ согласно неравенству:

$$\max_j n_j < m \le w. \qquad (9.3)$$

Для обеспечения приведенных условий, которым должен удовлетворять порождающий полином $\varphi(x)$, необходимо выполнить достаточно сложный анализ исследуемой схемы. Возможные подходы при осуществлении подобного анализа и оценка эффективности их конкретного применения будут рассмотрены ниже.

Исключение такого анализа из процедуры синтеза требуемой M-последовательности t_i, $i=\overline{1,2^m}$, в форме определения ее порождающего полинома возможно только в случае увеличения периода и соответственно длины псевдо-исчерпывающей тестовой последовательности. При этом можно ограничиться соотношением (9.1) или выражением:

$$\deg\varphi(x)=\max_j(\max_c Q_c^j-\min_c Q_c^j)+1, \qquad (9.4)$$

где Q_c^j – номер разряда регистра сдвига (см. рис. 9.1), выход которого подключен к c-му, $c=\overline{1,n_j}$, входу j-й подсхемы G_j.

Для примера, представленного на рис. 9.2, множество комбинационных подсхем G_j состоит из одной подсхемы G_1, для которой получим: $c=\overline{1,3}$, $Q_1^1=1, Q_2^1=4$ и $Q_3^1=5.$

Из соотношения (9.4) получим: deg $\varphi(x)$ = , $Q_3^1-Q_1^1+1=5-1+1=5$ откуда следует, что использование, например, полинома $\varphi(x) = 1 \oplus x^2 \oplus x^5$ позволит осуществить исчерпывающее тестирование рассматриваемой схемы G_1 (рис. 9.2). Длина l тестовой последовательности будет составлять величину $2^5 - 1 = 31$, определяемую периодом M-последовательности.

Таким образом, с помощью соотношения (9.4) можно найти значение старшей степени полинома $\varphi(x)$, который обеспечивает исчерпывающее тестирование рассматриваемой схемы без детального ее исследования. Причем любой примитивный полином $\varphi(x)$ имеющий степень, определенную согласно (9.4), соответствует условиям, предъявляемым к подобным полиномам.

Возвращаясь к рассматриваемому примеру, нетрудно показать, что исчерпывающее тестирование комбинационной схемы G_1 (см. рис. 9.2) будет обеспечиваться и при использовании полинома $\varphi(x) = 1 \oplus x^2 \oplus x^3$. Действительно, как видно из временной диаграммы состояний элементов памяти регистра сдвига (рис. 9.3), приведенной в таблице 9.1, на три входа комбинационной схемы G_1 подаются всевозможные комбинации (кроме нулевой), состоящие из трех двоичных символов.

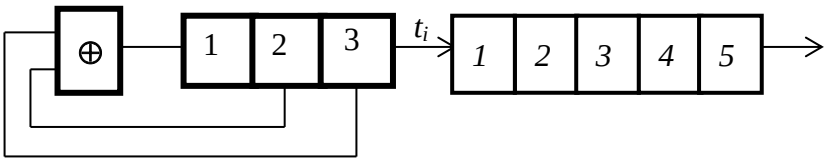


Рисунок 9.3 – *Пример реализации псевдо-исчерпывающего теста*

Как видно из приведенной диаграммы процедуры тестирования, первые $w = 5$ тактов функционирования генератора M-последовательности необходимы для инициализации регистра сдвига, к которому подключена тестируемая схема.

Таблица 9.1 – *Временная диаграмма тестирования*

№	Номе разряда LFSR			Номер разряда регистра сдвига				
	1	2	3	*1*	*2*	*3*	*4*	*5*
1	1	0	0	*X*	*X*	*X*	*X*	*X*
2	0	1	0	0	*X*	*X*	*X*	*X*
3	1	0	1	0	0	*X*	*X*	*X*
4	1	1	0	1	0	0	*X*	*X*
5	1	1	1	0	1	0	0	*X*
6	0	1	1	***1***	0	1	***0***	***0***
7	0	0	1	***1***	1	0	***1***	***0***
8	1	0	0	***1***	1	1	***0***	***1***
9	0	1	0	***0***	1	1	***1***	***0***
10	1	0	1	***0***	0	1	***1***	***1***
11	1	1	0	***1***	0	0	***1***	***1***
12	1	1	1	***0***	1	0	***0***	***1***

В течение последующих 7 тактов, на *1*, *4* и *5* разрядах регистра сдвига, и, соответственно, на входах тестируемой схемы будут сформированы 2^3-1 двоичные комбинации. При этом длина тестовой последовательности $l = 5 +$

7 = 12 будет значительно меньше соответствующей величины для случая использования полинома $\varphi(x) = 1 \oplus x^2 \oplus x^5$.

Как видно из предыдущего анализа, попытка исключить процедуру синтеза оптимальной *M*-последовательности в смысле минимально возможного ее периода приводит к существенному увеличению длины l тестовой последовательности. В этом случае псевдо-исчерпывающее тестирование реализуется любой *M*-последовательностью, старшая степень которой удовлетворяет соотношению (9.4). В то же время в результате целенаправленного синтеза *M*-последовательности возможно нахождение полинома $\varphi(x)$, старшая степень которого определяется величиной $m = \max_j n_j$ [14]. Тогда минимально возможную длину l тестовой последовательности $\{t_i\}$, представляющей собой *M*-последовательность, можно найти по выражению:

$$l = 2^m + w - 1, \qquad (9.5)$$

где w – количество разрядов регистра сдвига (см. рис. 9.1), а величины w и $2^m - 1$ – соответственно длина установочной фазы и фазы тестирования, сумма которых является длиной l тестовой последовательности $\{t_i\}$. В то же время величина l не учитывает нулевую комбинацию из m бит обеспечиваемую за счет начальной установки сдвигового регистра в нулевое состояние.

В случае практической сложности начальной установки в заданное состояние сдвигового регистра, в данном случае в нулевое состояние, может быть предложено альтернативное решение, основанное на последовательностях де Брейна [207, 301]. Подобные последовательности позволяют формировать полное множество из 2^m двоичных комбинаций на $m = \max_j n_j$ последовательных разрядах регистра сдвига.

Однако реализация генератора подобной последовательности требует дополнительного использования ($m - 1$)-входового элемента И и двухвходового элемента М2 (см. главу 7). Кроме того, минимальная длина *M*-последовательности будет увеличена на единицу.

Для примера, приведенного на рис. 9.3, на любом множестве последовательных разрядов регистра сдвига длиной 3 формируются всевозможные двоичные комбинации за исключением комбинации 0 0 0, которая отсутствует на трех разрядах LFSR (таблица 9.1). Это объясняется тем, что тестовой последовательностью служит *M*-последовательность, описываемая полиномом $\varphi(x) = 1 \oplus x^2 \oplus x^3$. Используя данный полином, можно реализовать генератор последовательности де Брейна, приведенный на рис. 9.4 [207, 301].

Данный генератор позволит обеспечить на любых трех последовательных разрядах регистра сдвига, приведенного на рис 9.3, всевозможные двоичные комбинации. Как видно из рис. 9.4, реализация такого генератора отличается несколько большими аппаратурными затратами.

Следовательно, основная задача, возникающая при реализации псевдо-исчерпывающего тестирования, в результате использования *M*-

последовательностей, является задача нахождения порождающего полинома $\varphi(x)$, формирующего последовательность $\{t_i\}$, которая обеспечивает исчерпывающее тестирование комбинационной части цифрового устройства, подключенного к единому регистру сдвига.

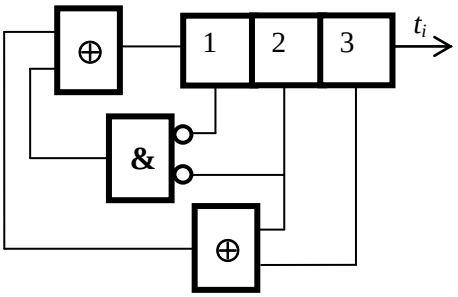


Рисунок 9.4 – *Пример реализации генератора последовательности де Брейна*

Рассмотрим ряд решений подобной задачи для случая комбинационных схем вычислительных систем, который легко обобщается для произвольного объекта тестирования.

9.2. Синтез генераторов псевдо-исчерпывающих тестов

Рассмотрим возможные подходы для решения задачи синтеза генераторов тестовых последовательностей на базе *M*-последовательностей, используемых при реализации псевдо-исчерпывающих тестов для вычислительных систем (см. рис. 9.1). Из анализа указанной структуры, общепринятой в рамках контролепригодного синтеза, следует, что топология подключения каждой комбинационной подсхемы G_j, $j=\overline{1,r}$, схемы G к регистру сдвига может принимать совершенно произвольный вид. При этом однозначной характеристикой, показывающей взаимосвязь схемы G_j с разрядами регистра сдвига, будет множество их номеров $\{Q_1^j, Q_2^j, Q_3^j, ..., Q_{n_j}^j\}$, где n_j представляет собой общее число входов схемы G_j.

На основании значений Q_c^j, $c=\overline{1,n_j}$ определяется величина:

$$Q^j = \max_c Q_c^j - \min_c Q_c^j, \qquad (9.6)$$

показывающая структуру связей рассматриваемой схемы. Так, при $Q^j = n_j - 1$ все входы G_j подключены к последовательным разрядам регистра сдвига, а для $Q^j = w - 1$ к определенным n_j разрядам, лежащим между первым и последним разрядами регистра сдвига. При этом первый случай наиболее благоприятный для синтеза генератора псевдо-исчерпывающей тестовой последовательности, представляющей собой *M*-последовательность. Действительно, для величины:

$$Q = \max_j Q^j, \tag{9.7}$$

принимающей небольшие значения (20−30) [207, 302], проблема синтеза генератора псевдослучайной тестовой последовательности сводится к тривиальному выбору полинома $\varphi(x)$, старшая степень которого соответствует выражению (9.4) и равна $Q + 1$. Для величин Q, имеющих большие значения, стоит задача синтеза подобного генератора, обеспечивающего всевозможные 2^m двоичные комбинации на произвольных $m = \max_j n_j$ разрядах w-разрядного регистра сдвига.

Решим задачу синтеза генератора тестовой последовательности для примера, приведенного на рис. 9.2. Предварительно докажем следующую теорему [320].

Теорема 9.1. Множество выражений вида $t_i = t_{i+p1} \oplus t_{i+p2}$, справедливых для символов $t_i \in \{0,\ 1\}$, $i = \overline{0, L-1}$, M-последовательности, описываемой примитивным полиномом $\varphi(x)$, не пересекается с множеством выражений $s_i = s_{i+g1} \oplus s_{i+g2,}$ выполняемых для символов $s_i \in \{0,\ 1\}$, $i = \overline{0, L-1}$, M-последовательности, описываемой полиномом $\varphi(x)^{-1}$, сопряженным полиному $\varphi(x)$, тогда и только тогда, когда $L \bmod 3 \neq 0$, где $L = 2^m - 1$; $m = \deg \varphi(x)$, а $p1, p2, g1, g2 \in \{0, 1, 2, \ldots, L-1\}$.

Доказательство теоремы 9.1. Предположим, что примитивный полином $\varphi(x)$ порождает M-последовательность $\{t_i\}$, $i = \overline{0, L-1}$, а сопряженный ему полином $\varphi(x)^{-1}$ последовательность $\{s_i\}$, $i = \overline{0, L-1}$. Учитывая, что $\varphi(x)^{-1}$ порождает инверсную последовательность, получаем [207, 295, 302]:

$$\{t_i\} = \{s_{L-i-1}\},\ i = \overline{0, L-1}. \tag{9.8}$$

На основании свойства сдвига и сложения для M-последовательности $\{t_i\}$, $i = \overline{0, L-1}$ можно записать следующее соотношение:

$$t_i = t_{i+p1} \oplus t_{i+p2},\ p1 > p2, \tag{9.9}$$

которое свидетельствует о линейной взаимосвязи между ее символами t_i, t_{i+p1} и t_{i+p2} для определенных $p1$ и $p2$.

Используя выражение (9.8), соотношение (9.9) преобразуем к виду:

$$s_{L-i-1} = s_{L-i-p1-1} \oplus s_{L-i-p2-1}. \tag{9.10}$$

В то же время для символов последовательности $\{s_i\}$, $i = \overline{0, L-1}$ справедливо соотношение, аналогичное равенству (9.9):

$$s_i = s_{i+p1} \oplus s_{i+p2},\ g1 > g2. \tag{9.11}$$

Выражения (9.10), (9.11) справедливы для любых значений $i = \overline{0, L-1}$, а их совместное выполнение для $g1 = p1$ и $g2 = p2$ свидетельствует о том, что множество соотношений (9.9) пересекается с множеством аналогичных тождеств (9.11).

Определим условия совместного выполнения соотношений (9.9) и (9.11). Согласно (9.10) при $i = L - p1 - 1$ и (9.11) при $i = 0$ и $g1 = p1$, $g2 = p2$, получим систему из двух тождеств:

$$s_{p1} = s_0 \oplus s_{p1-p2};$$
$$s_0 = s_{p1} \oplus s_{p2},$$

которая будет выполняться только при $p2 = p1 - p2$, т. е. для $p1 = 2 \times p2$.

При $i = L - p2 - 1$ из (9.10) и $i = 0$ из (9.11) для $g1 = p1$, $g2 = p2$ имеем [301]:

$$s_{p2} = s_{L+p2-p1} \oplus s_0,$$
$$s_0 = s_{p1} \oplus s_{p2}.$$

Последняя система тождеств справедлива для $p1 = L + p2 - p1$, т. е. для $2 \times p1 = L + p2$.

Таким образом, условием совместного выполнения соотношений (9.9) и (9.11), показывающих линейную взаимосвязь символов t_i, t_{i+p1} и t_{i+p2} последовательности $\{t_i\}$, $i = \overline{0, L-1}$, и s_i, s_{i+g1} и s_{i+g2} последовательности $\{s_i\}$, $i = \overline{0, L-1}$, являются равенства $p1 = 2 \times p2$ и $2 \times p1 = L + p2$, из которых следует, что признаком пересечения множеств соотношений (9.9) и (9.11) будет выражение $3 \times p2 = L$ или делимость величины L на 3, т. е. $L \bmod 3 = 0$. В противном случае, когда $L \bmod 3 \neq 0$, множества указанных соотношений являются непересекающимися. Что и требовалось доказать.

Следствие. 9.1. Последовательное использование M-последовательностей, описываемых полиномами $\varphi(x)$ и $\varphi(x)^{-1}$, позволяет формировать тестовые последовательности, обеспечивающие полное множество двоичных комбинаций из трех бит на любых трех разрядах регистра сдвига (см. рис. 9.1).

Необходимо отметить, что в данном случае нулевая комбинация обеспечивается начальной установкой регистра сдвига в нулевое состояние всех его разрядов.

Применение примитивных полиномов $\varphi(x)$ и $\varphi(x)^{-1}$, для которых $L \bmod 3 \neq 0$, позволяет обеспечить для любой подсхемы G_j схемы $\boldsymbol{G}$ всевозможные входные наборы на $n_j \leq 3$ ее входах, подключенных к w произвольным разрядам входного сдвигового регистра.

Для примера, приведенного на рис. 9.2, количество n_j входов единственной подсхемы G_1 схемы G равно 3, а величина $Q^1 = \max_c Q_c^1 - \min_c Q_c^1, c = \overline{1,3}$, так же как и $Q = \max_j Q^j$, равняется 4. Следова-

тельно, для реализации самотестирования исследуемой схемы возможны два альтернативных варианта.

Первый вариант основывается на использовании примитивного полинома $\varphi(x)$, для которого $\deg \varphi(x) = Q + 1 = 5$. Как показывалось ранее, применение, например, полинома $\varphi(x) = 1 \oplus x^2 \oplus x^5$ обеспечивает формирование всевозможных комбинаций из трех двоичных символов на входах схемы G_1 (см. рис. 9.2). Однако в данном случае длина $l = 31$ тестовой последовательности, без учета установочной фазы, значительно превышает минимально возможное значение $l = 12$, определяемое по выражению (9.5).

Второй вариант основывается на использовании следствия теоремы 9.1, согласно которому применение полиномов $\varphi(x) = 1 \oplus x^2 \oplus x^3$ и $\varphi(x)^{-1} = 1 \oplus x^1 \oplus x^3$ позволяет обеспечить всевозможные двоичные комбинации на трех входах схемы G_1 (см. рис. 9.2).

Аналогичный результат, заключающийся в формировании всевозможных комбинаций из $v \le 3$ двоичных переменных на v входах схемы G_j, может быть получен при последовательном использовании прямых t_i и инверсных $\bar{t}_i$ значений символов M-последовательности $\{t_i\}$, $i = \overline{0, L-1}$. Для них справедлива следующая теорема.

Теорема 9.2. Последовательное генерирование прямых t_i и инверсных $\bar{t}_i$ значений символов M-последовательности $\{t_i\}$, $i = \overline{0, L-1}$, описываемой полиномом $\varphi(x)$, позволяет формировать всевозможные двоичные комбинации на $v \le 3$ произвольных разрядах регистра сдвига, разрядность w которого удовлетворяет неравенству ю $w \le 2^m - 1$, где $m = \deg \varphi(x)$, $L = 2^m - 1$.

Доказательство теоремы 9.2. Генерирование прямых t_i значений символов M-последовательности $\{t_i\}$, $i = \overline{0, L-1}$ позволяет формировать всевозможные комбинации из $v \le 2$ двоичных символов. Это является следствием, вытекающим из свойств последовательностей, формируемых на основании примитивных полиномов [123, 159]. Любые комбинации из $v \le 2$ символов будут обеспечиваться на произвольных v разрядах сдвигового регистра, длина которого $w \le 2^m - 1$. В то же время для определенных $v = 3$ разрядов указанного регистра всевозможные комбинации будут обеспечиваться только при условии, если между символами t_i, t_{i+p1} и t_{i+p2}, формируемыми на них, будет отсутствовать линейная зависимость. В противном случае, когда значения указанных символов связаны линейным соотношением (9.9), на соответствующих разрядах регистра сдвига формируются только двоичные комбинации вида 0 0 0, 1 0 1, 0 1 1, 1 1 0 и отсутствуют комбинации 0 0 1, 1 0 0, 0 1 0 и 1 1 1.

При формировании инверсных значений $\bar{t}_i$ последовательности $\{t_i\}$, $i = \overline{0, L-1}$, на разрядах регистра сдвига, определенных соотношением (9.9), будут формироваться инверсные коды $\bar{t}_i$ $\overline{t_{i+p1}}$ $\overline{t_{i+p2}}$. При этом коду t_i t_{i+p1} t_{i+p2}

= 0 0 0 соответствует $\bar{t}_i$ $\overline{t_{i+p1}}$ $\overline{t_{i+p2}}$ = 1 1 1, коду 1 0 1 – 0 1 0, 0 1 1 – 1 0 0 и 1 1 0 – 0 0 1.

Таким образом, в результате использования прямых t_i и инверсных $\bar{t}_i$ значений символов *M*-последовательности $\{t_i\}$, $i=\overline{0,L-1}$ формируются всевозможные двоичные комбинации на $v \le 3$ произвольных разрядах регистра сдвига, количество которых удовлетворяет неравенству $w \le 2^m - 1$. Что и требовалось доказать.

Практическая реализация генератора, формирующего инверсные значения $\bar{t}_i$ *M*-последовательности $\{t_i\}$, $i=\overline{0,L-1}$, отличается от классической структуры генератора *M*-последовательности наличием дополнительного входа в сумматоре по модулю два, включенном в цепь обратной связи регистра сдвига. Для генератора *M*-последовательности, описываемой полиномом $\varphi(x) = 1 \oplus x^2 \oplus x^3$, функциональная схема подобного устройства, формирующего инверсную последовательность $\bar{t}_i$, приведена на рис. 9.5.

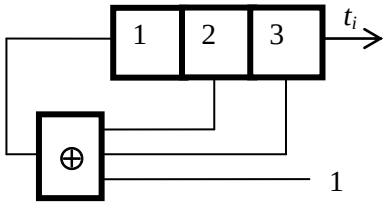


Рисунок 9.5 – *Функциональная схема генератора инверсной M-последовательности*

Данное устройство будет генерировать указанную последовательность при наличии логической единицы на дополнительном входе сумматора по модулю два. Для формирования последовательности t_i на управляющий вход вместо единицы подается значение логического нуля.

Достоинство использования прямых и инверсных значений *M*-последовательности для обеспечения всевозможных двоичных комбинаций на $v \le 3$ произвольных разрядах регистра сдвига заключается в возможности формирования нулевого кода, состоящего из трех символов, что подтверждается временной диаграммой состояний элементов памяти устройства (рис. 9.5).

Рассмотренные два подхода решения задачи синтеза генераторов псевдослучайных тестовых последовательностей, используемых при формировании псевдо-исчерпывающих тестов, эффективно применяются только для цифровых устройств, состоящих из множества комбинационных подсхем G_j, $j=\overline{1,r}$, для которых выполняется неравенство $\max\limits_j n_j \le 3$. Решение подобной задачи для более общего случая, когда $\max\limits_j n_j$ принимает произвольное значение, требует привлечения более фундаментальных результатов, получен-

ных в теории циклических кодов и комбинаторике. Исследования использования положений указанных теорий для построения генераторов псевдо-счерпывающих тестовых последовательностей приведены в разделе 8.3.

В ряде случаев положения теории кодирования позволяют получить приемлемые результаты. Например, применение классического кода Голея, описываемого полиномом $\varphi(x) = 1 \oplus x \oplus x^5 \oplus x^6 \oplus x^7 \oplus x^9 \oplus x^{11}$ и имеющего длину $w = 23$, обеспечивает всевозможные двоичные комбинации из $v \le 6$ двоичных бит на любых 6 из 23 разрядах регистра сдвига [207, 302].

Рассмотренные подходы и примеры использования M-последовательностей, а также теории циклических кодов для реализации псевдо-исчерпывающего тестирования показывают принципиальную возможность решения подобной задачи для заданного количества входов n_j комбинационной подсхемы G_j, $j = \overline{1,r}$, комбинационной части вычислительной системы, структура которой в режиме контроля имеет вид, приведенный на рис. 9.1. Однако возможности данного подхода существенно ограничиваются необходимостью предварительной установки начальных кодов для генерирования всех подмножеств полного множества кодовых слов, что предопределяет включение в состав вычислительной системы запоминающих устройств для хранения указанных кодов. Альтернативное решение, заключающееся в записи начальных кодов, поступающих извне системы, может заметно увеличить время проведения тестирования, которое для реальных значений n_j = 20-30 достигает нереально больших величин.

В результате анализа результатов, представленных в данном разделе можно сформулировать недостатки, аналогичные недостаткам метода, основанного на циклических кодах. Наиболее существенным недостатком всех методов синтеза генераторов псевдо-исчерпывающих тестовых последовательностей является большая длина формируемых последовательностей генераторами, построенными на их основе. Применяя рассмотренные методы синтеза генераторов псевдо-исчерпывающих тестовых последовательностей, можно решать задачу обеспечения исчерпывающего тестирования n_j-входовой схемы G_j, $j = \overline{1,r}$, входы которой подключаются к любым n_j из w разрядов регистра сдвига. В то же время реально ее входы подключаются к конкретным n_j разрядам, что упрощает постановку решаемой задачи. Рассмотрим возможные пути обеспечения псевдо-исчерпывающего тестирования для конкретных случаев топологии подключения комбинационной части к разрядам регистра сдвига.

9.3. Псевдо-исчерпывающие тесты для заданной топологии подключения

Как отмечалось выше, структура вычислительной системы, реализующей в режиме тестирования один из методов сканирования пути, однозначно

описывается множествами $\{Q_1^j, Q_2^j, Q_3^j, ..., Q_{n_j}^j\}$ номеров разрядов регистра сдвига (см. рис. 9.1), к которым подключены входы подсхем G_j, $j = \overline{1,r}$, комбинационной части вычислительной системы. Множества номеров разрядов регистра сдвига дают полную информацию, необходимую для синтеза генераторов псевдо-исчерпывающих тестовых последовательностей. Последнее, как правило, основывается на применении псевдослучайных тестовых *M*-последовательностей, формируемых в соответствии с примитивными порождающими полиномами.

Основной задачей синтеза генератора псевдо-исчерпывающей тестовой последовательности является нахождение порождающего полинома $\varphi(x)$, формирующего *M*-последовательность $\{t_i\}$, $i = \overline{0, L-1}$, обеспечивающую генерирование всевозможных двоичных комбинаций на всех множествах $Q_1^j, Q_2^j, Q_3^j, ..., Q_{n_j}^j$ разрядов регистра сдвига. Рассмотрим решение данной задачи для случая, когда комбинационная часть вычислительной системы состоит из одной подсхемы G_1, топология подключения входов которой описывается множеством $\{Q_1, Q_2, Q_3, ..., Q_n\}$.

Первоначально множеству $\{Q_1, Q_2, Q_3, ..., Q_n\}$ номеров разрядов регистра сдвига поставим в соответствие значения символов t_i, $t_{i+\tau(1)}$, $t_{i+\tau(2)}$, …, $t_{i+\tau(n-1)}$, $k = 0, 1, 2, ...$, искомой *M*-последовательности, где $\tau(j) = Q_{j+1} - Q_1$, $j = \overline{0, n-1}; Q_{j+1} > Q_1$, а $\tau(0) = 0$. Тогда исходная задача реализации псевдо-исчерпывающего тестирования комбинационной части G_1 сводится к задаче обеспечения линейной независимости между символами t_i, $t_{i+\tau(1)}$, $t_{i+\tau(2)}$, …, $t_{i+(n-1)}$ искомой *M*-последовательности. Ее порождающий полином $\varphi(x)$ имеет старшую степень *m*, определяемую согласно (9.2), минимальное значение которой равно *n*.

На основании свойства сдвига и сложения для *M*-последовательности $\{t_i\}$, $i = \overline{0, L-1}$, можно показать, что любой ее символ $t_{i+\tau}$, где $\tau \in \{0, 1, 2, ..., 2^m - 2\}$, представляется в виде суммы по модулю два некоторого подмножества символов t_i, t_{i+1}, t_{i+2}, …, t_{i+m-1}, определяемого коэффициентами $\delta_i(\tau) \in \{0, 1\}$, $i = \overline{1, m}$, выражения [301]:

$$t_{i+\tau} = \sum_{l=1}^{m} {}^{\oplus} \delta_l(\tau)\, t_{i+l-1}. \qquad (9.12)$$

Каждому значению τ будет соответствовать отличное от других множество коэффициентов $\delta_l(\tau)$ [301].

В качестве примера, иллюстрирующего соотношение (9.12), можно привести значения коэффициентов $\delta_l(\tau)$, $\tau \in \{0, 1, 2, ..., 30\}$, (см. таблицу 9.2) для символов *M*-последовательности, порождаемой полиномом $\varphi(x) = 1 \oplus x^3 \oplus x^5$.

Таблица 9.2 – *Значения коэффициентов* $\delta_1(\tau)$, $\delta_2(\tau)$, $\delta_3(\tau)$, $\delta_4(\tau)$, $\delta_5(\tau)$

τ	$\delta_1(\tau)$	$\delta_2(\tau)$	$\delta_3(\tau)$	$\delta_4(\tau)$	$\delta_5(\tau)$	τ	$\delta_1(\tau)$	$\delta_2(\tau)$	$\delta_3(\tau)$	$\delta_4(\tau)$	$\delta_5(\tau)$
0	1	0	0	0	0	*16*	1	1	0	1	1
1	0	1	0	0	0	*17*	1	1	0	0	1
2	0	0	1	0	0	*18*	1	1	0	0	0
3	0	0	0	1	0	*19*	0	1	1	0	0
4	0	0	0	0	1	*20*	0	0	1	1	0
5	1	0	1	0	0	*21*	0	0	0	1	1
6	0	1	0	1	0	*22*	1 .	0	1	0	1
7	0	0	1	0	1	*23*	1	1	1	1	0
8	1	0	1	1	0	*24*	0	1	1	1	1
9	0	1	0	1	1	*25*	1	0	0	1	1
10	1	0	0	0	1	*26*	1	1	1	0	1
11	1	1	1	0	0	*27*	1	1	0	1	0
12	0	1	1	1	0	*28*	0	1	1	0	1
13	0	0	1	1	1	*29*	1	0	0	1	0
14	1	0	1	1	1	*30*	0	1	0	0	1
15	1	1	1	1	1						

Конкретные значения коэффициентов $\delta_l(\tau)$, однозначно определяются величиной τ и видом порождающего полинома $\varphi(x)$ и могут быть вычислены по алгоритмам, приведенным в [301]. Отметим, что для $\tau < m$ независимо от порождающего полинома только один коэффициент $\delta_{l+\tau}(\tau)$ множества $\{\delta_l(\tau)\}$ искомых коэффициентов равен 1. Действительно, для τ = 1 и 3, по данным таблицы 9.2, имеем $\delta_1(\tau)\ \delta_2(\tau)\ \delta_3(\tau)\ \delta_4(\tau)\ \delta_5(\tau)$ = 0 1 0 0 0 и $\delta_1(\tau)\ \delta_2(\tau)\ \delta_3(\tau)\ \delta_4(\tau)\ \delta_5(\tau)$ = 0 0 0 1 0.

Любому множеству коэффициентов $\delta_l(\tau) \in \{0,\ 1\}$, $l = \overline{1, m}$, определяемому величиной τ, можно поставить в соответствие полином $\varphi_\tau(x) = \delta_1(\tau)x^0 \oplus \delta_2(\tau)x^1 \oplus \ldots \oplus \delta_{m-2}(\tau)x^{m-2} \oplus \delta_{m-1}(\tau)x^{m-1}$, который однозначно зависит от конкретных значений $\delta_l(\tau)$. Полином $\varphi_\tau(x)$ может быть получен как остаток от деления x^τ на полином $\varphi(x)^{-1}$, сопряженный порождающему полиному $\varphi(x)$ [301]:

$$\varphi_\tau(x) = x^\tau \bmod \varphi(x)^{-1}. \qquad (9.13)$$

Используя приведенные определения полиномов $\varphi_\tau(x)$, докажем следующую теорему.

Теорема 9.4. Примитивный полином $\varphi(x)$, используемый для построения генератора псевдо-исчерпывающей тестовой последовательности, позволяет обеспечить всевозможные двоичные комбинации на разрядах Q_1, Q_2, Q_3, ..., Q_n ($Q_{j+1} > Q_j$, $j = \overline{1, n-1}$) регистра сдвига только в том случае, когда любое подмножество полиномов $\varphi_{\tau(0)}(x) = x^{\tau(0)} \bmod \varphi(x)^{-1}$, $\varphi_{\tau(1)}(x) = x^{\tau(1)} \bmod \varphi(x)^{-1}$,

$\varphi_{\tau(2)}(x) = x^{\tau(2)} \bmod \varphi(x)^{-1}$, …, $\varphi_{\tau(n-1)}(x) = x^{\tau(n-1)} \bmod \varphi(x)^{-1}$, где $\tau(\gamma-1) = Q_\gamma - Q_1$, $\gamma = \overline{1,n}$) является линейно независимым над полем $GF(2)$.

Доказательство теоремы 9.4. Предположим, что выполняется противоположное утверждение, т. е. всевозможные двоичные комбинации будут обеспечиваться на разрядах Q_1, Q_2, Q_3, ..., Q_n ($Q_{j+1} > Q_j$, $j = \overline{1,n-1}$) регистра сдвига в случае, когда полиномы $\varphi_{\tau(0)}(x) = x^{\tau(0)} \bmod \varphi(x)^{-1}$, $\varphi_{\tau(1)}(x) = x^{\tau(1)} \bmod \varphi(x)^{-1}$, $\varphi_{\tau(2)}(x) = x^{\tau(2)} \bmod \varphi(x)^{-1}$, …, $\varphi_{\tau(n-1)}(x) = x^{\tau(n-1)} \bmod \varphi(x)^{-1}$ или любое их подмножество линейно зависимо над полем GF (2).

Для случая линейной зависимости полиномов $\varphi_{\tau(0)}(x)$, $\varphi_{\tau(1)}(x)$, $\varphi_{\tau(2)}(x)$, ..., $\varphi_{\tau(n-1)}(x)$ будет выполняться следующее равенство:

$$\varphi_{\tau(0)}(x) \oplus \varphi_{\tau(1)}(x) \oplus \varphi_{\tau(2)}(x) \oplus \ldots \oplus \varphi_{\tau(n-1)}(x) = 0, \tag{9.14}$$

из которого следует, что для коэффициентов $\delta_l(\tau_{(\gamma-1)})$, $l = \overline{1,m}$, $\gamma = \overline{1,n}$ определяющих вид полинома $\varphi_{\tau(\gamma-1)}(x) = \delta_1(\tau_{(\gamma-1)})x^0 \oplus \delta_2(\tau_{(\gamma-1)})x^1 \oplus \delta_3(\tau_{(\gamma-1)})x^2 \oplus \ldots \oplus \delta_{m-1}(\tau_{(\gamma-1)})x^{m-2} \oplus \delta_m(\tau_{(\gamma-1)})x^{m-1}$, выполняется система тождеств:

$$\begin{aligned}
&\delta_1(\tau_{(0)}) \oplus \delta_1(\tau_{(1)}) \oplus \delta_1(\tau_{(2)}) \oplus \ldots \oplus \delta_1(\tau_{(n-1)}) = 0,\\
&\delta_2(\tau_{(0)}) \oplus \delta_2(\tau_{(1)}) \oplus \delta_2(\tau_{(2)}) \oplus \ldots \oplus \delta_2(\tau_{(n-1)}) = 0,\\
&\qquad\qquad \ldots\\
&\delta_m(\tau_{(0)}) \oplus \delta_m(\tau_{(1)}) \oplus \delta_m(\tau_{(2)}) \oplus \ldots \oplus \delta_m(\tau_{(n-1)}) = 0.
\end{aligned} \tag{9.15}$$

Используя (9.15), можно показать, что для символов M-последовательности, описываемых полиномом $\varphi(x)$, справедливо следующее равенство:

$$t_{i+\tau(0)} \oplus t_{i+\tau(1)} \oplus t_{i+\tau(2)} \oplus \ldots \oplus t_{i+\tau(n-1)} = 0, \tag{9.16}$$

где каждый символ представляется в виде соотношения (9.12).

Равенство (9.16) показывает линейную зависимость между символами M-последовательности, формируемыми на разрядах регистра сдвига, номера которых определяются множеством $\{Q_1, Q_2, Q_3, ..., Q_n\}$, а также невозможность генерирования на них различных двоичных комбинаций. Их множество определяется выражением (9.16), из которого следует невозможность получения наборов $t_{i+\tau(0)}\ t_{i+\tau(1)}\ t_{i+\tau(2)}\ \ldots\ t_{i+\tau(n-1)}$, состоящих из нечетного числа единичных символов.

Аналогичным образом показывается несостоятельность утверждения, противоположного условию теоремы, для любого подмножества полиномов $\varphi_{\tau(0)}(x)$, $\varphi_{\tau(1)}(x)$, $\varphi_{\tau(2)}(x)$, ..., $\varphi_{\tau(n-1)}(x)$.

Всевозможные двоичные комбинации на разрядах Q_1, Q_2, Q_3, ..., Q_n регистра сдвига будут обеспечиваться только в случае отсутствия линейной зависимости символов M-последовательности, формируемой на нем. Но тогда для любого подмножества коэффициентов $\{\delta_l(\tau_{(\gamma-1)})\}$, $l = \overline{1,m}$, $\gamma = \overline{1,n}$, соот-

ветствующих некоторому подмножеству символов $\{t_{i+\tau(\gamma-1)}\}$ не будет выполняться система тождеств (9.15) и обеспечиваться линейная независимость полиномов $\{\varphi_{\tau(\gamma-1)}(x)\}$, что и требовалось доказать.

Таким образом, использование полинома $\varphi(x)$ для построения генератора псевдо-исчерпывающей тестовой последовательности для комбинационной схемы, топология которой описывается множеством $\{Q_1, Q_2, Q_3, ..., Q_n\}$ номеров разрядов регистра сдвига, требует проведения предварительного анализа, т. е. выполнения условий, сформулированных в теореме 9.4. При положительных результатах для полинома $\varphi(x)$, старшая степень которого $m = n$, на разрядах $Q_1, Q_2, Q_3, ..., Q_n$ регистра сдвига СБИС формируются всевозможные двоичные комбинации, за исключением нулевой. Последняя будет обеспечиваться только в случае, когда $m > n$.

Технически процедура анализа выполнения условий теоремы 9.4 основывается на использовании одного из трех соотношений (9.12), (9.14) и (9.15). В первом случае проверяется выполнение линейного соотношения для любого подмножества символов анализируемой M-последовательности посредством представления их в виде суммы символов, принадлежащих множеству $\{t_i, t_{i+1}, t_{i+2}, \ldots, t_{i+m-1}\}$. При использовании соотношения (9.14) алгоритм решения данной задачи базируется на процедуре деления двоичных полиномов над полем $GF(2)$ и, наконец, в случае применения соотношения (9.15) исследуется матрица размерности $n \times m$, для которой вычисляется ее ранг.

Несмотря на тесную связь между методами, реализующими указанные соотношения, их конкретное применение может отличаться трудоемкостью и соответственно временем, необходимым для исследования выполнения условий теоремы 9.4. В качестве примера проведения подобного исследования рассмотрим цифровую схему, приведенную на рис. 9.2, топология подключения комбинационной части которой к регистру сдвига описывается множеством {1, 4, 5}. Пусть значения символов M-последовательности, формируемой на них, обеспечиваются полиномом $\varphi(x) = 1 \oplus x \oplus x^4$. Рассматривая множество {1, 4, 5} номеров разрядов регистра сдвига, получаем величины $\tau(0) = 0$, $\tau(1) = 3$ и $\tau(2) = 4$ и соответствующие им полиномы $\varphi_{\tau(0)}(x) = x^0 \bmod (1 \oplus x^3 \oplus x^4) = 1$, $\varphi_{\tau(1)}(x) = x^3 \bmod (1 \oplus x^3 \oplus x^4) = x^3$, $\varphi_{\tau(2)}(x) = x^4 \bmod (1 \oplus x^3 \oplus x^4) = 1 \oplus x^3$, здесь $\varphi(x)^{-1} = 1 \oplus x^3 \oplus x^4$. Простейший анализ выражений для $\varphi_{\tau(0)}(x)$, $\varphi_{\tau(1)}(x)$ и $\varphi_{\tau(2)}(x)$ показывает, что выполняется следующее соотношение:

$$\varphi_{\tau(0)}(x) \oplus \varphi_{\tau(1)}(x) \oplus \varphi_{\tau(2)}(x) = 1 \oplus x^3 \oplus 1 \oplus x^3 = 0,$$

которое свидетельствует о невозможности обеспечения различных двоичных комбинаций на заданном множестве {1, 4, 5} разрядов регистра сдвига.

Аналогичный результат' может быт получен и при исследовании матрицы A, состоящей из коэффициентов $\delta_1(\tau_{(0)})\delta_2(\tau_{(0)})\delta_3(\tau_{(0)})\delta_4(\tau_{(0)}) = 1\ 0\ 0\ 0$, $\delta_1(\tau_{(1)})\delta_2(\tau_{(1)})\delta_3(\tau_{(1)})\delta_4(\tau_{(1)}) = 0\ 0\ 1\ 0$, $\delta_1(\tau_{(3)})\delta_2(\tau_{(3)})\delta_3(\tau_{(3)})\delta_4(\tau_{(3)}) = 1\ 0\ 1\ 0$.

$$\begin{vmatrix} 1 & 0 & 0 & 0 \end{vmatrix}$$

$$A = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{vmatrix}$$

Ранг данной матрицы равен двум, что свидетельствует о линейной зависимости символов M-последовательности, формируемых на основании коэффициентов $\{\delta_l(\tau_{(\gamma-1)})\}$, $l=\overline{1,4}$, $\gamma=\overline{1,3}$.

Приведенный пример характеризуется малой мощностью множества номеров разрядов регистра и вследствие этого его решение любым из указанных методов является тривиальным. Ситуация существенно изменяется при решении практических задач, для которых размерность множества номеров разрядов регистра сдвига, как правило, превышает 20. Для решения подобных задач используется специальное математическое обеспечение, реализующее один из рассмотренных ранее методов. В простейшем случае математическое обеспечение позволяет получить ответ на вопрос: *Удовлетворяет ли исследуемый полином условию обеспечения всевозможных двоичных комбинаций на заданном множестве разрядов регистра сдвига*? При отрицательном ответе анализируется применимость для данных целей другого примитивного полинома.

Как показано в работе [302], практически во всех случаях процедура поиска соответствующего полинома ограничивается незначительным числом примитивных полиномов, старшая степень m которых равна n. Только в исключительных ситуациях для решения данной задачи необходимо применять полином, для которого $m = n + 1$.

Более сложной является задача нахождения примитивного полинома $\varphi(x)$, обеспечивающего исчерпывающее тестирование комбинационной части СБИС, состоящей из множества подсхем G_j, $j=\overline{1,r}$, топология подключения которых к регистру сдвига описывается множествами $\{Q_1^j, Q_2^j, Q_3^j, ..., Q_{n_j}^j\}$. В этом случае процедура поиска соответствующего полинома носит итерационный характер. Для первой подсхемы G_1 комбинационной части цифрового устройства находится полином $\varphi(x)$, обеспечивающий всевозможные комбинации на разрядах $Q_1^1, Q_2^1, Q_3^1, ..., Q_{n_1}^1$ ее регистра сдвига. Затем для данного полинома проверяется условие обеспечения полного перебора входных наборов для остальных подсхем G_j, $j=\overline{2,r}$. В случае невыполнения данного условия хотя бы для одного множества $\{Q_1^j, Q_2^j, Q_3^j, ..., Q_{n_j}^j\}$, $j=\overline{2,r}$. инициализируется процедура поиска нового полинома для первой подсхемы G_1. Таким образом, данный процесс продолжается до тех пор, пока не будет найден полином $\varphi(x)$, удовлетворяющий условиям теоремы 9.4 для всех множеств $\{Q_1^j, Q_2^j, Q_3^j, ..., Q_{n_j}^j\}$, $j=\overline{1,r}$. В работе [320] приводится примитивный полином $\varphi(x) = 1 \oplus x^3 \oplus x^7 \oplus x^{10} \oplus x^{11} \oplus x^{18} \oplus x^{20}$, использование которого позволяет реализовать псевдо-исчерпывающее тестирование на шести (r = 6) множествах w = 81 разрядного регистра сдвига. Конкретные значения указанных множеств и их размерность (n_j) приведены в таблице 9.3.

Таблица 9.3 – *Множества разрядов регистра сдвига*

j	$Q_1^j,\quad Q_2^j,\quad Q_3^j,\quad ...,\quad Q_{n_j}^j$	n_j
1	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 41, 42, 43, 44, 45, 46, 75}	20
2	{0, 1, 2, 3, 4, 5, 6, 13,14, 15, 16, 17, 47, 48, 49, 50, 51, 76}	18
3	{0, 1, 2, 3, 4, 5, 6, 18, 19, 20, 21, 22, 52, 53, 54, 55, 56, 77}	18
4	{0, 1, 2, 3, 4, 5, 6, 23, 24, 25, 26, 27, 28, 57, 58, 59, 60, 62, 78}	19
5	{0, 1, 2, 3, 4, 5, 6, 29, 30, 31, 32, 33, 34, 63, 64, 65, 66, 67, 68, 79}	20
6	{0, 1, 2, 3, 4, 5, 6, 35, 36, 37, 38, 39, 40, 69, 70, 71, 72, 73, 74, 80}	20

Приведенные методы построения генераторов псевдо-исчерпывающих тестовых последовательностей для заданной топологии подключения цифрового устройства к регистру сдвига и конкретные результаты их применения показывают принципиальную возможность реализации идеи применения *M*-последовательностей для реализации псевдо-исчерпывающих тестов. В качестве основного подхода реализации данной идеи предлагается использовать методику псевдо-исчерпывающего тестирования, альтернативой которому может быть его аппроксимация почти псевдо-исчерпывающими тестами, основанными на использовании псевдослучайных *M*-последовательностей.

9.4. Почти псевдо-исчерпывающие *M*-последовательности

Понятие *почти псевдо-исчерпывающие* (*near pseudo-exhaustive*) тесты аналогично по своей сути термину *почти исчерпывающие* (*near exhaustive*) тесты, впервые определенному и использованному в работах [47, 52]. Под почти псевдо-исчерпывающими тестами понимают тесты, которые не являются псевдо-исчерпывающими, однако максимально близки к псевдо-исчерпывающим тестам, либо с большой вероятностью являются таковыми. Степень близости почти псевдо-исчерпывающих тестов к псевдо-исчерпывающим, как правило, оценивается вероятностью формирования всех возможных наборов, характерных для конкретного псевдо-исчерпывающего теста. Для почти псевдоисчерпывающих тестов эта вероятность близка к единице.

Псевдо-исчерпывающее тестирование современных вычислительных систем, реализующих в режиме тестирования структуру *test-per-scan*, как правило, базируется на формировании последовательностей равновероятных двоичных символов. Генераторами подобных последовательностей могут служить генераторы *M*-последовательностей, которые не требуют для своей реализации сколь-нибудь существенных дополнительных затрат. Оценка эффективности применения конкретной структуры подобного генератора дана в главах 7 и 8. Процедура получения информации о целесообразности использования рассмотренных ранее методов псевдо-исчерпывающего тестирования основывается на полном описании топологии комбинационной части вычислительной системы, что требует достаточно трудоемких вычислений и не

позволяет исследовать сравнительные характеристики различных генераторов *M*-последовательностей.

По аналогии с псевдо-исчерпывающим тестированием рассмотрим возможные подходы для оценки эффективности применения почти псевдо-исчерпывающего тестирования только на основании множеств $\{Q_1^j, Q_2^j, Q_3^j, ..., Q_{n_j}^j\}$, $j = \overline{1,r}$, описывающих топологию связей комбинационной части системы с ее элементами памяти. При этом комбинационная часть вычислительной системы будет представляться в виде r подсхем G_j, $j = \overline{1,r}$, а элементы памяти объединятся в сдвиговой регистр, длина w которого определяется их числом.

В качестве меры близости почти псевдо-исчерпывающего теста к псевдо-исчерпывающему тесту введем вероятность $P(m, v)$ обеспечения всевозможных двоичных комбинаций на $v \le m$ произвольных разрядах сдвигового регистра длины $w \ge m$.

При использовании *M*-последовательности, описываемой примитивным порождающим полиномом $\varphi(x)$, в качестве генератора почти псевдо-исчерпывающего теста, для вероятности $P(m, v)$ будет выполняться следующая теорема [302].

Теорема 9.5. Вероятность $P(m, v)$ обеспечения всевозможных двоичных комбинаций на $v \le m$ разрядах регистра сдвига, состоящего из $w \ge m$ разрядов, в случае генерирования *M*-последовательности, описываемой примитивным порождающим полиномом $\varphi(x)$, для которого $2^m - 1 \ge w$, где $m = \deg \varphi(x)$, определяется согласно соотношению:

$$P(m, v) = p(m, 3)\, p(m, 4)\, p(m, 5) \ldots p(m, v),$$

где $p(m, v) = (2^m - 2^{t-1})/(2^m - t)$, $t = \overline{1,v}$.

Доказательство теоремы 9.5. Предположим, что на $(t - 1)$-м из v разрядов регистра сдвига обеспечиваются всевозможные двоичные комбинации. Это свидетельствует о линейной независимости символов *M*-последовательности, соответствующих $(t - 1)$-м разрядам регистра. Определим вероятность того, что случайно выбранный t-й разряд приведет к тому, что он совместно с любым подмножеством из $(t - 1)$-х разрядов образует линейно независимую группу символов *M*-последовательности.

Для $(t - 1)$-х разрядов регистра сдвига существует $2^{t-1} - 1$ непустых подмножеств разрядов и соответствующих им подмножеств символов *M*-последовательности. Причем в силу того, что $t - 1$ символы *M*-последовательности образует линейно независимое множество, любое их подмножество будет также линейно независимым. Каждому из 2^{t-1} непустых подмножеств будет соответствовать только один t-й разряд, отличный от других, такой, что символ *M*-последовательности, формируемый на нем, будет линейно зависим от исходного подмножества. Исключение составляют $t - 1$ подмножества, состоящие из одного разряда регистра сдвига, в силу выполнения со-

отношения $w \le 2^m - 1$. Следовательно, общее количество случаев линейной зависимости t символов M-последовательности составит:

$$2^{t-1} - 1 - (t - 1) = 2^{t-1} - t,$$

а число возможных позиций, на которых может формироваться t-й символ, $2^m - 1 - (t - 1) = 2^m - t$. При равновероятном выборе t-го разряда регистра сдвига разрядности $w \le 2^m - 1$ вероятность $p(m, t)$ линейной независимости t-го символа M-последовательности от любого подмножества из $(t-1)$-го предварительно определенного набора находим из выражения:

$$p(m,t) = \frac{2^m - t - 2^{t-1} + t}{2^m - t} = \frac{2^m - 2^{t-1}}{2^m - t}, \quad t \le m. \tag{9.17}$$

Для $w < 2^m - 1$ соотношение (9.17) может быть использовано как математическое ожидание вероятности $p(m, t)$ и распространено на общий случай $w \le 2^m - 1$.

Вывод выражения (9.17) основывается на предположении, что $(t-1)$-й символ M-последовательности является линейно независимым так же, как и любое их подмножество. Так как вероятность $p(m, t-1)$ данного события вычисляется по выражению (9.17) и базируется на условии линейной независимости $t - 2$ символов и их любых подмножеств, то вероятность $P(m, v)$ обеспечения всевозможных двоичных комбинаций на v из m разрядах регистра сдвига будет определяться вероятностью совместного выполнения линейной независимости, всех подмножеств символов множеств, состоящих из одного, двух, трех и т. д., и v символов M-последовательности. Учитывая соотношение $w \le 2^m - 1$, окончательно получим:

$$P(m, v) = p(m, 3)\, p(m, 4)\, p(m, 5) \ldots p(m, v), \tag{9.18}$$

где $p(m, t)$, $t = \overline{1, v}$, вычисляется согласно (9.17). Что и требовалось доказать.

Очевидно, что значение вероятности $P(m, v)$ является функцией, зависящей от переменных m и v, где $v \le m$, и ее величина однозначно определяется их конкретным соотношением, о чем свидетельствуют ее численные значения для $m = 10$, 20 и 30, приведенные в таблице 9.4.

Как видно из приведенной таблицы 9.4, величина вероятности $P(m, v)$ зависит только от разности $m - v$. Подтверждением данного факта являются результаты, приведенные в таблице 9.5 для значений $m = \overline{10, 20}$ и разности $m - v = \overline{0, 5}$.

Таблица 9.4 – *Значение вероятности* $P(m, v)$

v	$m = 30$	$m = 20$	$m = 10$
2	1,000000	1,000000	1,000000
4	0,999999	0,999995	0,984361
6	0,999999	0,999959	0,959219
8	0,999999	0,999791	0,798514
10	0,999999	0,999077	0,305077
12	0,999996	0,996173	
14	0,999984	0,984555	
16	0,999939	0,938913	
18	0,999756	0,770227	
20	0,999023	0,288846	
22	0,996099		
24	0,984456		
26	0,938790		
28	0,770101		
30	0,288788		

Из таблиц 9.4 и 9.5 следует, что для практических значений $m > 14$ вероятность $P(m, m) = 0{,}29$. Это свидетельствует о том, что использование примитивного порождающего полинома $\varphi(x)$, для которого $\deg \varphi(x) = m$, позволяет обеспечить всевозможные двоичные комбинации (псевдо-исчерпывающую тестовую последовательность) на m заданных разрядах регистра сдвига (см. рис. 9.1) с вероятностью, равной 0,29. Для достижения большей вероятности реализации псевдо-исчерпывающего тестирования комбинационной части вычислительной системы, подключенной к v разрядам регистра, целесообразно использование полиномов, старшая степень которых $m > v$. Так, например, зависимость $P(m, v)$, приведенная в таблицах 9.4 и 9.5, показывает, что уже для $m = 14$ та же вероятность обеспечения псевдо-исчерпывающего тестирования на $v = 10$ заданных разрядах регистра сдвига достигает 0,94, тогда как для $m = 10$ она составляет 0,29.

Таблица 9.5 – *Значение вероятности* $P(m, v)$ для $m - v = \overline{0,5}$

m	$m - v$					
	0	1	2	3	4	5
10	0,31	0,60	0,80	0,91	0,96	0,98
11	0,30	0,59	0,79	0,90	0,95	0,98
12	0,29	0,59	0,78	0,89	0,95	0,98
13	0,29	0,58	0,78	0,89	0,94	0,97
14	0,29	0,58	0,77	0,88	0,94	0,97
…	…	…	…	…	…	…
20	0,29	0,58	0,77	0,88	0,94	0,97

Таким образом, основным результатом теоремы 9.5 является не только оценка эффективности реализации псевдо-исчерпывающего тестирования, но

и возможность синтеза генератора тестовой последовательности, сущность которого будет заключаться в определении старшей степени порождающего полинома $\varphi(x)$ на основании доверительной вероятности реализации псевдо-исчерпывающего теста. Кроме того, вероятность $P(m, v)$ может быть использована для вычисления трудоемкости поиска примитивного полинома $\varphi(x)$ согласно методам, приведенным в разделе 9.3.

Результаты теоремы 9.5 могут интерпретироваться следующим образом. Случайно выбранный примитивный полином $\varphi(x)$ степени m позволит реализовать псевдо-исчерпывающее тестирование с вероятностью $P(m, v)$, где $v \le m$ определяется количеством входов комбинационной части системы, подключенной к единому регистру сдвига. Причем данная вероятность $P(m, v)$ является вероятностью положительного исхода анализа полинома $\varphi(x)$ на предмет применимости его для реализации псевдо-исчерпывающего тестирования v-входовой комбинационной части системы. При этом процедура поиска соответствующего полинома $\varphi(x)$ среди множества полиномов, старшая степень которых равняется m, характеризуется достаточно невысокой вероятностью его нахождения, равной $P(m, m) \cong 0{,}29$. В данном случае m представляет собой количество входов единственной подсхемы G_1, подключенной к регистру сдвига.

Решение подобной задачи для общего случая, который характеризуется реализацией псевдо-исчерпывающего тестирования для r подсхем, максимальное количество входов которых не превышает m, может быть оценено вероятностью P_d нахождения необходимого полинома $\varphi(x)$.

$$P_d = P(m, m)^r = 0{,}29^r. \qquad (9.19)$$

Отсюда следует, что для $r > 10$ вероятность решения задачи поиска полинома принимает практически нулевое значение. Необходимо отметить, что выражение (9.19) является лишь нижней оценкой реального значения вероятности P_d.

Значительно улучшить полученные оценки P_d можно с помощью применения нескольких генераторов псевдослучайных тестовых M-последовательностей. Так, для двух генераторов M-последовательностей вероятность P_d, аналогичная величине, определяемой согласно (9.19), будет иметь следующий вид:

$$P_d = 1 - (1 - P(m, m)^r)(1 - P(m, m)^r) = 2P(m, m)^r - P(m, m)^{2r}.$$

С учетом полученных оценок можно сделать вывод, что более целесообразным является использовать несколько генераторов псевдослучайных тестовых последовательностей, старшая степень m которых удовлетворяет неравенству $m > v$, где v является максимальным количеством входов подсхем схемы G. При этом верхняя оценка величины m будет определяться

временем проведения тестового эксперимента, которое пропорционально величине $2^m - 1$.

Из изложенного выше следует, что применение множеств генераторов псевдослучайных *M*-последовательностей, а также увеличение старших степеней, порождающих их примитивных полиномов, позволяют повысить эффективность реализации псевдо-исчерпывающего тестирования.

Глава 10. Компактное тестирование

10.1. Определение компактного тестирования

Традиционные методы тестирования современных вычислительных систем основаны на формировании тестовых последовательностей, позволяющих обнаруживать заданное множество их неисправностей. Для проведения процедуры тестирования необходимо хранить как тестовые последовательности, так и эталонные выходные реакции на их воздействие. В процессе тестирования реальные выходные реакции сравниваются с эталонными значениями, и по результатам сравнения принимается решение о состоянии проверяемой системы. Только при равенстве полученных реакций эталонным их значениям система считается исправной.

Для современных вычислительных систем традиционный подход требует значительных временных затрат как на получение тестовых последовательностей, так и на реализацию самой процедуры тестирования. Кроме того, большие объемы тестовых воздействий и соответствующих им эталонных реакций предполагают наличие сложного оборудования для проведения тестового эксперимента. В связи с этим стоимостные и временные затраты на реализацию традиционного тестирования современных вычислительных систем растут быстрее, чем сложность самих систем, для которых они используются.

В настоящее время широкое применение находят методы, позволяющие минимизировать как сложность реализации процедуры тестирования, так и время, необходимое для ее проведения. В особенности это является актуальным для большого многообразия различных решений по организации самотестирования современных систем. Значительное снижение аппаратурной сложности, а также упрощение процедуры реализации тестирования достигается при использовании методов *компактного тестирования* (*compact testing*) [65, 157, 164, 267, 269, 302, 304, 305]. Обобщенная структурная схема методов компактного тестирования приведена на рис. 10.1.

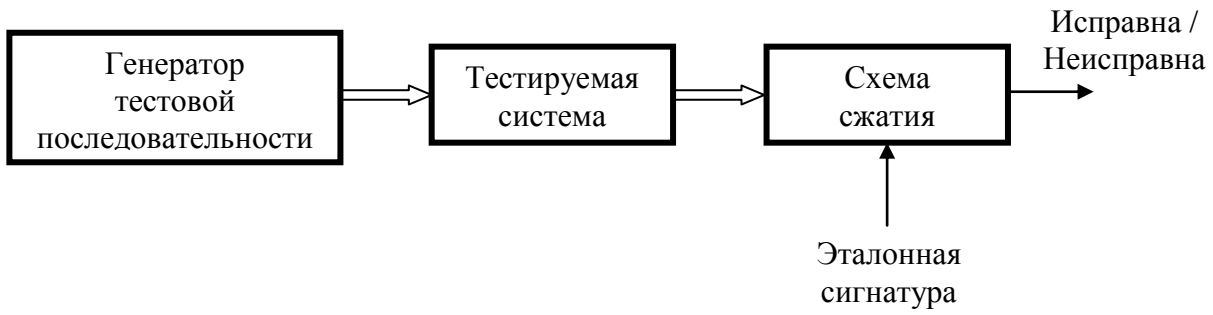


Рисунок 10.1 – *Структура реализации методов компактного тестирования*

Основным блоком реализации методов компактного тестирования системы является генератор тестовой последовательности, который должен предполагать компактную форму реализации, либо в виде алгоритма, либо в его аппаратном исполнении. Согласно приведенной схеме (рис. 10.1) на входы тестируемой системы подаются тестовые наборы, формируемые генератором тестовой последовательности. Выходные реакции системы на подаваемые тестовые воздействия сжимаются в короткие интегральные оценки, представляющие собой *компактные характеристики* сжимаемых последовательностей. Эти оценки называются *контрольными суммами* (*check sum*), *ключевыми словами* (*key words*), *синдромами* (*syndrome*) или *сигнатурами* (*signature*) [65, 157, 164, 267, 269, 302, 304, 305]. В дальнейшем будем использовать термин *сигнатура*, которая обозначает компактный результат тестирования системы, сформированный по одному из методов компактного тестирования (см. рис. 10.1). По результату сравнения эталонного значения сигнатуры с ее реальным значением принимается решение об исправности, либо неисправности проверяемой системы. Для этого на выходе схемы сравнения формируется выходной сигнал (Исправна / Неисправна). Термин *компактная характеристика* явился первопричиной появления общего названия для рассматриваемой методологии тестирования, а именно – *компактное тестирование*.

Под компактным *тестированием* понимают методы, основанные на получении компактных оценок результатов тестового эксперимента. Более широкая трактовка этого понятия подразумевает компактное представление не только результатов тестового эксперимента, но и сжатое представление входных тестовых воздействий. Так, для реализации генератора тестовой последовательности используются простейшие методы, предполагающие компактное представление таких генераторов в виде автономного устройства или алгоритма получения тестовых наборов. К таким методам можно отнести формирование следующих видов тестовых последовательностей [302, 304, 305]:

1. Детерминированных последовательностей.
2. Вероятностных тестовых последовательностей.
3. Псевдослучайных тестовых последовательной.
4. Квази-вероятностных тестовых последовательностей.
5. Исчерпывающих тестовых последовательностей.
6. Псевдо-исчерпывающих тестовых наборов.

Основным свойством приведенных алгоритмов формирования тестовых воздействий является очень большая длина воспроизводимых ими последовательностей. Поэтому очевидна необходимость использования алгоритмов сжатия выходных реакций тестируемых систем, полученных в результате формирования длинных тестовых последовательностей. Рассмотрим ряд классических методов сжатия, которые и используются для реализации компактного тестирования.

Метод счета единиц (*ones count compression*) основывается на использовании алгоритма сжатия двоичной последовательности $\{y(k)\} = y(0)\ y(1)\ y(2)\ \ldots\ y(k)\ \ldots$, состоящей из l последовательно формируемых символов согласно следующему выражению [157, 164, 170, 302, 304, 305]:

$$S = \sum_{k=0}^{l-1} y(k). \tag{10.1}$$

Сигнатура S, формируемая согласно выражению (10.1), может принимать значение в пределах от 0 до l в зависимости от количества единиц в последовательности $\{y(k)\}$. Переход от последовательности $\{y(k)\}$ к ее компактному эквиваленту S всегда однозначен, т. е. каждой двоичной последовательности соответствует единственное значение сигнатуры S. В то же время одну и ту же сигнатуру S можно получить на основании множества последовательностей $\{y(k)\}$. В случае метода счета единиц это множество включает последовательности, состоящие из одинакового количества единичных значений, равного S.

Метод счета переходов (*transition count testing*) очень близок по содержанию к методу счета единиц и считается одним из первых алгоритмов получения компактных оценок [94]. Данный метод описывается выражением:

$$S = \sum_{k=1}^{l-1} \left[y(k-1) \oplus y(k) \right], \tag{10.2}$$

где сигнатурой S является количество переходов в последовательности $\{y(k)\}$ двоичных символов. Так, например, для $y(0)\ y(1)\ y(2)\ y(3)\ y(4) = 1\ 0\ 0\ 1\ 1$ имеем $S = (1 \oplus 0) + (0 \oplus 0) + (0 \oplus 1) + (1 \oplus 1) = 2$. Модификациями метода счета переходов служат следующие отношения [94, 302, 304, 305]:

$$S_1 = \sum_{k=1}^{l-1} \left[\overline{y(k-1) \oplus y(k)} \right], \; S_2 = \sum_{k=1}^{l-1} \left[\overline{y(k-1)} y(k) \right], \; S_3 = \sum_{k=1}^{l-1} \left[y(k-1) \overline{y(k)} \right],$$

где S_1 представляет собой количество отсутствия переходов между последовательными символами в последовательности $\{y(k)\}$, S_2 и S_3, соответственно количество переходов из нуля в единицу и из единицы в ноль. Так, для рассмотренного выше примера: $S_1 = \overline{(1 \oplus 0)} + \overline{(0 \oplus 0)} + \overline{(0 \oplus 1)} + \overline{(1 \oplus 1)} = 2$, $S_2 = (\bar{1} \times 0) + (\bar{0} \times 0) + (\bar{0} \times 1) + (\bar{1} \times 1) = 1$, $S_3 = (1 \times \bar{0}) + (0 \times \bar{0}) + (0 \times \bar{1}) + (1 \times \bar{1}) = 1$. Для S, S_1, S_2, и S_3 выполняются следующие соотношения $S = l - S_1 - 1$, $S = S_2 + S_3$.

Так как максимальное количество переходов в последовательности $\{y(k)\}$, состоящей из l двоичных символов, равно $l - 1$, то сигнатуру для произвольного вида такой последовательности можно представить в виде позиционного кода, состоящего из $m = \lceil \log_2(l-1) \rceil$ бит.

При формировании вероятностных тестовых последовательностей и различных их модификаций в качестве компактной оценки S применяется вероятность $P(y(k) = 1)$ появления единицы в анализируемой последовательности [156, 158]:

$$S = P(y(k) = 1). \tag{10.3}$$

Использование псевдослучайных тестовых последовательностей, как правило, предполагает применение *сигнатурного анализа* (*signature analyses – SA*) как метода сжатия выходных реакций на тестовые воздействия [65]. Сигнатура $S = s_1(l)\,s_2(l)\,s_3(l)...s_m(l)$ для последовательности $\{y(k)\}$, состоящей из l двоичных символов, формируется по следующему алгоритму:

$$\begin{aligned} &s_1(0) = s_2(0) = s_3(0) = ... = s_m(0) = 0; \\ &s_1(k) = y(k-1) \oplus \sum_{i=1}^{m} \oplus\, \beta_i s(k-i); \\ &s_j(k) = s_{j-1}(k-1), \quad j = \overline{2,m}; \quad k = \overline{1,l}. \end{aligned} \tag{10.4}$$

Коэффициенты $\beta_i \in \{0,\ 1\}$, $i = \overline{1,m}$ определяются на основании примитивного порождающего полинома $\varphi(x) = 1 \oplus \beta_1 x^1 \oplus \beta_2 x^2 \oplus \ldots \oplus \beta_m x^m$, использованного для построения сигнатурного анализатора [65].

Значительное уменьшение объема информации, которую необходимо хранить для проведения процедуры тестирования, является главным достоинством рассмотренных методов сжатия. Однако отметим, что при сжатии различных последовательностей могут быть получены одинаковые значения их интегральных оценок, что при проведении тестового эксперимента приводит к снижению его достоверности [302, 304, 305]. Как отмечалось в ряде источников [65, 302, 304, 305], наибольшей достоверностью характеризуется сигнатурный анализ, который нашел широкое применение при тестировании современных вычислительных систем.

10.2. Сигнатурный анализ

Доминирующим методом сжатия информации при реализации компактного тестирования является сигнатурный анализ. Первый сигнатурный анализатор *HP*5004*A*, разработанный и примененный на практике фирмой *Hewlett-Packard*, использовал порождающий полином $\varphi(x) = 1 \oplus x^7 \oplus x^9 \oplus x^{12} \oplus x^{16}$ [65]. Основным назначением указанного анализатора, как и любого другого устройства подобного типа, является обнаружение ошибок в последовательности анализируемых данных, вызванных неисправностями тестируемого устройства либо системы. Высокая обнаруживающая способность схемы сжатия является весьма важной ее характеристикой.

При описании процедуры сжатия информации сигнатурным анализатором используются различные математические модели и алгоритмы [302,

304, 305]. Наиболее часто применяется модель, реализующая представление процедуры сжатия информации как операцию деления двоичных полиномов. В этом случае последовательность сжимаемых двоичных данных

$$\{y(k)\} = y(l-1)\, y(l-2)\, y(l-3)\, \ldots\, y(1)\, y(0),$$

где $y(k) \in \{0, 1\}$, $k = \overline{0, l-1}$, представляется в виде двоичного полинома

$$g(x) = y(l-1)x^{l-1} \oplus y(l-2)x^{l-2} \oplus y(l-3)x^{l-3} \oplus \ldots \oplus y(1)\, x^1 \oplus y(0)x^0. \qquad (10.5)$$

Например, последовательность двоичных данных 10011 можно представить в виде двоичного полинома:

$$g(x) = 1x^4 \oplus 0x^3 \oplus 0x^2 \oplus 1x^1 \oplus 1x^0 = x^4 \oplus x^1 \oplus x^0$$

Максимальный показатель степени переменной x полинома $g(x)$ называется его степенью. Так, для последнего примера имеем полином степени 4.

Для двоичного полинома (10.5) определены основные арифметические операции, выполняемые по модулю два [287].

В результате сложения двух полиномов $x^3 \oplus x^1 \oplus 1$ и $x^2 \oplus x^1 \oplus 1$ получим результирующий полином вида $x^3 \oplus x^2$.

Операция сложения по модулю два эквивалентна операции вычитания. Для данных операций выполняется следующее равенство:

$$x^r + x^r = x^r - x^r = 0 \bmod 2.$$

Умножение двоичных полиномов аналогично арифметическому умножению, отличием является лишь выполнение операции сложения при формировании результирующего произведения. Например, при умножении полиномов $x^3 \oplus x^1 \oplus 1$ и $x^2 \oplus x^1 \oplus 1$ получим:

$$\begin{array}{cccccc}
 & & x^3\oplus & & x^1\oplus & 1 \\
 & & & x^2\oplus & x^1\oplus & 1 \\
\hline
 & & x^3\oplus & & x^1\oplus & 1 \\
 & x^4\oplus & & x^2\oplus & x^1\oplus & \\
x^5\oplus & & x^3\oplus & x^2\oplus & & \\
\hline
x^5\oplus & x^4\oplus & & & & 1
\end{array}$$

При делении полинома $g(x)$ на полином $\varphi(x)$ имеем частное $q(x)$ и остаток $S(x)$ от операции деления, связанные между собой классическим разложением Евклида:

$$g(x) = q(x)\varphi(x) \oplus S(x).$$

В результате деления полинома $g(x) = x^7 \oplus x^6 \oplus x^5 \oplus x^4 \oplus x^2 \oplus 1$ на полином $\varphi(x) = x^3 \oplus x^2 \oplus 1$ получим:

$$
\begin{array}{llllll|l}
x^7 \oplus & x^6 \oplus & x^5 \oplus & x^4 \oplus & x^2 \oplus & 1 & x^3 \oplus\ x^2 \oplus\ 1 \\
\cline{7-7}
x^7 \oplus & x^6 \oplus & x^4 & & & & x^4 \oplus\ x^2 \oplus\ x^1 \oplus\ 1 \\
\cline{1-5}
 & & x^5 \oplus & x^2 \oplus & 1 & & \\
 & & x^5 \oplus & x^4 \oplus & x^2 & & \\
\cline{2-5}
 & & & x^4 \oplus & 1 & & \\
 & & & x^4 \oplus & x^3 \oplus & x^1 & \\
\cline{3-6}
 & & & & x^3 \oplus & x^1 \oplus\ 1 & \\
 & & & & x^3 \oplus & x^2 \quad 1 & \\
\cline{4-6}
 & & & & & x^2 \oplus\ x^1 &
\end{array}
$$

где $q(x) = x^4 \oplus x^2 \oplus x^1 \oplus 1$, а $S(x) = x^2 \oplus x^1$.

Представление двоичных данных в виде полинома используется только для удобства исследования свойств схем кодирования и декодирования, широко применяемых в теории циклических кодов [287]. Вместо полиномиального представления может использоваться первоначальное двоичное представление данных, для которого также выполняются вышеприведенные операции. Для предыдущего примера аналогичные преобразования при выполнении операции деления будут иметь вид:

$$
\begin{array}{cccccccc|c}
1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1\ 1\ 0\ 1 \\
\cline{9-9}
1 & 1 & 0 & 1 & & & & & 1\ 0\ 1\ 1\ 1 \\
\cline{1-4}
 & & 1 & 0 & 0 & 1 & 0 & 1 & \\
 & & 1 & 1 & 0 & 1 & & & \\
\cline{3-6}
 & & & 1 & 0 & 0 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
\cline{4-7}
 & & & & 1 & 0 & 1 & 1 & \\
 & & & & 1 & 1 & 0 & 1 & \\
\cline{5-8}
 & & & & & 1 & 1 & 0 &
\end{array}
$$

Здесь частное, равное двоичному коду 1 1 0, соответствует его полиномиальному представлению $S(x) = x^2 \oplus x^1$.

Процедура деления может быть описана процессом сжатия двоичных данных на регистре сдвига с внутренними сумматорами по модулю два.

Рассмотрим пример деления двоичных данных 1 1 1 1 0 1 0 1, соответствующих ранее использованному в качестве делимого полиному $g(x) = x^7 \oplus x^6 \oplus x^5 \oplus x^4 \oplus x^2 \oplus 1$, для сжатия которого применяется регистр сдвига, описываемый полиномом $\varphi(x) = x^3 \oplus x^2 \oplus 1$, соответствующим делителю (см. рис. 10.2).

Исходные состояния элементов памяти D_0, D_1 и D_2 регистра сдвига принимаются равными нулевому состоянию.

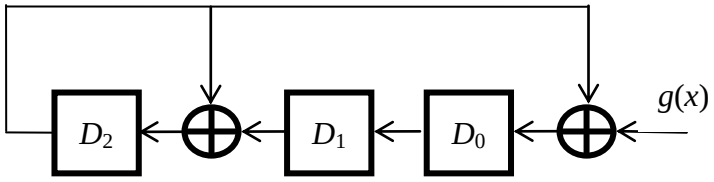


Рисунок 10.2 – *Структурная схема реализации сигнатурного анализатора*

Сжимаемые данные 1 1 1 1 0 1 0 1 последовательно подаются на вход регистра, в результате чего элементы памяти меняют свое состояние в соответствии с временной диаграммой, приведенной в таблице 10.1.

Таблица 10.1 – *Временная диаграмма функционирования сигнатурного анализатора*

					D_2	D_1	D_0	$g(x)$							
					0	0	0	1	1	1	1	0	1	0	1
					0	0	1	1	1	1	0	1	0	1	
					0	1	1	1	1	0	1	0	1		
					1	1	1	1	0	1	0	1			
				1	0	1	0	0	1	0	1				
			1	0	1	0	0	1	0	1					
		1	0	1	1	0	0	0	1						
	1	0	1	1	1	0	1	1							
1	0	1	1	1	1	1	0								
			$q(x)$			$S(x)$									

Остаток $S(x)$ от деления полинома $g(x) = x^7 \oplus x^6 \oplus x^5 \oplus x^4 \oplus x^2 \oplus 1$ на полином $\varphi(x) = x^3 \oplus x^2 \oplus 1$ фиксируется на элементах памяти регистра сдвига и принимает значение $S(x) = x^2 \oplus x^1$, в виде полинома, или $S(x)$ = 1 1 0 двоичного кода.

Полученные результаты можно сравнить с ранее рассмотренными примерами операции деления полиномов.

Схема сдвигового регистра, описываемого полиномом $\varphi(x) = x^3 \oplus x^2 \oplus 1$ и используемого для сжатия двоичной информации, называется *сигнатурным анализатором* (*signature analyzer*) [65]. Результатом сжатия является остаток от деления $S(x)$, называемый *сигнатурой* (*signature*).

Для практической реализации сигнатурного анализатора, описываемого полиномом $\varphi(x) = x^3 \oplus x^2 \oplus 1$, так же как, и для любого другого полинома, существует альтернативная структура (см. рис. 10.3), которая является более предпочтительной с точки зрения аппаратурного построения.

Результат сжатия $S^*(x)$ двоичных данных, полученный на сигнатурном анализаторе с внешними сумматорами по модулю два (рис.10.3), не совпадает с результатом $S(x)$ деления для делителя, описываемого полиномом $\varphi(x)$ =

$1 \oplus \beta_1 x^1 \oplus \beta_2 x^2 \oplus \beta_3 x^3 \ldots \oplus \beta_m x^m$, где $\beta_i \in \{0, 1\}$, $i = \overline{1, m}$. Однако между $S^*(x)$ и $S(x)$ существует однозначная связь, которая определяется выражением (10.6).

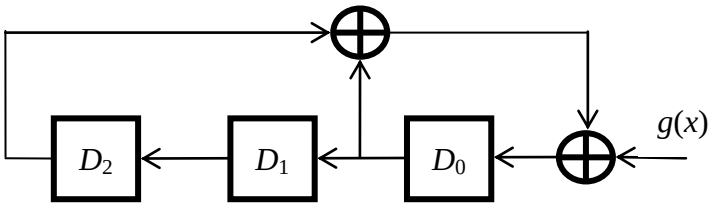


Рисунок 10.3 – *Структурная схема альтернативной реализации сигнатурного анализатора*

Приведенные примеры сигнатурных анализаторов и их математических моделей позволяют детально изучить процедуру формирования сигнатур и проследить взаимосвязь их теории синтеза и анализа с теорией циклических кодов и, в частности, теорией *M*-последовательностей. Одним из главных критериев, определивших широкое применение сигнатурного анализа, является его высокая достоверность, которая может быть оценена на основании ряда положений, вытекающих из свойств *M*-последовательностей:

$$S(x) = \begin{vmatrix} \beta_m & 0 & 0 & \ldots & 0 & 0 \\ \beta_{m-1} & \beta_m & 0 & \ldots & 0 & 0 \\ \beta_{m-2} & \beta_{m-1} & \beta_m & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \beta_2 & \beta_3 & \beta_4 & \ldots & \beta_m & 0 \\ \beta_1 & \beta_2 & \beta_3 & \ldots & \beta_{m-1} & \beta_m \end{vmatrix} \times S^*(x) \qquad (10.6)$$

Рассмотрим несколько подходов, используемых для оценки эффективности сигнатурного анализа.

10.3. Достоверность сигнатурного анализа

Прежде чем перейти к изучению достоверности сигнатурного анализа, отметим, что в работах, посвященных применению сигнатурного анализа, не принято показывать, что полнота обнаружения неисправностей вычислительных систем в первую очередь зависит от качества тестовых воздействий. Если определенная неисправность не проявляется в выходных последовательностях тестируемой системы в виде искажения их символов, то она не может быть обнаружена в результате применения сигнатурного анализа, который является не более чем эффективным методом сжатия последовательностей данных. Поэтому если выходная последовательность не несет информации о неисправности, то она и не появится после ее сжатия.

Таким образом, под достоверностью сигнатурного анализа будем понимать его эффективность обнаружения ошибки в последовательности сжимаемых данных. Для оценки этой характеристики сигнатурного анализа могут использоваться различные подходы и методы [278, 285, 302, 304, 305]. Наиболее широко применяемым является вероятностный подход, сущность которого заключается в определении вероятности P_n необнаружения ошибок в анализируемой последовательности данных [65]. Причем в рассматриваемом случае оценивается вероятность, зависящая только от метода сжатия, и не учитываются другие факторы.

Величина P_n рассчитывается для достаточно общего случая, приближенно соответствующего реальным примерам. Предполагается, что эталонная последовательность данных может равновероятно принимать разное значение, а любая конфигурация ошибочных бит может быть равновероятным событием. Далее, используя алгоритм деления полиномов как математический аппарат формирования сигнатуры, показывается, что для l-разрядного делимого вычисляются $(l-m)$-разрядное частное и m-разрядный остаток (сигнатура). При этом соответствие реальной последовательности, состоящей из l бит, ее эталонному виду, оценивается только по равенству их m-разрядных сигнатур. Для 2^{l-m} различных частных будет формироваться одинаковая сигнатура. Это свидетельствует о том, что $2^{l-m} - 1$ ошибочных l-разрядных последовательностей будут считаться соответствующими одной, а именно эталонной. Учитывая равную вероятность ошибочных последовательностей данных, можно заключить, что $2^{l-m} - 1$ ошибочных последовательностей, инициирующих эталонную сигнатуру, будут необнаруживаемыми. Таким образом, вероятность P_n необнаружения ошибок в анализируемой последовательности данных будет вычисляться как отношение:

$$P_n = \frac{2^{l-m} - 1}{2^l - 1}, \tag{10.7}$$

где величина $2^l - 1$ равняется общему числу ошибочных последовательностей.

Выражение (10.7) при $l >> m$ преобразуется к более простому виду:

$$P_n \approx \frac{1}{2^m}, \tag{10.8}$$

которое может служить основным аргументом для обоснования высокой эффективности сигнатурного анализа. Действительно, для случая сигнатурного анализатора фирмы *Hewlett-Packard*, для которого $m = 16$ имеем $P_n = 0{,}0000152...$, что свидетельствует о достаточно высокой эффективности сигнатурного анализа.

Величина $P_d = 1 - P_n = 0{,}999984...$ приводится практически во всех работах, посвященных сигнатурному анализу, и только в некоторых дается ре-

альная оценка данному соотношению, которое практически не позволяет оценить достоинства сигнатурного анализа. Простейший пример, показывающий несостоятельность применения интегральной оценки (10.8) в качестве основного критерия эффективности сигнатурного анализа по сравнению с другими методами компактного тестирования, дан в работе [179]. Сущность его заключается в том, что при использовании метода, основанного на анализе только m бит из l путем отбрасывания $l - m$ бит исследуемой последовательности, и учете допущения, принятого при выводе соотношения (10.8), вероятность P_n и в данном случае будет вычисляться по этой же формуле. Аналогичный пример для частного случая линейных комбинационных схем приведен в работе [278].

Наряду с интегральной характеристикой P_n в ряде случаев показывается, что еще одним достоинством сигнатурного анализа является его способность обнаруживать все одиночные ошибки для любого l и все двойные ошибки для $l < 2^m - 1$ [65, 179]. Уточненные значения P_n, приведенные в работе [285], не позволяют получить сравнительную характеристику сигнатурного анализа по отношению к другим методам компактного тестирования.

В качестве более точной меры оценки эффективности сигнатурного анализа рассматривается распределение вероятности необнаружения ошибки в зависимости от ее кратности μ, т. е. определяются значения P_n^{μ}, где $\mu = \overline{1, 2^m - 1}$ [302, 304, 305]. При этом используются основные положения теории генерирования M-последовательностей.

На основании линейности операции суммирования по модулю два полином $\chi^*(x)$, описывающий последовательность, содержащую ошибки, может быть представлен в виде суммы

$$\chi^*(x) = \chi(x) \oplus e(x), \qquad (10.9)$$

где $\chi(x)$ полином, описывающий эталонную последовательность, а $e(x)$ является полином ошибки. Причем очевидно, что ошибка будет необнаруживаемой только в случае делимости полинома $e(x)$ на полином $\varphi(x)$, описывающий структуру сигнатурного анализатора [301].

В случае возникновения одиночной ошибки в l-й позиции двоичной последовательности сжимаемых данных полином ошибки будет иметь вид: $e(x) = x^{l-1}$, где $1 \le l \le 2^m - 1$, $m = \deg\varphi(x)$, а $\varphi(x)$ является примитивным порождающим полиномом. Используя свойство M-последовательностей, заключающееся в формировании сдвинутой на l тактов копии на основании m ненулевых коэффициентов, можно показать, что каждому l соответствует отличное значение остатка от деления x^{l-1} на $\varphi(x)$. Другими словами, любой одиночной ошибке, описываемой полиномом $e(x) = x^{l-1}$, будет соответствовать ненулевая, отличная от других сигнатура $S(x)$. Таким образом, сигнатурный анализатор, построенный на базе примитивного порождающего полинома $\varphi(x)$, обнаруживает все одиночные ошибки [179, 285, 301].

При возникновении ошибок в l-й и r-й позициях двоичной последовательности данных полином ошибки будет иметь вид: $e(x) = x^{l-1} \oplus x^{r-1}$. Для примитивного порождающего полинома $\varphi(x)$ можно показать что любой паре целых чисел l и r где $l > r\ (1 \le l \le L, 1 \le l \le L), L = 2^m - 1$ соответствует ненулевая сигнатура, полученная как сумма над полем $GF(2)$ двух отличных ненулевых сигнатур для $e(x) = x^{l-1}$ и $e(x) = x^{r-1}$. В данном случае использовалось свойство линейности операции суммирования над полем $GF(2)$.

Следовательно, при допущении, что длина анализируемой последовательности данных не превышает величины $2^m - 1$ сигнатурный анализатор, описываемый полиномом $\varphi(x)$, обнаруживает все двойные ошибки. Это соответствует ранее полученным результатам [65].

Множеству L ненулевых сигнатур, вычисленных для полиномов ошибок вида $e(x) = x^{l-1} \oplus x^{r-1}$ соответствует множество полиномов $e(x) = x^{v-1}$, порождающих идентичные сигнатуры. Поэтому если сигнатуры для полиномов $e(x) = x^{l-1} \oplus x^{r-1}$ и $e(x) = x^{v-1}$ одинаковы, то сигнатура полученная для полинома $e(x) = x^{l-1} \oplus x^{r-1} \oplus x^{v-1}$ будет равна нулю, что не приведет к изменению результирующей сигнатуры для анализируемой последовательности содержащей тройную ошибку. Другими словами, конфигурации тройных ошибок, определяемых значениями l, r и v являются необнаруживаемыми. В терминах теории M-последовательностей величины l, r и v вычисляются согласно свойству сдвига и сложения (см. главу 7). Используя это свойство можно показать, что для необнаруживаемых тройных ошибок, описываемых полиномом $e(x) = x^{l-1} \oplus x^{r-1} \oplus x^{v-1}$, с помощью сигнатурного анализатора с порождающим полиномом $\varphi(x)$ для символов $a(k+l-1)$, $a(k+r-1)$ и $a(k+v-1)$ M-последовательности, выполняется тождество $a(k+l-1) \oplus a(k+r-1) = a(k+v-1)$. Отсюда следует, что число необнаруживаемых тройных ошибок будет определяться количеством тождеств состоящих из трех символов и выполняемых для M-последовательности. При $L = 2^m - 1$ это число равно [324]

$$V_n^3 = C_L^2 / 3 = \frac{(2^m - 1)(2^m - 2)}{3!}, \qquad (10.10)$$

где C_L^2-число сочетаний из $L = 2^m - 1$ по два, а наличие делителя 3 объясняется тем, что трем полиномам $x^{l-1} \oplus x^{r-1}$, $x^{l-1} \oplus x^{v-1}$ и $x^{r-1} \oplus x^{v-1}$, описывающим обнаруживаемые двойные ошибки, соответствует один полином $x^{l-1} \oplus x^{r-1} \oplus x^{v-1}$ описывающий необнаруживаемую ошибку, состоящую из трех неверных бит.

Общее число возможных конфигураций тройных ошибок составляет $V^3 = C_L^3$, а вероятность их необнаружения определяется отношением количества необнаруживаемых ошибок к общему их числу:

$$P_L^3 = \frac{1}{2^m - 3}.$$

Количество обнаруживаемых тройных ошибок определяется как разность их общего числа и количества необнаруживаемых ошибок.

$$V_d^3 = V^3 - V_n^3 = C_L^2 / 3 = \frac{(2^m - 1)(2^m - 2)(2^m - 4)}{3!}, \qquad (10.11)$$

Для обнаруживаемых тройных ошибок, описываемых полиномом $e(x) = x^{l-1} \oplus x^{r-1} \oplus x^{v-1}$, можно найти такой полином $e(x)=x^{\omega-1}$, который будет инициировать идентичную сигнатуру. Таким образом, полином $e(x) = x^{l-1} \oplus x^{r-1} \oplus x^{v-1} \oplus x^{\omega-1}$ описывает необнаруживаемую конфигурацию из четырех неверных бит. Значение ω получается на основании свойства сдвига и сложения и является функцией от переменных l, r и v, т. е. $\omega = f(l, r, v)$. Для $\omega \neq f(l, r, v)$ полином $e(x) = x^{l-1} \oplus x^{r-1} \oplus x^{v-1} \oplus x^{\omega-1}$ описывает обнаруживаемую ошибку из четырех неверных бит. Общее количество необнаруживаемых ошибок из четырех бит с учетом (10.11) определяется как:

$$V_n^4 = V_d^3 / 4 = \frac{(2^m - 1)(2^m - 2)(2^m - 4)}{4!}. \qquad (10.12)$$

Наличие делителя 4 в выражении (10.12), аналогично как и 3 в (10.11), объясняется тем, что четырем полиномам $e(x) = x^{l-1} \oplus x^{r-1} \oplus x^{v-1}$, $e(x) = x^{l-1} \oplus x^{r-1} \oplus x^{\omega-1}$, $e(x) = x^{l-1} \oplus x^{v-1} \oplus x^{\omega-1}$ и $e(x) = x^{r-1} \oplus x^{v-1} \oplus x^{\omega-1}$ удовлетворяющим обнаруживаемым ошибкам, соответствует один полином $e(x) = x^{l-1} \oplus x^{r-1} \oplus x^{v-1} \oplus x^{\omega-1}$, удовлетворяющий необнаруживаемой ошибке.

Окончательно вероятность необнаружения ошибок из четырех бит равна

$$P_n^4 = \frac{V_n^4}{V^4} = \frac{(2^m - 1)(2^m - 2)(2^m - 4)}{4! C_L^4} = \frac{1}{2^m - 3}.$$

Используя подобную методику, определяем количество необнаруживаемых ошибок из пяти бит:

$$V_n^5 = (V^4 - V_n^4 - V_n^3(2^m - 4)) / 5.$$

Появление члена $V_n^3(2^m - 4)$ свидетельствует о том, что любая ошибка из пяти бит, включающая необнаруживаемую ошибку из трех бит, всегда будет обнаруживаемой. Здесь $2^m - 4 = C_{L-3}^1$.

Число необнаруживаемых ошибок из шести неверных бит равно:

$$V_n^6 = (V^5 - V_n^5 - V_n^4(2^m - 5)) / 6.$$

Для общего случая можно показать, что количество необнаруживаемых ошибок, состоящих из μ бит, $\mu \in \{1, 2, 3, ..., 2^m - 1\}$, будет определяться выражением

$$V_n^1 = 0, \quad V_n^2 = 0,$$
$$V_n^\mu = \left[V^{\mu-1} - V_n^{\mu-1} - V_n^{\mu-2}(2^m + 1 - \mu)\right] / \mu. \tag{10.13}$$

Значения количества необнаруживаемых ошибок в зависимости от их кратности, определяемые согласно (10.13), позволяют получать распределение вероятностей необнаружения ошибок любой кратности. Выражение для вероятности необнаружения ошибки в данном случае имеет вид [301]:

$$P_n^1 = 0, \quad P_n^2 = 0,$$
$$P_n^\mu = \frac{V^{\mu-1} - V_n^{\mu-1} - V_n^{\mu-2}(2^m + 1 - \mu)}{\mu V^\mu}, \quad \mu = \overline{3, L}, \tag{10.14}$$

где $V^\mu = C_L^\mu$.

Преобразовав соотношение (10.14), окончательно получим:

$$P_n^1 = P_n^2 = 0,$$
$$P_n^\mu = \frac{1}{2^m - \mu}\left[1 - P_n^{\mu-1} - P_n^{\mu-2}(\mu - 1)\right], \quad \mu = \overline{3, 2^m - 1}. \tag{10.15}$$

Анализ полученных значений P_n^μ, а также соотношения (10.15) показывают, что для достаточно больших величин m $P_n^\mu \approx 1/2^m$, $\mu = \overline{3, 2^m - 2}$. Таким образом, ошибки кратности $\mu \in \{3, 4, 5, ..., 2^m - 2\}$, возникающие в последовательности данных длиной $2^m - 2$, обнаруживаются сигнатурным анализатором с вероятностью $P_d^\mu = 1 - P_n^\mu$, которая при $m > 7$ практически равна единице.

Основным следствием соотношения (10.15) является его инвариантность относительно сжимаемой последовательности и вида примитивного полинома $\varphi(x)$. Единственный параметр, влияющий на значения P_d^μ, является старшая степень m примитивного порождающего полинома $\varphi(x)$.

10.4. Структурный синтез сигнатурных анализаторов

При применении сигнатурного анализа необходима оценка его обнаруживающей способности относительно определенных типов неисправностей исследуемой вычислительной системы. Результатом решения данной задачи является ответ, показывающий количество и конфигурацию неисправностей, которые обнаруживаются конкретной структурой анализатора. При проведении подобных исследований предполагается, что неисправность заданного типа обязательно приводит к искажению выходных реакций системы.

Обычно процедура выбора наиболее эффективного сигнатурного анализатора заключается в применении метода проб и ошибок. В результате определяется анализатор, обнаруживающий максимальное число неисправностей вычислительной системы. Достаточно актуальной является разработка аналитических методов синтеза сигнатурных анализаторов. Исходным пунктом для решения этой задачи является определение разрядности m сигнатурного анализатора, позволяющего обнаруживать с требуемой вероятностью неисправности конкретной цифровой схемы.

Первоначально исследуется зависимость эффективности сигнатурного анализа от количества обнаруживаемых им неисправностей Q цифровой схемы и разрядности $m = \deg \varphi(x)$ сигнатурного анализатора, где полином $\varphi(x)$ используется в качестве порождающего полинома анализатора. В качестве меры эффективности сигнатурного анализа, как правило, используется вероятность P_d обнаружения неисправности схемы сигнатурным анализатором. Вероятность P_d на основании (10.8) определяется выражением:

$$P_d = 1 - \frac{1}{2^m}. \qquad (10.16)$$

Если предположить, что в исследуемой схеме возможно возникновение Q неисправностей, и все они являются независимыми, то факт их обнаружения характеризуется вероятностью $P_d(Q)$, которую можно определить из выражения:

$$P_d(Q) = \left(1 - \frac{1}{2^m}\right)^Q. \qquad (10.17)$$

Зависимость вероятности $P_d(Q)$ от величины Q и различных значений m рассмотрена в [304, 305], где количество неисправностей Q заменено на их относительную величину $q = Q/2^m$. В результате приведенного анализа показано, что с ростом m для всех q значения $P_d(q)$ стремятся к предельным значениям для случая $m = \infty$. Определим вид зависимости $P_d(q)$ для $m = \infty$. Для этого вычислим предел:

$$\lim_{m\to\infty}\left(1 - \frac{1}{2^m}\right)^{q2^m}.$$

Учитывая равенство:

$$\lim_{m\to\infty}\left(1 - \frac{1}{2^m}\right)^{q2^m} = \lim_{2^m\to\infty}\left(1 - \frac{1}{2^m}\right)^{q2^m}.$$

Произведя замену $2^m = x$, получим:

$$\lim_{x\to\infty}\left(1-\frac{1}{x}\right)^{qx}=e^{-q}.$$

Таким образом, окончательно для (10.17) имеем:

$$\lim_{m\to\infty}P_d(Q)=\lim_{m\to\infty}\left(1-\frac{1}{2^m}\right)^{q2^m}=e^{-q}. \qquad (10.18)$$

Используя полученную для $P_d(q)$ оценку, можно записать общее выражение для произвольного значения m. Для этого заменим правую часть выражения (10.18) на e^{-q}, где $q = Q/2^m$ получим

$$P_d(Q)=e^{-Q/2^m}. \qquad (10.19)$$

Соотношение (10.19) определяет эффективность $P_d(Q)$ применения сигнатурного анализа для обнаружения Q неисправностей вычислительной системы m-разрядным сигнатурным анализатором. Поскольку в качестве исходных данных используется количество Q неисправностей системы и заданная вероятность $P_d(Q)$ их обнаружения, то на основании этих данных в соответствии с (10.19) можно определить разрядность m сигнатурного анализатора, обеспечивающего требуемую вероятность. Для этого прологарифмируем обе части равенства (10.19). В результате имеем выражение:

$$\ln P_d(Q)=-\frac{Q}{2^m}.$$

Проведя преобразования и вновь прологарифмировав, а также принимая во внимание тот факт, что m может быть только целым, окончательно получим:

$$m=\lceil \log_2 Q-\log_2\ln(1/(P_d(Q))\rceil. \qquad (10.20)$$

Последнее равенство является расчетным соотношением, позволяющим вычислить основной параметр сигнатурного анализатора, а именно, его разрядность m, которая определяет старшую степень примитивного порождающего полинома. Более грубая оценка для вероятности $P_d(Q)$ может быть получена иным способом [304, 305]. На основании (10.17) можно получить соотношения, позволяющие определить вероятность обнаружения заданного количества неисправности Q. Рассматривая эти зависимости для Q = 1 и 2, получим, что $P_d(1) = (1 - 1/2^m)^1$, а $P_d(2) = (1 - 1/2^m)^2 = 1 - 2/2^m + 1/2^{2m}$. В последнем соотношении для достаточно больших значений m можно пренебречь слагаемым $1/2^{2m}$. Тогда $P_d(2) = 1 - 2/2^m$. Рассуждая аналогично, для Q = 3 получим: $P_d(3) = 1 - 3/2^m$.

Таким образом, вероятность $P_d(Q)$ обнаружения Q неисправностей m-разрядным сигнатурным анализатором в общем случае равна:

$$P_d(Q) = 1 - \frac{Q}{2^m}. \qquad (10.21)$$

Из выражения (10.21) получим расчетное соотношение для величины m:

$$m = \lceil \log_2 Q - \log_2(1 - P_d(Q)) \rceil. \qquad (10.22)$$

Сопоставив соотношения (10.20) и (10.22), можно заметить, что они отличаются лишь аргументами во втором слагаемом. Причем при значениях требуемой вероятности $P_d(Q)$, близких к единице, расчеты по формулам (10.20) и (10.22) дают практически одинаковый результат. Точность вычислений при этом зависит от того, насколько точно определен входящий в эти соотношения параметр Q – количество независимых неисправностей вычислительной системы. Определение этого параметра – достаточно сложная задача, но в большинстве случаев необходима только его верхняя оценка, которая для случая одиночных константных неисправностей составляет $Q = 2N$, где N – количество полюсов исследуемой схемы.

Рассмотрим пример цифровой схемы, в которой возможны только одиночные константные неисправности. Их общее число $Q = 2^{10}$. Для данной схемы необходимо синтезировать сигнатурный анализатор, который должен обнаруживать неисправное состояние схемы с вероятностью $P_d = 0{,}9922$. Другими словами, обнаруживать все $Q = 2^{10}$ его возможные неисправности.

С помощью формул (10.20), (10.22) получим: $m = 10 + 7 = 17$. Следовательно применяя примитивный полином $\varphi(x)$ степени $m = 17$ для построения сигнатурного анализатора, можно обнаружить $Q = 2^{10}$ неисправностей с вероятностью $P_d = 0{,}9922$.

10.5. Адаптивный сигнатурный анализ

В работах [211, 328] был предложен новый метод получения компактных оценок выходных последовательностей данных, получивший название *адаптивный сигнатурный анализ* (АСА). Данный метод устраняет главный недостаток классического сигнатурного анализа, который заключается в том, что любое изменение в последовательности сжатой на сигнатурном анализаторе предопределяет повторное ее сжатие для получения актуальной сигнатуры. Предложенный в работах [211, 328] адаптивный метод вычисления сигнатур позволяет корректировать текущее значение сигнатуры на реальное значение с учетом изменений в сжимаемой последовательности, не прибегая к ее полному повторному сжатию.

Основная идея предложенного метода адаптивного сигнатурного анализа заключается в использовании взаимообратных примитивных полиномов

для генератора тестовых последовательностей (ГТП), который, по сути, определяет порядок сжимаемых бит последовательности и сигнатурного анализатора (СА). Данный метод сжатия изначально был ориентирован на запоминающие устройства и определен при следующих ограничениях:

1. Тестируемое запоминающее устройство представляет собой бит-ориентированное ЗУ, содержащее $2^m - 1$ ячеек, адресуемых при помощи m-разрядных ненулевых адресов.

2. ГТП генерирует адреса ЗУ и управляющие сигналы для чтения данных. Функционирование ГТП описывается примитивным порождающим полиномом $\varphi(x)$ старшей степени m. Сигнатурный анализатор описывается примитивным взаимообратным порождающим полиномом $\varphi^{-1}(x)$ старшей степени m, т. е. $\deg\varphi(x) = \deg\varphi^{-1}(x) = m$.

Схема тестирования ЗУ с учетом приведенных ограничений имеет следующий вид:

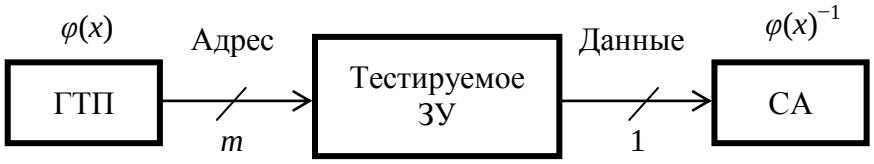


Рисунок 10.4 – *Схема тестирования ЗУ*

Для схемы тестирования ЗУ, приведенной на рис. 10.4 справедлива следующая теорема [211, 304, 305, 328].

Теорема 10.1. Сигнатура S бит-ориентированного ЗУ с $2^m - 1$ ячейками, которые содержат значение единицы только в одной ячейке с адресом $A = a_1 a_2 a_3 ... a_m$, $a_i \in \{0,1\}$; $i = 1, 2, 3, ..., m$, равна $S = s_1 s_2 s_3 ... s_m = a_m a_{m-1} a_{m-2} ... a_1$ для случая, когда:

1. ГТП и СА описываются примитивными порождающими полиномами $\varphi(x)$ и $\varphi^{-1}(x)$;

2. $\deg\varphi(x) = \deg\varphi^{-1}(x) = m$;

3. Начальные состояния ГТП и СА равны 1 0 0 … 0 и 0 0 0 … 0 соответственно;

4. Число псевдослучайных тестовых наборов, подаваемых на тестируемое ЗУ, равно $2^m - 1$.

Следует отметить, что $S = a_m a_{m-1} a_{m-2} ... a_1 \neq 000...0$.

Пример 10.1. ЗУ содержит $2^3 - 1 = 7$ ячеек. Временная диаграмма состояний ГТП и СА в процессе вычисления сигнатуры ЗУ приведена на рисунке 10.5.

Все ячейки содержат значение 0, исключение составляет ячейка с адресом $A = a_1\ a_2\ a_3 = 011$, значение которой равно единице. ГТП описывается

при помощи примитивного порождающего полинома $\varphi(x) = 1 + x + x^3$, СА – при помощи взаимообратного примитивного полинома $\varphi^{-1}(x) = 1 + x^2 + x^3$. Начальные состояния для ГТП и СА равны соответственно 1 0 0 и 0 0 0.

Для приведенного примера сигнатура, согласно теореме 10.1, равна $S = s_1\, s_2\, s_3 = a_3\, a_2\, a_1 = 1\ 1\ 0$, что соответствует результату, полученному на рисунке 10.5. Для общего случая справедлива следующая теорема [211, 328, 304, 305].

Теорема 10.2. Сигнатура S бит-ориентированного ЗУ с произвольным содержимым равна $S = s_1\, s_2\, s_3 \ldots s_m$, где $s_{m+1-i} = x_{1,i} \oplus x_{2,i} \oplus x_{3,i} \oplus \ldots \oplus x_{r,i}$; $x_{ij} \in \{0,1\}$, $i = 1,2,3,\ldots,m$, $j = 1,2,3,\ldots,r$ и $x_{j,1}$, $x_{j,2}$, $x_{j,3}$, …, $x_{j,m}$ есть адреса ячеек ЗУ, содержащих 1, и r есть число единиц для случая, когда:

1. ГТП и СА описываются примитивными порождающими полиномами $\varphi(x)$ и $\varphi^{-1}(x)$;

2. $\deg \varphi(x) = \deg \varphi^{-1}(x) = m$;

3. Начальные состояния ГТП и СА равны 1 0 0 … 0 и 0 0 0 … 0 соответственно;

4. Число псевдослучайных тестовых наборов, подаваемых на тестируемое ЗУ, равно $2^m - 1$.

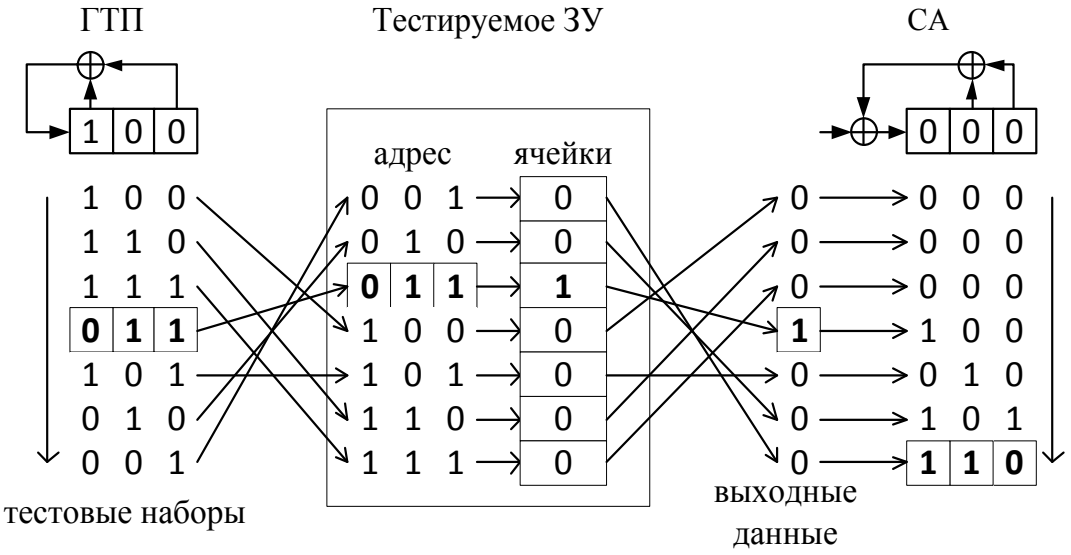


Рисунок 10.5 – *Временная диаграмма состояний ГТП и СА* (*пример* 10.1)

Пример 10.2. Рассмотрим предыдущий пример 10.1 для случая, когда только ячейки ЗУ с адресами A_2 = 0 1 0, A_3 = 0 1 1 и A_4 = 1 0 0 содержат значение 1. Временная диаграмма состояний ГТП и СА, выходные данные ЗУ и результирующая сигнатура показаны на рисунке 10.6.

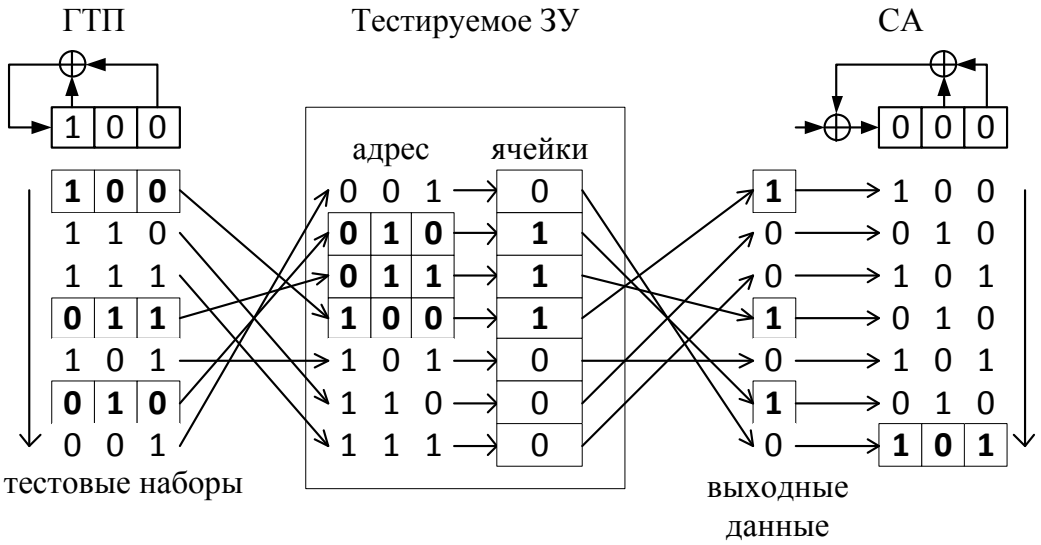


Рисунок 10.6 – *Временная диаграмма состояний ГТП и СА* (*пример* 10.2)

Эталонная сигнатура S ЗУ для рассмотренного примера равна 1 0 1. Согласно теореме 10.2 для вычисления сигнатуры S необходимо сложить по модулю 2 значения адресов тех ячеек, которые хранят 1. Для примера 10.2 получим $S = 0\,1\,0 \oplus 0\,1\,1 \oplus 1\,0\,0 = 1\,0\,1$.

Исходя из рассмотренных теорем 10.1 и 10.2, справедливо следующее утверждение.

Утверждение 10.1. Для случая бит-ориентированного ЗУ эталонная сигнатура может быть вычислена как побитная сумма по модулю 2 значений адресов всех ячеек, содержащих 1.

Аппаратурные затраты на применение адаптивного сигнатурного анализатора составляют m триггеров D-типа и m двухвходовых элементов сложения по модулю два (XOR).

Рассмотрим главную особенность адаптивного сигнатурного анализа, основываясь на предыдущих примерах 10.1 и 10.2. Возьмем для рассмотрения содержимое ЗУ из примера 10.1 в качестве начального состояния, а из примера 10.2 содержимое того же ЗУ после изменения содержимого, то в этом случае можно применить теорию АСА для коррекции эталонной сигнатуры. Пусть $S^{old} = 0\,1\,1$ есть значение сигнатуры ЗУ до модификации содержимого ЗУ, а $S^{new} = 0\,1\,0 \oplus 0\,1\,1 \oplus 1\,0\,1 = 1\,0\,1$ – значение сигнатуры ЗУ после модификации содержимого. Два варианта содержимого ЗУ отличаются лишь значениями по адресам 0 1 0 и 1 0 0. Тогда новое значение сигнатуры S^{new} может быть вычислено исходя из значения сигнатуры S^{old} на основании следующего утверждения.

Утверждение 10.2. Для коррекции значения эталонной сигнатуры при модификации содержимого ЗУ необходимо сложить по модулю 2 предыдущее значение эталонной сигнатуры и адреса ячеек ЗУ, содержимое которых изменились (модифицировалось) на противоположное значение.

В процедуре коррекции участвуют только адреса ячеек, значение содержимого которых было изменено на обратное значение из 0 в 1 либо, наоборот, из 1 в 0.

Таким образом, как отмечалось в предыдущих разделах, в работе [211] был предложен новый метод получения компактных оценок бинарных последовательностей данных, получивший название *адаптивный сигнатурный анализ* или *адаптивное сжатие выходных данных*. Данный метод, в сравнении с классическими методами вычисления сигнатур позволяет реализовывать модификацию предыдущего значения сигнатуры, не прибегая к ее полному пересчету. Основная идея предложенного метода адаптивного сигнатурного анализа заключается в использовании ряда принципиальных ограничений на размерность, как сжимаемой последовательности, так и на разрядность ее компактной оценки – сигнатуры [211].

Дальнейшее развитие теории адаптивного сигнатурного анализа нашло отражение в работах [331, 332, 339, 345], где рассмотрен *обобщенный адаптивный сигнатурный анализ*, который, в отличие от адаптивного сигнатурного анализа, не накладывает ограничения как на сжимаемые данные, так и на разрядность сигнатуры. Вопросы практического применения сигнатурного анализа на практике в части его достоверности, применения многоканальных анализирующих структур, а также применения в качестве средств самотестирования, подробно изложены в работах [147, 208, 211, 212, 283, 284, 288, 307, 309, 310].

Глава 11. Неразрушающее тестирование

11.1. Кольцевое тестирование

В традиционных подходах тестирования с ростом сложности вычислительных систем резко возрастает сложность тестового оборудования, например, за счет увеличения емкости памяти запоминающих устройств, используемых для хранения тестовых данных. Существенное уменьшение объема тестовых данных достигается за счет применения методов компактного тестирования, представленного в предыдущей главе. Подобные методы используются для представления информации в сжатом виде. Наиболее изученным классом компактных методов тестирования являются разомкнутые системы, в которых генератор тестовой последовательности, тестируемая система и схема сжатия (анализатор ответов) соединены последовательно, как это показано на рис. 10.1.

Подобные методы характеризуются рядом существенных достоинств и являются востребованными для построения схем самотестирования современных вычислительных систем и, в особенности, встроенных систем и систем на кристалле. Однако следует отметить, что и в этом случае необходимо заметное дополнительное оборудование для хранения как тестовых данных, включающих эталонные сигнатуры и начальные состояния генераторов тестов, так и текущего состояния тестируемой системы. Сохранение данных о текущем состоянии системы необходимо для ее дальнейшего использования по назначению. Схема компактного тестирования, приведенная на рис. 10.1, так же как и классические схемы тестирования, характеризуется разрушающим свойством по отношению к исходному состоянию системы. В процессе тестирования большинство исходной информации, например, состояния внутренних регистров системы разрушается, что требует выполнения процедуры инициализации вычислительной системы для ее последующего применения по назначению.

Дальнейшее снижение аппаратурной сложности достигается в классе замкнутых схем компактного тестирования (см. рис. 11.1), которые позволяют реализовать сохранение исходных данных системы, что эквивалентно *неразрушающему* (*transparent*) принципу тестирования.

В приведенной схеме замкнутого компактного тестирования функцию схемы сжатия реализует генератор тестовой последовательности. Его финальное состояние интерпретируется как результирующее значение компактной оценки (сигнатуры), а соответствие или несоответствие эталонному значению формирует результат тестирования вычислительной системы (Исправна / Неисправна).

Замкнутость компактных систем тестирования в значительной мере способствует разрешению противоречия, обусловленного отставанием временных характеристик классических средств тестирования от характеристик современных вычислительных систем. Поскольку в процессе функциониро-

вания встроенных средств таких систем отсутствуют обращения к запоминающим устройствам и периодическое сравнение фактических сигнатур с эталонными значениями, то возможно проведение процедуры тестирования на рабочей частоте системы. Это обстоятельство также является несомненным достоинством замкнутых схем компактного тестирования.

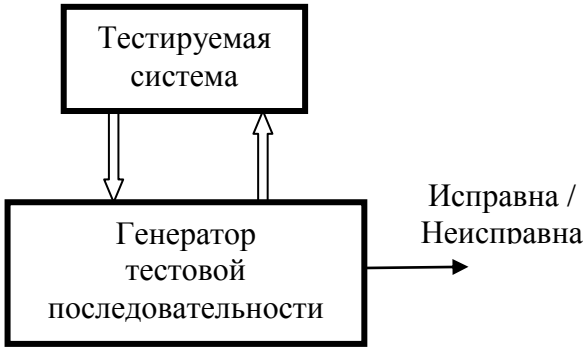


Рисунок 11.1 – *Структурная схема замкнутого компактного тестирования*

С развитием замкнутых схем тестирования связано появление схемы *кольцевого тестирования*, впервые рассмотренного в работах [59, 277, 279]. В кольцевых системах функции генератора и анализатора совмещаются в пространстве и во времени. Топология структуры кольцевого тестирования имеет форму кольца, а модели систем описываются в алгебре кольца многочленов и кольцевыми (циклическими) графами, что породило термин кольцевое тестирование [277, 279]. В процессе проверки исправная вычислительная система проходит свои состояния по циклу, возвращаясь в начальное состояние. Поэтому заключение об исправности системы делается на основании сравнения ее начального и конечного состояний.

Аппаратурная избыточность кольцевых схем тестирования зависит от свойств линейности и нелинейности объекта тестирования. За счёт совмещения функций генератора и анализатора избыточность систем для ряда объектов становится незначительной.

Реализация так называемого кольцевого тестирования послужила дальнейшим развитием применения псевдослучайных тестовых последовательностей для самотестирования вычислительных систем. Основная идея в использовании генераторов *M*-последовательностей для реализации компактного тестирования, заключается в том, что совместно с исследуемой комбинационной схемой подобный генератор образует автономную схему. Структура ее рассмотрена в [59]. В указанной структуре регистр сдвига с сумматором по модулю два в цепи обратной связи представляет собой генератор *M*-последовательности, на вход которого поступают значения, формируемые на выходах исследуемой комбинационной схемы. Последовательные состояния генератора *M*-последовательности подаются в качестве тестовых воздействий на входы схемы. Тестовая процедура подобной структуры реализуется в процессе

выполнения следующих трех этапов, а именно инициализации, контроля и анализа. Этап инициализации заключается в записи на регистр сдвига начального кода (начального тестового набора). Затем реализуется непосредственно этап контроля, который обеспечивается подачей на регистр сдвига управляющих сигналов сдвига. Последний этап состоит в сравнении кода, полученного на регистре сдвига, с его эталонным значением.

Одной из наиболее часто применяемых структур реализации кольцевого тестирования является структура, в которой исследуемая схема совместно с дополнительной логикой реализует линейную обратную связь регистра сдвига [279]. В качестве примера подобной структуры рассмотрим применение кольцевого тестирования для комбинационной схемы, реализующей булеву функцию $f = x_1 \oplus x_2 \oplus x_1 x_2 x_3 \oplus x_4$. Для этого используем генератор *M*-последовательности, описываемый полиномом $\varphi(x) = 1 \oplus x^1 \oplus x^4$. С целью обеспечения линейной обратной связи в регистре сдвига введем дополнительную логику, реализующую функцию $f^* = x_2 \oplus x_1 x_2 x_3$, тогда сумма по модулю два f и f^* определяется по выражению $f \oplus f^* = x^1 \oplus x^4$, что соответствует выбранному примитивному полиному $\varphi(x)$. Функциональная схема реализации данного примера показана на рис. 11.2 [302].

Полнота контроля схемы (рис. 11.2) так же, как и соответствующая характеристика для общей структуры, реализующей кольцевое тестирование, определяется следующими факторами:

1. Старшей степенью m примитивного полинома $\varphi(x)$, описывающего генератор *M*- последовательности.

2. Топологией подключения разрядов регистра сдвига к входам исследуемой схемы.

3. Начальным кодам регистра сдвига.

4. Количеством тактов реализации кольцевого тестирования.

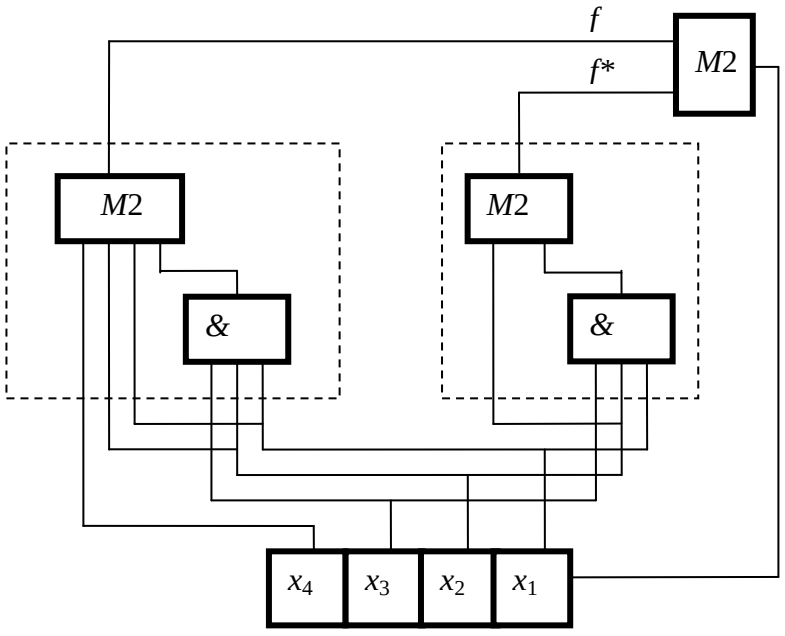


Рисунок 11.2 – *Пример реализации кольцевого тестирования*

Приведенные факторы находятся в сильной взаимной зависимости. Так, при количестве тактов, равном $2^m - 1$, где m представляет собой старшую степень порождающего полинома $\varphi(x)$, начальный код регистра сдвига может принимать любое значение, кроме нулевого. Однако при числе тактов меньшем, чем $2^m - 1$, начальный код уже имеет принципиальное значение. Поэтому в общем случае определение основных параметров кольцевого тестирования является достаточно важной и сложной задачей.

Для получения указанных параметров схемы, реализующей кольцевое тестирование, могут быть использованы результаты исследований [59, 302, 277, 279], которые базируются на применении теории так называемого сигнатурного анализа, рассмотренного в предыдущем разделе.

11.2. Неразрушающее тестирование запоминающих устройств

Важной вехой в эволюции функциональных тестов запоминающих устройств стали разработка и применение так называемых *неразрушающих методов тестирования*. Термин *неразрушающие* (*transparent*) стал использоваться для определения основного свойства новых методов, заключающихся в сохранении и восстановлении данных запоминающих устройств после проведения процедуры тестирования. Необходимость в применении неразрушающего тестирования обусловливается критическим применением многих современных приложений. В первую очередь к ним относятся бортовые компьютерные системы, сетевые серверы, автоматизированные системы управления и т. п., которые требуют *периодического тестирования* (*in field periodic testing*). Для периодического тестирования ЗУ основным критерием, помимо обнаруживающей способности, является способность восстанавливать оперативные данные после каждого тестового сеанса. Одни из первых разработанных систем периодического тестирования ЗУ использовали резервные копии содержимого модулей ЗУ. Процесс неразрушающего тестирования при этом выглядел следующим образом:

1. Перед очередным сеансом тестирования содержимое основного модуля ЗУ сохранялось как резервная копия.
2. Проводилось тестирование основного модуля ЗУ с применением разрушающих алгоритмических тестов.
3. После тестирования информация переписывалась обратно в основной модуль ЗУ с последующим продолжением его функционирования по назначению.

Данный подход требует резервного дублирования как самого запоминающего устройства, так и его содержимого, больших временных затрат на сохранение данных и практически не применяется в современных приложениях ввиду больших информационных объемов ЗУ.

Приведенные ограничения привели к поиску новых решений для осуществления неразрушающего тестирования запоминающих устройств. Одной из первых работ, в которой предлагался кардинально новый подход к тести-

рованию, была идея *Б. Конемана* (*B. Koeneman*) [148], представленная на семинаре *Design For Testability* в 1986 году. Предложенная технология основывалась на линейности сигнатурного анализатора и содержала следующие основные шаги.

1. Содержимое ЗУ представляется в виде линейного потока данных, которые сжимаются на сигнатурном анализаторе. Полученная сигнатура S^{old} сохраняется в регистре эталонной сигнатуры.

2. Содержимое ЗУ видоизменяется, согласно соотношению $M^{new} = M^{old} \oplus TP$, где M^{new} – содержимое ЗУ после изменения, M^{old} – исходное содержимое ЗУ, TP – тестовая маска, $\oplus$ – поразрядная логическая операция, *исключающее* ИЛИ.

3. Измененное содержимое ЗУ M^{new} сжимается на сигнатурном анализаторе. Полученная сигнатура S^{new} записывается в регистр рабочей сигнатуры.

Считалось, что ЗУ является исправным, если выполнялось следующее равенство.

$$S^{old} \oplus S^{new} = S^{TP}, \qquad (11.1)$$

где S^{TP} – сигнатура тестовой маски TP, полученная на том же анализаторе, что и две сигнатуры S^{old} и S^{new}.

Восстановление содержимого ЗУ осуществлялось благодаря линейности операции, *исключающее* ИЛИ, когда на основании M^{new} и TP содержимое памяти восстанавливается, то есть $M^{new} \oplus TP = M^{old} \oplus TP \oplus TP = M^{old}$.

Несмотря на возможность проведения неразрушающего тестирования, предложенный метод обладает рядом недостатков:

1. Данная технология применима лишь для случая использования линейного сигнатурного анализатора в качестве аппаратуры сжатия выходных данных.

2. При определенных условиях может иметь место маскирование неисправностей.

3. Большинство моделей неисправностей, которые используются для описания реальных дефектов ЗУ, достаточно сложны, и для их обнаружения необходимо использовать более сложные тесты, включающие разнообразные виды масок TP, что заметно усложняет процедуру тестирования.

Рассмотрим применение метода неразрушающего тестирования ЗУ, содержащего восемь бит, с использованием в качестве анализатора линейного сдвигового регистра с обратной связью. В качестве порождающего полинома для анализатора выберем полином третьей степени $\varphi(x) = 1 \oplus x \oplus x^3$. Перед сжатием данных начальное состояние анализатора равно 0 0 0. Начальное содержимое ЗУ выберем равным M^{old} = 1 0 1 1 0 0 0 1. Сигнатура тестовой маски TP = 1 1 1 1 1 1 1 1 равна S^{TP} = 1 0 0. Сжимая M^{old} на анализаторе, получим эталонную сигнатуру S^{old} = 1 1 1. После изменения содержимого ЗУ но-

вое значение будет равно M^{new} = 0 1 0 0 1 1 1 0, а значение рабочей сигнатуры S^{new} = 0 1 1. Согласно условию (11.1), $S^{old} \oplus S^{new}$ = 1 1 1 ⊕ 0 1 1 = 1 0 0 = S^{TP}, что свидетельствует об отсутствии неисправностей.

Предположим, что для предыдущего содержимого ЗУ M^{old} = 1 ***0*** ***1*** 1 ***0*** 0 0 1 в нем присутствуют константные и неисправности *SAF*0 в ячейках по адресам 1 и 4 и неисправность *SAF*1 по адресу 2. В результате сжатия значения M^{old} на анализаторе получим эталонную сигнатуру S^{old} = 1 1 1. После инвертирования содержимого ЗУ с применением тестовой маски *TP* = 1 1 1 1 1 1 1 1, новое значение содержимого памяти, с учетом наличия в ЗУ неисправностей, будет равно M^{new} = 0 ***0*** ***1*** 0 ***0*** 1 1 0. Это приведет к вычислению рабочей сигнатуры S^{new} = 0 1 1, вследствие чего значения рабочей и эталонной сигнатур будут удовлетворять условию (11.1), что свидетельствует об отсутствии неисправностей в ЗУ.

Инвертирование содержимого памяти приводит к тому, что неисправности отображаются в выходной последовательности ЗУ как ошибки определенной кратности и конфигурации. Ошибки можно представить в виде маски *E*, которая суммируется с эталонным содержимым ЗУ: $M^{new} = M^{old} \oplus TP \oplus \oplus E = M^{*}$. При этом условие необнаружения ошибки *E* согласно (11.1) выглядит следующим образом:

$$S^{old} \oplus S^{new} \oplus S^{E} = S^{TP}, \qquad (11.2)$$

где S^{E} – сигнатура маски *E*.

Из соотношения (11.2) следует, что равенство сигнатуры S^{E} – нулевой сигнатуре является условием необнаружения ошибки, описываемой маской *E*.

Достоверность данного метода неразрушающего тестирования оценивается способностью трансформировать неисправности в ошибки выходной последовательности данных ЗУ и способностью схемы анализа обнаруживать эти ошибки. В случае применения сигнатурного анализатора способность обнаруживать ошибки оценивается как вероятность необнаружения ошибочной выходной последовательности и выражается формулой (10.8).

Несмотря на ряд перечисленных недостатков, предложенная технология неразрушающего тестирования нашла применение в ряде зарубежных и отечественных систем тестирования.

11.3. Неразрушающее тестирование ЗУ по методу Николаидиса

Из-за описанных недостатков метод Конемана не нашел широкого применения на практике. Вместо него в большинстве случаев используются методы, основанные на преобразовании классических маршевых тестов к неразрушающему виду. Последовательность шагов, требуемых для такого преобразования, впервые была предложена *М. Николаидисом* (*M. Nicolaidis*) в работе [149].

Согласно его методике исходным является маршевый тест, для которого применяются следующие этапы преобразования [149, 259, 329, 338].

1. В начале каждого маршевого элемента исходного маршевого теста, начинающегося с операции записи, добавляется операция чтения. Это требуется для предотвращения потери текущего содержимого запоминающего устройства.

2. Если первый маршевый элемент теста используется только для записи в память инициализирующих значений, а это справедливо для большинства классических маршевых тестов, и этот элемент не обнаруживает никаких дополнительных неисправностей, то он удаляется из исходного маршевого теста.

3. На этом шаге все операции исходного теста заменяются неразрушающими аналогами. Ими являются операции ra и $r\bar{a}$, читающие значение текущей ячейки ЗУ, а также wa и $w\bar{a}$, записывающие в текущую ячейку ЗУ прямое или инверсное значение ранее прочитанного содержимого ячейки.

4. Если последняя операция записи помещает в память значения, обратные исходным значениям ЗУ, то в качестве последнего маршевого элемента применяется элемент $\Uparrow(ra, w\bar{a})$, или элемент $\Downarrow(ra, w\bar{a})$ для восстановления первоначального содержимого памяти.

Полученный на последнем шаге тест представляет собой неразрушающую версию исходного маршевого теста. Он называется *базовым неразрушающим тестом*. Данный тест, подобно, как и исходный маршевый тест, активизирует неисправности путем инвертирования содержимого ЗУ. Для обнаружения ошибок, вызванных присутствием в ЗУ неисправностей, результаты всех операций чтения базового неразрушающего теста сжимаются с помощью сигнатурного анализатора в *рабочую* (*реальную*) *сигнатуру*.

5. Для получения *эталонной сигнатуры*, соответствующей текущему содержимому ЗУ перед реализацией базового теста, реализуется *начальный неразрушающий тест*, состоящий только из операций чтения. Основой для построения начального теста является базовый неразрушающий тест, из которого удаляются все операции записи. Таким образом, получается тест, состоящий только из операций чтения данных, которые затем подаются на вход сигнатурного анализатора. В случае операции чтения ra, прочитанные данные без изменений подаются на вход анализатора, а в случае $r\bar{a}$ результат чтения предварительно инвертируется, а затем сжимается на сигнатурном анализаторе.

Пример 11.1. Рассмотрим применение приведенной методики для классического маршевого теста March $C-$ {⇑⇓($w0$); ⇑($r0$,$w1$); ⇑($r1$,$w0$); ⇓($r0$,$w1$); ⇓($r1$,$w0$); ⇑⇓($r0$)}:

1. Все маршевые элементы теста March $C-$, кроме первого, называемого фазой инициализации, начинаются с операции чтения.

2. Удаляется фаза инициализации ⇑⇓($w0$), в результате получим:

$$\{\Uparrow(r0,w1);\ \Uparrow(r1,w0);\ \Downarrow(r0,w1);\ \Downarrow(r1,w0);\ \Uparrow\Downarrow(r0)\}.$$

3. Операции $r0$ и $w0$ заменяются на операции ra и wa, а операции $r1$ и $w1$ заменяются на операции $r\bar{a}$ и $w\bar{a}$. Имеем:

$$\{\Uparrow(ra, w\bar{a});\ \Uparrow(r\bar{a}, wa);\ \Downarrow(ra, w\bar{a});\ \Downarrow(r\bar{a}, wa);\ \Uparrow\Downarrow(ra)\}.$$

4. Полученный тест в пункте 3 не требует дополнительной фазы записи, так как количество инверсных записей четно, что обеспечивает начальное состояние каждого запоминающего элемента ЗУ. Таким образом, полученный на предыдущем шаге тест и есть базовый неразрушающий тест March $C-$.

5. Соответствующий начальный неразрушающий тест March $C-$ имеет вид:

$$\{\Uparrow(ra);\ \Uparrow(r\bar{a});\ \Downarrow(ra);\ \Downarrow(r\bar{a});\ \Uparrow\Downarrow(ra)\}.$$

Процедура тестирования запоминающих устройств по методу Николаидиса с использованием базового и начального неразрушающих тестов состоит из трех последовательных процедур:

1. Вычисляется эталонная сигнатура S_F путем выполнения начального неразрушающего теста.

2. Вычисляется реальная сигнатура S_R путем выполнения базового неразрушающего теста.

3. Сигнатуры S_F и S_R сравниваются между собой. Их неравенство или равенство определяют, соответственно, наличие или отсутствие неисправностей в запоминающем устройстве.

Описанному методу неразрушающего тестирования также присущ ряд недостатков, среди которых наиболее значимыми является:

1. Существенно увеличивается сложность теста за счет добавления начального неразрушающего теста, используемого для вычисления эталонной сигнатуры S_F.

2. Не гарантируется 100%-ое обнаружение даже однократных неисправностей из-за эффекта маскирования [147, 212, 217, 223].

3. Третий недостаток рассмотренного неразрушающего тестирования связан с ухудшением диагностической способности неразрушающих тестов. Это вызвано сложностью вычисления адреса неисправной ячейки запоминающего устройства на основании эталонной и рабочей сигнатур [147].

Список некоторых из приведенных в таблице 2.3 классических маршевых тестов, преобразованных в неразрушающий вид, приведен в таблице 11.1. В графе «Описание тестов» указанной таблицы первым приведен начальный тест для каждого из классических тестов, а под ним базовый неразрушающий тест. Общая сложность неразрушающего теста складывается из двух слагаемых: сложности начального теста и сложности базового теста.

Таблица 11.1 – *Неразрушающие маршевые тесты Николаидиса*

Название теста	Сложность теста	Описание теста
TMATS	$2N + 3N = 5N$	$\{\Uparrow(ra); \Uparrow\Downarrow(r\bar{a})\}$ $\{\Uparrow(ra, w\bar{a}); \Uparrow\Downarrow(r\bar{a})\}$
TMATS+	$2N + 4N = 6N$	$\{\Uparrow(ra); \Downarrow(r\bar{a})\}$ $\{\Uparrow(ra, w\bar{a}); \Downarrow(r\bar{a}, wa)\}$
TMATS++	$3N + 5N = 8N$	$\{\Uparrow(ra); \Downarrow(r\bar{a}, ra)\}$ $\{\Uparrow(ra, w\bar{a}); \Downarrow(r\bar{a}, wa, ra)\}$
TMarch *X*	$3N + 5N = 8N$	$\{\Uparrow(ra); \Downarrow(r\bar{a}); \Uparrow\Downarrow(ra)\}$ $\{\Uparrow(ra, w\bar{a}); \Downarrow(r\bar{a}, wa); \Uparrow\Downarrow(ra)\}$
TMarch *Y*	$5N + 7N = 12N$	$\{\Uparrow(ra, r\bar{a}); \Downarrow(r\bar{a}, ra); \Uparrow\Downarrow(ra)\}$ $\{\Uparrow(ra, w\bar{a}, r\bar{a}); \Downarrow(r\bar{a}, wa, ra); \Uparrow\Downarrow(ra)\}$
TMarch *C*–	$5N + 9N = 14N$	$\{\Uparrow(ra); \Uparrow(r\bar{a}); \Downarrow(ra); \Downarrow(r\bar{a}); \Uparrow\Downarrow(ra)\}$ $\{\Uparrow(ra, w\bar{a}); \Uparrow(r\bar{a}, wa); \Downarrow(ra, w\bar{a}); \Downarrow(r\bar{a}, wa); \Uparrow\Downarrow(ra)\}$
TMarch *A*	$4N + 14N = 18N$	$\{\Uparrow(ra); \Uparrow(r\bar{a}); \Downarrow(r\bar{a}); \Downarrow(ra)\}$ $\{\Uparrow(ra, w\bar{a}, wa, w\bar{a}); \Uparrow(r\bar{a}, wa, w\bar{a}); \Downarrow(r\bar{a}, wa, w\bar{a}, wa);$ $\Downarrow(ra, w\bar{a}, wa)\}$
TMarch *B*	$6N + 16N = 22N$	$\{\Uparrow(ra, r\bar{a}, ra); \Uparrow(r\bar{a}); \Downarrow(r\bar{a}); \Downarrow(ra)\}$ $\{\Uparrow(ra, w\bar{a}, r\bar{a}, wa, ra, w\bar{a}); \Uparrow(r\bar{a}, wa, w\bar{a});$ $\Downarrow(r\bar{a}, wa, w\bar{a}, wa); \Downarrow(ra, w\bar{a}, wa)\}$
TMarch *U*	$6N + 12N = 18N$	$\{\Uparrow(ra, r\bar{a}); \Uparrow(ra); \Downarrow(r\bar{a}, ra); \Downarrow(r\bar{a})\}$ $\{\Uparrow(ra, w\bar{a}, r\bar{a}, wa); \Uparrow(ra, w\bar{a});$ $\Downarrow(r\bar{a}, wa, ra, w\bar{a}); \Downarrow(r\bar{a}, wa)\}$

Сравнивая сложность тестов, представленных в таблицах 2.3 и 11.1, можно сделать вывод о том, что алгоритмическая сложность неразрушающих тестов, построенных по методу Николаидиса, существенно возрастает (до 30-40 %) в сравнении с оригинальной разрушающей версией.

11.4. Минимальный неразрушающий тест

В работах [218, 233, 313] был предложен неразрушающий маршевый тест минимальной сложности. Было показано, что при условии неограниченного числа выполнения теста для изменяемого содержимого ЗУ тест обладает максимальной покрывающей способностью для заданного класса целевых неисправностей. Предложенный неразрушающий маршевый тест March 3*N* является минимально возможным. Он состоит из двух маршевых элементов, и его сложность оценивается как $Q(\text{March } 3N) = 3N$. Тест имеет следующий вид:

$$\{\Uparrow(ra, w\bar{a}); \Uparrow\Downarrow(r\bar{a})\}. \qquad (11.3)$$

Предложенный тест позволяет активизировать и обнаруживать константные неисправности *SAF*0 и *SAF*1. Активизация неисправностей происходит при выполнении первого маршевого элемента, а обнаружение при сжатии выходной последовательности, формируемой последней операцией чтения $r\bar{a}$. Аналогичное утверждение справедливо и для переходных неисправностей *TF*. В таблице 11.2 приведены условия, необходимые для обнаружения простых неисправностей взаимного влияния тестом March 3*N*.

Таблица 11.2 – *Условия обнаружения неисправностей взаимного влияния тестом March* 3*N*

1	Убывающая (∨)						Возрастающая (∧)					
2	⟨↑,0⟩	⟨↑,1⟩	$\langle\uparrow,\overline{a_j}\rangle$	⟨↓,0⟩	⟨↓,1⟩	$\langle\downarrow,\overline{a_j}\rangle$	⟨↑,0⟩	⟨↑,1⟩	$\langle\uparrow,\overline{a_j}\rangle$	⟨↓,0⟩	⟨↓,1⟩	$\langle\downarrow,\overline{a_j}\rangle$
3	0 0	0 1	0 X	1 0	1 1	1 X	0 1	0 0	0 X	1 1	1 0	1 X
4	1, 2	1, 2	1, 2	1, 2	1, 2	1, 2	1, 1	1, 1	1, 1	1, 1	1, 1	1, 1

В графе 1 таблицы 11.2 указана последовательность изменения адресов, в графе 2 – тип неисправности взаимного влияния, в графе 3 указано содержимое ячеек агрессора и жертвы, а в графе 4 – номера маршевых элементов теста, в которых неисправность активизируется и обнаруживается.

Для обнаружения константных неисправностей достаточно однократного выполнения теста March 3*N*, а для обнаружения переходных неисправностей – двукратного выполнения теста. Неисправность взаимного влияния конкретной конфигурации может быть обнаружена только при определенном состоянии ячейки агрессора и ячейки жертвы. Многократное выполнение, неразрушающее теста March 3*N* при условии изменения содержимого ЗУ при обычном функционировании по назначению, позволит повысить обнаруживающую способность теста. Для оценки зависимости числа обнаруживаемых неисправностей от состояния ЗУ разделим все неисправности взаимного влияния на две группы и три класса (см. таблицу 11.3).

Таблица 11.3 – *Классы и группы неисправностей взаимного влияния*

Классы неисправностей	Группа *A*	Группа *B*
Класс 1	⟨↑, 1⟩	⟨↓, 0⟩
Класс 2	⟨↑, 0⟩	⟨↓,1⟩
Класс 3	$\langle\uparrow,\overline{a_j}\rangle$	$\langle\downarrow,\overline{a_j}\rangle$

Число обнаруживаемых неисправностей, принадлежащих классу 1, не зависит от взаимного расположения 0 и 1 в ЗУ перед тестированием в силу того, что наводимое в жертве значение не совпадает с начальным значением агрессора. Для неисправностей группы *A* это число определяется, как:

$$T_A^1 = 2 \times \binom{N_0}{2}, \tag{11.4}$$

где N_0 – число ячеек ЗУ, содержащих значение 0. Для неисправностей группы *B* число обнаруживаемых неисправностей класса 1 составит:

$$T_B^1 = 2 \times \binom{N_1}{2}, \tag{11.5}$$

где N_1 – число ячеек, содержащих значение 1.

Количество обнаруживаемых неисправностей класса 2 для каждой из групп зависит от взаимного расположения 0 и 1 в ЗУ. Поскольку наводимое в жертве значение совпадает с начальным значением агрессора, то суммарное число неисправностей класса 1 может быть найдено по следующей формуле:

$$T^2 = 2 \times N_0 \times N_1. \tag{11.6}$$

К неисправностям класса 3 относятся инверсные неисправности взаимного влияния. Количество обнаруживаемых неисправностей каждой из групп данного класса равно сумме обнаруживаемых неисправностей, принадлежащих классам 1 и 2 соответствующей группы.

Общее количество неисправностей взаимного влияния, обнаруживаемых неразрушающим маршевым тестом March 3*N* для произвольно выбранного состояния ЗУ, оценивается как:

$$T = 2 \times \left(2 \times \binom{N_0}{2} + 2 \times \binom{N_1}{2} + 2 \times N_0 \times N_1 \right) = 2 \times (N^2 - N), \tag{11.7}$$

где N – количество ячеек тестируемого ЗУ.

Общее число неисправностей взаимного влияния, которые могут присутствовать в ЗУ, составляет:

$$T_{CF} = 12 \times \binom{N}{2} = 6 \times (N^2 - N), \tag{11.8}$$

где множитель 12 определяет число типов неисправностей взаимного влияния, а множитель $\binom{N}{2}$ – число пар агрессор-жертва.

Как видно из (11.7) и (11.8), отношение числа неисправностей взаимного влияния, обнаруживаемых однократным выполнением неразрушающего теста March 3*N*, к общему числу неисправностей взаимного влияния равно 1/3.

Исходя из данных, представленных в таблице 11.2, оценим обнаруживающую способность теста для неисправностей *CFid* и *CFin*:

$$\frac{T_{CFid}}{T} = 1 - (0{,}75)^k; \qquad (11.9)$$

$$\frac{T_{CFin}}{T} = 1 - (0{,}5)^k, \qquad (11.10)$$

где k – количество реализаций неразрушающего маршевого теста March 3*N*.

Общая оценка покрывающей способности теста для неисправностей взаимного влияния с ростом количества k реализаций неразрушающего маршевого теста выглядит весьма высокой.

11.5. Неразрушающие тесты на базе адаптивного сигнатурного анализа

С целью уменьшения временной сложности неразрушающего теста, с сохранением всех его достоинств, в данном разделе рассматриваются *алгоритмы адаптивного сжатия выходных данных* (*АСВД*) (*self-adjusting output data compression − SAODC*), представленные ранее в 10.5. В данном случае рассматривается применение *АСВД* для тестирования и диагностирования встроенных запоминающих устройств [211]. Такой подход позволяет полностью избежать временных издержек для вычисления эталонной сигнатуры. Далее будет показано, что алгоритмы адаптивного сжатия выходных данных могут быть использованы для эффективного обнаружения многократных константных неисправностей, переходных неисправностей и неисправностей взаимного влияния. При реализации встроенных средств самотестирования неразрушающих тестов, основанных на адаптивном сжатии выходных данных, аппаратные затраты сопоставимы с аппаратными затратами обычных неразрушающих тестов.

Концепция адаптивного сжатия выходных данных при реализации средств встроенного самотестирования ЗУ была предложена в [211]. Согласно этой концепции эталонная характеристика (сигнатура) S_F начального содержимого бит-ориентированного ЗУ вычисляется как сумма по модулю два всех адресов ячеек, содержащих значение 1 (см. раздел 10.5). Пример вычисления S_F для ЗУ с $2^m - 1 = 2^3 - 1 = 7$ ячейками представлен на рис. 11.3.

Согласно теории адаптивного сжатия выходных данных запоминающих устройств, тестируемое запоминающее устройство содержит все ячейки, кроме ячейки с нулевым адресом [211]. Пусть бит-ориентированное ОЗУ с $N = 2^m - 1$ ячейками будет представлено массивом $M[d]$, $d \in D$ с адресным пространством $D = \{1, 2, 3, \ldots, 2^m - 1\}$. Тогда $D_1 = \{d \in D \mid M[d] = 1\}$ обозначает все адреса ячеек, содержащих 1, а $D_0 = \{d \in D \mid M[d] = 0\}$, соответственно, обозначает все адреса ячеек, содержащих 0. Для D_1 и D_0 выполняется равенство $D_1 \cup D_0 = D$. Сумма по модулю два всех адресов D_1 и есть характеристика S_F [211], а именно:

$$S_F = \bigoplus_{d \in D_1} d. \qquad (11.11)$$

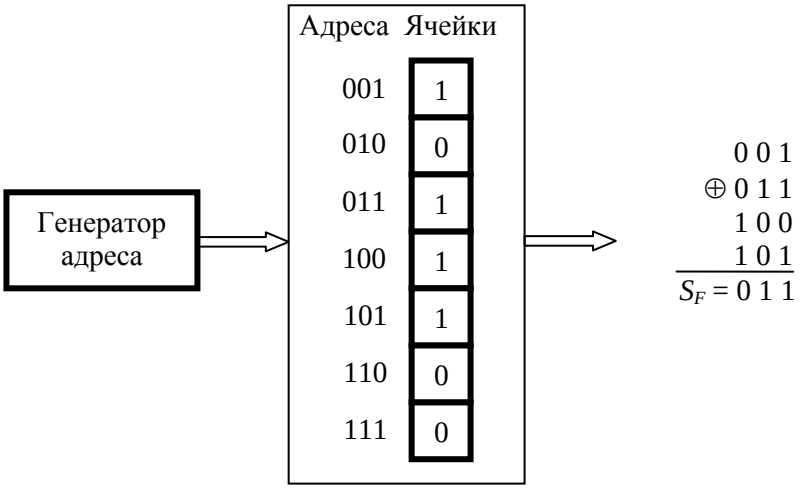


Рисунок 11.3 – *Вычисление эталонной сигнатуры S_F для бит-ориентированного ЗУ*

Для характеристики S_F справедливы следующие соотношения [211]:

$$S_F = \bigoplus_{d \in D_1} d = S_{F_1} = \bigoplus_{d \in D_1} d = S_{F_0} = \bigoplus_{d \in D_0} d; \quad S_F = \bigoplus_{d \in D} d = 0^m. \qquad (11.12)$$

Здесь, 0^m обозначает последовательность из m повторяющихся нулей.

Основываясь на соотношениях (11.12), можно сделать следующее заключение. Для исправного бит-ориентированного ЗУ с $N = 2^m - 1$ ячейками и произвольного его содержимого $A = a_1, a_2, a_3, \ldots, a_N$, $a_i \in \{0, 1\}$, $i \in \{1, 2, 3, \ldots, N\}$, эталонная сигнатура S_F равняется сумме по модулю два всех адресов ячеек, содержащих 1, или сумме по модулю два всех адресов ячеек, содержащих 0 ($S_F = S_{F1} = S_{F0}$). Для примера, представленного на рис. 11.3, $S_F = S_{F1} = 0\ 0\ 1 \oplus 0\ 1\ 1 \oplus 1\ 0\ 0 \oplus 1\ 0\ 1 = 0\ 1\ 1$ и $S_{F0} = 0\ 1\ 0 \oplus 1\ 1\ 0 \oplus 1\ 1\ 1 = 0\ 1\ 1$. Для дальнейшего рассмотрения сформулируем основное свойство *АСВД* [211].

Свойство 11.1. Для вычисления эталонной сигнатуры S_F согласно методу адаптивного сжатия выходных данных могут быть использованы любые алгоритмы формирования адресной последовательности и произвольный порядок генерирования адресов, в том числе возрастающий или убывающий.

Это свойство является следствием равенства (11.11).

Согласно данному свойству эталонная характеристика (сигнатура) S_F содержимого памяти определяется как сумма по модулю два всех адресов ячеек, содержащих 1, при этом порядок следования адресов не влияет на значение этой характеристики.

При реализации маршевых тестов выходные данные ЗУ, формируемые операциями чтения, имеют два значения, а именно, оригинальное значение, описываемое двоичной последовательностью $A = a_1, a_2, a_3, \ldots, a_{N-1}, a_N$, и инверсное значение $\overline{A} = \overline{a_1}, \overline{a_2}, \overline{a_3}, \ldots \ \ldots, \overline{a_{N-1}}, \ \overline{a_N} = 1 \oplus a_1, 1 \oplus a_2, 1 \oplus a_3, \ldots, \ 1 \oplus a_{N-1},$

$1 \oplus a_N$, где если $a_i = 0$, то $\overline{a_i} = 1$ и наоборот если $a_i = 1$, то $\overline{a_i} = 0$. Например, в случае неразрушающего маршевого теста March *C*– {⇑(*ra*, $w\overline{a}$); ⇑($r\overline{a}$,*wa*); ⇓(*ra*, $w\overline{a}$); ⇓($r\overline{a}$,*wa*); ⇑⇓(*ra*)}, в течение первой, третьей и пятой фаз считываются оригинальные данные $A = a_1, a_2, a_3, \ldots, a_{N-1}a_N$, в то время как во второй и четвертой фазах считываются инверсные значения $\overline{A} = \overline{a_1}, \overline{a_2}, \overline{a_3}, ..., \overline{a_{N-1}}, \overline{a_N}$.

Метод АСВД может быть использован при преобразовании оригинальных маршевых тестов в их неразрушающие версии. Предлагаемая процедура преобразования основывается на методике, предложенной Николаидсом [149]. Произвольный маршевый тест может быть преобразован в неразрушающий тест на базе *АСВД* согласно следующей процедуре:

1. Фаза инициализации ⇑⇓(*w*0) заменяется первичной фазой ⇑⇓(*ra*) вычисления эталонной характеристики S_F. Во время данной фазы для исследуемого ОЗУ вычисляется сигнатура S_F согласно (11.11).

2. Во всех остальных фазах маршевого теста все операции *r*0 (*r*1) заменяются операцией ra ($r\overline{a}$).

3. Аналогично, как и в предыдущем шаге, все операции *w*0 (*w*1) заменяются операцией wa ($w\overline{a}$). В результате получается неразрушающий маршевый тест.

Например, для маршевых тестов MATS+ и March *C*– их неразрушающие версии будут иметь вид: {⇑⇓(*ra*); ⇑(*ra*, $w\overline{a}$); ⇓($r\overline{a}$, *wa*)}, и {⇑⇓(*ra*); ⇑(*ra*, $w\overline{a}$); ⇑($r\overline{a}$,*wa*); ⇓(*ra*, $w\overline{a}$); ⇓($r\overline{a}$,*wa*); ⇑⇓(*ra*)}.

Следует отметить, что первичная фаза ⇑⇓(*ra*) используется во всех неразрушающих тестах, построенных согласно приведенной выше процедуре. Это значит, что эталонная сигнатура S_F, которая получается во время данной фазы, не зависит от вида используемого маршевого теста.

В общем случае, любой неразрушающий тест ЗУ, полученный согласно указанной процедуре, формирует регулярные выходные данные. Их вид зависит от трех аргументов, и в первую очередь от размера блока памяти, для которого применяется неразрушающий тест. Параметры блока ОЗУ определяются начальным адресом i_d и конечным адресом f_d ($i_d < f_d$). Кроме того, регулярные выходные данные зависят от содержимого блока памяти *D*, а также от количества и вида операций чтения.

В случае маршевых тестов ЗУ с более чем одной операцией чтения в одной фазе, как было показано в разделе 11.3, значения, полученные во время применения первой операции чтения более значимые по сравнению со значениями, полученными в результате остальных операций чтения. Для последующих операций чтения значения, полученные в результате первой операции чтения, используются как эталонные значения.

Подводя итог, можно сделать вывод, что независимо от числа операций чтения в фазе маршевого теста только результат первой операции чтения должен быть использован для получения как эталонной сигнатуры S_F, так и реального значения сигнатуры во всех фазах маршевого теста.

Для блока памяти, определенного двумя адресами i_d и f_d ($i_d < f_d$) и любого алгоритма генерирования адресов и их порядка следования в пределах данного блока, осуществляющего одно обращение (операцию чтения) ко всем ячейкам блока памяти, справедливы следующие утверждения.

Утверждение 11.1. Эталонная характеристика $S_F = S_F(i_d, f_d, A)$ для блока памяти, описываемого начальным адресом i_d и конечным адресом f_d, а также данными A, определяется как сумма по модулю два последовательных адресов блока памяти ячеек, содержащих 1, согласно выражению (11.11) [211].

Утверждение 11.2. Эталонная характеристика $S_F = S_F(i_d, f_d, (A \oplus \overline{A}))$ суммы по модулю два данных A и $\overline{A}$ тестируемого блока памяти равняется сумме по модулю два характеристик $S_F = S_F(i_d, f_d, A)$ и $S_F = S_F(i_d, f_d, \overline{A})$ или сумме по модулю два всех адресов блока памяти:

$$S_F(i_d, f_d, (A \oplus \overline{A})) = S_F(i_d, f_d, A) \oplus S_F(i_d, f_d, \overline{A}) = S_F(i_d, f_d, (1^{f_d - i_d + 1})). \qquad (11.13)$$

Утверждение 11.2 следует из свойства линейности, описываемого соотношениями (11.11) и (11.12) [211].

Утверждение 11.3. Эталонная характеристика S_F для тестируемого ЗУ с адресным пространством $D = \{1, 2, 3, \ldots, 2^m - 1\}$ (i_d = 0 0 0 … 0 1 = $0^{m-1}1$, f_d = 1 1 1 … 1 1 = 1^m) и содержащего во всех ячейках 1, имеет значение S_F = 0 0 0 … 0 0 = 0^m.

Например, в случае, представленном на рис. 11.3, и полном пространстве адресов памяти, получим 0 0 1 ⊕ 0 1 0 ⊕ 0 1 1 ⊕ 1 0 0 ⊕ 1 0 1 ⊕ 1 1 0 ⊕ 1 1 1 = 0 0 0, что соответствует утверждению 11.3 и основному свойству адаптивного сжатия выходных данных [211]. В то же время в случае блока памяти, состоящего из подмножества последовательных адресов памяти, результат может принимать значения, отличные от нулевого значения. Данное утверждение иллюстрируется примером на рисунке 11.3 для блока памяти, с начальным адресом i_d = 0 1 0, и конечным адресом f_d = 1 0 0. Действительно, для этого примера окончательное значение $S_F(i_d, j_d, (1^3)) = S_F(2, 4, (1^3))$ = 0 1 0 ⊕ 0 1 1 ⊕ 1 0 0 = 1 0 1.

Для блока ОЗУ с фиксированным начальным адресом i_d = 0 0 0 … 0 1 = $0^{m-1}1$ и произвольным конечным адресом f_d, содержащим единичные значения (A = 1 1 1 … 1 1 = $1^{fd+1-1} = 1^{fd}$), сигнатура $S_F = S_F(i_d, f_d, A) = S_F((0^{m-1}1, f_d, 1^{fd})$ зависит только от конечного адреса f_d.

Таким образом, $S_F = S_F(f_d)$, где $S_F(f_d)$ представляет собой сумму по модулю два всех адресов блока памяти, начиная с начального адреса $i_d = i_d(m - 1)\ i_d(m - 2)\ i_d(m - 3)\ \ldots\ i_d(1)\ i_d(0)$ = 0 0 0 … 0 1 = $0^{m-1}1$ и оканчивая адресом $f_d = f_d(m - 1)\ f_d(m - 2)\ f_d(m - 3)\ \ldots\ f_d(1)\ f_d(0)$ [213]. В таблице 11.4 приведены значения суммы по модулю два для последовательных адресов ЗУ, где, например, для f_d = 2 имеем $S_F(f_d = 2) = S_F$(0 0 0 … 0 1 0) = 0 0 0 … 0 0 1 ⊕ 0 0 0 … 0 1 0 = 0 0 0 … 0 1 1.

Таблица 11.4 – *Эталонная характеристика для тестируемого блока памяти*

f_d	$f_d = f_d(m–1)\ f_d(m–2)\ f_d(m–3)\ ...$ $f_d(2)\ f_d(1)\ f_d(0)$	$S_F(f_d) = s_{m–1}\ s_{m–2}\ s_{m–3}\ \ldots$ $s_2\ s_1\ s_0$
1	0 0 0 … 0 0 0 1	0 0 0 … 0 0 0 1
2	0 0 0 … 0 0 1 0	0 0 0 … 0 0 1 1
3	0 0 0 … 0 0 1 1	0 0 0 … 0 0 0 0
4	0 0 0 … 0 1 0 0	0 0 0 … 0 1 0 0
5	0 0 0 … 0 1 0 1	0 0 0 … 0 0 0 1
6	0 0 0 … 0 1 1 0	0 0 0 … 0 1 1 1
7	0 0 0 … 0 1 1 1	0 0 0 … 0 0 0 0
8	0 0 0 … 1 0 0 0	0 0 0 … 1 0 0 0
9	0 0 0 … 1 0 0 1	0 0 0 … 0 0 0 1
10	0 0 0 … 1 0 1 0	0 0 0 … 1 0 1 1
11	0 0 0 … 1 0 1 1	0 0 0 … 0 0 0 0
12	0 0 0 … 1 1 0 0	0 0 0 … 1 1 0 0
13	0 0 0 … 1 1 0 1	0 0 0 … 0 0 0 1
14	0 0 0 … 1 1 1 0	0 0 0 … 1 1 1 1
15	0 0 0 ... 1 1 1 1	0 0 0 … 0 0 0 0
…	…	…
$2^m–1$	1 1 1 … 1 1 1 1	0 0 0 … 0 0 0 0

Как было показано в [213, 214], для значения $S_F(f_d)$ справедливо следующее утверждение.

Утверждение 11.4. Для блока памяти, содержащего все 1 с начальным адресом $i_d = i_d(m-1)\ i_d(m-2)\ i_d(m-3)\ ...\ i_d(1)\ i_d(0) = 0\ 0\ 0 \ldots\ 0\ 1 = 0^{m-1}\ 1$ и произвольным конечным адресом $f_d = f_d(m-1)\ f_d(m-2)\ f_d(m-3)\ ...\ f_d(1)\ f_d(0)$, эталонная характеристика $S_F(f_d = f_d(m-1)\ f_d(m-2)\ f_d(m-3)\ ...\ f_d(1)\ f_d(0))$ принимает значение, которое зависит только от адреса $f_d = f_d(m-1)\ f_d(m-2)\ f_d(m-3)\ ...\ f_d(1)\ f_d(0)$ и определяется согласно выражению:

$$\begin{array}{ll} S_F(f_d) = f_d(m-1)\ f_d(m-2)\ f_d(m-3)\ldots f_d(2)\ f_d(1)\ f_d(1), & \text{для}\quad f_d(0)=0; \\ S_F(f_d) = 000\ldots00\ \overline{f_d(1)} & \text{для}\quad f_d(1)=1, \end{array} \qquad (11.14)$$

где $\overline{f_d(1)}$, есть инверсное значение $f_d(1)$.

Для вычисления значения $S_F(f_d)=s_{m-1}\ s_{m-2}\ s_{m-3}\ \ldots\ s_1\ s_0$, соответствующего приведенному выше утверждению, можно предложить следующий алгоритм.

Алгоритм 11.1.

Входные данные: $f_d = f_d(m-1)\ f_d(m-2)\ f_d(m-3)\ ...\ f_d(1)\ f_d(0)$.

Start:

If $f_d\ mod\ 4 = 0$, then $S_F(f_d) = f_d$;

Else if $f_d\ mod\ 4 = 1$, then $S_F(f_d) = 0\ 0\ 0\ \ldots\ 0\ 0\ 1$;

Else if $f_d\ mod\ 4 = 2$, then $S_F(f_d) = f_d + 1$;

Else if $f_d\ mod\ 4 = 3$, then $S_F(f_d) = 0\ 0\ 0\ \ldots\ 0\ 0\ 0$.

End

Основываясь на утверждении 11.4, можно сформулировать аналогичное утверждение и для произвольных начального и конечного адресов $i_d = i_d(m-1)\ i_d(m-2)\ i_d(m-3)\ ...\ i_d(1)\ i_d(0)$ и $f_d = f_d(m-1)\ f_d(m-2)\ f_d(m-3)\ ...\ f_d(1)\ f_d(0)$.

Утверждение 11.5. Эталонная характеристика $S_F(i_d, f_d, (1^{fd-id+1}))$ для блока памяти с начальным адресом i_d и конечным адресом f_d, содержащего все 1 (A = 1 1 1… 1 1 = $1^{fd-id+1}$), равняется сумме по модулю два характеристик $S_F(i_d - 1)$ и $S_F(f_d)$, определенных согласно утверждению 11.4.

Например, $S_F(3, 7,(1^{7-3+1})) = S_F(3-1) \oplus S_F(7) = S_F(2) \oplus S_F(7)$ = 0 0 0 ... 0 0 1 1 ⊕ 0 0 0 ... 0 0 0 0 = 0 0 0 ... 0 0 1 1 (см. таблицу 11.4).

Последние два утверждения позволяют сформулировать вычислительную процедуру для определения $S_F(i_d, f_d, (1^{fd-id+1}))$. Эта процедура может быть трансформирована в алгоритм, подобный алгоритму 11.1, принимая во внимание, что $(i_a - 1)\ mod\ 4 = (i_a + 3)\ mod\ 4$. В данном алгоритме будут использованы арифметические операции сложения (+) и вычитания (–), а также сумма по модулю два ⊕.

Алгоритм 11.2.

Входные данные: $i_d = i_d(m-1)\ i_d(m-2)\ i_d(m-3)\ ...\ i_d(1)\ i_d(0)$ и $f_d = f_d(m-1)\ f_d(m-2)\ f_d(m-3)\ ...\ f_d(1)\ f_d(0)$.

Start:

if $i_d\ mod\ 4 = 0$,
  if $f_d\ mod\ 4 = 0$, then $S_F = f_d$;
    Else if $f_a\ mod\ 4 = 1$, then S_F = 0 0 0 … 0 0 1;
      Else if $f_d\ mod\ 4 = 2$, then $S_F = f_d + 1$;
        Else if $f_d\ mod\ 4 = 3$, then S_F = 0 0 0 … 0 0 0.
  End if
End if
if $i_d\ mod\ 4 = 1$,
  if $f_d\ mod\ 4 = 0$, then $S_F = (i_d - 1) \oplus f_d$;
    Else if $f_d\ mod\ 4 = 1$, then $S_F = i_d$;
      Else if $f_d\ mod\ 4 = 2$, then $S_F = (i_d - 1) \oplus (f_d + 1)$;
        Else if $f_d\ mod\ 4 = 3$, then $S_F = i_d - 1$.
  End if
End if
if $i_d\ mod\ 4 = 2$,
  if $f_d\ mod\ 4 = 0$, then $S_F = f_d + 1$;
    Else if $f_d\ mod\ 4 = 1$, then S_F = 0 0 0 … 0 0 0;
      Else if $f_d\ mod\ 4 = 2$, then $S_F = f_d$;
        Else if $f_d\ mod\ 4 = 3$, then $S_F(f_d)$ = 0 0 0 … 0 0 1.
  End if
End if
if $i_d\ mod\ 4 = 3$,
  if $f_d\ mod\ 4 = 0$, then $S_F = i_d \oplus f_d$;

Else if f_d *mod* 4 = 1, then $S_F = i_d - 1$;

Else if f_d *mod* 4 = 2, then $S_F = i_d \oplus (f_d + 1)$;

Else if f_d *mod* 4=3, then $S_F = i_d$.

End if

End if

End

Выходные данные: $S_F(i_d, f_d, (1^{fd-id+1})) = S_F = s_{m-1}\, s_{m-2}\, s_{m-3} \ldots s_1\, s_0$.

Для примера определим значение $S_F(i_d, f_d, (1^{fd-id+1}))$ для $i_d = 3$ и $f_d = 13$. В соответствии с алгоритмом 11.2 получим, что i_d *mod* 4 = 3 *mod* 4 = 3 и f_d *mod* 4 = 13 *mod* 4 = 1. Тогда $S_F = S_F(i_d, f_d, (1^{fd-id+1})) = S_R(3,13,(1^{fd-id+1})) = i_d - 1 = 0\,0\,1\,1 - 0\,0\,0\,1 = 0\,0\,1\,0$, или как результат суммы по модулю два адресов рассматриваемого блока получим $S_F(i_d, f_d, (1^{fd-id+1})) = S_R(3,13,(1^{fd-id+1})) = 0\,0\,1\,1 \oplus 0\,1\,0\,0 \oplus 0\,1\,0\,1 \oplus 0\,1\,1\,0 \oplus 0\,1\,1\,1 \oplus 1\,0\,0\,0 \oplus 1\,0\,0\,1 \oplus 1\,0\,1\,0 \oplus 1\,0\,1\,1 \oplus 1\,1\,0\,0 \oplus 1\,1\,0\,1 = 0\,0\,1\,0$.

Приведенные выше утверждения и алгоритмы для адаптивного сжатия выходных данных позволяют сформулировать следующие основные свойства для *АСВД*, используемого для реализации неразрушающего тестирования ЗУ и определенного ранее приведенной процедурой.

Свойство 11.2. Эталонная характеристика $S_F(i_d, f_d, \overline{A})$, полученная согласно алгоритму адаптивного сжатия выходных данных для содержимого $\overline{A}$ блока тестируемого ЗУ, описываемого начальным i_d и конечным f_d адресами, может быть вычислена как:

$$S_F(i_d\, f_d, \overline{A}) = S_F(i_d\, f_d, A) \oplus S_F(i_d\, f_d, (1^{f_d - i_d + 1})), \qquad (11.15)$$

где $S_F(i_d, f_d, (1^{fd-id+1}))$ вычисляется согласно алгоритму 11.2.

Это свойство может быть рассмотрено как следствие соотношения (11.13).

Свойство 11.3. Эталонная характеристика S_F так же, как и значение реальной характеристики S_R для случая исправного состояния блока ЗУ для двух последовательных значений данных A и $\overline{A}$ не зависит от содержимого A блока ЗУ, а определяется только начальным i_d и конечным f_d адресами рассматриваемого блока, что следует из соотношения:

$$S_F(i_d\, f_d, A\overline{A}) = S_F(i_d\, f_d, A \oplus \overline{A}) = S_F(i_d\, f_d, (1^{f_d - i_d + 1})) \qquad (11.16)$$

Свойство 11.4. Неразрушающий тест на базе *АСВД* характеризуется аналогичной или большей обнаруживающей и диагностической способностью неисправностей ЗУ, по сравнению с классическим неразрушающим тестом в силу многократного сравнения S_R с эталонным значением S_F.

Для неразрушающих тестов, основанных на *АСВД*, это сравнение производится после каждой фазы маршевого теста ЗУ. В случае классических

неразрушающих тестов, S_R сравнивается с эталонным значением однократно по выполнению всех фаз теста.

Свойство 11.5. Средства неразрушающего самотестирования ЗУ на базе *АСВД* имеют низкие аппаратную и временную сложности.

Низкая аппаратная сложность определяется минимальным набором стандартных устройств, используемых для встроенного самотестирования. Так же, как и в случае известных технологий неразрушающего самотестирования, аппаратные средства для реализации неразрушающего самотестирования ЗУ на базе *АСВД* включают минимальный набор дополнительных устройств. В их число входит генератор адресов ЗУ, схема вычисления эталонной сигнатуры, схема вычисления реальной сигнатуры, схема хранения эталонной сигнатуры, схема сравнения реальной сигнатуры с эталонной и устройство управления самотестированием. По сравнению с известными методами неразрушающего самотестирования, неразрушающее самотестирование ЗУ на базе *АСВД* характеризуется минимальными затратами на реализацию схем вычисления эталонной и реальной сигнатур. Эти устройства представляют собой набор из m двухвходовых сумматоров по модулю два, где m – разрядность адреса ЗУ.

Минимальная временная сложность неразрушающего тестирования на базе *АСВД* обеспечивается минимальным временем для вычисления эталонной сигнатуры определяемым емкостью N ЗУ, которое равняется N циклам обращения к ЗУ. Для сравнения рассмотрим два стандартных неразрушающих теста MATS+ $\{\Uparrow(ra, w\bar{a});\ \Downarrow(r\bar{a},\ wa)\}$, и March *C*– $\{\Uparrow(ra, w\bar{a});\ \Uparrow(r\bar{a}, wa);\ \Downarrow(ra, w\bar{a});\ \Downarrow(r\bar{a}, wa);\ \Uparrow\Downarrow(ra)\}$. Эталонная сигнатура S_F для обоих стандартных тестов определяется с помощью сигнатурного анализатора, описываемого полиномом $\varphi(x)$. Эталонная сигнатура формируется как результат сжатия на анализаторе данных в соответствии с операциями чтения $\{\Uparrow(ra);\ \Downarrow(r\bar{a})\}$для MATS+ и $\{\Uparrow(ra);\ \Uparrow(r\bar{a});\ \Downarrow(ra);\ \Downarrow(r\bar{a});\ \Uparrow\Downarrow(ra)\}$ для March *C*–. В первом случае необходимо $2N$ циклов обращения к ЗУ, а во втором $5N$ циклов. Суммарная сложность выше приведенных тестов составляет $6N$ для MATS+ и $14N$ для March *C*–. Использование неразрушающих тестов MATS+ и March *C*– на базе *АСВД* характеризуется временной сложностью $5N$ и $10N$, соответственно. В данном случае для вычисления эталонного значения сигнатуры S_F, независимо от используемого маршевого теста необходимо только N дополнительных циклов обращения к ЗУ для чтения содержимого всех N ячеек ЗУ.

Глава 12. Симметричные тесты запоминающих устройств

12.1. Симметрия неразрушающих маршевых тестов

Определяющим недостатком существующих методов неразрушающего тестирования, как отмечалось в предыдущем разделе, является необходимость вычисления эталонного значения сигнатуры, что заметно увеличивает сложность процедуры тестирования по сравнению с классическими тестами. Одним из радикальных путей устранения данного недостатка является использование свойств маршевых тестов ЗУ, и в первую очередь неразрушающих маршевых тестов. Наиболее значимые характеристики неразрушающих маршевых тестов могут быть сформулированы следующим образом [214, 215, 222]:

1. Неразрушающий тест записывает в ячейки ЗУ только инверсное значение либо значение, которое хранилось в ячейке ЗУ.

2. Тест не может записать в ячейку ЗУ никакое заранее предопределенное значение, что является следствием отсутствия фазы инициализации.

3. Изменение содержимого ЗУ возможно только путем инвертирования содержимого его ячеек. Данное свойство является обобщением двух предыдущих характеристик неразрушающих маршевых тестов.

4. Первая фаза неразрушающего теста всегда начинается с операции чтения $\{\Uparrow(ra);\ \ldots\}$ таким образом, что первоначально прочитанное значение является эталоном для последующих фаз теста. Отметим, что это значение нигде не запоминается, а используется в структуре теста, опосредовано через фиксированную последовательность инверсных записей.

5. Все последующие фазы неразрушающего теста начинаются с операции чтения, что объясняется необходимостью обеспечения наблюдаемости проявления неисправностей в предыдущей фазе.

6. Структура теста, представленная операциями чтения, содержит как локально симметричные участки, так и глобальную симметрию.

Более подробно рассмотрим структуру маршевых тестов с точки зрения наличия в них симметрии. В качестве примера возьмем неразрушающий тест MATS + $\{\Uparrow(ra, w\bar{a});\Downarrow(r\bar{a}, wa)\}$. Тогда алгоритм вычисления эталонной сигнатуры выглядит следующим образом:

$$\{\Uparrow(ra);\Downarrow(r\bar{a})\}. \qquad (12.1)$$

Анализ последнего соотношения операций чтения показывает, что первоначально данные $a_0, a_1, a_2, \ldots, a_{N-1}$, $a_i \in \{0, 1\}$; $i = 0, 1, 2, \ldots, N-1$, которые ранее, в течение первой фазы, были записаны в ячейки ЗУ, в упорядоченной последовательности считываются и используются для формирования значения сигнатуры. Во время второй фазы эти же проинвертированные двоичные данные $\overline{a_{N-1}}, \overline{a_{N-2}}, \ldots, \overline{a_2}, \overline{a_1}, \overline{a_0}$, считываются, но уже строго в обратной после-

довательности, что свидетельствует о наличии симметрии в 13.1. В тоже время в тесте March *C*− $\{\Uparrow(ra,w\bar{a});\Uparrow(r\bar{a},wa);\Downarrow(ra,w\bar{a});\Downarrow(r\bar{a},wa);\Uparrow(ra)\}$ операции чтения

$$\{\Uparrow(ra);\Uparrow(r\bar{a});\Downarrow(ra);\Downarrow(r\bar{a});\Uparrow(ra)\}, \qquad (12.2)$$

образуют несколько симметричных участков с различным видом симметрии [260, 261]. Действительно, из теста можно выделить несколько симметричных фрагментов операций чтения, некоторые из которых приведены ниже [96, 97 303]:

Номер фаз теста	Фрагменты теста
1,2	$\Uparrow(ra);\Uparrow(r\bar{a});$
1,3	$\Uparrow(ra);\Uparrow(ra);$
1,5	$\Uparrow(ra);\Uparrow(ra);$
4,5	$\Downarrow(r\bar{a});\Uparrow(ra);$
1,2,3,4	$\Uparrow(ra);\Uparrow(r\bar{a});\Downarrow(ra);\Downarrow(r\bar{a});$

Данные, получаемые как результат чтения содержимого ЗУ и обладающие симметрией, можно сгруппировать в четыре вида симметрии, которые будут использоваться для построения симметричных тестов (см. рис.12.1).

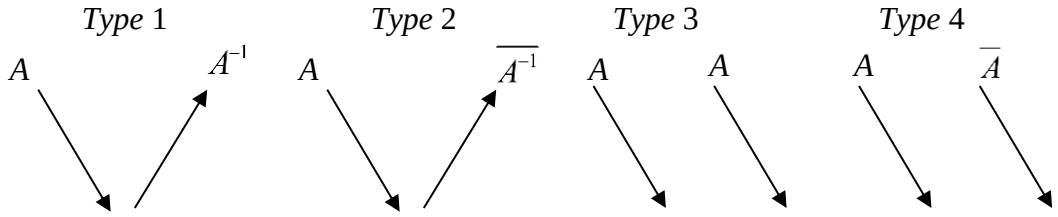


Рисунок 12.1 – *Виды симметрии данных*

Здесь $A = a_0, a_1, a_2, \ldots, a_{N-1}, a_i \in \{0, 1\}$; $i = 0, 1, 2, \ldots, N-1$ представляет собой некоторую последовательность данных. Тогда обратной последовательностью последовательность будет $A^{-1} = a_{N-1}, a_{N-2}, \ldots, a_2, a_1, a_0$, инверсной последовательностью - $\bar{A} = \bar{a_0}, \bar{a_1}, \bar{a_2}, ..., \overline{a_{N-2}}, \overline{a_{N-1}} = 1 \oplus a_0, 1 \oplus a_1, 1 \oplus a_2, \ldots, 1 \oplus a_{N-1}$, где если $a_i = 0$, то $\bar{a_i} = 1$ и наоборот, если $a_i = 1$, то $\bar{a_i} = 0$. Обратной инверсной последовательностью будет последовательность $\overline{A^{-1}} = \overline{a_{N-1}}, \overline{a_{N-2}}, ..., \bar{a_2}, \bar{a_1}, \bar{a_0}$. Отметим, что все указанные виды симметрии присутствуют в тесте March *C*−. Действительно, 1 и 3, а также 3 и 5 фазы образуют *Type* 1 симметрию; 1 и 4, 2 и 3, 4 и 5, а также 1, 2 и 3, 4 фазы образуют *Type* 2 симметрию. Симмет-

рию *Type* 3 образуют фазы 1 и 5, а симметрию *Type* 4 фазы 1 и 2, и фазы 3 и 4.

Наличие симметрии позволит в дальнейшем обеспечить формирование сигнатур данных, обладающих некоторыми видами симметрии, значение которых не будет зависеть от сжимаемой информации [214, 215, 303].

12.2. Получение сигнатур для симметричных данных

Как уже отмечалось в предыдущих разделах, значение сигнатуры S, получаемое как результат сжатия данных $A = a_0, a_1, a_2, \ldots, a_{N-1}$, $a_i \in \{0, 1\}$; $i = 0, 1, 2, \ldots, N-1$, на сигнатурном анализаторе, описываемом примитивным порождающим полиномом $\varphi(x) = \alpha_0 \oplus \alpha_1 x \oplus \alpha_2 x^2 \oplus \ldots \oplus \alpha_m x^m$, $\alpha_0 = \alpha_m = 1$, $\alpha_i \in \{0, 1\}$ $i = 1, 2, \ldots, m-1$, зависит от самих данных их вида и длины. Кроме того, сигнатура зависит от порождающего полинома сигнатурного анализатора, который определяет размерность сигнатуры и ее значение. Отметим, что изменения начального состояния S_0 сигнатурного анализатора приводит к изменению результирующей сигнатуры S сжимаемых данных. Таким образом, сигнатура S является функцией, зависящей от трех аргументов $S = S(A, S_0, \varphi(x))$, а именно: от сжимаемых данных A, начального состояния сигнатурного анализатора S_0 и его порождающего полинома $\varphi(x)$. Введем ряд определений, необходимых для дальнейшего изложения материала данного раздела.

1. Обозначим через $S = S(A, S_0, \varphi(x))$ сигнатуру, получаемую при сжатии последовательности данных A на сигнатурном анализаторе, описываемом примитивным порождающим полиномом $\varphi(x)$, степени m с начальным состоянием $S_0 = (s_0^1, s_0^2, s_0^3, \ldots, s_0^m)$, $s_0^j \in \{0, 1\}$; $j = 1, 2, 3, \ldots, m$.

2. Если $S = (s_i^1, s_i^2, s_i^3, \ldots, s_i^m)$, где $s_i^j \in \{0, 1\}$; $j = 1, 2, 3, \ldots, m$, представляет собой некоторое состояние сигнатурного анализатора в i-ый такт его функционирования, тогда обратным (транспонированным) состоянием сигнатурного анализатора назовем состояние $S = (s_i^m, s_i^{m-1}, \ldots, s_i^3, s_i^2, s_i^1)$.

3. Если $\varphi(x) = \alpha_0 \oplus \alpha_1 x \oplus \alpha_2 x^2 \oplus \ldots \oplus \alpha_m x^m$, $\alpha_0 = \alpha_m = 1$, $\alpha_i \in \{0, 1\}$ $i = 1, 2, \ldots, m-1$, представляет собой примитивный порождающий полином степени m. Тогда обратным полиномом $\varphi(x)^{-1}$ по отношению к полиному $\varphi(x)$ будет полином $x^m \varphi(x^{-1}) = \alpha_m \oplus \alpha_{m-1} x \oplus \alpha_{m-2} x^2 \oplus \ldots \oplus \alpha_1 x^{m-1} \oplus \alpha_0 x^m$. Здесь $\alpha_0 = \alpha_m = 1$. Используя приведенные определения, докажем следующую теорему.

Теорема 12.1. Если при сжатии последовательности данных A на сигнатурном анализаторе с начальным состоянием S_0, описываемом полиномом $\varphi(x)$, была получена сигнатура $S = S(A, S_0, \varphi(x))$. Тогда сигнатура S, полученная при сжатии последовательности данных A^{-1} на сигнатурном анализаторе, описываемом полиномом $\varphi(x)^{-1}$, с начальным состоянием $S^{-1} = S(A, S_0, \varphi(x))^{-1}$ будет ровняться обратному первоначальному состоянию S_0^{-1}, то есть:

$$S = S(A^{-1}, S(A, S_0, \varphi(x))^{-1}, \varphi(x)^{-1}) = S_0^{-1}.$$

Доказательство теоремы 12.1. Рассмотрим последовательность данных A, состоящую из одного бита (для последовательности данных большей длины доказательство будет следовать по индукции). При сжатии последовательности A на сигнатурном анализаторе, описываемом полиномом $\varphi(x) = 1 \oplus \alpha_1 x \oplus \alpha_2 x^2 \oplus \ldots \oplus \alpha_m x^m$ с начальным состоянием $S_0 = (s_0^1, s_0^2, s_0^3, \ldots, s_0^m)$, получается сигнатура S:

$$S = \left(A \oplus \sum_{i=1}^{m} {}^{\oplus} s_0^i \times \alpha_i,\ s_0^1,\ s_0^2,\ \ldots, s_0^{m-2}\ s_0^{m-1} \right).$$

Тогда:

$$S^{-1} = \left(s_0^{m-1},\ s_0^{m-2},\ \ldots,\ s_0^2,\ s_0^1, A \oplus \sum_{i=1}^{m} {}^{\oplus} s_0^i \times \alpha_i \right),$$

и $A^{-1} = A$.

Учитывая, что младший и старший коэффициенты α_0 и α_m полинома $\varphi(x)$, степени m всегда равны единице, сигнатура, получаемая на обратном сигнатурном анализаторе $\varphi(x)^{-1}$, будет описываться следующим соотношением:

$$\left(A^{-1} \oplus A \oplus \left(\sum_{i=1}^{m} {}^{\oplus} s_0^i \times \alpha_i \right) \times \alpha_m \oplus \sum_{i=1}^{m-1} {}^{\oplus} s_0^i \times \alpha_i,\ s_0^{m-2},\ s_0^{m-1},\ \ldots, s_0^2,\ s_0^1 \right) = \left(s_0^m, s_0^{m-1},\ \ldots, s_0^1 \right) = S_0^{-1}$$

.

Что и требовалось доказать [214, 215, 303].

В качестве иллюстрации данной теоремы рассмотрим следующий пример.

Пример 12.1. Рассмотрим сигнатурный анализатор, описываемый полиномом $\varphi(x) = 1 \oplus x \oplus x^4$ с начальным состоянием $S_0 = (s_0^1, s_0^2, s_0^3, s_0^4) = 1\ 1\ 0\ 0$, приведенный на рис. 12.2. В качестве сжимаемых данных используем последовательность $A = a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11} = 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1$. Тогда $S = S(A, S_0, \varphi(x)) = ((1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1), (1\ 1\ 0\ 0), (\varphi(x) = 1 \oplus x \oplus x^4))$, будет вычисляться в соответствии с диаграммой, приведенной в таблице 12.1. В результате получим $S = (s_{11}^1, s_{11}^2, s_{11}^3, s_{11}^4) = 1\ 0\ 1\ 1$.

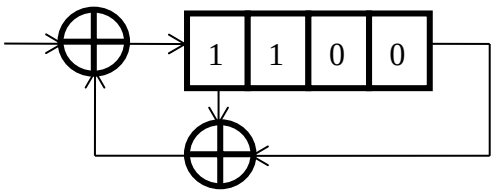


Рисунок 12.2 – *Сигнатурный анализатор для полинома* $\varphi(x) = 1 \oplus x \oplus x^4$

Таблица 12.1 – *Диаграмма функционирования сигнатурного анализатора, описываемого полиномом* $\varphi(x) = 1 \oplus x \oplus x^4$

i	A	$S = (s_i^1, s_i^2, s_i^3, \ldots, s_i^m)$
	S_0	**1 1 0 0**
0	$a_0 = 1$	0 1 1 0
1	$a_1 = 1$	1 0 1 1
2	$a_2 = 0$	0 1 0 1
3	$a_3 = 1$	0 0 1 0
4	$a_4 = 0$	0 0 0 1
5	$a_5 = 0$	1 0 0 0
6	$a_6 = 1$	0 1 0 0
7	$a_7 = 0$	0 0 1 0
8	$a_8 = 1$	1 0 0 1
9	$a_9 = 1$	1 1 0 0
10	$a_{10} = 1$	0 1 1 0
11	$a_{11} = 1$	**1 0 1 1**

Для порождающего полинома $\varphi(x)=1 \oplus x \oplus x^4$ взаимообратным полиномом является полином $\varphi(x)^{-1}= 1 \oplus x^3 \oplus x^4$. На его основе строиться сигнатурный анализатор, приведенный на рис. 12.3, а в качестве сжимаемых данных для этого сигнатурного анализатора используется последовательность $A^{-1}= a_{11}, a_{10}, a_9, a_8, a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0 = 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1$. Начальное состояние $S_0 = S^{-1} = (s_{11}^1, s_{11}^2, s_{11}^3, s_{11}^4)^{-1} = (1\ 0\ 1\ 1)^{-1} = 1\ 1\ 0\ 1$. Процедура сжатия данных сигнатурным анализатором на базе полинома $\varphi(x)^{-1}= 1 \oplus x^3 \oplus x^4$ и результирующая сигнатура приведены в таблице 12.2.

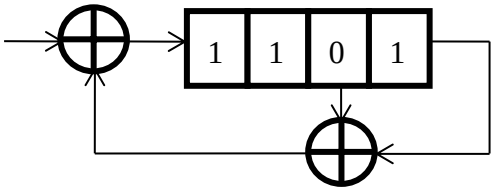


Рисунок 12.3 – *Сигнатурный анализатор для полинома* $\varphi(x) = 1 \oplus x^3 \oplus x^4$

Таким образом, рассмотренная технология сжатия последовательностей данных AA^{-1} состоит в использовании двух взаимообратных сигнатурных анализаторов SA и SA^{-1}. Причем для сжатия прямой последовательности данных A произвольной длины используется сигнатурный анализатор SA, значение сигнатуры которого является начальным значением сигнатуры для анализатора SA^{-1}, на котором сжимается обратная последовательность данных A^{-1}.

Основываясь на данном описании, несложно сформулировать свойства приведенной технологии сжатия [98, 214, 215, 221, 303].

Анализ результата, вытекающего из теоремы 12.1 и проиллюстрированного на примере, показывает инвариантность сигнатуры S сжимаемых данных AA^{-1}, имеющих симметрию *Type* 1 от вида данных A и их размерности

(длины). Значение сигнатуры S всегда равняется S_0^{-1}, где S_0 есть начальное состояние сигнатурного анализатора *SA*. Сформулируем данное свойство для наиболее реального случая, когда S_0 = 0 0 0 … 0.

Таблица 12.2 – *Диаграмма функционирования сигнатурного анализатора, для полинома* $\varphi(x) = 1 \oplus x^3 \oplus x^4$

i	A	$S = (s_i^1, s_i^2, s_i^3, \ldots, s_i^m)$
	S_0	**1 1 0 1**
0	$a_0 = 1$	0 1 1 0
1	$a_1 = 1$	0 0 1 1
2	$a_2 = 1$	1 0 0 1
3	$a_3 = 1$	0 1 0 0
4	$a_4 = 0$	0 0 1 0
5	$a_5 = 1$	0 0 0 1
6	$a_6 = 0$	1 0 0 0
7	$a_7 = 0$	0 1 0 0
8	$a_8 = 1$	1 0 1 0
9	$a_9 = 0$	1 1 0 1
10	$a_{10} = 1$	0 1 1 0
11	$a_{11} = 1$	**0 0 1 1**

Свойство 12.1. Для начального состояния S_0 = 0 0 0 … 0 результирующая сигнатура S сжимаемых данных AA^{-1} не зависит от вида данных A и их размерности (длины), а также от вида порождающего полинома $\varphi(x)$ и его степени $m = deg\varphi(x)$ и всегда равняется нулевому значению 0 0 0 … 0.

В тоже время из теории сигнатурного анализа следует, что любая последовательность данных, в которой отсутствует симметрия AA^{-1}, будет равновероятно отображаться в произвольную сигнатуру S таким образом, что вероятность равенства S = 0 0 0 … 0 равняется $1/2^m$. Очевидно, что для больших значений степени $m = deg\varphi(x)$ порождающего полинома $\varphi(x)$ эта величина практически равняется нулю. Все изменения (ошибки) в последовательности AA^{-1}, вызванные неисправностями ЗУ, будут обнаруживаться с большой вероятностью $P_d = 1 - 1/2^m$. Для случая ЗУ данный факт сформулируем в виде свойства.

Свойство 12.2. Все неисправности ЗУ, которые отображаются в ошибки, нарушающие симметрию *Type* 1 в сжимаемых данных AA^{-1}, будут обнаруживаться с вероятностью $P_d = 1 - 1/2^m$, а все неисправности, не нарушающие данную симметрию, будут необнаруживаемыми.

Анализ неисправностей ЗУ показывает, что неисправности взаимного влияния *CF*, а также кодочувствительные неисправности *PSF* нарушают симметрию AA^{-1} и соответственно являются обнаруживаемыми. В тоже время все константные неисправности *SAF*, и в некоторых случаях переходные неисправности *TF* являются необнаруживаемыми.

Дальнейшим развитием применения симметрии для целей тестирования ЗУ является следующая теорема [98, 214, 215, 221, 303].

Теорема 12.2. Если при сжатии последовательности данных A на сигнатурном анализаторе с начальным состоянием S_0, описываемом полиномом $\varphi(x)$, была получена сигнатура $S = S(A, S_0, \varphi(x))$. Тогда сигнатура S, получаемая при сжатии последовательности данных $\overline{A^{-1}} = \overline{a_{N-1}}, \overline{a_{N-2}}, ..., \overline{a_2}, \overline{a_1}, \overline{a_0}$ на сигнатурном анализаторе, описываемом полиномом $\varphi(x)^{-1}$, с начальным состоянием S^{-1} будет равняться поразрядной сумме по модулю два обратного S_0^{-1} начального состояния со значением сигнатуры $S((1\ 1\ 1\ \ldots\ 1), (0\ 0\ 0\ \ldots\ 0), \varphi(x)^{-1})$. единичной последовательности 1 1 1 … 1, то есть:

$$S = S(\overline{A^{-1}}, S(A, S_0, \varphi(x))^{-1}, \varphi(x)^{-1}) = S_0^{-1} \oplus S((1\ 1\ 1\ \ldots\ 1), (0\ 0\ 0\ \ldots\ 0), \varphi(x)^{-1}).$$

Доказательство теоремы 12.2. Последовательность сжимаемых данных $\overline{A^{-1}} = \overline{a_{N-1}}, \overline{a_{N-2}}, ..., \overline{a_2}, \overline{a_1}, \overline{a_0}$ может быть представлена как:

$$\overline{A^{-1}} = \overline{a_{N-1}}, \overline{a_{N-2}}, ..., \overline{a_2}, \overline{a_1}, \overline{a_0} = (1 \oplus a_{N-1}, 1 \oplus a_{N-2}, \ ..., 1 \oplus a_2, 1 \oplus a_1, 1 \oplus a_0),$$

где инверсное значение a_i равняется поразрядной сумме по модулю два единичного значения со значением a_i то есть $a_i = 1 \oplus a_i$. Тогда последовательность $\overline{A^{-1}}$ можно представить в виде поразрядной сумме по модулю два последовательности $A^{-1} = (a_{N-1}, a_{N-2}, \ldots, a_2, a_1, a_0)$ и единичной последовательности 1 1 1 … 1 такой же длины. То есть $\overline{A^{-1}} = A^{-1} \oplus (1\ 1\ 1\ \ldots\ 1)$.

В выражение для сигнатуры $S = S(\overline{A^{-1}}, S(A, S_0, \varphi(x))^{-1}, \varphi(x))^{-1}$ подставим $\overline{A^{-1}} = A^{-1} \oplus (1\ 1\ 1\ \ldots\ 1)$ и применим свойства линейности сигнатурного анализа, в результате получим, что:

$$S = S((A^{-1} \oplus (1\ 1\ 1\ \ldots\ 1)), S(A, S_0, \varphi(x))^{-1}, \varphi(x)^{-1}) = S(A^{-1}, S(A, S_0, \varphi(x))^{-1}, \varphi(x)^{-1}) \oplus S((1\ 1\ 1\ \ldots\ 1), (000\ldots0), \varphi(x)^{-1}).$$

Наконец используя теорему 12.1 для значения $S(A^{-1}, S(A, S_0, \varphi(x))^{-1}, \varphi(x)^{-1})$, получим, что $S = S_0^{-1} \oplus S((1\ 1\ 1\ \ldots\ 1), (0\ 0\ 0\ \ldots\ 0), \varphi(x)^{-1})$. Что и требовалось доказать.

Практическая реализация основной идеи, представленной в теореме 12.2 так же, как и в предыдущем случае, основана на использовании двух взаимообратных сигнатурных анализаторов SA и SA^{-1}, описываемых полиномами $\varphi(x)$ и $\varphi(x)^{-1}$. Отличием является только конструкция анализатора SA^{-1} (см. рис. 12.4), в котором на входной сумматор по модулю два подается значение единицы при сжатии последовательности $\overline{A^{-1}}$ для обеспечения инвариантности результирующей сигнатуры от сжимаемых данных.

Аналогичный результат будет получен, когда вместо дополнительного входа сжимаемая последовательность будет предварительно проинвертирована, что сопряжено с включением по входу анализатора инвертора.

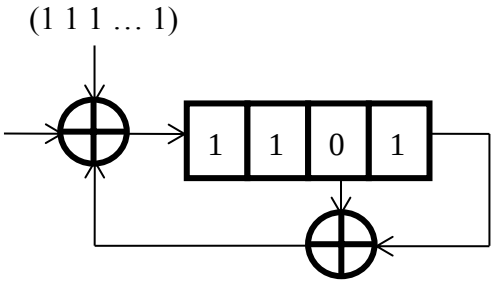


Рисунок 12.4 – *Модифицированный сигнатурный анализатор для полинома* $\varphi(x)^{-1} = 1 \oplus x^3 \oplus x^4$

Действительно, сумма сигнатуры $S(A^{-1}, S(A, S_0, \varphi(x))^{-1}, \varphi(x)^{-1})$ с сигнатурой $S((1\ 1\ 1\ \ldots\ 1), (0\ 0\ 0\ \ldots\ 0), \varphi(x)^{-1})$ в силу линейности сигнатурного анализа равняется S_0^{-1}. В дальнейшем в качестве S_0 всегда будем использовать S_0 = 0 0 0 … 0. Таким образом, начальное состояние S_0 анализатора SA всегда принимается нулевым.

Пример 12.2. Рассмотрим сигнатурный анализатор, описываемый полиномом $\varphi(x) = 1 \oplus x \oplus x^4$ с начальным состоянием $S_0 = (s_0^1, s_0^2, s_0^3, s_0^4) = 0\ 0\ 0\ 0$, приведенный на рис. 12.2 (см. пример 12.1), для которого временная диаграмма сжатия данных $A = a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}$ = 1 1 0 1 0 0 1 0 1 1 1 1 приведена в таблице 12.3.

Таблица 12.3 – *Диаграмма функционирования сигнатурного анализатора SA и модифицированного анализатора* SA^{-1}

i	A	SA	$\overline{A^{-1}} \oplus 1$	SA^{-1}
		$S = (s_i^1, s_i^2, s_i^3, \ldots, s_i^m)$		$S = (s_i^1, s_i^2, s_i^3, \ldots, s_i^m)$
		$S_0 =$ **0 0 0 0**		$S_0 =$ **1 0 0 1**
0	$a_0 = 1$	1 0 0 0	0 ⊕ 1 = 1	0 1 0 0
1	$a_1 = 1$	0 1 0 0	0 ⊕ 1 = 1	1 0 1 0
2	$a_2 = 0$	0 0 1 0	0 ⊕ 1 = 1	0 1 0 1
3	$a_3 = 1$	1 0 0 1	0 ⊕ 1 = 1	0 0 1 0
4	$a_4 = 0$	0 1 0 0	1 ⊕ 1 = 0	1 0 0 1
5	$a_5 = 0$	0 0 1 0	0 ⊕ 1 = 1	0 1 0 0
6	$a_6 = 1$	1 0 0 1	1 ⊕ 1 = 0	0 0 1 0
7	$a_7 = 0$	0 1 0 0	1 ⊕ 1 = 0	1 0 0 1
8	$a_8 = 1$	1 0 1 0	0 ⊕ 1 = 1	0 1 0 0
9	$a_9 = 1$	0 1 0 1	1 ⊕ 1 = 0	0 0 1 0
10	$a_{10} = 1$	0 0 1 0	0 ⊕ 1 = 1	0 0 0 1
11	$a_{11} = 1$	**1 0 0 1**	0 ⊕ 1 = 1	**0 0 0 0**

Как видно из приведенной таблицы, финальное значение сигнатуры ровняется нулевому значению. Здесь так же, как и в случае теоремы 12.1, нет необходимости в фазе определения эталонной сигнатуры для тестирования ЗУ, если считываемые данные обладают симметрией *Type* 2. Фактически эталонная сигнатура вычисляется за счет незначительной модификации сигна-

турного анализатора SA^{-1}, заключающейся в добавлении дополнительного входа, на который постоянно подается единичная последовательность.

Следует отметить, что анализаторы SA и SA^{-1}, могут быть реализованы с использованием одной структуры, построенной на сдвиговом регистре, выполняющем микрооперацию сдвига как в сторону младших разрядов, так и в сторону старших.

Основываясь на приведенном описании, несложно выделить несколько свойств метода сжатия симметричных данных $A\overline{A^{-1}}$, связанных с анализом его обнаруживающей способности.

Свойство 12.3. Для ошибок произвольной кратности, присутствующих только в одной из частей A или $\overline{A^{-1}}$ последовательности данных $A\overline{A^{-1}}$, обнаруживающая способность определяется так же, как и в случае классического сигнатурного анализа.

Это свойство вытекает из того факта, что присутствие ошибок только в одной половине последовательности $A\overline{A^{-1}}$ нарушает симметрию *Type* 2. Более того, если в последовательности данных A присутствует ошибка некоторой кратности E тогда $A_E = A \oplus E$. И если в результате сжатия последовательностей $A_E\overline{A^{-1}}$ получается сигнатура S_E, тогда в результате сжатия последовательности $A\overline{A_E^{-1}}$, где $\overline{A_E^{-1}} = \overline{A^{-1}} \oplus E^{-1}$, будет также получена сигнатура S_E.

Пример, иллюстрирующий данное свойство, приведен на рис. 12.5. Из приведенного рисунка видно, что в обоих случаях финальная сигнатура принимает одинаковое значение, равное S_E = 0 0 0 1.

Свойство 12.4. Если в последовательности данных A присутствует ошибка произвольной кратности $E1$ такая, что $A_{E1} = A \oplus E1$, а в последовательности данных $\overline{A^{-1}}$ также присутствует произвольная ошибка некоторой кратности $E2$, для которой $\overline{A_{E2}^{-1}} = \overline{A^{-1}} \oplus E2$. Тогда данная ошибка $E1E2$ не будет обнаружена рассмотренной технологией, в случае если $E2 = E1^{-1}$.

Это следует из того факта, что если выполняется равенство $E2 = E1^{-1}$, то симметрия *Type* 2 в последовательности $A\overline{A^{-1}}$ нарушена не будет. Симметричная последовательность $A\overline{A^{-1}}$ преобразуется в новую симметричную последовательность $A_{E1}\overline{A_{E2}^{-1}}$.

Анализ неисправностей ЗУ и их отображения в ошибки последовательности $A\overline{A^{-1}}$ показывает, что большинство неисправностей, включая константные неисправности, нарушают симметрию *Type* 2. Следовательно, все изменения (ошибки) в последовательности $A\overline{A^{-1}}$, вызванные всевозможными неисправностями ЗУ будут обнаруживаться с большой вероятностью $P_d = 1 - 1/2^m$, аналогично, как и для случая классических неразрушающих тестов. Для случая ЗУ, данный факт сформулируем в виде свойства [98, 214, 215, 221, 303]:

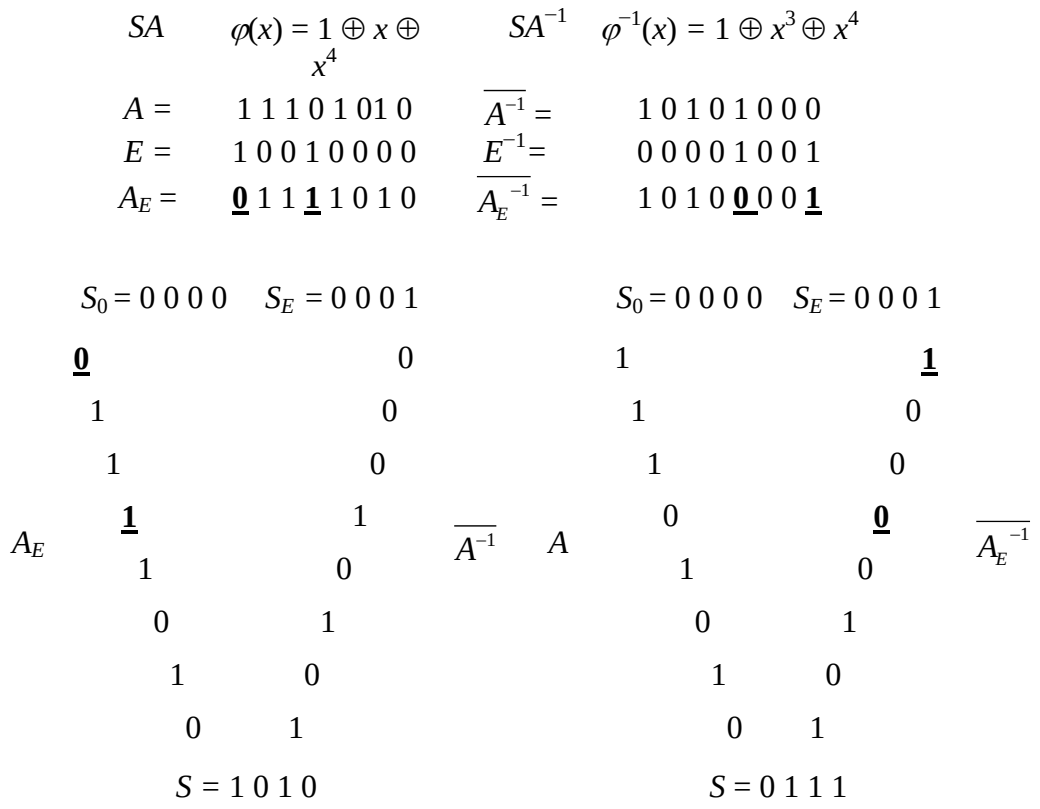


Рисунок 12.5 – *Иллюстрация свойства* 12.3

Свойство 12.5. Все неисправности ЗУ, которые отображаются в ошибки, нарушающие симметрию *Type* 2 в сжимаемых данных $A\overline{A^{-1}}$, будут обнаруживаться с вероятностью $P_d = 1 - 1/2^m$, а все неисправности, не нарушающие данную симметрию, будут необнаруживаемыми.

Учитывая высокую покрывающую способность тестов ЗУ, обладающих симметрией *Type* 2 (*symmetric transparent algorithm − STA*), сформулируем основные свойства неразрушающих тестов, построенных на их основе. Представим их в виде утверждений.

Утверждение 12.1. Процедура *STA* тестирования является неразрушающей.

Утверждение 12.2. Эталонная сигнатура S_{ref}, так же, как и реальная сигнатура S_{real}, полученная в соответствии с *STA*, при наличии неисправностей в ЗУ, имеет нулевое значение $S_{ref} = S_{real} = 0\,0\,0\,\ldots\,0$ и не зависит от:

1. Размера тестируемого ЗУ и его содержимого.
2. Используемого теста памяти, имеющего симметрию *Type* 2.
3. Вида примитивного порождающего полинома $\varphi(x)$.

Утверждение 12.3. Покрывающая способность симметричных неразрушающих тестов не ниже покрывающей способности классических неразрушающих тестов.

Более высокая покрывающая способность может быть получена в результате добавления к тесту ЗУ дополнительных операций чтения для обеспечения симметрии *Type* 2.

Утверждение 12.4. Реализация *STA* тестирования требует минимальных аппаратурных затрат, выражающихся в затратах на сигнатурный анализатор.

12.3. Синтез симметричных неразрушающих маршевых тестов

Проблему синтеза неразрушающих симметричных тестов (*STA*) рассмотрим для случая маршевых тестов в силу их широкого практического применения. Любой маршевый тест состоит из набора последовательных фаз. Каждая фаза маршевого теста содержит определенный набор операций записи и чтения, которые последовательно применяются ко всем ячейкам ЗУ. Как было показано в предыдущих разделах данной главы, данные, которые считываются при реализации маршевых тестов, обладают симметрией. Примеры локальной симметрии для тестов MATS + и March *C*− приведены в разделах 12.1 и 12.2. Однако если в случае теста MATS + имела место глобальная симметрия, то для теста March *C*− были рассмотрены только примеры локальной симметрии [237, 238, 261, 303].

С целью получения симметричных неразрушающих тестов для любого маршевого теста рассмотрим проблему преобразования исходного теста, таким образом, чтобы формируемую в результате выполнения операций чтения последовательность данных можно было бы представить в виде последовательности $A\overline{A^{-1}}$. Это может быть достигнуто путем введения в маршевый тест дополнительной операции чтения. Подобная возможность основывается на следующем утверждении.

Утверждение 12.5. Покрывающая способность маршевого теста не изменяется при добавлении к любой его фазе в произвольном месте произвольного количества операций чтения или фаз, содержащих только операции чтения.

Данное утверждение основывается на допущениях, которые принимаются при определении математических моделей неисправностей ЗУ (см. главу 2). Ни одна из рассматриваемых классических неисправностей не активизируется при выполнении операции чтения, следовательно, дополнительные операции чтения не могут вызывать маскирование неисправностей [147, 207, 212, 303, 311].

Очевидно, что внесение дополнительных операций чтения увеличивает сложность маршевого теста, но как будет показано далее, даже в самом неблагоприятном случае его сложность будет не больше, чем общая сложность алгоритма вычисления эталонной сигнатуры и базового маршевого теста для классических неразрушающих тестов. Действительно, для случая, когда исходный маршевый тест состоит только из однонаправленных фаз, он может быть преобразован для использования с новой технологией сжатия путем добавления перед его началом аналогичного теста, в котором фазы имеют обратное направление и отсутствуют операции записи. В этом случае сложность добавленного фрагмента теста будет полностью соответствовать слож-

ности алгоритма вычисления эталонной сигнатуры согласно классической технологии неразрушающего тестирования. Отметим, что на практике не встречаются маршевые тесты с однонаправленными фазами. Следовательно, сложность новых симметричных неразрушающих тестов будет меньше, чем сложность аналогичных тестов для классического неразрушающего тестирования.

Рассмотрим пример преобразования исходного маршевого теста в симметричный неразрушающий тест на примере теста March *X*. Исходный маршевый тест March *X*, сложность которого равняется 6*N*, имеет вид:

$$\{\Updownarrow(w0);\Uparrow(r0,\ w1);\Downarrow(r1,w0);\Uparrow(r0)\}. \qquad (12.3)$$

Классический неразрушающий тест на базе March *X* выглядит следующим образом:

1. Базовый неразрушающий тест: $\{\Uparrow(ra,\ w\bar{a});\Downarrow(r\bar{a},\ wa);\Uparrow(ra)\}$.

2. Алгоритм вычисления эталонной сигнатуры: $\{\Uparrow(ra);\Downarrow(r\bar{a});\Uparrow(ra)\}$.

Таким образом, общая сложность неразрушающего теста March *X* составляет $5N + 3N = 8N$ циклов обращения к памяти.

Для преобразования исходного теста проанализируем факт наличия в нем симметрии. С этой целью базовый неразрушающий тест представим в виде структуры операций чтения, показанной на рис. 12.6. Здесь так же, как и на рис.12.1, каждая стрелка соответствует отдельной операции чтения каждой фазы маршевого теста. Направление стрелки указывает направление изменения адресов в пределах каждой фазы теста, а фазы разделены вертикальной пунктирной линией.

Анализ диаграммы, приведенной на рис.5.6, свидетельствует только о наличии локальной симметрии для 1 и 2 фазы (*Type* 2) и такой же симметрии для 2 и 3 фаз. Задача преобразования сводится к преобразованию теста таким образом, что бы все его фазы в совокупности образовывали глобальную симметрию.

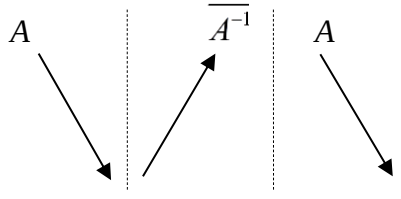


Рисунок 12.6 – *Операции чтения неразрушающего теста March X*

В качестве механизма преобразования используем принцип добавления дополнительных операций чтения в тест таким образом, чтобы представленная выше структура операций чтения была преобразована в симметричную структуру. В данном случае для этого достаточно добавить в качестве последней фазы теста фазу $\Downarrow(r\bar{a})$. Тогда структура операций чтения теста примет вид:

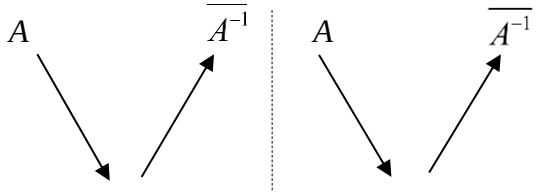


Рисунок 12.7 – *Операции чтения неразрушающего теста STAMarch X*_1

Тест March *X*, преобразованный в симметричный неразрушающий тест STAMarch *X*_1, выглядит следующим образом:

$$\{\Uparrow(ra, w\bar{a});\Downarrow(r\bar{a}, wa);\Uparrow(ra);\ \Downarrow\mathbf{(}\boldsymbol{r}\bar{a}\mathbf{)}\} \qquad (12.4)$$

Здесь и далее для выделения дополнительных операций чтения будем использовать полужирный шрифт. Теперь, при выполнении первой и второй фаз теста будет формироваться последовательность данных *A*, а при выполнении третьей и четвертой – также последовательность *A*, таким образом, получена глобальная симметрия *AA* для данных, формируемых операциями чтения теста STAMarch *X*_1.

Аналогичным образом дополнительная фаза чтения $\Downarrow(r\bar{a})$ может быть добавлена в качестве первой фазы симметричного неразрушающего теста. В результате получим тест:
STAMarch *X*_2 (6*N*):

$$\{\Downarrow\mathbf{(}\boldsymbol{r}\bar{a}\mathbf{)};\Uparrow(ra, w\bar{a});\Downarrow(r\bar{a}, wa);\Uparrow(ra)\}. \qquad (12.5)$$

Между двумя приведенными тестами STAMarch *X*_1 и STAMarch *X*_2 существует не только структурное отличие. Эти тесты будут характеризоваться и разной покрывающей способностью для неисправностей взаимного влияния.

Отдельные маршевые тесты уже изначально по своей сути обладают необходимой глобальной симметрией и не требуют каких-либо модификаций. Например, тест MATS + $\{\Uparrow\Downarrow(w0);\ \Uparrow(r0, w1);\ \Downarrow(r1, w0)\}$, преобразованный в классический неразрушающий тест $\{\Uparrow(ra, w\bar{a});\ \Downarrow(r\bar{a}, wa)\}$, сам по себе обладает симметрией $A\,\overline{A^{-1}}$ и не нуждается в дополнительных операциях чтения. В этом случае сложность неразрушающего теста даже меньше, чем сложность базового маршевого теста.

Далее рассмотрим случай, когда дополнительные операции чтения добавляются в уже существующие фазы теста. Для этого рассмотрим пример теста MATS ++ $\{\Uparrow\Downarrow(w0);\ \Uparrow(r0, w1);\ \Downarrow(r1, w0, r0)\}$. В этом тесте вторая фаза содержит две операции чтения. Для обеспечения глобальной симметрии необходимо модифицировать тест таким образом, чтобы структура его операций чтения, например, приняла следующий вид [303]:

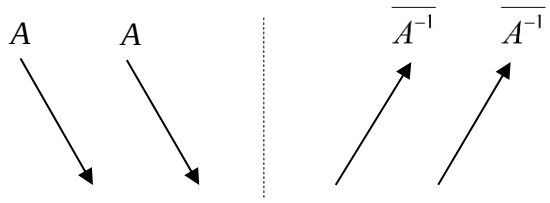


Рисунок 12.8 – *Симметричная последовательность операций чтения*

Простейшим способом преобразования теста MATS ++ в STAMATS ++ является добавление в первую фазу операции чтения $r\bar{a}$. Тогда неразрушающий тест примет следующий вид STAMATS ++ {⇑(***r***$\bar{a}$, *ra*, *w*$\bar{a}$); ⇓(*r*$\bar{a}$, *wa*, *ra*)}. Как видно, сложность этого теста такая же, как и у исходного маршевого теста. Однако при реализации данного алгоритма для встроенного тестирования временные затраты могут быть уменьшены за счет совмещения выполнения двух операций чтения. Это основывается на том факте, что между двумя операциями чтения первой фазы STAMATS ++ нет операций записи и, следовательно, отсутствует вероятность проявления неисправности в текущей ячейке памяти. Тогда можно выполнить цикл чтения из памяти только для первой операции чтения, а считываемое значение для второй операции чтения автоматически используется как результат первой операции чтения.

В качестве реализации совмещенного выполнения двух операций чтения используем хорошо апробированную технологию синтеза многоканальных сигнатурных анализаторов, а именно двухканальных сигнатурных анализаторов [207, 208, 301, 302, 304, 305].

В таблице 5.6 приведены результаты синтеза симметричных неразрушающих тестов на основе принципов изложенных в настоящем разделе. Для некоторых исходных маршевых тестов приведено несколько вариантов их преобразования в STA тесты. Как уже отмечалось, с целью получения симметричных неразрушающих тестов в классические тесты дополнительно вводились только операции чтения. Вновь введенные операции чтения выделены жирным шрифтом, а группы операций, которые могут выполняться параллельно, подчеркнуты [303].

Отметим, что в таблице 5.6 приведены лишь простейшие модификации исходных тестов с целью обеспечения свойств симметрии и только за счет дополнительных операций чтения.

Таблица 12.4 – *Симметричные неразрушающие маршевые тесты*

Исходный тест	Симметричный неразрушающий тест STA
MATS	STAMATS (3*N*): {⇑(*ra*, $w\bar{a}$); ⇓(*r*$\bar{a}$)}
MATS +	STAMATS + (4*N*): {⇑(*ra*, $w\bar{a}$); ⇓(*r*$\bar{a}$, *wa*)}
MATS ++	STAMATS ++ (6*N*): {⇑(**r**$\bar{a}$), *r*a, $w\bar{a}$); ⇓(*r*$\bar{a}$, *wa*, *ra*)}
Marching 1/0	STAMarching 1/0 (13*N*): {⇑(*ra*, $w\bar{a}$, *r*$\bar{a}$); ⇓(*r*$\bar{a}$, *wa*, *ra*); ⇑($w\bar{a}$); ⇑(*r*$\bar{a}$, *wa*, *ra*); ⇓(*ra*, $w\bar{a}$, *r*$\bar{a}$)}
March *X*	STAMarch *X*_1 (6*N*): {⇑(*ra*, $w\bar{a}$); ⇓(*r*$\bar{a}$, *wa*); ⇑(*ra*); ⇓(**r**$\bar{a}$)} STAMarch *X*_2 (6*N*): {⇓(**r**$\bar{a}$); ⇑(*ra*, $w\bar{a}$); ⇓(*r*$\bar{a}$, *wa*); ⇑(*ra*)}
March *Y*	STAMarch *Y* (10*N*): {⇓(**r**$\bar{a}$); ⇑(**r**$\bar{a}$, *r*a, $w\bar{a}$, *r*$\bar{a}$); ⇓(**r**a, *r*$\bar{a}$, *wa*, *ra*); ⇑(*ra*)}
March *C*–	STAMarch *C*– (10*N*): {⇑(**r**$\bar{a}$); ⇑(*ra*, $w\bar{a}$); ⇑(*r*$\bar{a}$, *wa*); ⇓(*ra*, $w\bar{a}$); ⇓(*r*$\bar{a}$, *wa*); ⇓(*ra*)}
March *C*	STAMarch *C*_1 (12*N*): {⇑(*ra*, $w\bar{a}$); ⇑(*r*$\bar{a}$, *wa*); ⇑(*ra*); ⇑(**r**$\bar{a}$); ⇓(*ra*, $w\bar{a}$); ⇓(*r*$\bar{a}$, *wa*); ⇓(*ra*); ⇓(**r**$\bar{a}$)} STAMarch *C*_2 (12*N*): {⇑(**r**$\bar{a}$); ⇑(*ra*, $w\bar{a}$); ⇑(**r**$\bar{a}$); ⇑(*r*$\bar{a}$, *wa*); ⇓(*ra*); ⇓(*ra*, $w\bar{a}$); ⇓(*r*$\bar{a}$,*wa*); ⇓(*ra*);}
March A	STAMarch *A*_1 (16*N*): {⇑(*ra*, $w\bar{a}$, *wa*, $w\bar{a}$); ⇑(*r*$\bar{a}$, *wa*, $w\bar{a}$); ⇑(**r***a*); ⇓(*r*$\bar{a}$, *wa*, $w\bar{a}$, *wa*); ⇓(*ra*, $w\bar{a}$, *wa*); ⇓(**r**$\bar{a}$)} STAMarch *A*_2 (16*N*): {⇑(**r**$\bar{a}$); ⇑(*ra*, $w\bar{a}$, *wa*, $w\bar{a}$); ⇑(*r*$\bar{a}$, *wa*, $w\bar{a}$); ⇓(**r***a*); ⇓(*r*$\bar{a}$, *wa*, $w\bar{a}$, *wa*); ⇓(*ra*, $w\bar{a}$, *wa*)} STAMarch *A*_3 (18*N*): {⇑(*r*a, **r**$\bar{a}$, *wa**, *wa*, $w\bar{a}$); ⇑(*r*$\bar{a}$, **r***a*, *wa*, $w\bar{a}$); ⇓(*r*$\bar{a}$, **r***a*, *wa*, $w\bar{a}$, *wa*); ⇓(*ra*, **r**$\bar{a}$, $w\bar{a}$, *wa*)}
March B	STAMarch *B*_1(20*N*): {⇑(*ra*, $w\bar{a}$, *r*$\bar{a}$, *wa*, *ra*, $w\bar{a}$); ⇑(*r*$\bar{a}$,*wa*, $w\bar{a}$); ⇑(**r**$\bar{a}$); ⇓(*r*$\bar{a}$,*wa*, $w\bar{a}$, *wa*); ⇓(*r*$\bar{a}$, $w\bar{a}$, *wa*); ⇓(**r**$\bar{a}$, *r*a, **r**$\bar{a}$)} STAMarch *B*_2(20*N*): {⇑(*ra*, $w\bar{a}$, *r*$\bar{a}$, *wa*, *ra*, $w\bar{a}$); ⇑(*r*$\bar{a}$, **r***a*, *wa*, $w\bar{a}$); ⇓(*r*$\bar{a}$, **r***a*, *wa*, $w\bar{a}$, *wa*); ⇓(**r**$\bar{a}$, *r*a, **r**$\bar{a}$, $w\bar{a}$, *wa*)}
Algorithm B	STAAlgorithm *B* (18*N*): {⇑(*r*a, **r**$\bar{a}$, $w\bar{a}$, *wa*, $w\bar{a}$); ⇑(*r*$\bar{a}$, *wa*, *ra*, $w\bar{a}$); ⇓(*r*$\bar{a}$, **r***a*, *wa*, $w\bar{a}$, *wa*); ⇓(*ra*, $w\bar{a}$, *r*$\bar{a}$, *wa*)}

12.4. Оценка эффективности неразрушающих симметричных тестов

При оценке эффективности симметричных неразрушающих тестов отметим, что полнота обнаружения неисправностей складывается из двух составляющих, во-первых, из эффективности используемого теста, то есть его способности отображать неисправности ЗУ в ошибки последовательности данных, считываемых из ЗУ, и во-вторых из обнаруживающей способности используемого метода сжатия данных.

Обнаруживающей способности классических маршевых тестов посвящено большое число работ [69, 70, 103, 303, 332] и это подробно рассматривалось в предыдущих главах. Поэтому основное внимание уделим зависимости между используемым классическим маршевым тестом и конечной обнаруживающей способностью его реализации в виде неразрушающего симметричного теста [103, 218, 220, 223, 303]. Как уже отмечалось ранее, основное назначение теста состоит в том, чтобы инициировать проявления неисправностей в ошибки сжимаемой последовательности данных. Если полученная в результате выполнения теста конфигурация ошибки E_1E_2 является несимметричной относительно центральной линии симметрии теста, то вероятность обнаружения инициировавшей появление такой ошибки неисправности будет определяться достоверностью классического сигнатурного анализа. В случае неразрушающих симметричных тестов эта достоверность определяется длиной сжимаемой последовательности, равной произведению емкости ЗУ N на количество операций чтения в тесте и разрядностью сигнатурного анализатора.

Если формируемая в результате выполнения теста конфигурация ошибки E_1E_2 является симметричной (то есть $E_2 = E_1^{-1}$), то инициировавшая ее неисправность не будет обнаружена. Следовательно, основной задачей, при оценке обнаруживающей способности симметричных неразрушающих тестов является анализ отображения конкретных неисправностей в конфигурации ошибок для заданного маршевого теста. Неисправности, которые не инициируются маршевым тестом и, не проявляются в ошибках выходной последовательности данных, будем относить к группе $G1$ данного маршевого теста. Неисправности, которые отображаются маршевым тестом в несимметричные конфигурации ошибок, и вероятность обнаружения которых будет определяться достоверностью сигнатурного анализатора, будем называть неисправностями группы $G2$ данного маршевого теста. А неисправности, которые отображаются маршевым тестом в симметричную конфигурацию ошибки и, следовательно, не обнаруживаются новой технологией сжатия, будем относить к группе $G3$. Очевидно, что симметричный неразрушающий маршевый тест, который не будет иметь неисправностей в группе $G3$, будет характеризоваться максимальной эффективностью [303]. Отметим, что приведенный анализ рассматривается в рамках симметрии *Type* 2. Анализ эффективности начнем с наиболее простых константных неисправностей, для которых сформулируем следующее утверждение.

Утверждение 12.6. Константные неисправности, обнаруживаемые исходным маршевым тестом, обнаруживаются и симметричным (*Type* 2) неразрушающим тестом, построенным на основе исходного теста, и относятся к группе $G2$.

Действительно, вследствие того, что сжимаемые последовательности данных A и $\overline{A^{-1}}$ являются обратными и инверсными по отношению друг к другу, константная неисправность будет отражаться в несимметричную конфигурацию ошибки.

Пример 12.3. Рассмотрим утверждение 12.6 на примере маршевого теста STAMarch *C*−: {⇑(***r*** $\bar{a}$); ⇑(*ra*, *w* $\bar{a}$); ⇑(*r* $\bar{a}$, *wa*); ⇓(*ra*, *w* $\bar{a}$); ⇓(*r* $\bar{a}$,*wa*); ⇓(*ra*)}.

Предположим, что содержимое ЗУ емкостью 8 бит, перед началом выполнения теста, имело следующий вид $A = a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$ = 0 0 0 1 1 0 0 1 и пусть в ячейке с адресом *4* присутствует константная неисправность SAF1. Тогда полученная в результате выполнения теста сжимаемая последовательность данных будет иметь вид, представленный на рис. 12.9.

Как видно из рис.12.9, константная неисправность в приведенном примере отражается в несимметричную конфигурацию ошибки и, следовательно, относится к группе *G*2. Это происходит в силу того, что в обеих частях теста из четвертой ячейки памяти будет считываться несимметричная (*Type* 2) последовательность данных, что нарушит симметрию сжимаемой последовательности данных:

111			*0*			*1*			*0*		*1*			*0*			*1*	
110		*1*			*0*			*1*			*0*			*1*			*0*	
101		*1*			*0*			*1*			*0*			*1*			*0*	
100		**1**			*1*			**1**			*1*			**1**			*1*	
011		*0*			*1*			*0*			*1*			*0*			*1*	
010		*1*			*0*			*1*			*0*			*1*			*0*	
001		*1*			*0*			*1*			*0*			*1*			*0*	
000		*1*			*0*			*1*			*0*			*1*			*0*	
Адрес		⇑(***r*** $\bar{a}$)			⇑(*ra*, *w* $\bar{a}$)			⇑(*r* $\bar{a}$, *wa*);			⇓(*ra*, *w* $\bar{a}$)			⇓(*r* $\bar{a}$, *wa*)			⇓(*ra*)	

Рисунок 12.9 – *Реализация теста STAMarch C*−

Подобное утверждение относится и к переходным неисправностям. Следовательно, константные и переходные неисправности, обнаруживаемые исходным маршевым тестом, будут относиться к группе *G*2 преобразованного маршевого теста [303].

Далее рассмотрим отображение неисправностей взаимного влияния в ошибки выходной последовательности. Для этого пронумеруем все возможные типы неисправностей взаимного влияния: 1) ∧⟨↑, 0⟩ 2) ∧⟨↑, 1⟩ 3) ∧⟨↓, 0⟩ 4) ∧⟨↓, 1⟩ 5) ∨⟨↑, 0⟩ 6) ∨⟨↑, 1⟩ 7) ∨⟨↓, 0⟩ 8) ∨⟨↓, 1⟩ 9) ∧⟨↑, a_j⟩ 10) ∧⟨↓, a_j⟩ 11) ∨⟨↑, a_j⟩ 12) ∨⟨↓, a_j⟩. Эффективность обнаружения подобных неисправностей рассмотрим на примере симметричного неразрушающего теста STAMarch *X*_1: {⇑(*ra*, *w* $\bar{a}$); ⇓(*r* $\bar{a}$, *wa*); ⇑(*ra*); ⇓(***r*** $\bar{a}$)}. Предположим, что первоначальное содержимое ЗУ нулевое. Тогда для каждой неисправности взаимного влияния можно определить фазу либо фазы теста STAMarch *X*_1, в которых конкретная неисправность отображается в “ошибку”. Результаты анализа представим в виде таблицы 12.5, где символ “+” означает возникновение ошибки, символ “-” ее отсутствие для конкретной неисправности номер которой указан в первом столбце [312, 313, 322].

Таблица 12.5 – *Эффективность теста STAMarch X_1*

№	$\Uparrow(ra, w\bar{a})$	$\Downarrow(r\bar{a}, wa)$	$\Uparrow(ra)$	$\Downarrow(\boldsymbol{r}\bar{a})$
1	-	-	-	-
2	+	+	+	+
3	-	-	-	-
4	-	-	+	+
5	-	+	+	+
6	-	-	-	-
7	-	+	+	+
8	-	-	-	-
9	+	+	+	+
10	-	-	+	+
11	-	+	+	+
12	-	+	+	+

Как видно из таблицы 12.5, все множество неисправностей взаимного влияния для теста STAMarch *X*_1 распределилось следующим образом: *G*1 = {1, 3, 6, 8}, *G*2 = {4, 5, 7, 10, 11, 12} и *G*3 = {2, 9}. Неисправности 2 и 9 множества *G*3 не были обнаружены симметричным неразрушающим тестом из-за того, что тест STAMarch *X*_1 отразил их в симметричные ошибки. Простейшим решением, позволяющим избежать подобного маскирования ошибок, является перенос дополнительной фазы с операцией чтения в начало теста, как это сделано в тесте STAMarch *X*_2: $\{\Downarrow(\boldsymbol{r}\bar{a}); \Uparrow(ra, w\bar{a}); \Downarrow(r\bar{a}, wa); \Uparrow(ra)\}$, что следует из таблицы 12.6.

Все множество неисправностей взаимного влияния для теста STAMarch *X*_2 перераспределилось между группами *G*1, *G*2 и *G*3 следующим образом: *G*1 = {1, 3, 6, 8}, *G*2 = {2, 4, 5, 7, 9, 10, 11, 12} и *G*3 = {∅}. Преобразованный маршевый тест STAMarch *X*_2, обеспечивает максимально возможную покрывающую способность относительно неисправностей взаимного влияния [303].

Следует отметить, что тест STAMarch *X*_1 не обнаруживал неисправности 2 и 9 из-за того, что они проявлялись и трансформировались в ошибку в первой фазе теста, а затем эта ошибка присутствовала в памяти до конца выполнения теста. Поэтому дополнительная фаза с операцией чтения, поставленная в конец теста, приводила к тому, что ошибка, являющаяся проявлением неисправностей 2 и 9, присутствовала во всех фазах теста и, соответственно, являлась симметричной. В то же время в тесте STAMarch *X*_2 дополнительная фаза с операцией чтения, находящаяся в начале теста, не может являться причиной проявления неисправности. Следовательно, в первой фазе ошибка, являющаяся проявлением неисправностей 2 и 9, отсутствовала, что привело к возникновению несимметричности конфигурации ошибки и, соответственно, к ее обнаружению.

Этот факт указывает на необходимость использования дополнительной фазы с операцией чтения перед началом теста, а не в его конце. Это действи-

тельно справедливо для коротких маршевых тестов (таких, как: MATS, MATS +, March *X* и др.), в которых некоторые неисправности взаимного влияния проявляются и отражаются в ошибку только в первой фазе. Однако для более длинных тестов, в которых неисправности взаимного влияния проявляются неоднократно, ситуация будет не такая однозначная.

Таблица 12.6 – *Эффективность теста STAMarch X_2*

№	$\Downarrow(r\bar{a})$	$\Uparrow(ra, w\bar{a})$	$\Downarrow(r\bar{a}, wa)$	$\Uparrow(ra)$
1	-	-	-	-
2	-	+	+	+
3	-	-	-	-
4	-	-	-	+
5	-	-	+	+
6	-	-	-	-
7	-	-	+	+
8	-	-	-	-
9	-	+	+	+
10	-	-	-	+
11	-	-	+	+
12	-	-	+	+

Для подтверждения описанной в предыдущих разделах методики оценки, покрывающей способности симметричного неразрушающего тестирования, было проведено экспериментальное моделирование работы ЗУ с внесением различных неисправностей. Для эксперимента использовались следующие исходные данные [312, 313, 322]:

1. Размер ЗУ – 32 *Кбит*.
2. Разрядность сигнатурного анализатора – 15 *бит*.
3. Моделируемые неисправности: одночные *SAF*, *TF*, *CFid* и *CFin*.
4. Используемые маршевые тесты: STAMarch *X*_1, STAMarch *X*_2, STAMarch *C*–, STAMarch *C*_1, STAMarch *C*_2 и соответствующие им классические версии неразрушающих тестов TMarch *X*, TMarch *C*–, TMarch *C*,
5. Число экспериментов – 10000.

Результаты экспериментов приведены в таблицах 12.7, 12.8 и 12.9.

Таблица 12.7 – *Покрывающая способность различных версий теста March X*

Неисправность	March *X*	STAMarch *X*_1	STAMarch *X*_2
SAF	100 %	100 %	100 %
TF	100 %	100 %	100 %
CFid	49.993 %	37.492 %	49.996 %
CFin	99.992 %	74.992 %	99.997 %

Таблица 12.8 – *Покрывающая способность различных версий теста March C*

Неисправность	March *C*	STAMarch *C*_1	STAMarch *C*_2
SAF	99.992 %	100 %	100 %
TF	99.991 %	100 %	100 %
CFid	99.995 %	100 %	100 %
CFin	99.996 %	100 %	100 %

Таблица 12.9 – *Покрывающая способность различных версий теста March C–*

Неисправность	March *C–*	STAMarch *C–*
SAF	99.992 %	100 %
TF	99.991 %	100 %
CFid	99.996 %	100 %
CFin	99.997 %	100 %

Как видно из приведенных таблиц, симметричные неразрушающие тесты обеспечивают более высокую покрывающую способность, по сравнению с классическими неразрушающими тестами. Отметим, что длина сжимаемых последовательностей и разрядность используемых сигнатурных анализаторов одинакова для обоих видов тестов.

Более высокая покрывающая способность симметричных неразрушающих тестов связана с уменьшением кратности конфигурации ошибки, в которую отражается каждая конкретная неисправность. Так, для теста March *C* и неисправности взаимного влияния $\wedge(\uparrow, 1)$ при использовании классического неразрушающего теста происходит отображение данной неисправности в четырехкратную ошибку. Вероятность обнаружения ошибки такой кратности классическим сигнатурным анализатором ниже 100 % и для рассмотренного случая равняется $(1 - 1/2^{15})\times 100$ %. В то же время для симметричного тестирования кратность конечной ошибки будет меньше. И для рассмотренного примера это значение равняется 1. Как известно, вероятность обнаружения однократных ошибок сигнатурным анализатором равна 100 %. Симметричные неразрушающие тесты обеспечивают более высокую покрывающую способность, что особенно заметно для неисправностей небольшой кратности. С увеличением кратности неисправностей покрывающая способность симметричных тестов будет приближаться к значениям, сравнимым с классической технологией неразрушающего тестирования.

12.5. Диагностические симметричные неразрушающие тесты

Рассмотрим процедуры диагностирования неисправностей ЗУ, которые построены на использовании преимуществ симметричных неразрушающих тестов. Для этих целей используем симметрию *Type* 2, а в качестве объекта диагностирования ЗУ с $N = 2^k$ ячейками. Обозначим как $S_{test}(n, l)$ сигнатуру, полученную с использованием STA тестов для блока памяти ЗУ, состоящего

из n ячеек с начальным адресом l. Отметим, что $S_{test}(n, l)$ представляет собой функцию, не зависящую от значений n и l. Ее величина в случае отсутствия неисправностей в блоке ЗУ всегда ровняется нулю, то есть $S_{test}(n, l) = S_{ref} =$ 0 0 0 … 0 в соответствии со свойством 12.5 симметричных неразрушающих тестов. В результате выполнения симметричного теста STA генерируется один из двух возможных результатов $S_{test}(n, l)\_PASS$ в случае выполнения равенства $S_{test}(n, l) = 000…0$ и $S_{test}(n, l)\_FAIL$ в противном случае [303].

Для обнаружения простейших неисправностей, включающих переходные и константные неисправности, может быть использован диагностический тест *Divide and Test*, представленный ниже [303]. Исходной информацией для его выполнения является получение отрицательного результата при тестировании ЗУ. Симметричный неразрушающий тест выработал сигнал $S_{test}(N, 0)\_FAIL$.

Тест *Divide and Test*

Входные данные: объем памяти N бит, Тест STA.

Выходные данные: адреса неисправных ячеек ЗУ.

Begin

1. Задается размер первого блока ЗУ $n = N/2$ представляющий собой половину объема исходного ЗУ и нулевой начальный адрес блока $l = 0$.

2. Выполняется тест STA, которым может быть любой симметричный неразрушающий маршевый тест. По результатам выполнения теста формируется значение $S_{test}(n, l)$ и проверяется выполнение равенства $S_{test}(n, l) = S_{ref} =$ 0 0 0 … 0.

Если равенство $S_{test}(n, l) =$ 0 0 0 … 0 не выполняется, формируется результат $S_{test}(n, l)\_FAIL$. Если $n > 1$, определяется новое значение $n = n/2$ и повторяется выполнение текущего этапа 2. Для случая, когда $n = 1$ переходят к выполнению этапа 4.

В случае выполнения равенства $S_{test}(n, l) =$ 0 0 0 … 0, формируется сигнал $S_{test}(n, l)\ \_PASS$ и для $n > 1$ определяется новое значение $n = n/2$, задается начальный адрес $l = l + n$, затем повторяется выполнение текущего этапа 2. Для случая, когда $n = 1$, задается $l = l + 1$ и переходят к выполнению этапа 3.

3. Реализуется тест STA для блока ЗУ $(1, l)$. Если $S_{test}(n, l)\_PASS$ формируется сигнал *Diagnoses Unsuccessful* и переходят к *End*. Если $S_{test}(n, l)\_FAIL$ переходят к выполнению этапа 4.

4. Неисправная ячейка находится по адресу l. Полученный адрес является выходной информацией данного алгоритма, которая используется алгоритмом саморемонтирования ЗУ.

5. Неразрушающий тест STA выполняется для всего ЗУ. Если $S_{test}(N,0)\_FAIL$, выполняется переход на начало *Begin* диагностической процедуры. В противном случае на ее конец (*End*).

End.

В качестве примера применения теста *Divide and Test* рассмотрим запоминающее устройство с $N = 8$ ячейками, содержащее три константные неисправности в ячейках с адресами 3, 4 и 7, как показано на рис. 12.10.

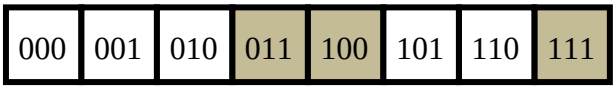


Рисунок 12.10 – *Модель запоминающего устройства, содержащего неисправности*

По результатам тестирования данного ЗУ с использованием STA тестов будет получен отрицательный результат в виде сигнала $S_{test}(8, 0)\_FAIL$. Наличие подобного сигнала инициирует выполнение теста *Divide and Test.* Будем считать, что после локализации неисправной ячейки ЗУ включается механизм саморемонтирования ЗУ и неисправность устраняется.

Пример 12.4. Используем тест *Divide and Test* для поиска неисправностей ЗУ, представленного на рис. 12.10.

Входные данные: объем памяти $N = 8$, Тест STA.

Выходные данные: адреса неисправных ячеек ЗУ.

Begin

1.1. $n = 4$, $l = 0$.

2.1. $S_{test}(4, 0)\_FAIL$ и $n = 4 > 1$; тогда $n = n/2 = 2$ и выполняется этап 2.

2.2. $S_{test}(2, 0)\_PASS$ и $n = 2 > 1$; $l = l + n = 0 + 2 = 2$, $n = n/2 = 1$ и опять выполняется этап 2.

2.3. $S_{test}(1, 2)\_PASS$ и $n = 1$; $l = l + 1 = 2 + 1 = 3$, и выполняется переход к этапу 3.

3.1. $S_{test}(1, 3)\_FAIL$ и переходим к 4.

4.1. Неисправная ячейка имеет адрес $l = 3 = 011$. Включается механизм ее ремонта и неисправность устраняется.

5.1. $S_{test}(8, 0)\_FAIL$, так как в ЗУ еще присутствуют две неисправности, осуществляется переход на начало процедуры *Divide and Test*.

1.1. $n = 4$, $l = 0$.

2.1. $S_{test}(4, 0)\_PASS$ и $n = 4 > 1$; тогда $n = n/2 = 2$; $l = l + n = 0 + 4 = 4$ и выполняется этап 2.

2.2. $S_{test}(2, 4)\_FAIL$ и $n = 2 > 1$; $n = n/2 = 1$ и выполняется этап 2.

2.3. $S_{test}(1, 4)\_FAIL$ и $n = 1$; и переходим к 4.

4.1. Неисправная ячейка имеет адрес $l = 4 = 100$. Включается механизм ее ремонта и неисправность устраняется.

5.1. $S_{test}(8, 0)\_FAIL$, так как в ЗУ еще присутствуют одна неисправность осуществляется переход на начало процедуры *Divide and Test*.

1.1. $n = 4$, $l = 0$.

2.1. $S_{test}(4, 0)\_PASS$ и $n = 4 > 1$; $l = l + n = 0 + 4 = 4$, $n = n/2 = 2$ и выполняется этап 2.

2.2. $S_{test}(2, 4)\_PASS$ и $n = 2 > 1$; $l = l + n = 4 + 2 = 6$, $n = n/2 = 1$ и выполняется этап 2.

2.3. $S_{test}(1, 6)\_PASS$ и $n = 1$; $l = l + 1 = 6 + 1 = 7$, и выполняется переход к этапу 3.

3.1. $S_{test}(1, 7)\_FAIL$ и переходим к 4.

4.1. Неисправная ячейка имеет адрес $l = 7 = 111$. Включается механизм ее ремонта и неисправность устраняется.

5.1. $S_{test}(8, 0)\_PASS$, и переходят к *End*.

End

Сложность $O(C_{D\&T})$ диагностического теста *Divide and Test* (*D*&*T*) для локализации неисправной ячейки ЗУ (одиночной константной неисправности), когда во внимание принимаются только операции обращения к ЗУ, оценивается как:

$$O(D\&T) = O(STA(N)), \qquad (12.6)$$

где $O(STA(N))$ – сложность реализации STA теста для ЗУ с N запоминающими ячейками.

Выбор емкости ЗУ как величина, равная 2^k, позволяет существенно уменьшить аппаратурную сложность реализации алгоритма диагностирования ЗУ. В этом случае для выполнения операции деления на два достаточно реализовать микрооперацию сдвига на один бит. Операция сложения может быть осуществлена с использованием логической операции ИЛИ в силу того, что размер n блока ЗУ всегда выражается как 2^k.

Учитывая замечания, приведенные выше, дополнительные аппаратурные затраты на реализацию теста *Divide and Test* будут включать.

1. Регистр для хранения размера n блока ЗУ. Разрядность регистра равняется $k = \log_2 N$. Данный регистр должен выполнять две микрооперации, микрооперацию сдвига на один разряд и микрооперацию хранения кода размера блока.

2. Статический регистр для хранения начального адреса блока ЗУ. Разрядность регистра равняется k.

3. k двухвходовых элементов ИЛИ.

4. Устройство управления для реализации диагностического теста *Divide and Test.*

Отметим невысокую эффективность теста *Divide and Test* для обнаружения более сложных неисправностей, например, таких как: неисправности взаимного влияния. Проиллюстрируем это утверждение на примере.

Пример 12.5. Используем тест *Divide and Test* для поиска неисправности взаимного влияния $\langle\uparrow, a_j\rangle$, когда переход из нуля в единицу в ячейке с адресом 011 инвертирует содержимое ячейки 100 (см. рис. 12.11).

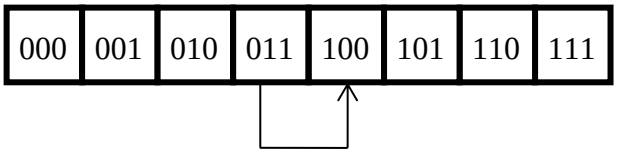


Рисунок 12.11 – *Модель запоминающего устройства, содержащего неисправность взаимного влияния*

Используем тест *Divide and Test* для поиска неисправностей ЗУ представленного на рис.12.11. В результате тестирования получим $S_{test}(8, 0)\_FAIL$.

Входные данные: объем памяти $N = 8$, Тест STA.

Выходные данные: адреса неисправных ячеек ЗУ.

Begin

1.1. $n = 4$, $l = 0$.

2.1. $S_{test}(4, 0)\_PASS$ и $n = 4 > 1$; $l = l + n = 0 + 4 = 4$, $n = n/2 = 2$ выполняется этап 2.

2.2. $S_{test}(2, 4)\_PASS$ и $n = 2 > 1$; $l = l + n = 4 + 2 = 6$, $n = n/2 = 1$ выполняется этап 2.

2.3. $S_{test}(1, 6)\_PASS$ и $n = 1$; $l = l + 1 = 6 + 1 = 7$, переход к этапу 3.

3.1. $S_{test}(1, 7)\_PASS$ формируется сигнал *Diagnoses Unsuccessful* и переходят к *End*.

End.

В результате неисправность взаимного влияния $\langle\uparrow, a_j\rangle$ приведенная на рис. 12.11, является необнаруживаемой тестом *Divide and Test*, что свидетельствует о невысокой его эффективности для сложных неисправностей. В первую очередь для неисправностей взаимного влияния и кодочувствительных неисправностей.

Для целей диагностирования сложных неисправностей, в частности неисправностей взаимного влияния, может быть использован диагностический тест *Divide and Check.*

Тест *Divide and Check*

Тест *Divide and Check* является диагностическим тестом, позволяющим определять адреса неисправных ячеек в случае переходных и константных неисправностей, а также адреса пассивных ячеек в неисправностях взаимного влияния. Подобно, как и в случае теста *Divide and Test,* данный алгоритм использует свойство 12.5 симметричных неразрушающих тестов. Отличием является последовательность реализации симметричного неразрушающего теста STA и процедура формирования реального значения сигнатуры $S_{test}(n, l)$. Так STA тест для блока (n, l) ЗУ выполняется $\log_2 n$ раз, и в худшем случае $\log_2 N$ раз для всех ячеек блока ЗУ. В тоже время содержимое только определенной части ячеек ЗУ используется для формирования $S_{test}(w, r)$, где значения w и r определяют окно измерения количеством ячеек w блока ЗУ участвующих в определении $S_{test}(w, r)$ и начальным адресом r окна.

Непосредственно тест *Divide and Check* состоит из следующих этапов.

Входные данные: размер n блока ЗУ. Начальный адрес l блока ЗУ. Тест *STA.*

Выходные данные: адреса неисправных ячеек ЗУ.

Begin

1. Задается размер первого окна измерения как значение $w = n/2$, представляющее собой половину объема неисправного (n, l) блока ЗУ и начальный адрес окна $r = l$, равный начальному адресу неисправного (n, l) блока ЗУ.

2. Выполняется тест STA для (n, l) блока ЗУ. В процессе выполнения теста формируется сигнатура $S_{test}(w, r)$ как результат сжатия данных, формируемых в заданном окне измерения (w, r). По окончании выполнения теста проверяется выполнение равенства $S_{test}(w, r) = S_{ref} = 0\ 0\ 0\ \ldots\ 0$.

Если равенство $S_{test}(w, r) = 0\ 0\ 0\ \ldots\ 0$ не выполняется, формируется сигнал $S_{test}(w, r)\_FAIL$ и если $w > 1$, определяется новое значение размера окна $w = w/2$ и повторяется выполнение текущего этапа 2. Для случая, когда $w = 1$, переходят к выполнению этапа 3.

В случае выполнения равенства $S_{test}(w, r) = 0\ 0\ 0\ \ldots\ 0$, формируется сигнал $S_{test}(w, r)\_PASS$ и для $w > 1$ определяется новое значение для размера окна $w = w/2$ и задается новый начальный адрес окна измерения $r = r + w$ затем повторяется выполнение текущего этапа 2. Для случая, когда $w = 1$, задается $r = r + 1$ и переходят к выполнению этапа 3.

3. Неисправная ячейка находится по адресу r. Полученный адрес является выходной информацией данного алгоритма, которая используется алгоритмом саморемонтирования ЗУ.

4. Неразрушающий тест STA выполняется для всего ЗУ. Если $S_{test}(N, 0)\_FAIL$ выполняется переход на начало *Begin* диагностической процедуры. В противном случае на ее конец (*End*).

End.

Для оценки приведенного теста *Divide and Check* рассмотрим случай неисправности взаимного влияния, рассмотренной в примере 12.5.

Пример 12.6. Используем тест *Divide and Check* для поиска неисправностей в ЗУ, представленном на рис. 12.11.

Входные данные: объем исходного блока равняется объему ЗУ $n = N = 8$. Начальный адрес $l = 0$. Тест *STA* представляет собой произвольный симметричный тест ЗУ, который использовался для тестирования ЗУ, и на основании его было получено значение $S_{test}(N, 0)\_FAIL$. А затем при реализации *Divide and Test* получен сигнал *Diagnoses Unsuccessful.*

Выходные данные: адреса неисправных ячеек ЗУ.

Begin

1.1. $w = 4$, $r = 0$.

2.1. Реализуется тест для блока (8, 0), а сигнатура формируется путем сжатия данных, полученных в окне измерения (4, 0). $S_{test}(4, 0)\_PASS$ и $w = 4 > 1$; тогда $w = w/2 = 2$; $r = r + 4 = 0 + 4 = 4$ и выполняется этап 2.

2.2. Реализуется тест для блока (8, 0), а сигнатура формируется путем сжатия данных, полученных в окне измерения (2, 4). $S_{test}(2, 4)\_FAIL$ и $w = 2 > 1$; $w = w/2 = 1$ и выполняется этап 2.

2.3. Реализуется тест для блока (8, 0), а сигнатура формируется путем сжатия данных полученных в окне измерения (1, 4). $S_{test}(1, 4)\_FAIL$, $w = 1$ и выполняется переход к этапу 3.

3.1. Неисправная ячейка имеет адрес $l = 4 = 100$. Включается механизм ее ремонта и неисправность устраняется. Выполняется переход на завершение процедуры *Divide and Check.*

End.

Временная сложность реализации данного теста $C_{D\&C}$ в сильной мере зависит от результатов предварительного использования теста *Divide and Test* (*D*&*C*), который не локализовал местонахождение неисправной ячейки ЗУ и может быть оценена выражением для размерности блока памяти $n < N$.

$$O(D\&C) = (\log_2 n)O(STA(n)). \qquad (12.7)$$

Для (12.7) выполняется неравенство $O(STA(2)) < O(D\&C) < (\log_2 N)O(STA(N))$.

Рассмотренный тест *Divide and Check* может быть эффективно использован для локализации ячеек ЗУ с константными и переходными неисправностями. Кроме того, данный тест локализует местоположение пассивной ячейки (жертвы) неисправностей взаимного влияния. Основным недостатком данного алгоритма является его временная сложность, которая для реальных значений размеров ЗУ принимает большие значения. Для уменьшения влияния данного недостатка можно применить тест *Search Area*, основная идея которого заключается в определения исходного (n, l) блока ЗУ, который содержит неисправность для минимально возможного значения n. Сигнал *Diagnoses Unsuccessful*, полученный по результатам теста *Divide and Test*, является сигналом запуска теста *Search Area*.

Тест *Search Area*

Входные данные: размер ЗУ N. Тест STA.

Выходные данные: неисправный (n, l) блок ЗУ, с его начальным адресом l и объемом n.

Begin

1. $n = N/2$, $l = 0$.

2. Реализуется тест для блока (n, l) ЗУ и блока (n, $l + 1$). Если $S_{test}(n, l)\_PASS$ и $S_{test}(n, l + 1)\_PASS$, тогда $n = 2n$ и переходят к выполнению этапа 3. Если $S_{test}(n, l)\_FAIL$ и $S_{test}(n, l + 1)\_PASS$, тогда $n = n/2$ и переходят к повторному выполнению этапа 2. Если $S_{test}(n, l)\_PASS$ и $S_{test}(n, l + 1)\_FAIL$, тогда $n = n/2$; $l = l + n$ и переходят к выполнению этапа 2.

3. Формируется сигнал *Search Area is* (n, l), где n есть размер неисправного блока ЗУ, а l – его начальный адрес.

End.

Пример 12.7. Используем тест *Search Area* для определения неисправного (*n, l*) блока ЗУ для примера, представленного на рис. 12.11.

Входные данные: объем памяти $N = 8$, Тест *STA.*

Выходные данные: неисправный (*n, l*) блок ЗУ. Его начальный адрес *l* и объем *n.*

Begin

1.1 $n = N/2$, $l = 0$.

2.1. Реализуется тест для блока (4, 0) и блока (4, 4), получим, что $S_{test}(4, 0)\_PASS$ и $S_{test}(4, 4)\_PASS$, тогда $n = 2n = 8$ и выполняется этап 3.

3.1. Формируется сигнал *Search Area is* (8, 0).

End.

Временная сложность $O(SA)$ теста *Search Area* (*SA*), выраженная количеством операций обращения к ЗУ, оценивается выражением:

$$O(STA(N)) < O(SA) < 2O(STA(N)). \qquad (12.8)$$

Здесь $O(STA(N))$ есть временная сложность симметричного неразрушающего теста.

Рассмотренные тесты ЗУ, использующие симметрию маршевых тестов ЗУ, показывают высокую эффективность как по их временной сложности, так и по аппаратурным затратам на их реализацию.

Глава 13. Многократные маршевые тесты ЗУ

13.1. Анализ эффективности маршевых тестов

При дальнейшем изложении в качестве анализируемого ЗУ будет рассматриваться запоминающее устройство, состоящее из N однобитных запоминающих ячеек. Для тестирования подобных устройств, как правило, используются только маршевые тесты, как единственно возможное решение для тестирования современных ЗУ. В качестве неисправностей, которые должны быть выявлены тестами, в дальнейшем будем рассматривать модель кодочувствительных неисправностей *PSFk* или ее разновидности, определяемые, как: *NPSFk*, которые эффективно покрывают более простые модели неисправностей ЗУ. Наиболее адекватной моделью для данного анализа, как показано в ряде источников [73, 331, 333], являются *PNPSFk*, количество которых для произвольных k ячеек ЗУ и фиксированной базовой ячейки равняется $2\times2^{k-1} = 2^k$, а общее их число определяется выражением (1.2).

Все множество маршевых тестов можно представить в виде трех подмножеств, условно разделенных на тесты трех классов [234]. В качестве классификационного критерия используем количество двоичных комбинаций в $k - 1$ соседних запоминающих элементах (ЗЭ) при обращении к базовому ЗЭ для k произвольных фиксированных ЗЭ [331]. Под обращением к базовому ЗЭ понимается выполнение операции записи инверсного значения, результат выполнения которой может быть проверен в текущей или последующей фазе теста.

Проанализируем неразрушающий маршевый тест MATS+ для $k = 5$. Предположим, что фиксированными ячейками ОЗУ являются ячейки $b_\alpha\, b_\beta\, b_\chi\, b_\delta\, b_\varepsilon$ где $b_\gamma \in \{0, 1\}$, $\gamma \in \{\alpha, \beta, \chi, \delta, \varepsilon\}$, представляет текущее состояние ЗЭ, а их адреса $\alpha, \beta, \chi, \delta, \varepsilon$ расположены в возрастающей последовательности ($\alpha < \beta < \chi < \delta < \varepsilon$). Предположим, что ячейка b_χ является базовой ячейкой, а ячейки $b_\alpha\, b_\beta\, b_\delta\, b_\varepsilon$ – соседними ячейками, тогда для указанных ЗЭ существует $2^k = 2^5 = 32$ кодочувствительные неисправности *PNPSF5*, которые различаются $2^{k-1} = 2^4 = 16$ двоичными комбинациями, записанными в соседних ячейках. Для текущего состояния $b_\alpha b_\beta b_\chi b_\delta b_\varepsilon$ = 1 0 1 0 1 в указанных ячейках при реализации неразрушающего теста MATS+ $\{\Uparrow\Downarrow (wb);\ \Uparrow(rb,\ w\bar{b});\ \Downarrow(r\bar{b}, wb)\}$ получим следующие состояния ЗЭ (см. таблицу 13.1) [333]:

Активные ячейки, для которых выполняются операции чтения и записи, выделены жирным шрифтом. Как видно из приведенной таблицы, тестом MATS+ активизируется и обнаруживается только одна из 32 возможных *PNPSF*5 неисправностей, а именно 01↓01. Неисправность 01↑01 не будет обнаружена тестом MATS+, так как после фазы активизации базовой ячейки b_χ в указанном тесте отсутствует операция чтения ее состояния.

Рассмотрим случай произвольного состояния ЗУ $B = b_0\, b_1\, b_2\, \ldots\, b_{N-2}\, b_{N-1}$ и произвольной *PNPSFk*, которая включает запоминающие элементы $b_{\gamma(j)}$, $j \in \{0, 1, \ldots, k-1\}$ с адресами, представленными в возрастающей последова-

тельности $\gamma(0) < \gamma(1) < \gamma(2) < \ldots < \gamma(k-1)$, а базовая ячейка имеет адрес $\gamma(i)$, где $0 \le i \le k-1$.

Таблица 13.1 – *Состояния ячеек ОЗУ при реализации теста MATS+*

Фаза теста MATS+	Содержимое ячеек					Фаза теста MATS+	Содержимое ячеек				
	b_α	b_β	b_χ	b_δ	b_ε		b_α	b_β	b_χ	b_δ	b_ε
	1	0	1	0	1		0	1	0	1	0
$\Uparrow(rb, w\bar{b})$	**0**	0	1	0	1	$\Downarrow(r\bar{b}, wb)$	0	1	0	1	**1**
	0	**1**	1	0	1		0	1	0	**0**	1
	0	1	**0**	0	1		0	1	**1**	0	1
	0	1	0	**1**	1		0	**0**	1	0	1
	0	1	0	1	**0**		**1**	0	1	0	1

Анализируемый тест MATS+ будет формировать только одну комбинацию в соседних ячейках. Эта комбинация $\bar{b}_{\gamma(0)}\,\bar{b}_{\gamma(1)}\,\bar{b}_{\gamma(2)}\ldots\,\bar{b}_{\gamma(i-1)}\,\bar{b}_{\gamma(i)}\,b_{\gamma(i+1)}\ldots\,b_{\gamma(k-2)}\,b_{\gamma(k-1)}$ зависит от исходного состояния $B = b_0\,b_1\,b_2\,\ldots\,b_{N-2}\,b_{N-1}$. Таким образом, MATS+ подобные тесты позволяют обнаруживать только одну из 2^k возможных *PNPSFk*, тогда полнота покрытия подобных тестов, вычисляемая как процентное отношение обнаруживаемых неисправностей к их общему числу [351]:

$$FC_{MATS+}(PNPSFk) = \frac{1}{2^k}100\%. \qquad (13.1)$$

Увеличение покрывающей способности маршевых тестов возможно за счет обнаружения двух кодочувствительных неисправностей, зависящих от состояния базовой ячейки. Полнота покрытия, например, теста MATS++ $\{\Uparrow(rb, w\bar{b});\ \Downarrow(r\bar{b}, wb, rb)\}$, в сравнении с тестом MATS+, будет в два раза больше, так как будут обнаружены обе *PNPSFk* неисправности, а именно: $\bar{b}_{\gamma(0)}\,\bar{b}_{\gamma(1)}\,\bar{b}_{\gamma(2)}\ldots\,\bar{b}_{\gamma(i-1)}\uparrow_{\gamma(i)}\,b_{\gamma(i+1)}\ldots\,b_{\gamma(k-2)}\,b_{\gamma(k-1)}$ и $\bar{b}_{\gamma(0)}\,\bar{b}_{\gamma(1)}\,\bar{b}_{\gamma(2)}\ldots\,\bar{b}_{\gamma(i-1)}\downarrow_{\gamma(i)}\,b_{\gamma(i+1)}\ldots\,b_{\gamma(k-2)}\,b_{\gamma(k-1)}$.

В случае теста March *C*– $\{\Uparrow\Downarrow(wb);\ \Uparrow(rb, w\bar{b});\ \Uparrow(r\bar{b}, wb);\ \Downarrow(rb, w\bar{b});\ \Downarrow(r\bar{b}, wb);\ \Uparrow\Downarrow(rb)\}$ обнаруживаются четыре типа *PNPSFk*

$$\begin{aligned}
&\bar{b}_{\gamma(0)}\,\bar{b}_{\gamma(1)}\,\bar{b}_{\gamma(2)}\ldots\,\bar{b}_{\gamma(i-1)} \quad \uparrow_{\gamma(i)} \quad b_{\gamma(i+1)}\ \ldots\,b_{\gamma(k-2)}\ \ b_{\gamma(k-1)},\\
&b_{\gamma(0)}\ b_{\gamma(1)}\ b_{\gamma(2)}\ \ldots\,b_{\gamma(i-1)} \quad \uparrow_{\gamma(i)} \quad \bar{b}_{\gamma(i+1)}\ldots\,\bar{b}_{\gamma(k-2)}\,\bar{b}_{\gamma(k-1)},\\
&b_{\gamma(0)}\ b_{\gamma(1)}\ b_{\gamma(2)}\ \ldots\,b_{\gamma(i-1)} \quad \downarrow_{\gamma(i)} \quad \bar{b}_{\gamma(i+1)}\ldots\,\bar{b}_{\gamma(k-2)}\,\bar{b}_{\gamma(k-1)},\\
&\bar{b}_{\gamma(0)}\,\bar{b}_{\gamma(1)}\,\bar{b}_{\gamma(2)}\ldots\,\bar{b}_{\gamma(i-1)} \quad \downarrow_{\gamma(i)} \quad b_{\gamma(i+1)}\ \ldots\,b_{\gamma(k-2)}\ \ b_{\gamma(k-1)}.
\end{aligned}$$

Это достигается за счет генерирования двух комбинаций в соседних ячейках и проверки выполнения различных переходов базового ЗЭ [351]. Тогда полнота покрытия будет вычисляться как:

$$FC_{MarchC-}(PNPSFk) = \frac{1}{2^{k-2}}100\%. \qquad (13.2)$$

Существенно большее значение покрывающей способности может быть получено на основании теста PS(23*N*) [351] и с помощью тестов, представленных в [26, 37, 225, 232], для которых обнаруживаемыми являются все типы *PNPSFk:*

$$\begin{array}{lll}
\bar{b}_{\gamma(0)}\ \bar{b}_{\gamma(1)}\ \bar{b}_{\gamma(2)} \ldots\ \bar{b}_{\gamma(i-1)} & \uparrow_{\gamma(i)} & b_{\gamma(i+1)} \ldots b_{\gamma(k-2)}\ b_{\gamma(k-1)}, \\
\bar{b}_{\gamma(0)}\ \bar{b}_{\gamma(1)}\ \bar{b}_{\gamma(2)} \ldots\ \bar{b}_{\gamma(i-1)} & \downarrow_{\gamma(i)} & b_{\gamma(i+1)} \ldots b_{\gamma(k-2)}\ b_{\gamma(k-1)}, \\
b_{\gamma(0)}\ b_{\gamma(1)}\ b_{\gamma(2)} \ldots b_{\gamma(i-1)} & \uparrow_{\gamma(i)} & \bar{b}_{\gamma(i+1)} \ldots\ \bar{b}_{\gamma(k-2)}\ \bar{b}_{\gamma(k-1)}, \\
b_{\gamma(0)}\ b_{\gamma(1)}\ b_{\gamma(2)} \ldots b_{\gamma(i-1)} & \downarrow_{\gamma(i)} & \bar{b}_{\gamma(i+1)} \ldots\ \bar{b}_{\gamma(k-2)}\ \bar{b}_{\gamma(k-1)}, \\
\bar{b}_{\gamma(0)}\ \bar{b}_{\gamma(1)}\ \bar{b}_{\gamma(2)} \ldots\ \bar{b}_{\gamma(i-1)} & \uparrow_{\gamma(i)} & \bar{b}_{\gamma(i+1)} \ldots\ \bar{b}_{\gamma(k-2)}\ \bar{b}_{\gamma(k-1)}, \\
\bar{b}_{\gamma(0)}\ \bar{b}_{\gamma(1)}\ \bar{b}_{\gamma(2)} \ldots\ \bar{b}_{\gamma(i-1)} & \downarrow_{\gamma(i)} & \bar{b}_{\gamma(i+1)} \ldots\ \bar{b}_{\gamma(k-2)}\ \bar{b}_{\gamma(k-1)}, \\
b_{\gamma(0)}\ b_{\gamma(1)}\ b_{\gamma(2)} \ldots b_{\gamma(i-1)} & \uparrow_{\gamma(i)} & b_{\gamma(i+1)} \ldots b_{\gamma(k-2)}\ \ b_{\gamma(k-1)}, \\
b_{\gamma(0)}\ b_{\gamma(1)}\ b_{\gamma(2)} \ldots b_{\gamma(i-1)} & \downarrow_{\gamma(i)} & b_{\gamma(i+1)} \ldots b_{\gamma(k-2)}\ \ b_{\gamma(k-1)}.
\end{array}$$

Для нулевого начального состояния ЗУ $B = b_0\, b_1\, b_2 \ldots b_{N-2}\, b_{N-1}$ = 0 0 0 … 0 0 все множество обнаруживаемых неисправностей *PNPSF*5 приведено в таблице 13.2.

Анализ приведенных *PNPSFk* и *PNPSF*5 позволяет сделать вывод о том, что максимальное обнаруживаемое количество таких неисправностей равняется 8*k* – 8. Это следует из того факта, что различные типы *PNPSFk*, например #1 и #5, включают одинаковые неисправности, в данном случае ↑ 0 0 0 … 0 0. Множество повторяющихся неисправностей *PNPSFk* включает: ↑ 0 0 0 … 0 0, ↓ 0 0 0 … 0 0, 0 0 0 … 0 0 ↑, 0 0 0 … 0 0 ↓, ↓ 1 1 1 … 1 1, ↑ 1 1 1 … 1 1, 1 1 1 … 1 1 ↑, 1 1 1 … 1 1 ↓ [232].

Таким образом, полнота покрытия теста PS(23*N*) является максимально достижимой полнотой покрытия для случая маршевых тестов.

Таблица 13.2 – *Неисправности PNPSF*5

Типы *PNPSF*5	*PNPSF*5
#1	↑ 0 0 0 0, 1 ↑ 0 0 0, 1 1 ↑ 0 0, 1 1 1 ↑ 0, 1 1 1 1 ↑
#2	↓ 1 1 1 1, 0 ↓ 1 1 1, 0 0 ↓ 1 1, 0 0 0 ↓ 1, 0 0 0 0 ↓
#3	0 0 0 0 ↑, 0 0 0 ↑ 1, 0 0 ↑ 1 1, 0 ↑ 1 1 1, ↑ 1 1 1 1
#4	1 1 1 1 ↓, 1 1 1 ↓ 0, 1 1 ↓ 0 0, 1 ↓ 0 0 0, ↓ 0 0 0 0
#5	↑ 0 0 0 0, 0 ↑ 0 0 0, 0 0 ↑ 0 0, 0 0 0 ↑ 0, 0 0 0 0 ↑
#6	↓ 0 0 0 0, 0 ↓ 0 0 0, 0 0 ↓ 0 0, 0 0 0 ↓ 0, 0 0 0 0 ↓
#7	↑ 1 1 1 1, 1 ↑ 1 1 1, 1 1 ↑ 1 1, 1 1 1 ↑ 1, 1 1 1 1 ↑
#8	↓ 1 1 1 1, 1 ↓ 1 1 1, 1 1 ↓ 1 1, 1 1 1 ↓ 1, 1 1 1 1 ↓

Для различных значений *k* полнота покрытия PS(23*N*) вычисляется как:

$$FC_{PS(23N)}(PNPSFk) = FC_{MAX} = \frac{8k-8}{k2^k}100\% = \frac{k-1}{k2^{k-3}}100\%. \qquad (13.3)$$

Следует отметить, что вне зависимости от начального состояния ЗУ и адресной последовательности одноразовое применение маршевого теста произвольной структуры и сложности в силу последовательного обращения к ячейкам памяти не может обеспечить большей полноты покрытия, чем (13.3). Численные значения полноты покрытия для различных *PNPSFk* приведены в таблице 13.3.

Таблица 13.3 – *Полнота покрытия* (%) *PNPSFk*

Тест	*PNPSF*3	*PNPSF*5	*PNPSF*9
MATS	12,50	3,125	0,195
MATS++	25,00	6,250	0,390
March *C*–	50,00	12,50	0,781
March PS(23*N*)	66,66	20,00	1,388

Данные, приведенные в таблице, свидетельствуют о невысокой эффективности обнаружения *PNPSFk*, в особенности для больших значений *k*. Предельные величины полноты покрытия достигаются за счет большей сложности теста *Q* или ее приведенной величины $Q_{Test} = Q/N$. Для рассмотренных ранее тестов их приведенная временная сложность определяется как: $Q_{MATS} = 4$, $Q_{MATS++} = 6$, $Q_{March\ C-} = 10$ и $Q_{PS(23N)} = 23$.

В качестве более адекватной характеристики оценки эффективности обнаружения *PNPSFk* рассмотрим взвешенную полноту покрытия:

$$WFC_{Test}(PNPSFk) = \frac{FC_{Test}(PNPSFk)}{Q_{Test}}. \qquad (13.4)$$

Для тестов, рассмотренных ранее, значения данной характеристики приведены в таблице 13.4.

Таблица 13.4 – *Взвешенная полнота покрытия* (%) $WFC_{Test}(PNPSFk)$

Тест	*PNPSF*3	*PNPSF*5	*PNPSF*9
MATS	3,12	0,781	0,0487
MATS++	4,16	1,041	0,0650
March *C*–	5,00	1,250	0,0781
March PS(23*N*)	2,89	0,869	0,0603

Анализ данных, приведенных в двух предыдущих таблицах, позволяет сделать следующие выводы. Во-первых, дальнейшее усложнение маршевых

тестов с целью увеличения покрывающей способности *PNPSFk* оказывается невозможным, так как существующие тесты [27, 38] позволяют достичь максимально возможных значений полноты покрытия (13.3). Во-вторых, предельные величины полноты покрытия *PNPSFk* даже для малых *k* не достигают значений 100 %, а для больших *k* составляют доли процентов. В-третьих, относительная полнота покрытия, являющаяся наиболее адекватной метрикой, свидетельствует о практически одинаковой и невысокой эффективности любых маршевых тестов, начиная от простейших тестов до тестов, специально ориентированных на обнаружение *PNPSFk* (см. таблицу 13.4). Таким образом, для обнаружения кодочувствительных неисправностей *PNPSFk* необходимы новые подходы при реализации процедуры тестирования современных ЗУ.

13.2. Многократное тестирование ЗУ

Невысокая покрывающая способность *PNPSFk* неисправностей традиционными тестами объясняется последовательной процедурой доступа к ячейкам памяти. Как было показано в предыдущем разделе, в $k - 1$ соседних ячейках, участвующих в *PNPSFk*, в рамках последовательного доступа генерируется не более четырех двоичных комбинаций, что и определяет максимально возможную эффективность классических тестов. Очевидно, что для стандартного начального состояния ЗУ эти комбинации будут повторяться, а при различных начальных состояниях и повторяющихся процедурах тестирования их количество будет увеличиваться [349].

Идея многократного тестирования ЗУ изначально была сформулирована в рамках исчерпывающего и псевдо-исчерпывающего тестирования, основанного на использовании повторяющейся тестовой процедуры для различных предопределенных состояний ОЗУ [317]. Неразрушающее тестирование запоминающих устройств по своей сути также можно считать разновидностью многократного тестирования с произвольными состояниями ЗУ [106, 148, 149, 303, 316, 331].

Как отмечалось в ряде источников [149, 303, 331], неразрушающее тестирование теоретически позволяет обнаружить любые неисправные состояния ЗУ, что на практике требует многократного повторения теста. Если предположить, что однократное применение маршевого теста позволяет обнаруживать *PNPSFk* с полнотой покрытия $FC_{Test}(PNPSFk)$, тогда l-кратное его использование при произвольных (случайных) начальных состояниях ЗУ позволяет достичь полноты покрытия вычисляемой согласно выражению [331]:

$$FC_{Test}(PNPSFk, l) = \left(1 - \left(1 - \frac{FC_{Test}(PNPSFk)}{100\%}\right)^{l}\right)100\%. \qquad (13.5)$$

Очевидно, что максимальная полнота покрытия, равная 100 %, достижима только для $l \to \infty$, а реальные ее величины, близкие к 100 %, – для больших величин l, что следует из рисунка 13.1 для *PNPSF*3 и таблицы 13.5.

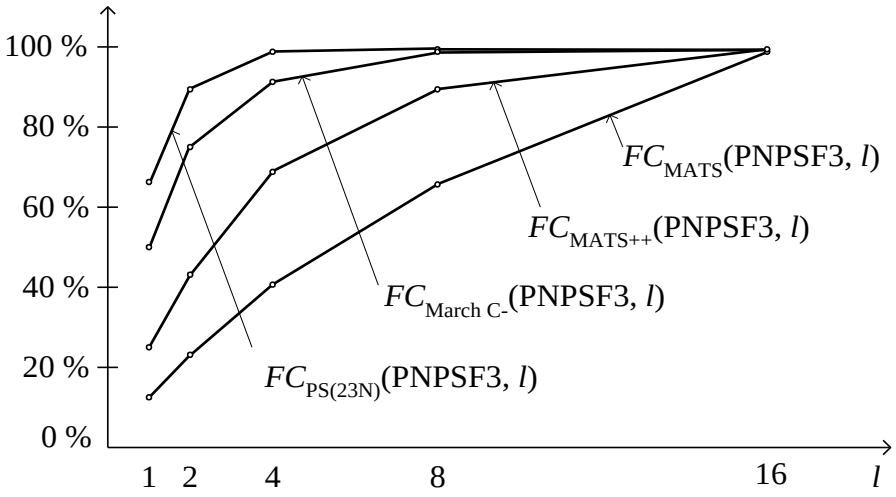


Рисунок 13.1 – *Полнота покрытия* (%) FC_{Test}(*PNPSF*3, *l*)

Как видно из представленных данных, необходимая полнота покрытия даже в случае простейших кодочувствительных неисправностей для $k = 3$ требует достаточного количества итераций маршевого теста. Так, полнота покрытия 99,9 % неисправностей *PNPSF*3 тестом MATS++ достигается после 25 итераций, а более сложный тест March *C*– требует 10 итераций. При больших значениях k достижение заданного значения полноты покрытия предполагает существенно большее количество итераций. Та же полнота покрытия 99,9 % неисправностей *PNPSF*5 и *PNPSF*9 тестом MATS++ достигается в результате 72 и 1176 итераций соответственно. Тест March *C*– для аналогичных неисправностей требует 35 и 588 итераций. Очевидно, что эффективность данного подхода в значительной мере зависит не только от количества итераций, но и от изменений содержимого ЗУ перед повторным применением маршевого теста [331].

Таблица 13.5 – *Количество l итераций маршевых тестов для получения заданной полноты покрытия* FC_{Test}(*PNPSF*3, *l*)

FC_{Test}(*PNPSF*3, *l*), %	50,0	90,0	95,0	99,0	99,9
MATS	6	18	23	45	52
MATS++	3	9	11	17	25
March *C*–	1	4	5	7	10
March PS(23*N*)	1	3	3	5	7

Вторым аргументом, который влияет на полноту покрытия *PNPSFk*, является адресная последовательность $A = A(0)\ A(1)\ A(2)\ \ldots\ A(N-1)$, $(A(j) \in \{0, 1, 2, \ldots, N-1\}, j \in \{0, 1, 2, \ldots, N-1\})$, которая применяется при тестировании для последовательного обращения к ЗЭ ЗУ [317,318,319]. Изменяя ал-

горитм формирования адресов, можно формировать новые, по сравнению с предыдущими итерациями многократного маршевого теста, двоичные комбинации в соседних ячейках *PNPSFk* [349]. Даже простейшее изменение начальных адресов A_0 для каждой последующей итерации позволяет заметно увеличить суммарную полноту покрытия кодочувствительных неисправностей [242].

13.3. Тестирование ЗУ с изменяемыми начальными адресами

Как показано в [151], количество обнаруживаемых неисправностей маршевыми тестами не зависит от последовательности адресов, используемых при реализации теста. С изменением адресной последовательности происходит только перераспределение между множеством обнаруживаемых и необнаруживаемых неисправностей. В качестве примера, иллюстрирующего данное утверждение, рассмотрим тестовую процедуру для тестов MATS+ и March *C*–, когда начальный адрес A_0 для второй итерации тестирования равен $A(1)$. Отметим, что для первой итерации $A_0 = A(0)$, а начальное состояние ЗУ $B = b_0\, b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7$ = 0 0 0 0 0 0 0 0 [349, 351]. Последовательные состояния ячеек ЗУ для данного случая приведены в таблице 13.6. Для March *C*– рассмотрены только две первые фазы теста.

Таблица 13.6 – *Состояния ячеек ЗУ для* A_0=$A(1)$

MATS+	Содержимое ОЗУ								March *C*–	Содержимое ОЗУ							
	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7		b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7
	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0
$\Uparrow(rb,w\bar{b})$	0	***1***	0	0	0	0	0	0	$\Uparrow(rb,w\bar{b})$	0	***1***	0	0	0	0	0	0
	0	1	***1***	0	0	0	0	0		0	1	***1***	0	0	0	0	0
	0	1	1	***1***	0	0	0	0		0	1	1	***1***	0	0	0	0
	0	1	1	1	***1***	0	0	0		0	1	1	1	***1***	0	0	0
	0	1	1	1	1	***1***	0	0		0	1	1	1	1	***1***	0	0
	0	1	1	1	1	1	***1***	0		0	1	1	1	1	1	***1***	0
	0	1	1	1	1	1	1	***1***		0	1	1	1	1	1	1	***1***
	1	1	1	1	1	1	1	1		***1***	1	1	1	1	1	1	1
$\Downarrow(r\bar{b},wb)$	***0***	1	1	1	1	1	1	1	$\Uparrow(r\bar{b},wb)$	1	***0***	1	1	1	1	1	1
	0	1	1	1	1	1	1	***0***		1	0	***0***	1	1	1	1	1
	0	1	1	1	1	1	***0***	0		1	0	0	***0***	1	1	1	1
	0	1	1	1	1	***0***	0	0		1	0	0	0	***0***	1	1	1
	0	1	1	1	***0***	0	0	0		1	0	0	0	0	***0***	1	1
	0	1	1	***0***	0	0	0	0		1	0	0	0	0	0	***0***	1
	0	1	***0***	0	0	0	0	0		1	0	0	0	0	0	0	***0***
	0	***0***	0	0	0	0	0	0		***0***	0	0	0	0	0	0	0

Для теста MATS+ получено одно состояние (двоичная комбинация), а для теста March *C*– два состояния в $N - 1 = 7$ ячейках ЗУ для всевозможных фиксированных положений базовой ячейки. Для каждой базовой ячейки b, исключая случай, когда $b = b_0$, имеем новое значение в одной из соседних ячеек, ячейке b_0, что изменит множество обнаруживаемых и не обнаружива-

емых *PNPSFk*. Например, для первой итерации теста MATS+, базовой ячейки $b = b_1$ и начального адреса $A_0 = A(0)$ обнаруживаются два класса *PNPSF*5, а именно, 15 неисправностей $b_{i0}\, n_{i1}\, n_{i2}\, n_{i3}\, n_{i4} = b\ 0\ 0\ 0\ 0$ и 20 неисправностей $n_{i0}\, b_{i1}\, n_{i2}\, n_{i3}\, n_{i4} = 1\ b\ 0\ 0\ 0$. Здесь символ *n* используется для обозначения соседних ячеек *i*-ой неисправности *PNPSFk* конкретного класса. Для нового начального адреса $A_0 = A(1)$ имеем 15 неисправностей $b_{i0}\, n_{i1}\, n_{i2}\, n_{i3}\, n_{i4} = b\ 0\ 0\ 0\ 0$, ранее обнаруженных при $A_0 = A(0)$, и 20 новых неисправностей $n_{i0}\, b_{i\,1} n_{i2}\, n_{i3}\, n_{i4} = 0\ b\ 0\ 0\ 0$. Новыми являются неисправности, которые не были обнаружены в результате первой итерации теста MATS+. Если базовая ячейка имеет другую фиксированную позицию, предположим $b = b_3$, тогда для $A_0 = A(0)$ имеем четыре класса *PNPSF*5. Они включают одну неисправность класса $b_{i0}\, n_{i1}\, n_{i2}\, n_{i3}\, n_{i4} = b\ 0\ 0\ 0\ 0$, 12 неисправностей $n_{i0}\, b_{i1}\, n_{i2}\, n_{i3}\, n_{i4} = 1\ b\ 0\ 0\ 0$, 18 неисправностей $n_{i0}\, n_{i1}\, b_{i2}\, n_{i3}\, n_{i4} = 1\ 1\ b\ 0\ 0$ и 4 неисправности $n_{i0}\, n_{i1}\, n_{i2}\, b_{i3}\, n_{i4} = 1\ 1\ 1\ b\ 0$. Для нового начального адреса $A_0 = A(1)$ получим другое распределение неисправностей (см. таблицу 13.6). В этом случае имеем одну неисправность $b_{i0}\, n_{i1}\, n_{i2}\, n_{i3}\, n_{i4} = b\ 0\ 0\ 0\ 0$, 12 неисправностей $n_{i0}\, b_{i1}\, n_{i2}\, n_{i3}\, n_{i4}$ (8 неисправностей $1\ b\ 0\ 0\ 0$ и 4 - $0\ b\ 0\ 0\ 0$), а также 18 неисправностей $n_{i0}\, n_{i1}\, b_{i2}\, n_{i3}\, n_{i4}$ (6 неисправностей $1\ 1\ b\ 0\ 0$ и 12 неисправностей $0\ 1\ b\ 0\ 0$) и 4 неисправности $n_{i0}\, n_{i1}\, n_{i2}\, b_{i3}\, n_{i4} = 0\ 1\ 1\ b\ 0$.

Для нового начального адреса $A_0 = A(1)$ и базовой ячейки $b = b_3$ количество новых *PNPSF*5 неисправностей, обнаруживаемых только во время второй процедуры тестирования, равняется такому же количеству 20, как и для случая базовой ячейки $b = b_1$, и состоит из 4 неисправностей $0\ b\ 0\ 0\ 0$, 12 неисправностей $0\ 1\ b\ 0\ 0$ и 4 неисправности $0\ 1\ 1\ b\ 0$. Совершенно другой результат имеем для случая, когда базовая ячейка $b = b_0$ и начальный адрес $A_0 = A(1)$. В этом случае все неисправности, обнаруживаемые в течение второй итерации тестирования, являются отличными от неисправностей, обнаруженных в течение первой процедуры тестирования тестом MATS+ для $A_0 = A(0)$. Действительно, в первой процедуре обнаруживаются только неисправности $b\ 0\ 0\ 0\ 0$, а во время второй – только $b\ 1\ 1\ 1\ 1$. С учетом всех возможных позиций для базовой ячейки, общее количество неисправностей *PNPSF*5, обнаруживаемых только во время второй процедуры тестирования, равняется 175.

Как показано в [235 349], количество дополнительных неисправностей *PNPSFk*, обнаруженных во время второй итерации теста MATS+ с начальным адресом $A_0 = A(1)$, вычисляется согласно:

$$(N-1)\binom{N-2}{k-2}+\binom{N-1}{k-1}.$$

Для $A_0 = A(2)$ это значение будет равно:

$$(N-2)\left[\binom{2}{1}\binom{N-3}{k-2}+\binom{2}{2}\binom{N-3}{k-3}\right]+2\left[\binom{N-2}{k-1}+\binom{N-2}{k-2}\right].$$

А для $A_0 = A(3)$ получим:

$$(N-3)\left[\binom{3}{1}\binom{N-4}{k-2}+\binom{3}{2}\binom{N-4}{k-3}+\binom{3}{3}\binom{N-4}{k-4}\right]+3\left[\binom{N-3}{k-1}+\binom{2}{1}\binom{N-3}{k-2}+\binom{2}{2}\binom{N-3}{k-3}\right].$$

Отметим, что новый начальный адрес $A_0 = A(r)$, $r \in \{1, 2, 3, \ldots, N-1\}$, в случае теста MATS+, делит все множество из N возможных базовых ячеек на две группы. Это деление реализуется в зависимости от расстояния $s = A(r) - A(0)$ между начальными адресами $A(0)$ и $A(r)$. В соответствии с приведенной метрикой первая группа включает $(N - s)$ ячеек с s различными битами в $N - 1$ ячейках ЗУ (исключая базовую ячейку), вторая группа включает s ячеек с $(N - s)$ различными значениями в соседних ячейках. Тогда общее количество $Q_1(PNPSFk(s))$ неисправностей $PNPSFk$ обнаруживаемых только во время второй итерации теста MATS+ для $N >> k$, определяется как [235]:

$$Q_1(PNPSFk(s)) = (N-s)\sum_{i=1}^{\min(s,(k-1))}\binom{s}{i}\binom{N-s-1}{k-i-1}+s\sum_{i=1}^{\min(s,(k-1))}\binom{s-1}{i-1}\binom{N-s}{k-i}. \qquad (13.6)$$

Выражение (13.6) справедливо для любых начальных адресов A_0, удовлетворяющих равенству $s = A(r) - A(0)$ для $A(r) > A(0)$ [183, 235].

Анализ последнего соотношения показывает, что количество $Q_1(PNPSFk(s))$ неисправностей, обнаруживаемых в течение второй тестовой процедуры, зависит только от значения s и принимает максимум при $s = N/2$. Такой же результат может быть получен при максимизации количества новых значений в соседних $N - 1$ ячейках для всевозможных фиксированных положений базовой ячейки. Это есть то же самое, что и максимизация суммы хэмминговых расстояний S_HD для всевозможных положений базовой ячейки в двух последовательных итерациях теста для начальных адресов $A(0)$ и $A(r)$. Значение $S\_HD(s)$ вычисляется как: $S\_HD(s) = s(N - s) + (N - s)s = 2s(N - s)$ [183, 235]. Тогда максимум значения $S\_HD(s)$ будет достигаться для величины s, значение которой определяется из соотношения:

$$\frac{\partial(S\_HD(s))}{\partial s} = \frac{\partial(2s(N-s))}{\partial s} = (2N - 4s) = 0. \qquad (13.7)$$

В результате получим, что $s = N/2$. Таким образом, максимальная покрывающая способность при проведении только двух итераций тестирования будет достигнута при использовании начальных адресов, удовлетворяющих соотношению: $s = A(r) - A(0) = N/2$ [351].

Для общего случая, когда тестирование ЗУ проводится в результате выполнения q последовательных итераций, оптимальное соотношение начальных адресов $A(r_1)$, $A(r_2)$, $A(r_3)$, ..., $A(r_q)$, где $A(r_1) < A(r_2) < A(r_3) < \ldots < A(r_q)$, для каждой из итераций, будет определяться максимальным значением $S\_HD(q)$. Здесь $S\_HD(q)$ есть сумма хэмминговых расстояний $S\_HD(s)$ для всевозможных положений базовой ячейки во всех парах последовательных

процедур тестирования, то есть $S\_HD(q) = S\_HD(s_1) + S\_HD(s_2) + \ldots + S\_HD(s_q) + S\_HD(s_0)$, где $s_i = A(r_{i+1}) - A(r_i)$, $i \in \{1, 2, \ldots, q-1\}$, а $s_q = s_1 + s_2 + \ldots + s_{q-1}$. Окончательно для $S\_HD(q)$ получим [235, 351]:

$$S\_HD(q) = 2\sum_{i=1}^{q} s_i(N - s_i).$$

Таким образом, для трехкратного применения маршевых тестов сумма хэмминговых расстояний будет равна:

$$S\_HD(q) = 4s_1N + 4s_2N - 4s_1^2 - 4s_2^2 - 4s_1s_2. \qquad (13.8)$$

Данное выражение достигает максимума при $s_1 = s_2 = N/3$, т. е. максимальная покрывающая способность трехкратного маршевого теста MATS+ достигается для случая, когда разность между начальными адресами двух следующих друг за другом итераций теста равняется $N/3$ [235, 351].

Для подтверждения полученных соотношений был проведен ряд вычислительных экспериментов. Эксперимент состоял в случайном выборе базовой ячейки и $k - 1$ соседних (влияющих) ячеек *PNPSFk* со случайно выбранным кодом их состояний, после чего применялись однократно, двукратно, трехкратно или q-кратно тесты MATS++ и March *C*– и фиксировался факт обнаружения или не обнаружения данной конкретной неисправности. После проведения большого числа подобных экспериментов (порядка 10^6) вычислялись процентные отношения обнаруженных кодочувствительных неисправностей ЗУ к общему числу сгенерированных неисправностей.

В таблицах 13.7 – 13.9 приведены экспериментальные данные для двукратного применения маршевого теста MATS++ и первых двух фаз с последующей фазой чтения маршевого теста March *C*– для *PNPSF5* и различной емкости N ЗУ. Символом s обозначена разность между начальным адресом первой и второй итерации теста ($s = A(r) - A(0)$). В таблицах 13.8 и 13.9 включен дополнительный столбец MATS++т, который содержит теоретические значения для маршевого теста MATS++, полученные по выражению (13.6) для конкретного объема ЗУ и типа кодочувствительной неисправности.

Из таблиц 13.7 – 13.9 следует, что наибольшая эффективность двукратных маршевых тестов достигается в том случае, когда разность s между начальными адресам двух последовательных итераций теста равняется $N/2$ [235, 351].

Таблица 13.7 – *Покрывающая способность* (%) *для* $N = 8$

s	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
MATS++	6,20	10,10	11,76	12,30	**12,40**	12,30	11,76	10,10
March *C*–	12,40	18,50	22,20	23,90	**24,50**	23,90	22,20	18,50

Таблица 13.8 – *Покрывающая способность* (%) *для* $N = 16$

s	MATS++	MATS++т	March $C-$	s	MATS++	MATS++т	March $C-$
0	6,12	6,250	12,50	8	**12,33**	**12,34**	**23,90**
1	8,10	8,203	15,50	9	12,20	12,29	23,50
2	9,50	9,635	18,00	10	12,00	12,13	23,15
3	10,6	10,66	20,00	11	11,70	11,83	22,50
4	11,3	11,37	21,40	12	11,30	11,36	21,40
5	11,7	11,84	22,50	13	10,60	10,66	20,00
6	12,0	12,13	23,15	14	9,500	9,680	18,00
7	12,2	12,29	23,50	15	8,100	8,554	15,50

Таблица 13.9 – *Покрывающая способность* (%) *для* $N = 32$

s	MATS ++	MATS ++т	March $C-$	s	MATS ++	MATS ++т	March $C-$	s	MATS ++	MATS ++т	March $C-$
0	6,20	6,250	12,50	11	11,70	11,85	22,60	22	11,60	11,67	22,20
1	7,20	7,227	14,00	12	11,85	11,99	23,00	23	11,30	11,45	21,75
2	8,00	8,077	15,40	13	11,95	12,09	23,20	24	11,00	11,17	21,10
3	8,70	8,814	16,60	14	12,00	12,17	23,40	25	10,70	10,85	20,40
4	9,30	9,449	17,80	15	12,05	12,21	23,48	26	10,40	10,45	19,60
5	10,0	9,994	18,80	16	**12,07**	**12,22**	**23,53**	27	10,00	9,994	18,80
6	10,4	10,46	19,60	17	12,05	12,21	23,48	28	9,300	9,449	17,80
7	10,7	10,85	20,40	18	12,00	12,17	23,40	29	8,700	8,814	16,60
8	11,0	11,18	21,10	19	11,95	12,09	23,20	30	8,000	8,081	15,40
9	11,3	11,45	21,75	20	11,85	11,99	23,00	31	7,200	7,298	14,00
10	11,6	11,67	22,20	21	11,70	11,85	22,60	–			

В таблице 13.10 приведены результаты трехкратного применения маршевых тестов для $N = 8$, где приведена покрывающая способность *PNPSF*5. Символами s_1 и s_2 обозначается разность между начальными адресами первой и второй (s_1), а также второй и третьей (s_2) итерациями тестов.

Таблица 13.10 – *Покрывающая способность* (%) *для* $N = 8$

s_1	s_2	MATS++	March $C-$	s_1	s_2	MATS++	March $C-$	s_1	s_2	MATS++	March $C-$
1	1	14,00	24,87	2	2	17,33	31,90	3	4	16,20	30,10
1	2	15,67	28,37	2	3	**17,70**	**32,88**	4	1	16,20	29,80
1	3	16,13	29,87	2	4	17,33	31,90	4	2	17,30	31,90
1	4	16,13	29,90	2	5	15,60	28,66	4	3	16,20	30,15
1	5	15,60	28,40	3	1	16,20	30,10	5	1	15,60	28,66
1	6	14,00	24,90	3	2	**17,70**	**32,90**	5	2	15,70	28,64
2	1	15,60	28,30	3	3	17,70	32,91	6	1	14,00	27,87

В результате максимальная покрывающая способность трехкратных маршевых тестов достигается при $s_1 = 2$, $s_2 = 3$ или, наоборот, при $s_1 = 3$, $s_2 = 2$, что подтверждает полученные выше ранее теоретические результаты и, в частности, выражение (13.8) [235, 331, 351].

Увеличение количества итераций маршевых тестов будет увеличивать полноту покрытия *PNPSFk*, однако предельные значения, приведенные в таблице 13.11 для $N = 16$, свидетельствуют об ограниченных возможностях данного метода тестирования ЗУ.

Таким образом, можно сделать вывод, что 100 % покрывающая способность неисправностей *PNPSFk* при использовании маршевых тестов достигается только для $k = 3$ и теста March *C*– при реализации $q = N$ кратного тестирования ЗУ. Увеличение кратности q теста до величин, близких к емкости ЗУ, нереально для современных ЗУ. В то же время эффективное увеличение полноты покрытия достигается только для малых значений q [235, 331, 351].

Таблица 13.11 – *Покрывающая способность* (%) 16-*кратных тестов*

PNPSFk	*PNPSF*3	*PNPSF*4	*PNPSF*5	*PNPSF*6	*PNPSF*7
MATS++	75	50	31,10	18,40	11,00
March *C*–	100	75	50,00	31,00	18,20

Как отмечалось ранее, не только изменение начальных адресов при многократном тестировании ЗУ позволяет повысить эффективность обнаружения кодочувствительных неисправностей *PNPSFk*, в качестве альтернативного подхода может быть применена модификация последовательностей адресов [335]. В качестве примера модификации адресов рассмотрим случай двукратного маршевого теста. Первая итерация теста реализуется при использовании произвольного генератора адресов ЗУ. Единственным требованием к нему является требование перебора всех адресов без повторений в некой произвольной упорядоченной последовательности. Для второй итерации используется та же последовательность адресов ЗУ, что и для первой итерации, но с различными модификациями. Первая модификация (модификация № 1) состоит в инвертировании старшего бита исходной адресной последовательности, использованной в первой итерации теста, а вторая модификация (модификация № 2) – в инвертировании младшего бита адресной последовательности.

Результаты двух экспериментов для двух различных модификаций адресной последовательности приведены в таблице 13.12. В графе *вторая итерация* приведена суммарная полнота покрытия, полученная в результате выполнения двух итераций. Анализ приведенных результатов свидетельствует о зависимости полноты покрытия от метода модификации последовательности адресов.

Таблица 13.12 – *Полнота покрытия* (%) *PNPSF*5 *двукратными тестами*

Маршевый тест	Модификация №1		Модификация №2	
	первая итерация	вторая итерация	первая итерация	вторая итерация
MATS++	6,25	12,5	6,25	9,82
March *C*–	12,5	24,6	12,5	19,6

Таким образом, возникает задача выбора оптимальной, с точки зрения максимальной полноты покрытия, модификации адресов для следующей итерации многократного теста. Такая же задача существует и в случае выбора оптимальных начальных состояний ЗУ для увеличения полноты покрытия. Для выбора оптимальных адресных последовательностей ЗУ и его начальных состояний при реализации многократного тестирования необходимо использовать формальные подходы, основанные на использовании численных характеристик, как адресных последовательностей, так и начальных состояний ЗУ.

13.4. Анализ эффективности изменяемых начальных состояний ЗУ

При дальнейшем исследовании будем рассматривать эффективность маршевых тестов типа MATS, формирующих только одну двоичную комбинацию в соседних ЗЭ *PNPSFk*. Как отмечалось ранее, результаты, полученные для данного случая, легко обобщаются на более сложные маршевые тесты. Как было показано в [234], множество начальных состояний ЗУ, позволяющее эффективно обнаруживать *PNPSFk*, формируется на основании численных метрик, основанных на расстоянии Хэмминга. При использовании многократных маршевых тестов множество начальных состояний ЗУ должно отвечать условию, сформулированному в утверждении 13.1 [234].

Утверждение 13.1. Для случая q-кратного маршевого теста, формирующего только одну двоичную комбинацию в соседних ЗЭ произвольной кодочувствительной неисправности *PNPSFk*, оптимальное множество начальных состояний ЗУ B_0, B_1, B_2, …, B_{q-1} должно иметь максимальное минимальное хэммингово расстояние $HD(B_k, B_j)$ для любой пары (B_k, B_j) начальных состояний $k \neq j \in \{0, 1, 2, ..., q-1\}$.

Традиционным решением, удовлетворяющим утверждению 13.1, является множество начальных состояний ЗУ, для которых минимальное хэммингово расстояние равняется $N/2$. Для ЗУ, состоящего из N бит, это множество содержит $q = 2(\lceil \log_2 N \rceil + 1)$ начальных состояний со стандартной и хорошо известной структурой [73, 123]. Если $N = 2^m$, то количество подобных двоичных векторов равняется $2(m + 1)$. В качестве примера соответствующие 10 начальных состояний ЗУ для $m = 4$ представлены в таблице 13.13.

Для упрощения анализа эффективности начальных состояний, приведенных в таблице 13.13, при обнаружении кодочувствительных неисправностей *PNPSFk* маршевыми тестами, генерирующими только одну двоичную

комбинацию в соседних ячейках, предположим, что $k << N$, тогда $N - k \approx N$ [331].

Таблица 13.13 – *Начальные состояния ЗУ*

B	$b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8\ b_9\ b_{10}\ b_{11}\ b_{12}\ b_{13}\ b_{14}\ b_{15}$
B_0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
B_2	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
B_3	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
B_4	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
B_5	1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
B_6	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
B_7	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
B_8	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
B_9	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

Применение первого начального состояния B_0 = 0 0 0 … 0 0 позволяет сформировать нулевые комбинации во всевозможных произвольных k из N запоминающих элементах ЗУ. Тогда количество $Q(B_0)$ обнаруживаемых MATS подобным тестом неисправностей *PNPSFk* при использовании начального состояния ЗУ B_0 определяется как:

$$Q(B_0) = k\binom{N}{k}.$$

Начальное состояние B_1 формирует двоичные комбинации, состоящие из k единиц в произвольных k запоминающих ячейках ЗУ. Это позволяет обнаруживать другие неисправности *PNPSFk* в сравнении с неисправностями, обнаруженными при применении B_0. Однако количество неисправностей обоих множеств одинаково, т. е. $Q(B_1) = Q(B_0)$, а общее их число вычисляется как: $Q(B_0,B_1) = Q(B_1) + Q(B_0)$. Принимая во внимание, что N является большим целым числом, для которого $N - k \approx N$ и $N^k >> N^{k-1}$, последнее соотношение принимает вид:

$$Q(B_0, B_1) = 2k\binom{N}{k} = \frac{2N^k}{(k-1)!}.$$

При тех же допущениях для величин N и k выражение для общего количества $Q(PNPSFk)$ (1.2) неисправностей *PNPSFk* может быть упрощено:

$$Q(PNPSFk) = k2^k\binom{N}{k} = \frac{2^k N^k}{(k-1)!}. \qquad (13.9)$$

Тогда полнота покрытия $FC(B_0, B_1)$ неисправностей *PNPSFk* двукратным MATS подобным тестом с использованием B_0, B_1 вычисляется как:

$$FC(B_0, B_1) = \frac{Q(B_0, B_1)}{Q(PNPSFk)}100\% = \frac{1}{2^{k-1}}100\% = \left(1 - \left(\frac{2^{k-1}-1}{2^{k-1}}\right)^1\right)100\%. \qquad (13.10)$$

Следующая пара начальных состояний B_2 и B_3 позволяет сформировать в некоторых группах из k произвольных ЗЭ ОЗУ емкостью N бит новые двоичные комбинации. В сравнении с ранее сгенерированными на основании векторов B_0 и B_1 двоичными комбинациями новые комбинации будут состоять из векторов, включающих как 0, так и 1 [331]. В силу инверсной природы начальных состояний B_2 и B_3 их использование позволит обнаружить одинаковое количество неисправностей *PNPSFk*, которые отличаются инверсными состояниями соседних ячеек. Общее количество неисправностей $Q(B_2, B_3)$ будет вычисляться как:

$$Q(B_2, B_3) = 2k\sum_{i=1}^{k-1}\binom{N/2}{k-i}\binom{N/2}{i}.$$

Тогда общее число обнаруживаемых неисправностей *PNPSFk* после четырех итераций MATS подобного теста, основанных на начальных состояниях B_0, B_1, B_2 и B_3, равняется:

$$Q((B_0, B_1),(B_2, B_3)) = 2k\left(\binom{N}{k} + \sum_{i=1}^{k-1}\binom{N/2}{k-i}\binom{N/2}{i}\right).$$

Используя ранее принятые допущения, что $N^k >> N^{k-1}$ и $k << N$ для больших величин $N = 2^m$, окончательно получим:

$$Q((B_0, B_1),(B_2, B_3)) = kN^k\left(\frac{2}{k!} + \frac{2}{2^k}\sum_{i=1}^{k-1}\frac{1}{(k-i)!i!}\right).$$

Полнота покрытия $FC((B_0, B_1),(B_2, B_3))$ неисправностей *PNPSFk* равняется

$$FC((B_0, B_1),(B_2, B_3)) = \frac{Q((B_0, B_1),(B_2, B_3))}{Q(PNPSFk)}100\% = \left(\frac{2}{2^k} + \frac{2k!}{2^{2k}}\sum_{i=1}^{k-1}\frac{1}{(k-i)!i!}\right)100\% =$$

$$= \frac{2^k - 1}{2^{2k-2}}100\% = \left(1 - \left(\frac{2^{k-1}-1}{2^{k-1}}\right)^2\right)100\%. \qquad (13.11)$$

Для упрощения последнего соотношения использовалось равенство:

$$\sum_{i=1}^{k-1} \frac{k!}{(k-i)!i!} = 2^k - 2. \qquad (13.12)$$

Начальные состояния B_4 и B_5 в сравнении с B_0, B_1, B_2, и B_3 имеют более сложную структуру, приведенную в таблице 13.14.

Таблица 13.14 – *Начальные состояния B_4 и B_5*

	X_1	X_2	X_3	X_4
B_4	0 0 0 … 0	1 1 1 … 1	0 0 0 … 0	1 1 1 … 1
B_5	1 1 1 … 1	0 0 0 … 0	1 1 1 … 1	0 0 0 … 0

Так же, как и в случае предыдущих начальных состояний, состояния B_4 и B_5 являются инверсными векторами, поэтому количество обнаруженных неисправностей *PNPSFk* как на основании B_4, так и на основании B_5, одинаково. Проанализируем эффективность начального основания B_4 для ЗУ емкостью $N = 2^m$ [331]. Так как $N = 2^m$ нацело делится на 4, то начальное состояние B_4 будет представляться четырьмя равными частями X_1, X_2, X_3, и X_4, приведенными в таблице 13.14. В сравнении с начальными состояниями B_0, B_1, B_2 и B_3 состояние B_4 формирует в k определенных ячейках ОЗУ новые двоичные комбинации P_i, $i \in \{1, 2, 3, \ldots, 7\}$, представленные в таблице 13.15.

Таблица 13.15 – *Двоичные комбинации P_i*

	X_1	X_2	X_3	X_4
P_1	0 0 0 … 0	1 1 1 … 1		
P_2			0 0 0 … 0	1 1 1 … 1
P_3	0 0 0 … 0	1 1 1 … 1	0 0 0 … 0	
P_4	0 0 0 … 0	1 1 1 … 1		1 1 1 … 1
P_5	0 0 0 … 0		000…0	1 1 1 … 1
P_6		111…1	000…0	1 1 1 … 1
P_7	0 0 0 … 0	111…1	000…0	1 1 1 … 1

Для k = 4 формируются: P_1 = {0_111, 00_11, 000_1 (X_1_X_2)}; P_2 = {0_111, 00_11, 000_1 (X_3_X_4)}; P_3 = {00_1_0, 0_11_0, 0_1_00 (X_1_X_2_X_3)}; P_4 = {00_1_1, 0_11_1, 0_1_11 (X_1_X_2_X_4)}; P_5 = {00_0_1, 0_00_1, 0_0_11 (X_1_X_3_X_4)}; P_6 = {11_0_1, 1_00_1, 1_0_11 (X_2_X_3_X_4)} и P_7={0_1_0_1 (X_1_X_2_X_3_X_4)}. Такое же количество двоичных комбинаций, как инверсных копий комбинаций, полученных на основании B_4, формируется и при использовании B_5 [331].

Число обнаруживаемых неисправностей *PNPSFk* типа P_1 и P_2 равняется:

$$Q(P_1, P_2) = 4k \sum_{i=1}^{k-1} \binom{N/4}{k-i} \binom{N/4}{i}.$$

Дополнительное количество обнаруживаемых неисправностей *PNPSFk*, формируемых на основании P_3, P_4, P_5, и P_6, вычисляется как:

$$Q(P_3, P_4, P_5, P_6) = 8k \sum_{j=1}^{k-2} \binom{N/4}{j} \sum_{i=1}^{k-j-1} \binom{N/4}{k-j-i} \binom{N/4}{i}.$$

Формирование двоичных комбинаций P_7 начальными состояниями B_4 и B_5 позволяет обнаруживать дополнительное количество неисправностей *PNPSFk*:

$$Q(P_7) = 2k \sum_{j=1}^{k-3} \binom{N/4}{j} \sum_{r=1}^{k-j-2} \binom{N/4}{r} \sum_{i=1}^{k-j-r-1} \binom{N/4}{k-j-r-i} \binom{N/4}{i}.$$

Окончательно общее количество *PNPSFk* неисправностей, обнаруживаемых в результате шестикратного применения теста MATS, используя начальные состояния B_0, B_1, B_2, B_3, B_4 и B_5, равняется:

$$Q((B_0, B_1), (B_2, B_3), (B_4, B_5)) = 2k \left(\binom{N}{k} + \sum_{i=1}^{k-1} \binom{N/2}{k-i} \binom{N/2}{i} + 2 \sum_{i=1}^{k-1} \binom{N/4}{k-i} \binom{N/4}{i} + \right.$$

$$\left. + 4 \sum_{j=1}^{k-2} \binom{N/4}{j} \sum_{i=1}^{k-j-1} \binom{N/4}{k-j-i} \binom{N/4}{i} + \sum_{j=1}^{k-3} \binom{N/4}{j} \sum_{r=1}^{k-j-2} \binom{N/4}{r} \sum_{i=1}^{k-j-r-1} \binom{N/4}{k-j-r-i} \binom{N/4}{i} \right).$$

На основании соотношения (13.12) и принятых ранее допущениях полнота покрытия $FC((B_0, B_1), (B_2, B_3), (B_4, B_5))$ неисправностей *PNPSFk* вычисляется как:

$$FC((B_0, B_1), (B_2, B_3), (B_4, B_5)) = \frac{Q((B_0, B_1), (B_2, B_3), (B_4, B_5))}{Q(PNPSFk)} 100\ \% =$$

$$= \left(1 - \left(\frac{2^{k-1} - 1}{2^{k-1}} \right)^3 \right) 100\ \%\ . \qquad (13.13)$$

На основании $l + 1$ пары начальных состояний (B_0, B_1), (B_2, B_3), …, (B_{2l}, B_{2l+1}), для $l \in \{1, 2, 3, \ldots, m\}$ будет достигнута полнота покрытия, вычисляемая как:

$$FC((B_0, B_1), (B_2, B_3), \ldots, (B_{2l}, B_{2l+1})) =$$

$$= \frac{Q((B_0, B_1), (B_2, B_3), \ldots, (B_{2l}, B_{2l+1}))}{Q(PNPSFk)} 100\ \% = \left(1 - \left(\frac{2^{k-1} - 1}{2^{k-1}} \right)^{l+1} \right) 100\ \% \qquad (13.14)$$

Последнее соотношение (13.14) позволяет оценить максимально возможную полноту покрытия $FC_{MAX} = FC((B_0, B_1), (B_2, B_3), \ldots, B_{2m}, B_{2m+1})$. Эта величина достигается при использовании $2(\lceil \log_2 N \rceil + 1)$ начальных состояний $(B_0, B_1), (B_2, B_3), \ldots, (B_{2m}, B_{2m+1})$, где $m = \lceil \log_2 N \rceil$ (см. таблицу 13.13), и вычисляется как:

$$FC_{MAX} = \left(1 - \left(\frac{2^{k-1} - 1}{2^{k-1}}\right)^{\lceil \log_2 N \rceil + 1}\right) 100\ \% \qquad (13.15)$$

Для $N = 2^m$ полнота покрытия FC_{MAX} неисправностей *PNPSFk* при использовании стандартных начальных состояний $(B_0, B_1), (B_2, B_3), \ldots, (B_{2m}, B_{2m+1})$ приведена в таблице 13.16.

Из приведенной таблицы следует, что, например, при $2(m + 1) = 2(30 + 1) = 62$ итерации теста типа MATS, оказывается возможным достижение 99,98 % полноты покрытия неисправностей *PNPSF*3, 98,40 % неисправностей *PNPSF*4 и 86,47 % для неисправностей *PNPSF*5. Следует отметить, что такая высокая полнота покрытия достигается за счет большого количества итераций теста [331].

Таблица 13.16 – *Значение* FC_{MAX} (%) *для различных значений N*

$N = 2^m$	2^5	2^{10}	2^{20}	2^{30}
$k = 3$	82,20	95,77	99,76	99,98
$k = 4$	63,87	76,98	93,94	98,40
$k = 5$	32,10	49,16	74,21	86,47

При том же количестве итераций $l = 2(m + 1)$ достигаемая полнота покрытия несущественно превышает полноту покрытия, которая может быть получена с использованием случайных начальных состояний ЗУ [331]. Для *PNPSF*3 сравнительные данные для различных значений *N* приведены в таблице 13.17, где $FC_{MATS}(PNPSF3, l)$, представляет собой полноту покрытия для $l = 2(m + 1)$ случайных начальных состояниях ЗУ.

Таблица 13.17 – *Полнота покрытия* (%) *для стандартных и случайных состояний ЗУ*

$N = 2^m$	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9
$FC_{MAX}(PNPSF3)$	57,81	68,35	76,26	82,20	86,65	89,98	92,49	94,36
$FC_{MATS}(PNPSF3, l)$	55,12	65,63	73,69	79,85	84,57	88,19	90,96	93,07

Анализ данных, приведенных в таблицах 13.16 и 13.17, позволяет сделать вывод о невысокой эффективности как случайных начальных состояний ЗУ, так и стандартных состояний.

Глава 14. Псевдо-исчерпывающее тестирование

14.1. Исчерпывающее и псевдо-исчерпывающее тестирование ЗУ

Как отмечалось в главе 8, изначально исчерпывающее тестирование рассматривалось только для комбинационных цифровых устройств [16, 289, 302]. Для его реализации в случае комбинационного устройства, имеющего N входов, необходимо сформировать 2^N тестовых двоичных вектора, что практически всегда является нереальной задачей. Поэтому, в качестве реальной альтернативы исчерпывающему тестированию во многих случаях применяется псевдо-исчерпывающее тестирование [12, 105, 118, 186, 187], которое основывается на формировании множества тестовых наборов $T(N, k)$, обеспечивающих 2^k возможных двоичных комбинаций на любых k из N входах тестируемого устройства. Примеры подобных тестов приведены в таблице 8.1 [304].

Исчерпывающее тестирование ЗУ предполагает использование всевозможных 2^N состояний его ЗЭ, только для записи которых необходимо применение $N \times 2^N$ операций обращения к ЗУ [331]. Поэтому под исчерпывающим тестированием ЗУ всегда понимают исчерпывающее обнаружение кодочувствительных неисправностей *PSFk* (*k-coupling*) для ограниченного количества k ячеек ЗУ [254, 325, 331, 344, 350]. Как было показано в литературных источниках, данная модель является обобщающей моделью, покрывающей более простые известные модели неисправностей, а также позволяющей обнаруживать широкий спектр физических дефектов ЗУ, которые нельзя описать более простыми известными моделями неисправностей. Однако и для случая исчерпывающего обнаружения *PSFk* существенными являются ограничения, применяемые для этих неисправностей.

В одной из первых работ по исчерпывающему тестированию *PSFk* рассматривались только *несвязные* (*disjoint*) *PSFk* для $k = 3$ в ЗУ с известной топологией [42, 43]. Сложность предложенного теста равняется $Q(T(N, 3)) = N + 32N\log_2 N$. При тех же ограничениях на *PSFk* в [331] рассматривался тест, сложность которого $Q(T(N, 3)) = 36N + 24N\log_2 N$. Для случая произвольных $k = 3$ из N ячеек ЗУ без ограничений лучший результат представлен в [331]. Сложность теста $T(N, 3)$, формируемого в соответствии с приведенным алгоритмом, оценивается величиной $(\log_2 N)^2 + \log_2 N + 2$.

Результаты, представленные в пионерских работах [43, 331,351] по исчерпывающему тестированию *PSFk*, были существенно улучшены за счет применения псевдо-исчерпывающих неразрушающих тестов $E(N, 3)$, сложность которых оценивается величиной $Q(E(N, 3)) = k2^k N \times \log_2 N$ [331]. Как аппроксимация предложенных методов псевдо-исчерпывающего тестирования в данных работах предлагаются методики *циклического неразрушающего тестирования* (*circular transparent testing*) и *псевдослучайного циклического тестирования* (*circular pseudorandom testing*), которые имеют заметно меньшую сложность за счет уменьшения покрывающей способности тестов [331].

Применение кодов *Рида-Соломона* (*Reed-Solomon*) (см. главу 8) позволило распространить идею исчерпывающего тестирования *PSFk* на слово-ориентированные ЗУ [331]. Сложность исчерпывающего теста *PSFk* для ЗУ с w ЗЭ в ячейке (w бит в слове ЗУ) оценивается как: $Q(T(N, k, w)) = 2^{kw+1}n + 2n$, где $n < N$ и определяется параметрами кода Рида-Соломона.

Применение симплексных кодов, описываемых *матрицей Адамара* (*Hadamard matrix*), позволило построить *почти исчерпывающие тесты* ЗУ (*near-exhaustive tests*), основанные на следующей теореме [52].

Теорема 14.1. Для любого $s \geq k$ и произвольного $a < N$, где $N = a\ 2^s$, существует σ-исчерпывающий тест $T(N, k, \sigma)$ с $Q(T(N, k, \sigma)) = 2^{s+1}$ и величиной $1 - \sigma = (1 - \sigma_v)(1 - \sigma_h)$, где:

$$\sigma_v = 1 - \left(2^s \prod_{i=0}^{k-2} (2^s - 2^i) \right) / \left(\prod_{i=0}^{k} (2^s - i) \right); \quad \sigma_h = 1 - a^k \binom{2^s}{k} / \binom{a2^s}{k}.$$

Величина σ может принимать сколь угодно малое значение, определяющее степень близости к исчерпывающему тесту. Значение $(1 - \sigma)$ 100% представляет процентное соотношение сгенерированных двоичных комбинаций для любых k из N ячеек ЗУ к общему их числу. Следует отметить, что в данном случае уменьшение сложности теста достигается за счет снижения его эффективности обнаруживать *PSFk*.

Дальнейшее развитие исследований, направленных на создание тестов для обнаружения *PSFk*, были связаны с введением еще более значимых ограничений на модели обнаруживаемых неисправностей. Так, в случае полного соответствия логических адресов ЗЭ их физическому расположению, неисправности *PNPSF*5 согласно подходу, предложенному в [331], могут быть обнаружены тестом, сложность которого определяется как: $Q(PNPSF5) = 97(5N/9)$. Расширение данного подхода для обнаружения и других разновидностей кодочувствительных неисправностей увеличивает сложность теста до $Q(PNPSF5) = 121(5N/9)$.

Абсолютное соответствие физического расположения ЗЭ их логическим адресам, а также фиксированные значения k позволили создать серию современных тестов, гарантированно обнаруживающих кодочувствительные неисправности. Так, тест PS($23N$) [225], сложность которого пропорциональна $23N$, обнаруживает всевозможные кодочувствительные неисправности для $k = 5$ при последовательном использовании стандартных начальных состояний ОЗУ.

Достаточно интересны решения, представленные в [26, 27], где были предложены два новых теста MT-*R*3CF и ME-*R*4CF для обнаружения неисправностей для k, равного 3 и 4. Сложность этих тестов равняется соответственно $30N$ и $41N$. В данном случае уменьшение сложности обнаружение кодочувствительных неисправностей достигается за счет еще больших ограничений.

Дальнейшее развитие исследований по проблеме обнаружения кодочувствительных неисправностей было направлено на создание не только контролирующих тестов, но и на разработку их диагностических версий [331].

В качестве хорошей аппроксимации исчерпывающего и псевдоисчерпывающего тестирования широко используется вероятностное тестирование ЗУ (см. главы 3, 4 и 5) [331]. В этом случае каждый тестовый вектор выбирается случайным образом, независимо от предыдущих ранее сгенерированных тестовых наборов, которыми в случае ЗУ являются состояния ячеек ЗУ и адресные последовательности, используемые маршевым тестом. Преимущества вероятностного тестирования включают в себя его низкую стоимость внедрения и возможность автоматической генерации множества тестовых данных. Кроме того, при вероятностном тестировании обычно используются тестовые наборы, которые генерируются генератором псевдослучайных или квази-случайных чисел с определенным начальным состоянием, чтобы можно было воспроизвести любое поведение ЗУ при его неисправном состоянии. Вероятностное тестирование и его вариации широко использовались и изучались как для аппаратной части вычислительных систем, так и для программного обеспечения [135, 204].

Первая публикация [53], касающаяся вероятностного тестирования памяти, была направлена на тестирование постоянных запоминающих устройств, которые могут рассматриваться как комбинационные цифровые схемы. Как было показано, время необходимое для реализации вероятностного теста, не превышает допустимого времени и может считаться незначительным по сравнению со временем подготовительной процедуры [53]. Фундаментальные исследования вероятностного тестирования памяти, представленные в [53], основаны на применении модели *цепей Маркова*. Авторы теоретически, а также экспериментальными исследованиями доказали, что вероятностное тестирование памяти имеет линейную сложность, и поэтому более эффективно, чем детерминированные тесты со сложностью $N^{3/2}$ и N^2.

Дальнейшим развитием вероятностного тестирования памяти является, предложенное авторами статьи [209], псевдо-исчерпывающее тестирование памяти на основе многократного повторения процедуры тестирования. Вначале было предложено многократное применение тестов памяти для случая изменения порядка адресов для небольшого числа итераций теста [209, 331]. Были проанализированы двукратные и трехкратные тесты памяти с разными адресными последовательностями. Эффективность в терминах сложности сложных неисправностей памяти была исследована для некоторых детерминированных адресных последовательностей, таких как: последовательностей на базе двоичных счетчиков; последовательностей Грея; последовательностей анти-Грея; последовательностей с максимальным расстоянием Хэмминга; последовательности Соболя и др. [182, 209, 229]. Отправной точкой для изменения содержимого памяти был оптимальный выбор второго начального состояния для двукратного тестирования памяти, рассмотренный в [234]. Расширением идеи вариации начального состояния ЗУ было определение и

формирование оптимального набора состояний памяти для многократного тестирования ЗУ [226].

Все упомянутые выше существующие подходы были направлены на достижение высокого уровня покрытия для некоторых моделей неисправностей памяти или их подмножества. Во всех случаях покрытие неисправностей определялось как процент обнаруженных неисправностей F_C или как вероятность обнаружения P_d. Эти два показателя являются хорошими оценками для любого подхода к тестированию памяти. Во избежание сложной процедуры анализа и оценки эффективности теста памяти рассмотрим псевдо-исчерпывающие методы тестирования памяти. Как упоминалось ранее, эти подходы являются реальной альтернативой исчерпывающему тестированию. Их применение позволяет существенно снизить сложность тестов памяти. Обычно неисправности ЗУ включают только ограниченное число ячеек памяти $k << N = 2^m$, где N – общее количество ячеек памяти, а m – размер адреса ЗУ. Значение k обычно меньше 10, поэтому псевдо-исчерпывающий подход к генерации тестов является многообещающим.

Согласно определению псевдо-исчерпывающего теста (определение 8.2) набор двоичных векторов $E(N, k)$ в двоичном N-мерном пространстве (ЗУ с N ячейками) полностью охватывает все указанные k-мерные подпространства (k ячеек памяти), если проекция $E(N, k)$ на эти k-мерные подпространства содержит все 2^k двоичные векторы в каждом подмножестве из k ячеек памяти.

Изучая эффективность тестов ЗУ, всегда принимается во внимание сложность генерирования всех 2^k комбинаций для k ячеек памяти, что является необходимым, а во многих случаях достаточным условием, позволяющим обнаруживать различные сложные неисправности, задаваемые параметром k. В качестве эффективного способа формирования 2^k комбинаций в k ячейках может быть применена идея генерировании *орбит* (*orbits*) как результата применения маршевого теста памяти ЗУ [182].

14.2. Изменение начального состояния запоминающих устройств

Маршевые тесты запоминающих устройств состоят из последовательных фаз с конкретными комбинациями операций записи (w) и чтения (r). Каждая фаза реализует последовательное обращение ко всем ячейкам запоминающего устройства согласно их адресам. Напомним, что символ $\Uparrow$ обозначает прямую (возрастающую) последовательность адресов памяти, символ $\Downarrow$ обозначает обратную (убывающую) последовательность, а два символа $\Uparrow\Downarrow$ обозначают возрастающую или убывающую последовательность адресов. В качестве примера рассмотрим два наиболее часто применяемых на практике маршевых теста, а именно: MATS ++ $\{\Uparrow\Downarrow(w0); \Uparrow(r0, w1); \Downarrow(r1, w0, r0)\}$ и March $C-$ $\{\updownarrow (w0); \Uparrow(r0, w1); \Uparrow(r1, w0); \Downarrow(r0, w1); \Downarrow(r1, w0); \Uparrow\Downarrow(r0)\}$. Сложность теста MATS ++ равна $6N$, а сложность March $C-$ составляет $10N$, где N представляет собой емкость запоминающего устройства в битах.

Как уже отмечалось в предыдущем разделе, маршевые тесты запоминающих устройств, при их реализации, генерируют, в k произвольных ячейках памяти, так называемую орбиту, которая представляет собой набор двоичных k-разрядных векторов [318]. Конкретный набор векторов в орбите зависит от трех основных факторов [318]. Прежде всего, используемый маршевый тест однозначно определяет структуру орбиты, так как он формулирует правила, по которым генерируются двоичные векторы в ячейках запоминающего устройства. Эти правила представляются набором фаз маршевого теста, их структурой и используемой последовательностью адресов. Начальное состояние запоминающего устройства является вторым аргументом, который принимается во внимание при формировании конкретной орбиты. Адресная последовательность, представляющая собой порядок адресов запоминающих ячеек памяти, формируемых маршевым тестом, является третьим фактором. Порядок адресов в последовательности может быть произвольным с единственным ограничением, что каждый адрес формируется ровно один раз, а их последовательность обратима [318].

Анализ классических маршевых тестов показывает, что существует только одна или две отличающиеся орбиты для k произвольных ячеек ЗУ, которые генерируются во время выполнения теста [318]. Отличающимися орбитами считаются такие орбиты, которые формируют неповторяющиеся наборы в k ячейках ЗУ. Действительно, для фаз классических маршевых тестов применяются два возможных начальных состояния в произвольных k ячейках памяти, а именно все нули и все единицы. Как правило, во время выполнения теста используются фазы, для которых данные из ячеек памяти считываются, и в них записываются инверсные значения. Наряду с двумя возможными вариантами начальных состояний (единичного и нулевого) для k из N произвольных ячеек памяти, во время выполнения фаз применяется прямая последовательность адресов ($\Uparrow$) или обратная последовательность ($\Downarrow$). Поэтому в маршевых тестах существует четыре разные орбиты. Количество орбит и их порядок во время выполнения теста зависит от применяемого маршевого теста. Все возможные комбинации начальных состояний и адресных последовательностей для классических маршевых тестов и соответствующие им орбиты, формируемые в произвольных k ячейках памяти, приведены в таблице 14.1.

Например, маршевый тест MATS ++ генерирует две орбиты, а именно O_0 во время реализации его фазы $\Uparrow(r0, w1)$ и O_3 в результате применения фазы $\Downarrow(r1, w0, r0)$, которые состоят из одинаковых двоичных наборов. В то же время *March C−* позволяет генерировать все четыре возможные орбиты, что доказывает его более высокую эффективность для тестирования запоминающих устройств.

Для орбиты O, порожденной маршевым тестом верны следующие общие утверждения [318].

Таблица 14.1 – *Орбиты, формируемые классическими маршевыми тестами*

Орбиты	O_0	O_1	O_2	O_3
Начальное состояние P_0	0 0 0 … 0 0	0 0 0 … 0 0	1 1 1 … 1 1	1 1 1 … 1 1
Последовательность адресов	Прямая (⇑)	Обратная (⇓)	Прямая (⇑)	Обратная (⇓)
P_0	0 0 0 … 00	0 0 0 … 0 0	1 1 1 … 1 1	1 1 1 … 1 1
P_1	0 0 0 … 01	1 0 0 … 0 0	1 1 1 … 1 0	0 1 1 … 1 1
P_2	0 0 0 … 11	1 1 0 … 0 0	1 1 1 … 0 0	0 0 1 … 1 1
…	…	…	…	…
P_{k-2}	0 0 1 … 11	1 1 1 … 0 0	1 1 0 … 0 0	0 0 0 … 1 1
P_{k-1}	0 1 1 … 11	1 1 1 … 1 0	1 0 0 … 0 0	0 0 0 … 0 1
P_k	1 1 1 … 11	1 1 1 … 1 1	0 0 0 … 0 0	0 0 0 … 0 0

Утверждение 14.1. Орбита O, сформированная в произвольных k ячейках памяти, как результат применения маршевого теста, состоит из k + 1-го k-разрядного двоичного вектора P_0, P_1, ..., P_k.

Данное утверждение следует из структуры орбиты, показанной в таблице 14.1. Следует отметить, что для маршевого теста из-за фазы инициализации ⇑⇓($w0$) или любых фаз теста, таких как: ⇑⇓(..., $w0$, ...), P_0 принимает нулевое значение P_0 = 0 0 0 ... 0, то есть все k ячейки памяти устанавливаются в нулевое состояние. Значение начального состояния ячеек P_0 = 1 1 1 ... 1 может быть сгенерировано тестом, который имеет фазу инициализации ⇑⇓($w1$) или включает фазу ⇑⇓(..., $w1$, ...).

Утверждение 14.2. Расстояние Хэмминга $HD(P_i, P_{i+j})$ между двумя векторами состояний P_i и P_{i+j} орбиты O для k произвольных ячеек запоминающего устройства, где $i \in \{0, 1, 2, ..., k-1\}$ и $j \in \{0 , 1, 2, ..., k-i\}$ равно j.

Это утверждение верно для различных 2^k начальных состояний в k ячейках памяти и произвольной адресной последовательности.

Следует отметить важное следствие утверждения 14.2 о том, что в пределах орбиты O нет двух одинаковых двоичных векторов P_i и P_l для $i \neq l$. Для любого начального состояния P_0 расстояние Хэмминга между любыми двумя векторами P_i и P_{i+j} всегда одинаково и определяется соотношением $HD(P_i, P_{i+j}) = j$, например, если j = 1, то $HD(P_i, P_{i+1}) = 1$.

Для случая псевдо-исчерпывающих тестов основная цель состоит в генерировании всех возможные 2^k двоичных векторов для произвольных k ячеек памяти. При этом последовательность векторов не имеет значения. Оптимальным решением является создание другой орбиты, отличной от сформированных ранее орбит, для получения максимального количества дополнительных тестовых векторов. Все орбиты должны быть максимально разными с точки зрения генерируемых двоичных векторов.

Утверждение 14.3. Две орбиты O_i и O_l считаются эквивалентными или равными ($O_i = O_l$), если обе орбиты включают одинаковые двоичные векторы, независимо от их порядка.

В случае классических маршевых тестов $O_0 = O_3$ и $O_1 = O_2$ (см. таблицу 14.1).

Все множество классических маршевых тестов, как отмечалось ранее [106, 318, 333], делится на два основных множества. Первое множество включает так называемые тесты типа MATS ++. Эти тесты характеризуются тем, что генерируют одну орбиту из четырех возможных или две, но эквивалентные (O_0 и O_3 или O_1 и O_2). Второе множество, называемое маршевыми March *C*− подобными тестами, характеризуется формированием при их реализации, по меньшей мере, двух различных орбит, а именно: O_0, O_1; O_0, O_2; O_1, O_3 или O_2, O_3.

В результате приведенного анализа можно сделать следующий вывод. Однократное применение маршевого теста позволяет получить только одну или две орбиты, которые зависят от начального состояния памяти и используемой адресной последовательности. В случае тестов типа MATS ++ применение маршевого теста позволяет генерировать $k + 1$ различных k-разрядных двоичных векторов или $2k$ для тестов типа March *C*−. Очевидно, что для получения всех возможных 2^k двоичных векторов для k произвольных ячеек памяти, маршевый тест должен применяться более одного раза при использовании различных начальных состояний или различных адресных последовательностей. Подход, основанный на изменении адресных последовательностей, был всесторонне исследован ранее [152, 318, 333].

Первоначально рассмотрим бит-ориентированное запоминающее устройство, которое содержит $N = 2^m$ ячеек, каждая из которых имеет уникальный m-разрядный адрес. Для случая произвольного количества $k < 2^m$ ячеек памяти с адресами $\beta, \gamma, ..., \delta, \varepsilon, \eta$, где $2^m > \beta > \gamma > ... > \delta > \varepsilon > \eta \geq 0$, оригинальная орбита O_0 формируется согласно процедуре, показанной на таблице 14.2 [318].

Таблица 14.2 – *Орбита O_0, состоящая из $k + 1$ двоичного вектора, для k ячеек памяти*

Орбита O_0						
Вектор	Индексы разрядов вектора					
	$k-1$	$k-2$	…	2	1	0
P_0	a_β	a_γ	…	a_δ	a_ε	a_η
P_1	a_β	a_γ	…	a_δ	a_ε	$\overline{a_\eta}$
P_2	a_β	a_γ	…	a_δ	$\overline{a_\varepsilon}$	$\overline{a_\eta}$
P_3	a_β	a_γ	…	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$
	...	...	…	...	...	
P_{k-1}	a_β	$\overline{a_\gamma}$	…	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$
P_k	$\overline{a_\beta}$	$\overline{a_\gamma}$	…	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$

Начиная от исходного состояния памяти $P_0 = a_\beta\, a_\gamma\, ...\, a_\delta\, a_\varepsilon\, a_\eta$ где $a_i \in \{0, 1\}$, $i \in \{\beta, \gamma, ..., \delta, \varepsilon, \eta\}$ и $\overline{a_i}$ принимает инверсное значение относительно a_i, в орбите формируются $k + 1$ различных двоичных векторов, которые удовле-

творяют утверждениям 14.1 и 14.2. В качестве примера рассмотрим случай, когда $k = 3$.

Для всех возможных $2^3 = 8$ начальных состояний восемь орбит, полученных согласно процедуре, приведенной в таблице 14.2, приведены ниже в таблице 14.3.

Таблица 14.3 – *Восемь возможных орбит для* $k = 3$

O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0 0 1	0 0 0	0 1 1	0 1 0	1 0 1	1 0 0	1 1 1	1 1 0
0 1 1	0 1 0	0 0 1	0 0 0	1 1 1	1 1 0	1 0 1	1 0 0
1 1 1	1 1 0	1 0 1	1 0 0	0 1 1	0 1 0	0 0 1	0 0 0

Для случая изменяемого начального состояния в k ячейках запоминающего устройства справедливы следующие утверждения.

Утверждение 14.4. Для произвольных $k > 1$ ячеек запоминающего устройства и различных 2^k начальных состояний $P_0 = a_\beta a_\gamma \ldots a_\delta a_\varepsilon a_\eta$, произвольный маршевый тест при использовании фиксированной адресной последовательности генерирует 2^k уникальных неэквивалентных орбит.

Доказательство утверждения 14.4. Так как $a_i \in \{0, 1\}$, $i \in \{\beta, \gamma, \ldots, \delta, \varepsilon, \eta\}$, то для k ячеек запоминающего устройства с k адресами $\beta, \gamma, \ldots, \delta, \varepsilon, \eta$ существует 2^k различных начальных состояний. Для случая фиксированного порядка адресов каждое начальное состояние генерирует только одну орбиту. Таким образом, для случая неизменяемого порядка адресов количество различных орбит не может быть больше 2^k. Число орбит менее 2^k может быть только в случае, когда для двух различных начальных состояний P_0 две орбиты эквивалентны (утверждение 14.3). Это означает, что для орбиты O_0, показанной в таблице 14.2, с начальным состоянием $P_0 = a_\beta a_\gamma \ldots a_\delta a_\varepsilon a_\eta$ существует эквивалентная орбита O_n с $P_0 \neq a_\beta a_\gamma \ldots a_\delta a_\varepsilon a_\eta$.

Согласно утверждению 14.3, для случая, когда $O_n = O_0$, орбита O_n должно быть построена в результате перестановки двоичных векторов, используемых в орбите O_0. Это означает, что двоичные векторы $P_0 = a_\beta a_\gamma \ldots a_\delta a_\varepsilon a_\eta$ и $P_k = \overline{a_\beta}\,\overline{a_\gamma} \ldots \overline{a_\delta}\,\overline{a_\varepsilon}\,\overline{a_\eta}$ орбиты O_0 с расстоянием Хэмминга $\mathrm{HD}(P_0, P_k) = k$, также должны входить в состав орбиты O_n. Единственная новая позиция для вектора $P_k = \overline{a_\beta}\,\overline{a_\gamma} \ldots \overline{a_\delta}\,\overline{a_\varepsilon}\,\overline{a_\eta}$ – это позиция P_0 в орбите O_n, а для вектора $P_0 = a_\beta a_\gamma \ldots a_\delta a_\varepsilon a_\eta$ местоположение P_k, что следует из утверждения 14.2. Если в орбите O_n начальное состояние ровняется $P_0 = \overline{a_\beta}\,\overline{a_\gamma} \ldots \overline{a_\delta}\,\overline{a_\varepsilon}\,\overline{a_\eta}$, тогда для той же последовательности адресов $P_1 = \overline{a_\beta}\,\overline{a_\gamma} \ldots \overline{a_\delta}\,\overline{a_\varepsilon}\,a_\eta$, которое при $k > 1$ не является двоичным вектором принадлежащим O_0, что означает неэквивалентность орбит O_0 и O_n, т.е. $O_0 \neq O_n$, что и требовалось доказать.

Утверждение 14.5. Для заданного k и постоянного порядка адресов существует $2^k - (k^2 + k)/2 - 1$ орбит O_n с иными начальными состояниями P_0,

состоящие из двоичных векторов, которые не используются в орбите O_0 и $(k^2+k)/2$ орбиты, включающие по два вектора, входящие в орбиту O_0.

Доказательство утверждения 14.5. Для общего случая орбиты O_0 с начальным состоянием $P_0 = a_\beta\, a_\gamma \ldots a_\delta\, a_\varepsilon\, a_\eta$ новая орбита O_n будет получена в результате инвертирования r ($k \geq r \geq 1$) разрядов для всех двоичных векторов исходной орбиты O_0. В качестве примера две новые орбиты O_w и O_v, как результат инвертирования определенных разрядов в векторах орбиты O_0, для $k = 5$, показаны на рис. 14.1. В данном случае использовалось инвертирование соседних разрядов векторов, так что отрицания представляют собой единый непрерывный блок.

	O_0					O_w					O_v				
	4	*3*	*2*	*1*	*0*	*4*	*3*	*2*	*1*	*0*	*4*	*3*	*2*	*1*	*0*
P_0	a_β	a_γ	a_δ	a_ε	a_η	a_β	a_γ	a_δ	$\overline{a_\varepsilon}$	a_η	a_β	$\overline{a_\gamma}$	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	a_η
P_1	a_β	a_γ	a_δ	a_ε	$\overline{a_\eta}$	a_β	a_γ	a_δ	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	a_β	$\overline{a_\gamma}$	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$
P_2	a_β	a_γ	a_δ	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	a_β	a_γ	a_δ	a_ε	$\overline{a_\eta}$	a_β	$\overline{a_\gamma}$	$\overline{a_\delta}$	a_ε	$\overline{a_\eta}$
P_3	a_β	a_γ	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	a_β	a_γ	$\overline{a_\delta}$	a_ε	$\overline{a_\eta}$	a_β	$\overline{a_\gamma}$	a_δ	a_ε	$\overline{a_\eta}$
P_4	a_β	$\overline{a_\gamma}$	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	a_β	$\overline{a_\gamma}$	$\overline{a_\delta}$	a_ε	$\overline{a_\eta}$	a_β	a_γ	a_δ	a_ε	$\overline{a_\eta}$
P_5	$\overline{a_\beta}$	$\overline{a_\gamma}$	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	$\overline{a_\beta}$	$\overline{a_\gamma}$	$\overline{a_\delta}$	a_ε	$\overline{a_\eta}$	$\overline{a_\beta}$	a_γ	a_δ	a_ε	$\overline{a_\eta}$

Рисунок 14. 1 – *Результат инвертирования соседних разрядов векторов орбиты*

Орбита O_w является результатом отрицания всего лишь одного разряда с индексом *1* для всех векторов орбиты O_0, а орбита O_v, получена в результате инвертирования блока разрядов векторов с индексами *1*, *2* и *3*. В результате обе орбиты O_w и O_v имеют три новых вектора и включают в себя два вектора из оригинальной орбиты O_0. Так в орбите O_w используются векторы P_1, P_2 из орбиты O_0, а в орбите O_v, соответственно P_1, P_4. В обоих случаях эти пары векторов заменяют свои позиции в орбите. Для новой орбиты O_w вектор P_1 занимает положение P_2, и наоборот P_2 находится на позиции P_1. То же самое можно заключить и для векторов P_1 и P_4 в случае орбиты O_v. Остальные $k - 1$ векторы из-за примененных отрицаний являются новыми по сравнению с множеством векторов орбиты O_0.

Обобщая приведенный анализ, можно сделать следующий вывод. Для случая одного блока последовательных отрицаний примененных для O_0, с целью получения новой орбиты O_n, как это было проиллюстрировано примерами построения O_w и O_v, результирующая новая орбита включает только два двоичных вектора исходной орбиты O_0. Остальные векторы отличаются от векторов орбиты O_0. Учитывая утверждения 14.3 и 14.4, в новой орбите, очевидно, будут повторяться двоичные векторы P_i и P_j из множества векторов P_0, P_1, P_2, …, P_k. При этом вектор P_i займет позицию вектора P_j и, наоборот, P_j позицию P_i. Отметим, что для любого k существует k блоков с одним отрицанием, $k - 1$ блок с двумя последовательными отрицаниями, $k - 2$ блоков с тремя последовательными отрицаниями и так далее. Тогда число M_1 новых

орбит O_n, которые включают в себя два вектора из оригинальной орбиты O_0, может быть рассчитано как:

$$M_1 = k + (k-1) + (k-2) + ... + 2 + 1 = (k^2 + k)/2. \quad (14.1)$$

Применение других наборов отрицаний для получения новой орбиты, которые включают в себя более одного блока последовательных отрицаний, приводит к получению абсолютно новой орбиты. Например, орбиты O_x и O_z включают другие наборы двоичных векторов по сравнению с векторами орбиты O_0, как показано для случая $k = 5$ на рис. 14.2. Таким образом, можно сделать вывод, что применение более чем одного блока последовательных отрицания при получении новой орбиты O_n разрушают два вектора P_i и P_j, которые сохраняются в случае одного блока отрицаний.

	O_0					O_x					O_z				
	4	*3*	*2*	*1*	*0*	*4*	*3*	*2*	*1*	*0*	*4*	*3*	*2*	*1*	*0*
P_0	a_β	a_γ	a_δ	a_ε	a_η	$\overline{a_\beta}$	$\overline{a_\gamma}$	a_δ	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	a_β	$\overline{a_\gamma}$	a_δ	$\overline{a_\varepsilon}$	$\overline{a_\eta}$
P_1	a_β	a_γ	a_δ	a_ε	$\overline{a_\eta}$	$\overline{a_\beta}$	$\overline{a_\gamma}$	a_δ	$\overline{a_\varepsilon}$	a_η	a_β	$\overline{a_\gamma}$	a_δ	$\overline{a_\varepsilon}$	a_η
P_2	a_β	a_γ	a_δ	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	$\overline{a_\beta}$	$\overline{a_\gamma}$	a_δ	a_ε	a_η	a_β	$\overline{a_\gamma}$	a_δ	a_ε	a_η
P_3	a_β	a_γ	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	$\overline{a_\beta}$	$\overline{a_\gamma}$	$\overline{a_\delta}$	a_ε	a_η	a_β	$\overline{a_\gamma}$	$\overline{a_\delta}$	a_ε	a_η
P_4	a_β	$\overline{a_\gamma}$	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	$\overline{a_\beta}$	a_γ	$\overline{a_\delta}$	a_ε	a_η	a_β	a_γ	$\overline{a_\delta}$	a_ε	a_η
P_5	$\overline{a_\beta}$	$\overline{a_\gamma}$	$\overline{a_\delta}$	$\overline{a_\varepsilon}$	$\overline{a_\eta}$	a_β	a_γ	$\overline{a_\delta}$	a_ε	a_η	$\overline{a_\beta}$	a_γ	$\overline{a_\delta}$	a_ε	a_η

Рисунок 14.2 – *Результат применения более одного блока инверсий*

Число M_2 новых орбит, состоящее из двоичных векторов, которые не участвуют в исходной орбите O_0, можно оценить как:

$$M_2 = 2^k - (k^2 + k)/2 - 1. \quad (14.2)$$

Для случая $k = 3$, согласно (14.1) и (14.2), получим $M_1 = 6$ и $M_2 = 1$, что подтверждается результатами, приведенными в таблице 14.3. Действительно, только одна орбита O_5 имеет совершенно разные двоичные векторы, в сравнении с O_0. Остальные орбиты O_1, O_2, O_3, O_4, O_6 и O_7 включают ровно два двоичных вектора из орбиты O_0. Численные значения M_1 и M_2 для разных значений k представлены в таблице 14.4.

Таблица 14.4 – *Численные значения M_1 и M_2*

k	2	3	4	5	6	7	8	9	10
M_1	3	6	10	15	21	28	36	45	55
M_2	0	1	5	16	42	99	219	466	968

Ограничение количества разновидностей новых орбит только двумя видами и их точное число M_1 и M_2 позволяет оценить среднее число M_{ave}

двоичных векторов, формируемых в результате двукратного маршевого теста. В случае двукратного теста MATS ++ это значение можно получить следующим образом.

Для произвольного значения k, однократное применение теста MATS ++, формирует в них $k + 1$ двоичный вектор (см. утверждение 14.1). Основываясь на общей идее многократного тестирования запоминающих устройств [333], исходные состояния памяти, перед последующим применением маршевого теста, формируются как случайные равномерно распределенные данные. Это означает, что для тех же k ячеек памяти новая орбита, из всех возможных 2^k орбит, генерируется с равной вероятностью. Тогда среднее количество новых двоичных векторов формируемых во время повторного применения теста MATS ++ будет определяться как: $[M_1 \times (k-1) + M_2 \times (k+1)]/2^k$, а значение среднего число M_{ave} двоичных векторов, формируемых в результате двукратного маршевого теста MATS ++ ,определяется соотношением:

$$M_{ave} = (k+1) + \frac{M_1 \times (k-1) + M_2 \times (k+1)}{2^k} = 2(k+1) - \frac{(k+1)^2}{2^k}. \qquad (14.3)$$

Численные значения $M_{ave}/2^k$ для различных значений k приведены в таблице 14.5.

Таблица 14.5 – *Численные значения* $M_{ave}/2^k$

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
$M_{ave}/2^k$	0,937	0,750	0,527	0,339	0,206	0,121	0,069	0,038	0,021

Представленное значение в таблице 14.5, умноженное на 100%, может быть интерпретировано как число генерируемых двоичных векторов в процентном исчислении как результат двукратного применения маршевого теста MATS ++. Для экспериментальной проверки, приведенных выше в таблице 14.5 аналитических результатов, были получены реальные экспериментальные значения для различных значений k. Эти значения являются средней величиной $M_{ave}/2^k$ в процентах, полученной на основе 100 000 ее экспериментальных значений. Результаты экспериментальных результатов приведены в таблице 14.6.

Таблица 14.6 – *Экспериментальные значения* $M_{ave}/2^k$

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
$(M_{ave}/2^k)\times 100\%$	93,76	75,04	52,74	33,98	20,64	12,12	6,91	3,87	2,14

Как можно видеть, экспериментальные и теоретические значения $M_{ave}/2^k$ очень близки и практически одинаковы (см. таблицы 14.5 и 14.6), что объясняется точностью полученных аналитических соотношений (14.1), (14.2) и (14.3). Очевидно, что аналогичные оценки могут быть получены и для трехкратного, четырехкратного и т. д. применений маршевых тестов.

Однако для этого нужно построить достаточно громоздкие и сложные комбинаторные соотношения, сложность которых возрастает с ростом k.

Поэтому в следующем разделе рассмотрим более общую модель для любого значения кратности многократных маршевых тестов.

Утверждение 14.6. Минимальное количество Q_{min} орбит, полученных в результате многократного применения маршевого теста с изменяемыми начальными состояниями, необходимыми для получения всех 2^k возможных двоичных векторов в произвольных k ячейках, должно удовлетворять неравенству:

$$Q_{min} \geq \left\lceil \frac{2^k}{k+1} \right\rceil k = 2, 3, \ldots . \qquad (14.4)$$

Значение Q_{min} может рассматриваться как нижняя граница для минимального числа маршевых тестов типа MATS ++ для достижения исчерпывающего покрытия 2^k двоичными векторами всех состояний в k произвольных ячейках памяти. Эта нижняя граница была получена для наилучшего случая, когда все последующие орбиты включают по k + 1-му новому двоичному вектору, ранее не использованному в предыдущих орбитах, генерируемых MATS ++ подобными маршевыми тестами. В случае маршевых тестов типа March $C-$, учитывая, что каждая последующая итерация их применения формирует не более чем $2k$ новых двоичных векторов Q_{min}, принимает следующий вид:

$$Q_{min} \geq \left\lceil \frac{2^{k-1}}{k} \right\rceil k = 2, 3, \ldots . \qquad (14.5)$$

Согласно (14.4) для k = 3 минимальное число орбит Q_{min} = 2. Это означает, что минимальное число кратности, например, теста MATS + для достижения исчерпывающего покрытия в трех ячейках памяти равно двум. Действительно, следующие пары орбит: O_0, O_5; O_1, O_4; O_2, O_7 и O_3, O_6 генерируют все 2^3 = 8 двоичные векторы для k = 3 (см. таблицу 14.3).

14.3. Модификация адресных последовательностей маршевых тестов

В качестве примера изменения порядка адресов для случая k = 3 в таблице 14.7 приведены 6 орбит [318]. Эти орбиты были получены в результате всех возможных перестановок адресов ЗУ. Первая орбита O_0 является общей орбитой O_g, сгенерированной на основе стандартного возрастающего порядка адресов 0, 1, 2 (см. таблицу 14.7). Остальные орбиты были получены на основе всех возможных перестановок порядка адресации. Для случая изменения порядка адресов верно следующее утверждение [318].

Таблица 14.7 – *Орбиты для* $k = 3$ *и различных последовательностей адресов*

Адреса	0 1 2	0 2 1	1 0 2	1 2 0	2 0 1	2 1 0
Орбиты	O_0	O_1	O_2	O_3	O_4	O_5
P_0	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)	000 (0)
P_1	100 (4)	100 (4)	010 (2)	001 (1)	010 (2)	001 (1)
P_2	110 (6)	101 (5)	110 (6)	101 (5)	011 (3)	011 (3)
P_3	111 (7)	111 (7)	111 (7)	111 (7)	111 (7)	111 (7)

Утверждение 14.7. Для k произвольных ячеек памяти с постоянным начальным состоянием P_0 существует $k!$ орбит как результат изменения порядка адресов k ячеек памяти.

Доказательство утверждения 14.7. Общая орбита O_g имеет структуру, показанную в таблице 14.2. Не нарушая общности, как и в предыдущем случае, предположим, что O_g будет иметь структуру, показанную в таблице 14.2. Эта орбита была получена на основе начального набора, состоящего из всех нулей $P_0 = 0\ 0\ 0\ \ldots\ 0$ и включает $k + 1$ двоичных векторов. Последний набор P_k, независимо от изменения порядка адресов, будет постоянным, и представлять собой двоичный вектор, состоящий из всех единиц. Поэтому любое изменение порядка адресов приводит к не более чем $(k - 1)$-му новому набору в пределах новой орбиты.

Изменение порядка адресов состоит из обычной перестановки столбцов в исходной орбите O_g, для получения новой орбиты O_n. Произвольный столбец орбиты O_g с индексом l состоит из последовательности всех нулей от первого бита начального набора P_0 до первого единичного бита из последовательных всех единиц. Согласно перестановке адресов в O_g в новой орбите P_0, как минимум, два столбца изменили свои позиции. Поэтому всегда существует столбец, новая позиция которого в O_n будет справа по сравнению с исходной позицией в O_g, а второй столбец будет занимать левую позицию. Пусть l-ый столбец сместится вправо, тогда набор P_{l+1} будет иметь вид $1\ \ldots\ 1\ 1\ 1\ \underline{\boldsymbol{0}}\ 0\ 0\ 0\ \ldots\ 0\ \underline{\boldsymbol{1}}\ 0\ \ldots\ 0$. Если из-за перестановки адресов столбец с индексом l изменил свое положение, новая орбита будет иметь хотя бы один новый набор, а именно P_{l+1}. Есть две возможности для P_{l+1}, а именно $P_{l+1} = 1\ \ldots\ 1\ 1\ 1\ \underline{\boldsymbol{0}}\ 0\ 0\ 0\ \ldots\ 0\ \underline{\boldsymbol{1}}\ 0\ \ldots\ 0$ или $P_{l+1} = 1\ \ldots\ 1\ \underline{\boldsymbol{0}}\ 1\ \ldots\ 1\ 1\ 1\ \underline{\boldsymbol{1}}\ 0\ 0\ 0\ \ldots\ 0\ 0\ 0\ \ldots\ 0$. Очевидно, что дальнейшее перемещение другого столбца не возвращает P_{l+1} обратно к его начальному значению $P_{l+1} = 1\ \ldots\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ \ldots\ 0\ 0\ 0\ \ldots\ 0$, за исключением обратной перестановки столбцов. Следовательно, любая перестановка столбцов, вызванная изменением порядка адресов в исходной орбите O_g с $P_0 = 0\ 0\ 0\ \ldots\ 0$, приводит к новой орбите O_n, которая не равна O_g. Что и требовалось доказать.

Структура всех орбит, полученных в результате перестановки адресов, включает в себя два постоянных набора P_0 и P_k, которые зависят только от используемого фона (первоначального состояния памяти). Для всех орбит, представленных в таблице 14.7, $P_0 = 0\ 0\ 0$ и $P_k = 1\ 1\ 1$. По сравнению с изменением начального состояния изменение порядка адресов для $k > 3$ позволяет

получить существенно больше новых орбит. Количество N_B орбит, полученных в результате изменения начального состояния, и число N_A орбит, сгенерированных перестановкой адресов для разных значений k, показаны в таблице 14.8.

Таблица 14.8 – *Количество орбит при изменении начального состояния ЗУ и адресной последовательности*

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
$N_B = 2^k$	4	8	16	32	64	128	256	512	1024
$N_A = k!$	2	6	24	120	720	5040	40320	362880	3628800

Для случая изменения порядка адресов верно следующее утверждение [318].

Утверждение 14.8. Минимальное число орбит Q_{min}, полученное в результате многократного тестирования с изменением порядка адресов на основе MATS ++ подобного теста, которое необходимо для получения всех возможных двоичных наборов в k произвольных ячейках памяти, определяется как:

$$Q_{min} = \binom{k}{\lceil k/2 \rceil}. \qquad (14.6)$$

Доказательство утверждения 14.8. Каждая орбита генерирует только один набор, состоящий из заранее определенного числа единиц и нулей. Если начальный набор P_0 состоит из всех нулей, то набор P_1 для любой новой орбиты всегда будет содержать одну единицу и $k - 1$ нулей. Например, для k = 4 орбиты, полученные в соответствии с перестановкой адресов, образуют множество наборов P_1 { 0 0 0, 0 1 0 0, 0 0 1 0, 0 0 0 1}, состоящих из одной 1 и трех 0. Для тестового набора P_2 множество возможных значений состоит из {1 0 0 1, 1 0 1 0, 1 1 0 0, 0 1 0 1, 0 1 1 0, 0 0 1 1}, включающих в себя две 1 и два 0. Для P_3 генерируются следующие значения {1 1 1 0, 1 1 0 1, 1 0 1 1, 0 1 1 1}, состоящие из комбинаций трех 1 и одного 0. Для обеспечения псевдоисчерпывающего теста для k ячеек памяти необходимо сгенерировать все двоичные наборы, состоящие из $\lceil k/2 \rceil$ единиц и $\lfloor k/2 \rfloor$ нулей. Количество таких наборов максимально по сравнению с наборами, имеющими другое соотношение нулей и единиц. Это следует из комбинаторных свойств r-комбинаций для заданного множества k, где $k > 1$, $0 < r < k$ и $r \neq \lceil k/2 \rceil$:

$$\binom{k}{\lceil k/2 \rceil} > \binom{k}{r}.$$

Таким образом, Q_{min} будет равно числу двоичных векторов, составленных из $\lceil k/2 \rceil$ единиц и $\lfloor k/2 \rfloor$ нулей (14.6). Эта сумма будет одинаковой для случая $\lfloor k/2 \rfloor$ единиц и $\lceil k/2 \rceil$ нулей, для которых $\lfloor k/2 \rfloor + \lceil k/2 \rceil = k$, что следует из следующего свойства комбинаторных комбинаций:

$$\binom{k}{\lceil k/2 \rceil} = \binom{k}{k - \lceil k/2 \rceil}.$$

Что и требовалось доказать.

Далее определим максимальное количество орбит Q_{max}, необходимое для формирования псевдо-исчерпывающего теста для произвольного значения k, которое позволяет сгенерировать все 2^k тестовых комбинаций. Как следует из утверждения 14.1, применение одной орбиты независимо от ее типа, обеспечивает $k + 1$ двоичных наборов, каждый из которых состоит из k бит. В худшем случае каждая последующая орбита, будет содержать, по меньшей мере, один новый двоичный набор по отношению к двоичным наборам, сгенерированным ранее использованными орбитами. Тогда верхняя граница для Q_{max} будет вычислена как (14.7):

$$Q_{max} = 1 + (2^k - (k+1)) = 2^k - k. \qquad (14.7)$$

Оценка (14.7) Q_{max} представляет собой верхнюю границу кратности маршевых тестов, необходимой для реализации псевдо-исчерпывающего тестирования.

14.4. Математическая модель псевдо-исчерпывающего тестирования ЗУ

Как было показано в предыдущих разделах, для генерирования всех 2^k возможных двоичных комбинаций в произвольных k ячейках памяти, возможно применение многократного маршевого теста. Все последующие итерации применения выбранного теста в рамках многократного тестирования должны реализовываться с разными начальными состояниями или/и различными адресными последовательностями, что является необходимым условием для обеспечения формирования новых орбит и, соответственно, псевдо-исчерпывающего тестирования ЗУ.

Предположим, что последовательность начальных состояний и адресные последовательности формируются случайным образом, а начальные состояния и адреса представляют собой случайные равномерно распределенные и независимые переменные. Тогда рассматриваемая задача определения среднего числа количества орбит Q_{ave}, которые необходимы для формирования исчерпывающего теста на произвольных k ячейках памяти, как было показано в [318], сводится к классической задаче теории комбинаторных вероятностей. А именно, к *задаче собирателя купонов* (*Coupon Collector's Problem*), когда в качестве купонов рассматривались равномерно распределенные орбиты, формируемые маршевыми тестами [318]. Предположив, что в качестве купона будет рассматриваться один двоичный вектор, а не орбита, постановка классической задачи собирателя купонов в терминах многократного

тестирования (псевдо-исчерпывающего тестирования) запоминающих устройств, может быть сформулирована следующим образом.

Определение 14.1. Применение одной итерации многократного маршевого теста генерирует один из 2^k возможных двоичных наборов для k произвольных ячеек памяти, который является равномерно распределенной случайной величиной с вероятностью $1/2^k$.

Отметим, что определение 14.1 апеллирует не орбитой, а двоичным вектором, что, по сути, означает применение только одной фазы, фазы инициализации применяемого произвольного маршевого теста. Эти ограничения позволяют использовать задачу собирателя купонов для описания случая изменяемых начальных состояний памяти при формировании псевдо-исчерпывающих тестов на базе многократного применения маршевых тестов памяти. В качестве меры сложности псевдо-исчерпывающего теста ранее была показана эффективность применения среднего числа кратности Q_{ave}, используемого многократного теста. Эта величина Q_{ave}, для случая формирования одного двоичного вектора для каждой итерации теста в соответствии с определением 14.1, показывает среднее количество случайной выборки купонов (двоичных векторов) для получения всех купонов (2^k векторов), как минимум по одному разу. Таким образом, обеспечивается необходимое условие получения исчерпывающего теста [318]. Это значение вычисляется в соответствии с выражением [63, 318]:

$$Q_{ave} = 1 + \frac{2^k}{2^k - 1} + \frac{2^k}{2^k - 2} + ... + \frac{2^k}{2} + 2^k = 2^k \sum_{n=1}^{2^k} \frac{1}{n} . \qquad (14.8)$$

При больших значениях 2^k можно использовать *аппроксимацию Эйлера гармонического ряда* (14.8) в виде формулы.

$$Q_{ave} = 2^k (\log_e 2^k + \gamma) .$$

Величина $\gamma \approx 0{,}57722$ является *постоянной величиной Эйлера-Маскерони* [63].

Численные значения Q_{ave} для ряда значений k показаны в таблице 14.9.

Таблица 14.9 – *Численные значения* Q_{ave}

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
2^k	4	8	16	32	64	128	256	512	1024
Q_{ave}	8,33	21,74	54,09	129,87	303,61	695,44	1567,83	3490.05	7689,39

Приведенные численные значения Q_{ave} показывают, что многократное применение маршевого теста с ограничениями, принятыми в определении 14.1, позволяет воспроизводить псевдо-исчерпывающий тест для произвольных k ячеек памяти с умеренной временной сложностью. Среднее число Q_{ave} кратности теста принимает приемлемые значения. Например, для формирования $2^4 = 16$ двоичных векторов, необходимых для обнаружения сложных

неисправностей запоминающих устройств, таких как: *PPSF*4 [318], среднее значение кратности Q_{ave} многократного маршевого теста равно 54,09.

Следует отметить, что в рамках многократного тестирования памяти с изменяемыми начальными состояниями и/или изменяемыми адресными последовательностями во время одной итерации теста генерируется не один двоичный вектор, а так называемая орбита, состоящая из $k + 1$ двоичных векторов. Поэтому обобщенная математическая модель, полученная для случая одного двоичного вектора на одну итерацию теста, должна быть адаптирована для реальной процедуры тестирования памяти.

14.5. Псевдо-исчерпывающее тестирование запоминающих устройств

Как было показано в предыдущих разделах, классический маршевый тест позволяет генерировать орбиты, содержащие $k + 1$ различных двоичных векторов в случае тестов типа MATS ++ или $2k$ векторов для March $C-$ подобных тестов. Все двоичные векторы, в обоих случаях, различны и зависят от исходного начального состояния памяти, определяемого для k ячеек памяти начальным вектором P_0. Применяемый маршевый тест генерирует орбиту исходя из структуры теста и начального вектора P_0, который является случайной равномерно распределенной величиной. Для упрощения дальнейших рассуждений предположим, что в одной орбите двоичные векторы являются независимыми и равномерно распределенными случайными величинами, а вероятность формирования каждого из них равняется $1/2^k$. Основываясь на принятых допущениях, и предположении, что оценка Q_{ave} (14.8), полученная для общей математической модели, может быть адаптирована для многократных классических маршевых тестов с изменяемым начальным состоянием тестируемых запоминающих устройств. Тогда среднее значение Q_{ave}(MATS ++) кратности для многократных тестов типа MATS ++, необходимое для генерирования всех 2^k двоичных векторов, можно вычислить как Q_{ave}(MATS ++) = $Q_{ave}/(k + 1)$. В случае тестов типа March $C-$ может использоваться следующее соотношение: Q_{ave}(March $C-$) = $Q_{ave}/(2k)$. Отметим, что независимо от применяемого теста, двоичные векторы, входящие в орбиту, формируются без восстановления, т. е. они не повторяются, а математическая модель задачи собирателя купонов рассматривается для случая с восстановлением. Тогда теоритические оценки кратности многократных маршевых тестов Q_{ave}(MATS ++) = $Q_{ave}/(k + 1)$ и Q_{ave}(March $C-$) = $Q_{ave}/(2k)$ могут рассматриваться как верхние оценки, соответственно Q_{max}(MATS ++) и Q_{max}(March $C-$).

В следующих двух таблицах 14.10 и 14.11 представлены теоретические и экспериментальные оценки кратности для двух множеств многократных маршевых тестов памяти, обеспечивающих формирование всевозможных 2^k двоичных наборов для k произвольных ячеек памяти (псевдо-исчерпывающий тест). Наряду с экспериментальной оценкой Q_{ave} были полу-

чены значения максимальной Q_{max} и минимальной кратности используемых тестов. Величина Q_{ave} представляет собой среднее значение количества итераций тестов в процентах, полученное на основании 100 000 экспериментальных значениях.

Таблица 14.10 – *Теоретические* (Теор.) и *экспериментальные* (Эксп.) *значения для* Q_{ave}(MATS++)

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
2^k	4	8	16	32	64	128	256	512	1024
Теор. Q_{ave}	2,77	5,44	10,82	21,64	43,37	86,93	174,20	349.00	699,03
Эксп. Q_{ave}	-	4,44	9,42	19,54	40,90	83,57	170,36	341.10	680,86
Эксп. Q_{min}	-	2	4	7	17	41	96	201	413
Эксп. Q_{max}	-	16	31	60	110	201	366	706	1405

Таблица 14.11 – *Теоретические* (Теор.) *и экспериментальные* (Эксп.) *значения для* Q_{ave}(March *C*–)

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
2^k	4	8	16	32	64	128	256	512	1024
Теор. Q_{ave}	2,77	3,62	6,76	12,98	25,30	49,67	97,98	193.89	384,47
Эксп. Q_{ave}	-	2,33	4,42	9,34	19,55	40,55	82,62	169.32	339,48
Эксп. Q_{min}	-	2	2	4	7	18	37	86	213
Эксп. Q_{max}	-	7	17	28	65	113	207	391	769

Анализ приведенных данных позволяет сделать два основных вывода. Прежде всего, численные значения показывают, что псевдо-исчерпывающее тестирование, реализуемое многократными маршевыми тестами на основе изменяемого начального состояния памяти, можно описать в терминах вероятностных математических моделей, использующих геометрическое распределение. А именно, применяя результаты, полученные для задачи собирателя купонов можно оценить среднее число Q_{ave} многократных тестов, которое очень близко к реальным экспериментальным значениям. Эти значения имеют вполне приемлемые величины, что позволяет использовать многократные тесты памяти с изменяемым начальным состоянием для реализации псевдо-исчерпывающего тестирования памяти даже для случаев, когда $k = 10$.

Следует подчеркнуть, что среднее значение Q_{ave} для тестов MATS ++ примерно в два раза выше, в сравнении с тестами типа March *C*–. Это объясняется тем фактом, что второе множество тестов генерирует приблизительно в два раза больше двоичных векторов в произвольных k ячейках памяти.

Оценим среднее число Q_{ave} количества орбит, которые необходимы для формирования псевдо-исчерпывающего теста на произвольных k ячейках ЗУ за счет изменения последовательности адресов маршевого теста. Эту оценку получим для случая выборки орбит с восстановлением. Для этого нужно проанализировать всевозможные сочетания орбит, которые обеспечат 2^k тестовых наборов в k ячейках ЗУ, что даже для малых значений k является затруднительно в силу большого числа орбит, равного $k!$.

Поэтому, используя результаты, полученные при выводе оценки Q_{min} (14.6), сформулируем необходимое условие формирования исчерпывающего теста на произвольных k ячейках ОЗУ в виде утверждения 14.9.

Утверждение 14.9. Необходимым условием для формирования псевдо-исчерпывающего теста в произвольных k из N ячейках ЗУ является формирование для этих ячеек ЗУ множества орбит, в которых все двоичные коды, содержащие $\lfloor k/2 \rfloor$ единиц и $k - \lfloor k/2 \rfloor$ нулей, будут сгенерированы хотя бы по одному разу.

Количество таких орбит, в зависимости от k, определяется величиной Q_{min} (14.6). Для небольших величин k их значения приведены в таблице 4.12, где Q_{max} представляет собой оценку (14.7), а Q_{tou} общее количество орбит.

Например, как следует из таблицы 14.12, для k = 4, необходимым является формирование такого подмножества орбит, в которых будут присутствовать все 6 двоичных векторов (0 0 1 1, 0 1 0 1, 1 0 0 1, 0 1 1 0, 1 0 1 0, 1 1 0 0). Для случая k = 6, соответственно, необходимо сгенерировать орбиты, содержащие все 20 двоичных кодов вида (0 0 0 1 1 1, 0 0 1 0 1 1, 0 1 0 0 1 1, …, 1 1 1 0 0 0).

Таблица 14.12 – *Численные значения Q_{tou}, Q_{max} и Q_{min}*

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Q_{tou}	2	6	24	120	720	5040	40320	362880	3628800
Q_{min}	2	3	6	10	20	35	70	126	252
Q_{max}	3	5	12	27	58	121	248	503	1014

Отметим, что конкретная орбита для произвольного k содержит только один двоичный вектор, который включает $\lfloor k/2 \rfloor$ единиц и $k - \lfloor k/2 \rfloor$ нулей. Указанные двоичные векторы присутствуют одинаковое число раз во всевозможных орбитах для заданного значения k. Таким образом, при равновероятной выборке одной из орбит с вероятностью $1/Q_{min}$ будет выбран один из Q_{min} векторов, который включает $\lfloor k/2 \rfloor$ единиц и $k - \lfloor k/2 \rfloor$ нулей.

В подобной интерпретации рассматриваемая задача определения среднего числа количества орбит Q_{ave}, которые необходимы для формирования исчерпывающего теста на произвольных k ячейках ЗУ, сводится к классической *задаче собирателя купонов* (*Coupon Collector's Problem*) [63]. Для случая одной коллекции равновероятных купонов (Q_{min} орбит) среднее количество Q_{ave} случайной выборки купонов (орбит) для получения всех купонов, как минимум по одному разу, (обеспечение необходимого условия получения псевдо-исчерпывающего теста см. утверждение 14.9) определяется согласно соотношению (14.8) [63]. Численные значения Q_{ave} для случая изменения последовательности адресов приведены в таблице 14.13.

Таблица 14.13 – *Численные значения* Q_{ave}

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Q_{min}	2	3	6	10	20	35	70	126	252
Q_{ave}	3	5,50	14,70	29,29	71,95	145,14	338,06	683,52	1538,58

Анализ величин Q_{ave} показывает, что для обеспечения исчерпывающего множества двоичных комбинаций для произвольных k из N ячеек ЗУ и, соответственно, реализации псевдо-исчерпывающего теста ЗУ для заданного k, необходимо использовать многократный тест ЗУ с изменяемыми адресными последовательностями. При этом, как видно из таблицы 14.13, среднее значение кратности теста принимает приемлемые значения. Например, для обнаружения всего множества сложных неисправностей *PPSF*5, среднее значение кратности теста равняется 29,29, что обеспечивает их 100% обнаружение. Отметим, что данное утверждение справедливо для случая генерирования адресов без повторений и простейшего теста ЗУ типа $\{\Uparrow (w0); \Uparrow(r0, w1)\}$.

В качестве меры эффективности псевдо-исчерпывающих тестов ЗУ была выбрана ее средняя сложность Q_{ave}, определяемая средним количеством многократных тестов ЗУ с изменяемыми адресными последовательностями. При оценке данной величины в предыдущем разделе, принимался ряд ограничений и допущений, что позволило аналитически оценить ее значение, а также получить оценки значений Q_{min} и Q_{max} (см. таблицы 14.12 и 14.13).

Для экспериментального подтверждения полученных аналитических результатов был проведен статистический анализ, для определения среднего значения Q_{ave} многократного применения теста $\{\Uparrow(w0); \Uparrow(r0, w1)\}$ с изменяемыми адресными последовательностями. В первом случае адреса ЗУ генерировались без повторений, что несколько усложняло их формирование, а во втором с повторениями. В обоих случаях средние оценочные значения Q_{ave} были получены на основании 10 000 экспериментов, а их конкретные величины приведены, соответственно, в таблицах 14.14 и 14.15.

Таблица 14.14 – *Экспериментальные численные значения* Q_{ave} *для случая без повторения адресов*

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Q_{min}	2	3	6	14	31	79	197	499	998
Q_{ave}	2,99	6,69	15,53	35,02	77,21	168,06	362,96	762	1634,98
Q_{max}	14	32	66	122	242	499	898	1850	3037

Таблица 14.15 – *Экспериментальные численные значения Q_{ave} для случая с повторением адресов*

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Q_{min}	2	3	6	15	37	98	235	559	1267
Q_{ave}	3,96	9,14	20,50	45,69	98,89	211,60	446,86	950,15	1987,92
Q_{max}	21	44	78	183	300	5669	1364	2466	4135

Графически значения Q_{ave} для двух случаев формирования случайных адресов (с повторением и без повторения) представлены на рис. 14.3.

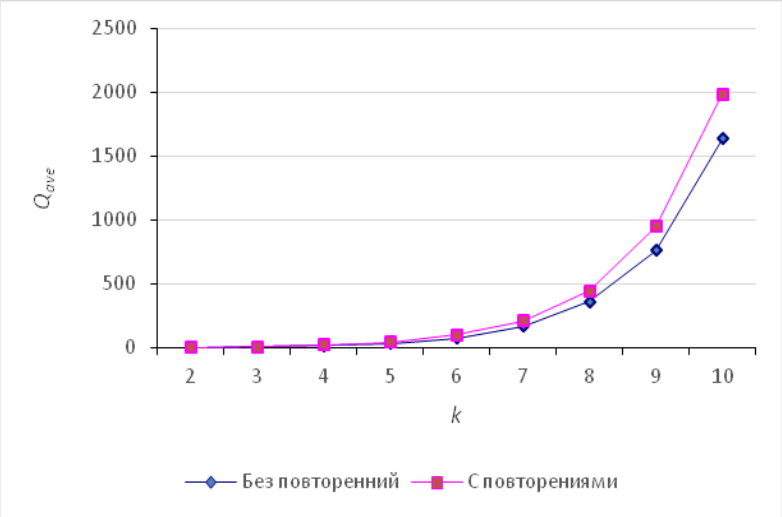


Рисунок 14.3 – *Экспериментальные значения Q_{ave}*

Анализ приведенных результатов показывает, что аналитическая оценка среднего количества маршевых тестов подтверждается данными, приведенными в таблицах 14.14 и 14.15, а также на рис. 14.3. Аналитические значения Q_{ave} по сути являются нижней оценкой средней сложности многократного применения простейшего маршевого теста вида $\{\Uparrow(w0); \Uparrow(r0, w1)\}$ для реализации псевдо-исчерпывающего теста ЗУ для заданного k. Подобный псевдо-исчерпывающий тест ЗУ является результатом многократного применения маршевого теста с изменяемыми адресными последовательностями. Аналогичный результат будет получен для любого MATS+ подобного теста. В таблице 14.16 приведены экспериментальные результаты для теста MATS+ и адресных последовательностей без восстановления.

Таблица 14.16 – *Численные значения Q_{ave} для теста* MATS+ *без повторения адресов*

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Q_{min}	2	3	6	15	35	82	217	479	1028
Q_{ave}	3	6,7	15,42	32,02	76,77	166,9	361	771	1624
Q_{max}	11	27	52	89	214	384	991	1765	3488

Очевидно, что использование классического маршевого теста, такого, как March *C−*, позволит обеспечить псевдо-исчерпывающее тестирование за меньшее число итераций, что подтверждается экспериментальными результатами, приведенными в таблице 14.17.

Как отмечалось ранее, все множество маршевых тестов ЗУ, применяемых для тестирования ЗУ, основано на использовании последовательной выборки адресов ячеек ЗУ, в том числе и случайной с повторяющимися адресами и без повторяющихся адресов. Во втором случае активно используются генераторы псевдослучайных последовательностей, которые могут гарантировать процедуру выборки без восстановления.

Таблица 14.17 – *Численные значения Q_{ave} для теста* March *C−* *без повторения адресов*

k	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Q_{min}	1	2	3	6	12	33	82	185	469
Q_{ave}	1	2,5	5,88	13,79	31,93	72,3	157	345	730
Q_{max}	1	8	17	49	89	181	361	853	1619

В то же время неповторяющиеся адресные последовательности для всего ЗУ могут формировать повторяющиеся адресные последовательности и соответственно одинаковые орбиты для произвольных k из N ячеек ЗУ, где значение k существенно меньше емкости ЗУ. Последнее утверждение и иллюстрируется близостью результатов эксперимента для двух способов формирования случайных адресных последовательностей (см. рис. 14.3).

Сравнивая полученные результаты с результатами для случая псевдоисчерпывающих тестов на базе многократных маршевых тестов с изменяемыми адресными последовательностями [63], можно отметить, что при изменении начальных состояний в среднем кратность маршевых тестов уменьшается в два раза. Например, экспериментальное среднее значение Q_{ave} для многократных тестов MATS ++, при изменении адресов без их повторения, для $k = 6$, равняется 76,77, а $Q_{min} = 35$ и $Q_{max} = 214$ [318]. В случае изменяемых начальных состояний запоминающих устройств, при применении того же теста, для $k = 6$, имеем: $Q_{ave} = 43{,}37$, $Q_{min} = 17$ и $Q_{max} = 110$ (см. таблицу 14.10).

Глава 15. Исчерпывающие детерминированные тесты

15.1. Детерминированные тестовые воздействия

В предыдущих разделах подробно рассматривался достаточно широкий набор различных детерминированных тестовых последовательностей, таких как: счетчиковые (пересчетные) последовательности (см. главу 5), квазислучайные последовательности (см. главу 6) псевдослучайные последовательности (см. главу 7) и различные их модификации, которые применяются для тестирования современных вычислительных систем. Практически все представленные ранее последовательности можно рассматривать как детерминированные последовательности, которые, в каждом конкретном случае характеризуются своими специфическими свойствами. Среди многообразия различных свойств таких последовательностей выделяют свойство случайности, равномерности, максимальной длины и ряд других их характеристик и особенностей. Общим, принципиально важным свойством всех детерминированных тестовых последовательностей является их свойство воспроизводимости (повторяемости), которое, по сути, и является объединяющим их критерием. Для детерминированных тестовых последовательностей справедливо следующее определение.

Определение 15.1. Детерминированной тестовой последовательностью является последовательность, заданная аналитически в виде рекуррентной функции, зависящей от предыдущих тестовых наборов и дискретного времени, представленного в виде последовательных индексов (тактов) наборов, которая инициализируется начальными тестовыми наборами.

В качестве примера рассмотрим детерминированную тестовую последовательность *Фибоначчи* (*Fibonacci*), в которой первые два инициализирующие начальные значения равны либо 1 и 1, либо 0 и 1, а каждое последующее значение набора равно сумме двух предыдущих. Аналитически такая последовательность задается следующим рекуррентным соотношением:

$$F_0 = 1,\ F_1 = 1, \quad F_k = F_{k-1} + F_{k-2}, \quad k = 2,3,4,... \tag{15.1}$$

Для $F_0 = 1$ и $F_1 = 1$ первые семь тестовых наборов последовательности Фибоначчи приведены в десятичной и двоичной системах счисления в таблице 15.1.

Таблица. 15.1 – *Тестовая последовательность Фибоначчи*

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_6	F_6	F_6
1	1	2	3	5	8	13	21	34	55
000001	000001	000010	000011	000101	001000	001101	010101	100010	110111

Для детерминированной тестовой последовательности и определенного начального ее значения в конкретный дискретный момент времени текущие значения ее наборов строго предопределены.

Несомненным достоинством детерминированных тестовых последовательностей является наличие их абстрактных математических моделей, что позволяет эффективно реализовывать их в виде программных приложений, а также в виде аппаратурных устройств вычислительных систем.

Среди большого многообразия детерминированных тестовых последовательностей доминируют так называемые *M*-последовательности и разнообразные их модификации. Они нашли широкое применение для реализации самотестирования современных вычислительных систем. В рамках современных методологий самотестирования подобные последовательности применяются для реализации вероятностного тестирования и управляемого вероятностного тестирования (см. главы 3 и 4), псевдослучайного тестирования (см. главу 7), исчерпывающего и псевдо-исчерпывающего тестирования (см. главы 8 и 9). Доминирующее положение *M*-последовательности занимают при разработке как генераторов тестовых последовательностей, так и сигнатурных анализаторов в рамках методов компактного тестирования (см. главу 10). Большое множество примеров использования таких последовательностей можно привести и для случая тестирования запоминающих устройств, в особенности для генерирования адресных последовательностей маршевых тестов (см. главы 11 и 12).

Вторым по значимости и применимости в качестве детерминированных тестовых последовательностей, является множество счетчиковых двоичных тестовых последовательностей. Подобные последовательности уже нашли свое отражение в данной работе в части реализации многократного вероятностного тестирования (см. главу 5), а также как одна из разновидностей неслучайных тестовых последовательностей, формируемых в рамках обобщенной модели генерирования квази-случайных последовательностей (см. раздел 6.5)

Методы формирования различных детерминированных тестовых последовательностей приведены в главе 6. Основное внимание в данной главе уделено квази-случайным тестовым последовательностям. В тоже время обобщение метода формирования квази-случайных последовательностей Соболя позволило применить этот метод и для формирования большого спектра других детерминированных последовательностей, таких как, например, счетчиковых, формируемых на базе двоичных счетчиков. Важной особенностью счетчиковых последовательностей является возможность формирования на их основе исчерпывающих тестовых последовательностей, включающих всевозможные двоичные комбинации заданной разрядности.

15.2. Счетчиковые тестовые последовательности и их модификации

В дальнейшем под счетчиковой тестовой последовательностью $A = A(0)\ A\ (1)\ A(2)\ \ldots\ A(N-2)\ A(N-1)$, где $A(j) \in \{0, 1, 2, \ldots, N-1\}$, $j \in \{0, 1, 2, \ldots, N-1\}$ и $N = 2^m - 1$, будем понимать последовательность, состоящую из 2^m

двоичных наборов $A = a_{m-1}\, a_{m-2} \ldots a_2\, a_1\, a_0$. Эта последовательность формируется в соответствии с алгоритмом двоичного суммирующего счетчика разрядностью m, в соответствии со следующим рекуррентным соотношением:

$$A_C(j) = A_C(j-1), \quad j = 1, 2, 3, \ldots, 2^m - 1. \tag{15.2}$$

Начальным тестовым набором $A(0)$ (исходное состояние счетчика) может быть любой m-разрядный двоичный код, однако, как правило, принимают, что $A(0) = 0\ 0\ 0 \ldots 0$.

Отметим, что на основании (15.2) генерируется циклическая тестовая последовательность такая, что для произвольного начального тестового набора $A(0) = A(j)$ конечным набором будет $A((j + 2^m - 1) \bmod 2^m) = A(j) - 1$. Так для начального тестового набора $A(0) = (0)_{10} = (0\ 0\ 0 \ldots 0)_2$ имеем $A(2^m - 1) = (2^m - 1)_{10} = (1\ 1\ 1 \ldots 1)_2$, где индексы определяют основание системы счисления.

Для получения модифицированных счетчиковых последовательностей возможно применение двух простейших приемов, а именно перестановку битовых (разрядных) последовательностей a_i, $i \in \{0, 1, 2, \ldots, m-1\}$, и перестановку тестовых наборов $A(j)$, $j \in \{0, 1, 2, \ldots, N-1\}$ [331]. Для m-разрядной счетчиковой последовательности существует $m!$ последовательностей, полученных в результате перестановки бит a_i исходного тестового набора. Например, для двухразрядных наборов $A = a_1\, a_0$ имеем $m! = 2! = 2$ тестовые последовательности, а для $m = 3$ количество модификаций тестовых наборов $A = a_2\, a_1\, a_0$ достигает $m! = 3! = 6$. Результаты модификации счетчиковых тестовых последовательностей для $m = 2$ и $m = 3$ приведены в таблице 15.2 [216, 331], где A_{b1} означает первый результат перестановки бит a_i тестовых наборов исходной тестовой последовательности A.

Количество последовательностей, получаемых согласно описанному алгоритму, велико и в зависимости от величины m численные значения этого количества $m!$ приведены в таблице 15.3.

Таблица 15.2 – *Тестовые последовательности как результат перестановок разрядов исходной последовательности*

$m = 2$		$m = 3$					
A_{b1} $a_1\, a_0$	A_{b2} $a_0\, a_1$	A_{b1} $a_2\, a_1\, a_0$	A_{b2} $a_2\, a_0\, a_1$	A_{b3} $a_1\, a_2\, a_0$	A_{b4} $a_1\, a_0\, a_2$	A_{b5} $a_0\, a_2\, a_1$	A_{b6} $a_0\, a_1\, a_2$
0 0	0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
		0 0 1	0 1 0	0 0 1	0 1 0	1 0 0	1 0 0
0 1	1 0	0 1 0	0 0 1	1 0 0	1 0 0	0 0 1	0 1 0
		0 1 1	0 1 1	1 0 1	1 1 0	1 0 1	1 1 0
1 0	0 1	1 0 0	1 0 0	0 1 0	0 0 1	0 1 0	0 0 1
		1 0 1	1 1 0	0 1 1	0 1 1	1 1 0	1 0 1
1 1	1 1	1 1 0	1 0 1	1 1 0	1 0 1	0 1 1	0 1 1
		1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1

Таблица 15.3 – *Количество счетчиковых тестовых последовательностей*

m	5	6	8	10	12	16	18	20
$m!$	120	720	$4{,}0\times10^{4}$	$3{,}6\times10^{6}$	$4{,}8\times10^{8}$	$2{,}1\times10^{13}$	$6{,}4\times10^{15}$	$2{,}4\times10^{18}$

Для больших величин m значение $m!$ можно оценить на основании формулы *Стиглинга* (*Stirling's approximation*):

$$m! \approx m^m e^{-m} \sqrt{2\pi m}.$$

Несомненным недостатком описанного подхода является сложность его аппаратной реализации, которая предполагает как минимум m мультиплексоров с m адресными входами и m m-разрядных регистров [331].

Новые счетчиковые тестовые последовательности могут быть также получены в результате перестановки тестовых наборов в счетчиковой последовательности [216, 331]. Количество таких последовательностей равняется $2^m!$ [331]. Например, для $m = 2$ можно получить $2^2! = 24$ различных счетчиковых тестовых последовательностей. Эти последовательности приведены в таблице 15.4, где A_{a1} означает исходную последовательность A, а остальные последовательности являются результатом перестанови наборов в A_{a1}.

Таблица 15.4 – *Результат перестановок тестовых наборов для* $m = 2$

A_{a1}	A_{a2}	A_{a3}	A_{a4}	A_{a5}	A_{a6}	A_{a7}	A_{a8}	A_{a9}	A_{a10}	A_{a11}	A_{a12}
0 0	0 0	0 0	0 0	0 0	0 0	0 1	0 1	0 1	0 1	0 1	0 1
0 1	0 1	1 0	1 0	1 1	1 1	0 0	0 0	1 0	1 0	1 1	1 1
1 0	1 1	0 1	1 1	0 1	1 0	1 0	1 1	0 0	1 1	0 0	1 0
1 1	1 0	1 1	0 1	1 0	0 1	1 1	1 0	1 1	0 0	1 0	0 0
A_{a13}	A_{a14}	A_{a15}	A_{a16}	A_{a17}	A_{a18}	A_{a19}	A_{a20}	A_{a21}	A_{a22}	A_{a23}	A_{a24}
1 0	1 0	1 0	1 0	1 0	1 0	1 1	1 1	1 1	1 1	1 1	1 1
0 0	0 0	0 1	0 1	1 1	1 1	0 0	0 0	0 1	0 1	1 0	1 0
0 1	1 1	0 0	1 1	0 0	0 1	0 1	1 0	0 0	1 0	0 0	0 1
1 1	0 1	1 1	0 0	0 1	0 0	1 0	0 1	1 0	0 0	0 1	0 0

Из приведенной выше таблицы 15.4 видно, что все 24 последовательности тестовых наборов были получены из 6 оригинальных последовательностей бит d_i, где $i \in \{1, 2, \ldots, 6\}$. Эти последовательности бит d_i можно описать, используя две оригинальные последовательности бит a_1 и a_0 исходной счетчиковой последовательности $A = a_1\, a_0$. Тогда получим, что $d_1 = a_1 = 0\ 0\ 1\ 1$, $d_2 = a_0 = 0\ 1\ 0\ 1$, $d_3 = a_1 \oplus a_0 = 0110$, $d_4 = \overline{a_1} = 1\ 1\ 0\ 0$, $d_5 = \overline{a_0} = 1\ 0\ 1\ 0$ и $d_6 = \overline{(a_1 \oplus a_2)} = 1\ 0\ 0\ 1$. Таким образом, все 24 последовательности можно сформировать из 2 оригинальных последовательностей бит a_1 и a_0. Покажем это на примере, приведенном в таблице 15.4. Действительно, $A_{a1} = d_1\, d_2 = a_1\, a_0$; $A_{a2} = d_1\, d_3 = a_1(a_1 \oplus a_0)$; $A_{a3} = d_2\, d_1 = a_0\, a_1$; $A_{a4} = d_3\, d_1 = (a_1 \oplus a_0)\, a_1$; $A_{a5} = d_2\, d_3 = a_0\, (a_1 \oplus a_0)$; $A_{a6} = d_3\, d_2 = (a_1 \oplus a_0)\, a_0$; $A_{a7} = d_1\, d_6 = a_1\, \overline{(a_1 \oplus a_2)}$; $A_{a8} = d_1\, d_5 = a_1\, \overline{a_0}$;

$A_{a9} = d_2\, d_6 = a_0\, \overline{(a_1 \oplus a_2)}$; $A_{a10} = d_3\, d_5 = (a_1 \oplus a_0)\ \overline{a_0}$; $A_{a11} = d_2\, d_4 = a_0\, \overline{a_1}$; $A_{a12} = d_3\, d_4 = (a_1 \oplus a_0)\ \overline{a_1}$; $A_{a13} = d_6\, d_1 = \overline{(a_1 \oplus a_2)}\, a_1$; $A_{a14} = d_5\, d_1 = \overline{a_0}\, a_1$; $A_{a15} = d_6\, d_2 = \overline{(a_1 \oplus a_2)}\, a_0$; $A_{a16} = d_5\, d_3 = \overline{a_0}\ (a_1 \oplus a_0)$; $A_{a17} = d_4\, d_2 = \overline{a_1}\, a_0$; $A_{a18} = d_4\, d_3 = \overline{a_1}\ (a_1 \oplus a_0)$; $A_{a19} = d_6\, d_5 = \overline{(a_1 \oplus a_2)}\ \overline{a_0}$; $A_{a20} = d_5\, d_6 = \overline{a_0}\ \overline{(a_1 \oplus a_2)}$; $A_{a21} = d_6\, d_4 = \overline{(a_1 \oplus a_2)}\ \overline{a_1}$; $A_{a22} = d_5\, d_4 = \overline{a_0}\ \overline{a_1}$; $A_{a23} = d_4\, d_6 = \overline{a_1}\ \overline{(a_1 \oplus a_2)}$; $A_{a24} = d_4,\, d_5 = \overline{a_1}\ \overline{a_0}$.

Несмотря на большое количество новых счетчиковых тестовых последовательностей, получаемых в результате перестановок тестовых наборов, данный подход сложно реализовать практически из-за большой аппаратурной избыточности подобного генератора. Поэтому на практике применяются более простые модификации исходных детерминированных тестовых последовательностей, которые используют различные свойства счетчиковых последовательностей. Рассмотрим некоторые из них.

Свойство 15.1. Счетчиковая тестовая последовательность $A = a_{m-1}\, a_{m-2}\, \ldots\, a_2\, a_1\, a_0$, состоит из всех возможных 2^m тестовых наборов (двоичных комбинаций $a_{m-1}\, a_{m-2}\, \ldots\, a_2\, a_1\, a_0$) в произвольной упорядоченной последовательности, причем каждый тестовый набор генерируется только один раз.

Последовательности бит a_i, $i \in \{0, 1, 2, \ldots, m-1\}$ тестовой последовательности $A = a_{m-1}\, a_{m-2}\, \ldots\, a_2\, a_1\, a_0$ обладают следующими простейшими свойствами [19, 105, 199, 314].

Свойство 15.2. Для любой последовательности бит a_i последовательности A существует 2^{m-1} различных двоичных комбинаций $a_{m-1}\, a_{m-2}\, \ldots\, a_{i+1}\, a_{i-1}\, \ldots\, a_2\, a_1\, a_0$ при $a_i = 0$ и такое же число комбинаций $a_{m-1}\, a_{m-2}\, \ldots\, a_{i+1}\, a_{i-1}\, \ldots\, a_2\, a_1\, a_0$ при $a_i = 1$.

Свойство 15.3. Для любых двух последовательностей бит a_i и a_j, где $i \neq j$ в последовательности A присутствует 2^{m-2} различных двоичных комбинаций, когда a_i и a_j принимают значения 00, 01, 10 и 11, соответственно.

Для общего случая, когда число последовательностей бит более чем два, это свойство можно сформулировать следующим образом.

Свойство 15.4. Для любого числа r последовательностей бит a_i, a_j,…, a_q, где $i \neq j \neq \ldots \neq q$, такого, что $r < m$, в счетчиковой последовательности $A = a_{m-1}\, a_{m-2}\, \ldots\, a_2\, a_1\, a_0$ для всевозможных комбинаций 0 0 … 0, 0 0 … 1, …, и 1 1 … 1, значений a_i, a_j, …, a_q, существует 2^{m-r} различные двоичные комбинации в остальных $m - r$ последовательностях бит.

Следующее утверждение позволит сформулировать алгоритм для генерирования счетчиковых тестовых последовательностей.

Утверждение 15.1. Упорядоченная счетчиковая тестовая последовательность $A = a_{m-1}\, a_{m-2}\, \ldots\, a_2\, a_1\, a_0$ может быть получена при использовании m линейно независимых последовательностей бит, которые содержат 2^{m-1} нулей и 2^{m-1} единиц.

Доказательство утверждения 15.1. Необходимым условием для последовательности бит a_i является равное количество в них единиц и нулей, что вытекает из свойства 15.2. Предположим, что некоторое подмножество r

< m последовательностей бит a_i, a_j, …, a_q линейно зависимо. Тогда можно записать, что $a_i \oplus a_j \oplus \ldots \oplus a_q$ = const. Это означает, что $a_i \oplus a_j \oplus \ldots \oplus a_q = 0$ либо $a_i \oplus a_j \oplus \ldots \oplus a_q = 1$. В первом случае, очевидно, что не существует такой двоичной комбинации для $a_i\, a_j \ldots a_q$ и любого r, которая содержала бы нечетное число единиц, что противоречит свойству 15.4. Для второго случая отсутствует такая двоичная комбинация для $a_i\, a_j \ldots a_q$., которая содержала бы нечетное число нулей для нечетного r, и четное число нулей для четного r, что также противоречит свойству 15.4. Таким образом, линейная независимость является достаточным условием для последовательности бит счетчиковой тестовой последовательности.

Например, для последовательности $A_{a2} = d_1\, d_3 = a_1\, (a_1 \oplus a_0)$ (см. таблицу 15.3) имеется две последовательности бит d_1 и d_3, которые содержат по равному числу единиц и нулей и являются линейно независимыми, так как $d_1 \oplus d_3 = a_1 \oplus (a_1 \oplus a_0) = a_0 \neq 0$. В то же время легко показать, что нельзя получить полное множество тестовых наборов, используя последовательности бит d_2 и d_5, так как они являются линейно зависимыми, т. е. $d_5 = d_2 \oplus 1$.

Используя свойства 15.2 и 15.4, а также утверждение 15.1, можно сформулировать алгоритм генерирования последовательностей бит, используемых для формирования счетчиковых тестовых последовательностей. В общем случае подмножество подобных тестовых последовательностей может быть получено из $2(2^m - 1)$ последовательностей бит, которые определяются по формуле:

$$d_i = (\delta_{i(m-1)} a_{m-1} \oplus \delta_{i(m-2)} a_{m-2} \oplus \ldots \oplus \delta_{i2} a_2 \oplus \delta_{i1} a_1 \oplus \delta_{i0} a_0)^{\lambda}, \qquad (15.3)$$

где $\delta_{ij} \in \{0, 1\}$; $\delta_{i(m-1)}\, \delta_{i(m-2)}\, \delta_{i(m-3)} \ldots \delta_{i1}\, \delta_{i0} \neq 0\ 0\ 0\ \ldots\ 0\ 0$. При $\lambda = 1$ присутствует отрицание, формирующее инверсное значение выражения (15.3), представленного в скобках, а при $\lambda = 0$ отрицание отсутствует.

Как частный случай описанного выше метода рассмотрим последовательности бит $b_i \in \{a_i, \overline{a_i}\}$. Тогда векторы $\delta_{i(m-1)}\, \delta_{i(m-2)}\, \delta_{i(m-3)} \ldots \delta_{i(i+1)}\, \delta_{ii}\, \delta_{i(i-1)} \ldots \delta_{i1}\ \delta_{i0}$ примут значения 0 0 0…0 1 0 … 0 0 для $i \in \{0, 1, 2, \ldots, m-1\}$, а тестовая последовательность будет иметь вид:

$$A_m = a_{m-1}^{\lambda m-1}\, a_{m-2}^{\lambda m-2} \oplus \ldots \oplus a_2^{\lambda 2} \oplus a_1^{\lambda 1} \oplus a_0^{\ 0}, \qquad (15.4)$$

где при $\lambda_I = 1$ присутствует отрицание над a_i, а при $\lambda_i = 0$ - отсутствует.

В этом случае получим 2^m счетчиковых тестовых последовательностей, которые легко генерируются на основании исходной счетчиковой последовательности как ее простейшие модификации.

Примеры подобных последовательностей были рассмотрены в главе 5. В этой главе представлен метод, основанный на применении масок, определяющих инверсии бит исходного базового теста (5.6), который используется для формирования многократных вероятностных тестов.

Операция отрицания широко применяется и для получения оптимальных управляемых вероятностных тестов малой длины, которые подробно рассмотрены в главе 4.

Как отмечалось ранее, для упрощения проблемы реализации генератора счетчиковых последовательностей необходимо использовать простейшие подходы, не требующие больших аппаратных затрат [331]. Одним из подобных методов является модификация счетчиковых последовательностей за счет циклического сдвига последовательностей бит тестовых наборов [226, 230, 331]. Данный метод базируется на следующих свойствах [216].

Свойство 15.5. Последовательность бит a_i, $i \in \{0, 1, 2, \ldots, m-1\}$ тестовых наборов $A = a_{m-1}\, a_{m-2} \ldots a_2\, a_1\, a_0$, формируемых счетчиковой последовательностью, является циклической последовательностью с периодом 2^l, где $l \in \{1, 2, \ldots, m\}$.

Например, для счетчиковой последовательности $A = a_2\, a_1\, a_0$ при $m = 3$ можно выделить битовую последовательность $a_0 = 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1$ с периодом $2^1 = 2$, $a_1 = 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1$ с периодом $2^2 = 4$, и $a_2 = 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1$, для которой период $2^3 = 8$ максимален. Отметим, что свойство 15.5 справедливо не только для счетчиковой последовательности, оно также выполнимо и для последовательности Грея, однако очевидно, что оно выполняется не для любой детерминированной тестовой последовательности. Используя данное свойство, докажем следующую теорему [336].

Теорема 15.1. Для любой последовательности бит a_i $i \in \{0, 1, 2, \ldots, m-1\}$ тестовых наборов $A = a_{m-1}\, a_{m-2} \ldots a_2\, a_1\, a_0$ счетчиковой последовательности выполняется равенство $a_{m-1}(j)\, a_{m-2}(j) \ldots a_{i+1}(j)\, a_{i-1}(j) \ldots a_2(j)\, a_1(j)\, a_0(j) = a_{m-1}(j + 2^i)\, a_{m-2}(j + 2^i) \ldots a_{i+1}(j + 2^i)\, a_{i-1}(j + 2^i) \ldots a_2(j + 2^i)\, a_1(j + 2^i)\, a_0(j + 2^i)$, где $j \in \{0, 1, 2, \ldots, 2^m - 1\}$, для любого $j \in \{k2^{i+1} + 0, k2^{i+1} + 1, k2^{i+1} + 2, \ldots k2^{i+1} + 2^i - 1\}$ и $k \in \{0, 1, 2, \ldots, 2^{m-i} - 1\}$, где 2^{i+1} является периодом последо.вательности a_i, а значение $j + 2^i$ вычисляется по модулю 2^m.

Доказательство теоремы 15.1. Согласно свойству 15.5 для любого $j \in \{0, 1, 2, \ldots, 2^m - 1\}$ и $q \in \{0, 1, 2, \ldots, m-1\}$ справедливы равенства $a_q(j) = 0$ для $j \bmod 2^{q+1} < 2^q$ и $a_q(j) = 1$ для $j \bmod 2^{q+1} \geq 2^q$, которые позволяют сделать вывод, что для $q > i$ выполняется $a_{m-1}(j)\, a_{m-2}(j) \ldots a_{i+1}(j) = a_{m-1}(j + 2^i)\, a_{m-2}(j + 2^i) \ldots a_{i+1}(j + 2^i)$. Принимая во внимание, что период последовательности бит a_q, $q \in \{0, 1, 2, \ldots, m-1\}$ счетчиковой числовой последовательности $A = a_{m-1}\, a_{m-2} \ldots a_2\, a_1\, a_0$ равен 2^{q+1}, будет выполняться равенство $a_q(j) = a_q((j + g2^{q+1}) \bmod 2^m)$ для любого $g > 0$. Таким образом, можно заключить, что для $q < i$ имеем $a_{i-1}(j) \ldots a_2(j)\, a_1(j)\, a_0(j) = a_{i-1}(j + 2^i) \ldots a_2(j + 2^i)\, a_1(j + 2^i)\, a_0(j + 2^i)$, что и требовалось доказать.

Теорема 15.1 позволяет сформулировать следующее свойство [336].

Свойство 15.6. Для любой последовательности бит a_i $i \in \{0, 1, 2, \ldots, m-1\}$ тестовых наборов $A = a_{m-1}\, a_{m-2} \ldots a_2\, a_1\, a_0$ счетчиковой последовательности отрицание любого числа пар $a_i(j)$ и $a_i\ (j + 2^i)$ бит, где $j \in \{k2^{i+1} + 0, k2^{i+1} + 1, k2^{i+1} + 2, \ldots, k2^{i+1} + 2^i - 1\}$ и $k \in \{0, 1, 2, \ldots, 2^{m-i} - 1\}$, приведет к формированию новой счетчиковой последовательности.

В качестве примера рассмотрим счетчиковую последовательность $A = a_3\, a_2\, a_1\, a_0$ и для различных значений i и k получим отрицания пар бит $a_i(j)$ и a_i $(j + 2^i)$. Результирующие последовательности представлены в таблице 15.5.

Таблица 15.5 – *Последовательности тестовых наборов с отрицанием пар бит*

A(*j*)					$i = \mathbf{0}$, $k = \mathbf{0}$. $i = \underline{\mathbf{0}}$, $k = \underline{\mathbf{3}}$. $i = \underline{\boldsymbol{0}}$, $k = \underline{\boldsymbol{7}}$.	$i = \mathbf{0}$, $k = \mathbf{0}$. $i = \underline{\mathbf{1}}$, $k = \underline{\mathbf{0}}$. $i = \underline{\boldsymbol{1}}$, $k = \underline{\boldsymbol{3}}$.	$i = \mathbf{2}$, $k = \mathbf{0}$.
j	$i = 3$	$i = 2$	$i = 1$	$i = 0$			
0	0	0	0	0	0 0 0 **1**	0 0 **1** **1**	0 **1** 0 0
1	0	0	0	1	0 0 0 **0**	0 0 **1** **0**	0 **1** 0 1
2	0	0	1	0	0 0 1 0	0 0 **0** 0	0 **1** 1 0
3	0	0	1	1	0 0 1 1	0 0 **0** 1	0 **1** 1 1
4	0	1	0	0	0 1 0 0	0 1 0 0	0 **0** 0 0
5	0	1	0	1	0 1 0 1	0 1 0 1	0 **0** 0 1
6	0	1	1	0	0 1 **1** 1	0 1 1 0	0 **0** 1 0
7	0	1	1	1	0 1 **1** 0	0 1 1 1	0 **0** 1 1
8	1	0	0	0	1 0 0 0	1 0 0 0	1 0 0 0
9	1	0	0	1	1 0 0 1	1 0 0 1	1 0 0 1
10	1	0	1	0	1 0 1 0	1 0 1 0	1 0 1 0
11	1	0	1	1	1 0 1 1	1 0 1 1	1 0 1 1
12	1	1	0	0	1 1 0 0	1 1 **1** 0	1 1 0 0
13	1	1	0	1	1 1 0 1	1 1 **1** 1	1 1 0 1
14	1	1	1	0	1 1 **1** 1	1 1 **0** 0	1 1 1 0
15	1	1	1	1	1 1 **1** 0	1 1 **0** 1	1 1 1 1

Общее количество детерминированных тестовых последовательностей подобного вида вычисляется как: $2^{m2^{m-1}} - 1$ [336]. Следует отметить, что практическая реализация данного метода достаточно трудоемка.

За счет существенного уменьшения количества модифицированных счетчиковых последовательностей предложенный метод может быть адаптирован на основании свойства 15.7 [336].

Свойство 15.7. В результате циклического сдвига последовательностей бит a_i, $i \in \{0, 1, 2, \ldots, m-1\}$ тестовых наборов $A = a_{m-1}\, a_{m-2} \ldots a_2\, a_1\, a_0$ счетчиковой последовательности генерируется $2^{m+1} - 2$ различных двоичных последовательностей d_j, $j \in \{0, 1, 2, \ldots, 2^{m+1} - 3\}$, состоящих из 2^{m-1} нулей и 2^{m-1} единиц.

Так, для счетчиковой последовательности $A = a_2\, a_1\, a_0$, согласно свойству 15.7 существует $2^{3+1} - 2 = 14$ последовательностей бит d_j, $j \in \{0, 1, 2, \ldots, 13\}$ (таблица 15.6).

Последовательности бит d_j, полученные путем сдвига, составляют подмножество последовательностей, которые удовлетворяют теореме 15.1 и могут быть использованы для формирования исчерпывающих детерминированных тестовых последовательностей [336].

Таблица 15.6 – *Битовые последовательности d_j для $m = 3$*

$d_0 = a_0$	d_1	$d_2 = a_1$	d_3	d_4	d_5	$d_6 = a_2$	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}
0	1	0	0	1	1	0	0	0	0	1	1	1	1
1	0	0	1	1	0	0	0	0	1	1	1	1	0
0	1	1	1	0	0	0	0	1	1	1	1	0	0
1	0	1	0	0	1	0	1	1	1	1	0	0	0
0	1	0	0	1	1	1	1	1	1	0	0	0	0
1	0	0	1	1	0	1	1	1	0	0	0	0	1
0	1	1	1	0	0	1	1	0	0	0	0	1	1
1	0	1	0	0	1	1	0	0	0	0	1	1	1

В таблице 15.7 приведены примеры подобных последовательностей $A = a_{m-1}\{l_{m-1}\}\ a_{m-2}\{l_{m-2}\}\ \ldots\ a_2\{l_2\}\ a_1\{l_1\}\ a_0\{l_0\}$, зависящих от $l_i \in \{0, 1, 2, \ldots, 2^{i+1} - 1\}$, где l_i определяет возможное значение циклического сдвига i-го разряда счетчиковой последовательности.

Таблица 15.7 – *Модифицированные счетчиковые последовательности*

$a_3\{0\}a_2\{0\}a_1\{0\}a_0\{1\}$	$a_3\{0\}a_2\{5\}a_1\{0\}a_0\{0\}$	$a_3\{7\}a_2\{5\}a_1\{2\}a_0\{1\}$
0 0 0 1 (1)	0 0 0 0 (0)	1 0 1 1 (11)
0 0 0 0 (0)	0 1 0 1 (5)	1 1 1 0 (14)
0 0 1 1 (3)	0 1 1 0 (6)	1 1 0 1 (13)
0 0 1 0 (2)	0 1 1 1 (7)	1 1 0 0 (12)
0 1 0 1 (5)	0 1 0 0 (4)	1 1 1 1 (15)
0 1 0 0 (4)	0 0 0 1 (1)	1 0 1 0 (10)
0 1 1 1 (7)	0 0 1 0 (2)	1 0 0 1 (9)
0 1 1 0 (6)	0 0 1 1 (3)	0 0 0 0 (0)
1 0 0 1 (9)	1 0 0 0 (8)	0 0 1 1 (3)
1 0 0 0 (8)	1 1 0 1 (13)	0 1 1 0 (6)
1 0 1 1 (11)	1 1 1 0 (14)	0 1 0 1 (5)
1 0 1 0 (10)	1 1 1 1 (15)	0 1 0 0 (4)
1 1 0 1 (13)	1 1 0 0 (12)	0 1 1 1 (7)
1 1 0 0 (12)	1 0 0 1 (9)	0 0 1 0 (2)
1 1 1 1 (15)	1 0 1 0 (10)	0 0 0 1 (1)
1 1 1 0 (14)	1 0 1 1 (11)	1 0 0 0 (8)

Следует отметить, что для $l_{m-1} = l_{m-2} = \ldots = l_2 = l_1 = l_0 = 0$ модифицированная последовательность $A = a_{m-1}\{0\}\ a_{m-2}\{0\}\ldots\ a_2\{0\}\ a_1\{0\}\ a_0\{0\}$ представляет собой оригинальную последовательность $A = a_{m-1}\, a_{m-2} \ldots\ a_2\, a_1\, a_0$.

Рассмотренный метод позволяет получить большее число модифицированных счетчиковых последовательностей по сравнению с методом модификации последовательностей, представленным ранее. Для его реализации необходимо использовать $m + m - 1 + \ldots + 3 + 2 + 1$ запоминающих элементов типа D-триггеров для m двоичных счетчиков, разрядность которых определяется индексом i битовой последовательности a_i. Каждый двоичный счетчик используется для генерирования битовых циклических последовательно-

стей $a_i\{l_i\}$. В режиме инициализации двоичные счетчики объединяются в единый регистр сдвига для записи начальных состояний l_i.

Эффективность предложенного метода подтверждается экспериментальными результатами, приведенными в [331] для случая тестирования ЗУ различной емкости в битах и различных модификаций счетчиковой тестовой последовательности в качестве адресной последовательности маршевых тестов.

15.3. Децимация детерминированных тестовых последовательностей

Термин децимация широко используется в различных технических приложениях. Содержательно он означает прореживание по индексу q исходной последовательности или последовательную выборку каждого q-го элемента заданной последовательности. В случае взаимной простоты длины последовательности и значения индекса q различают циклическую децимацию, в результате которой формируются все элементы исходной последовательности в ином порядке. В случае детерминированных тестовых последовательностей, например счетчиковых, децимация по индексу $q \geq 2$ означает получение модифицированной счетчиковой последовательности. Очевидно, что для любого индекса q, аппаратная реализация генератора такой модифицированной последовательности тестовых наборов не требует больших затрат. В простейшем случае счетчикового генератора тестовых последовательностей необходим счетчик с двумя функциональными режимами. В первом режиме реализуется обычный реверсивный двоичный счетчик, выполняющий две микрооперации, а именно: +1 и –1. Такой режим необходим для стандартной реализации детерминированных тестов, где необходима возрастающая и убывающая последовательности тестовых наборов. Для формирования децимированной по индексу q счетчиковой последовательности необходима реализация двух дополнительных микроопераций, состоящих в увеличении состояния счетчика на величину, равную q ($+q$), и уменьшении на это же значение ($–q$).

Первоначально рассмотрим случай децимации по индексу $q = 2$. Тогда новая модифицированная счетчиковая последовательность $A_k = A_k(0)\ A_k(1)\ A_k(2)\ \ldots\ A_k(N-2)\ A_k(N-1)$, как результат децимации по индексу $q = 2$ исходной последовательности $A_j(i) \in \{0, 1, 2, \ldots, N-1\}$, $i \in \{0, 1, 2, \ldots, N-1\}$, где $N = 2^m$ будет иметь вид:

$$\begin{aligned} A_k &= A_j(2i), && i \in \{0,1,2,\ldots,2^{m-1}-1\} \\ A_k &= A_j(2(i-2^{m-1})+1), && i \in \{2^{m-1}, 2^{m-1}+1, 2^{m-1}+2, \ldots, 2^m-1\}. \end{aligned} \qquad (15.5)$$

Например, в случае стандартной счетчиковой последовательности для $m = 4$ $A_j = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$ последовательность A_k примет вид $A_k = 0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15$. В зависимости

от начального тестового набора s существует $N = 2^m$ версий оригинальной последовательности A_k. Все такие последовательности A_k для $m = 3$ представлены таблице 15.8.

Как было показано ранее (см. главу 4 и 5), для реализации эффективного двукратного тестирования, необходимо выбрать две оптимальные тестовые последовательности [140, 142, 331]. Причем, как правило, исходная последовательность является базовой стандартной тестовой последовательностью. В данном случае базовой является счетчиковая тестовая последовательность A_j, для которой начальное состояние генератора тестов, формирующего такую последовательность, равняется нулевому состоянию. Как критерий для выбора второй тестовой последовательности может быть использовано арифметическое расстояние (5.8).

Результаты вычисления данной характеристики $AD(A_j, A_k(s))$, где s является начальным состоянием генератора счетчиковой последовательности для $m = 3$ и $m = 4$, приведены в таблицах 15.9 и 15.10.

Таблица 15.8 – *Результат децимации оригинальной счетчиковой последовательности для* $m = 3$

i	$A_j(i)$	A_k							
		$s = 0$	$s = 2$	$s = 4$	$s = 6$	$s = 1$	$s = 3$	$s = 5$	$s = 7$
0	0	0	2	4	6	1	3	5	7
1	1	2	4	6	1	3	5	7	0
2	2	4	6	1	3	5	7	0	2
3	3	6	1	3	5	7	0	2	4
4	4	1	3	5	7	0	2	4	6
5	5	3	5	7	0	2	4	6	1
6	6	5	7	0	2	4	6	1	3
7	7	7	0	2	4	6	1	3	5

Таблица 15.9 – *Значения расстояния* $AD(A_j, A_k(s))$ *для* $m = 3$

s	0	2	4	6	1	3	5	7
$AD(A_j, A_k(s))$	12	20	24	24	20	24	24	20

Таблица 15.10 – *Значения расстояния* $AD(A_j, A_k(s))$ *для* $m = 4$

s	$s = 0$	$s = 2$	$s = 4$	$s = 6$	$s = 8$	$s = 10$	$s = 12$	$s = 12$
$AD(A_j, A_k(s))$	56	72	84	92	96	96	92	84
s	$s = 1$	$s = 3$	$s = 5$	$s = 7$	$s = 9$	$s = 11$	$s = 13$	$s = 15$
$AD(A_j, A_k(s))$	72	84	92	96	96	92	84	72

Краткий анализ, приведенных выше значений характеристики различия счетчиковых последовательностей A_j и $A_k(s)$, позволяет сделать вывод, что

оптимальными парами счетчиковых тестовых последовательностей A_j и $A_k(s)$ для $m = 3$ будут $(A_j, A_k(4))$, $(A_j, A_k(6))$, $(A_j, A_k(3))$ и $(A_j, A_k(5))$, а для $m = 4$, соответственно, $(A_j, A_k(8))$, $(A_j, A_k(10))$, $(A_j, A_k(7))$ и $(A_j, A_k(9))$.

Так как структура последовательности $A_k(s)$ известна (15.5), то легко показать, что для произвольного значения $N = 2^m$ выше приведенная характеристика различия $AD(A_j, A_k(s))$ и любого начального тестового набора s может быть легко определена с помощью соотношений:

$$AD(A_j, A_k(s)) = \begin{cases} \dfrac{s(2^m - s + 2)}{2} + 2^{2m-2} - 2^{m-1}, & \text{для четных } s; \\ \dfrac{s(2^m - s) + 1}{2} + 2^{2m-2}, & \text{для нечетных } s. \end{cases} \quad (15.6)$$

Как было показано ранее [331], оптимальное значение s с точки зрения максимального значения характеристики $AD(A_j, A_k(s))$ может быть получено для четных значений s из следующей выражения:

$$\frac{\partial AD(A_j, A_k(s))}{\partial s} = 2^{m-1} - s + 1 = 0.$$

Наиболее близкие четные значения s решения равенства $s = 2^{m-1} + 1$ будут $s = 2^{m-1}$ и $s = 2^{m-1} + 2$. В случае $m = 3$ и $m = 4$, соответственно 4, 6 и 8, 10 (см. таблица 15.9 и 15.10).

Оптимальные нечетные значения s могут быть получены из формулы:

$$\frac{\partial AD(A_j, A_k(s))}{\partial s} = 2^{m-1} - s = 0.$$

Тогда $s = 2^{m-1} - 1$ и $s = 2^{m-1} + 1$, что подтверждает полученные выше значения арифметического расстояния, которые приведены в таблицах 15.9 и 15.10. Оптимальные значения для $m = 3$ равняются 3, 5 и для $m = 4$, соответственно, 7, 9.

На основе полученных результатов, можно сделать вывод, что наиболее эффективным сочетанием счетчиковых тестовых последовательностей, являются исходная счетчиковая последовательность и последовательность, полученная с помощью децимации по индексу 2 и начальным тестовым набором $s \in \{2^{m-1} - 1, 2^{m-1}, 2^{m-1} + 1, 2^{m-1} + 2\}$.

Прежде чем определить счетчиковую тестовую последовательность, получаемую с помощью децимации по индексу $q = 3$, приведем ряд положений из теории чисел, сформулированных в виде теорем [140, 142, 331].

Теорема 15.2. Неравенство $(2^m - 1) \bmod 3 \neq 2$ верно для любого целого числа m.

Доказательство теоремы 15.2. Пусть $(2^m - 1) \bmod 3 = 2$, тогда $(2^m - 1) = 3p + 2$, откуда следует, что $2^m - 3 = 3p$, где p – целое положительное число. Из последнего равенства следует, что 2^m делится на 3 без остатка, что не яв-

ляется верным утверждением. Тогда, $(2^m - 1) \bmod 3 \neq 2$, что и требовалось доказать.

Из данной теоремы вытекает два следствия.

Следствие 15.1. $(2^m - 1) \bmod 3 \in \{0, 1\}$.

Принимая во внимания следствие 15.1 и равенство (15.7):

$$(a \otimes b) \bmod d = ((a \bmod d) \otimes (b \bmod d)) \bmod d, \qquad (15.7)$$

где a, b и d – целые числа, и $\otimes \in \{+, \times\}$, верно следующее следствие.

Следствие 15.2. $((2^m - 1) + 2) \bmod 3 = (2^m + 1) \bmod 3 \in \{0, 2\}$.

Теорема 15.3. $(2^m - 1) \bmod 3 = 0$ для четных значений m и $(2^m - 1) \bmod 3 = 1$ для нечетных значений m.

Доказательство теорема 15.3. Пусть $m = 2n$ – четное целое число, тогда $(2^m - 1) = (2^{2n} - 1) = (2^n + 1)(2^n - 1)$. Очевидно, что для $(2^n - 1) \bmod 3 = 0$ или 1 значение $(2^n + 1) \bmod 3$ соответственно равняются 2 или 0. Тогда согласно (15.7) равенство $((2^n + 1) \times (2^n - 1)) \bmod 3 = (((2^n + 1) \bmod 3) \times ((2^n - 1) \bmod 3)) \bmod 3$ всегда равно 0.

В случае нечетного значения m легко показать, что $(2 \times (2^m - 1)) \bmod 3 = ((2^{m+1} - 1) - 1) \bmod 3 = 2$, исходя из соотношений $(2^{m+1} - 1) \bmod 3 = 0$ ($m + 1$ четно) и $(-1) \bmod 3 = 2$. Тогда на основе соотношения (15.7) для $(2^m - 1) \bmod 3$ единственно возможным значением является 1.

Исходя из последней теоремы, можно привести определение счетчиковой тестовой последовательности с децимацией по индексу $q = 3$. Так, последовательность A_k с децимацией по индексу $q = 3$ исходной счетчиковой последовательности $A_j = A_j(0)\, A_j(1)\, A_j(2) \ldots A_j(2^m - 2)\, A_j(2^m - 1) = 0, 1, 2, \ldots, 2^m - 2, 2^m - 1$ имеет вид:

$$A_k(i) = A_j(3i \bmod (2^m - 1)) = 0, 3, 6, \ldots, 2^m - 4, 2^m - 1, 2, 5, 8, \ldots, 2^m - 5, 2^m - 2, 1, 4, 7, \ldots, 2^m - 6, 2^m - 3 \qquad (15.8)$$

для четных m,

$$A_k(i) = A_j(3i \bmod (2^m - 1)) = 0, 3, 6, \ldots, 2^m - 5, 2^m - 2, 1, 4, 7, \ldots, 2^m - 4, 2^m - 1, 2, 5, 8, \ldots, 2^m - 6, 2^m - 3 \qquad (15.9)$$

для нечетных m.

Результаты вычисления характеристики $AD(A_j, A_k(s))$ для счетчиковой тестовой последовательности A_j и последовательности $A_k(s)$ с децимацией A_j по индексу $q = 3$ при $m = 3$ и $m = 4$ и различных начальных тестовых наборов s второй последовательности приведены в таблицах 15.11 и 15.12.

Таблица 15.11 – *Значения расстояния* $AD(A_j, A_k(s))$ *для* $m = 3$

s	0	1	2	3	4	5	6	7
$AD(A_j, A_k(s))$	16	20	16	24	24	20	24	24

Таблица 15.12 – *Значения расстояния* $AD(A_j, A_k(s))$ *для* $m = 4$

s	$s = 0$	$s = 1$	$s = 2$	$s = 3$	$s = 4$	$s = 5$	$s = 6$	$s = 7$
$AD(A_j, A_k(s))$	72	68	72	88	80	84	96	88
s	$s = 8$	$s = 9$	$s = 10$	$s = 11$	$s = 12$	$s = 13$	$s = 14$	$s = 15$
$AD(A_j, A_k(s))$	88	100	88	88	96	84	80	88

Как видно из приведенных выше данных, значения арифметического расстояния двух последовательностей A_j и $A_k(s)$ для разных начальных тестовых наборов s второй последовательности различны.

Определим характеристику $AD(A_j, A_k(s))$ аналитически, как функцию, зависящую от длины $N = 2^m$ тестовой последовательности и начального набора второй счетчиковой последовательности s, полученной с помощью децимации по индексу $q = 3$. Для четных значений m, используя подстановку $a = (N - 1)/3$, последовательности A_j будут представлены как:

$$A_j = 0, 1, \ldots, a, a + 1, a + 2, \ldots, (3a + 1)/2, (3a + 1)/2 + 1,(3a + 1)/2 + 2, \ldots,$$

$$2a, 2a + 1,2a + 2, \ldots, 3a.$$

В свою очередь последовательности $A_k(s = 0)$ имеют вид:

$$A_k(s = 0) = 0, 3, \ldots, 3a, 2, 5, \ldots, (3a + 1)/2, (3a + 1)/2 + 3, (3a + 1)/2 + 6,\ldots,$$
$$3a - 1, 1, 4, ..., 3a - 2.$$

Тогда для четных значений m и $N = 2^m$ арифметическое расстояние будет вычисляться как:

$$AD(A_j, A_k(s = 0)) = \sum_{i=0}^{N-1} \left| A_j(i) - A_k(i) \right| = 2 + 4 + \ldots + 2a + 2 + 4 + \ldots + a - 1 + 2 + 4 + \ldots$$

$$+ a - 1 + 2 + 4 + \ldots + 2a = \frac{5a^2 + 4a - 1}{2} = \frac{5N^2 + 2N - 16}{18}. \qquad (15.10)$$

Аналогично для нечетных значений m получим:

$$AD(A_j, A_k(s = 0)) = \sum_{i=0}^{N-1} \left| A_j(i) - A_k(i) \right| = \frac{5N^2 - 2N - 16}{18}. \qquad (15.11)$$

Для $s \neq 0$ получаются более сложные соотношения. Так, для четных значений m арифметическое расстояние вычисляется как (15.12):

$$AD(A_j, A_k(s)) = \begin{cases} (5\times 2^{2m} + 2^{m+1} + 3\times 2^{m+1}\times s - 16 + 12\times s - 6\times s^2)/18, \\ s \in \{0,6,12,\ldots,2^m - 4\}; \\ \\ (5\times 2^{2m} + 2^{m+1} + 3\times 2^{m+1}\times s + 2 + 12\times s - 6\times s^2)/18, \\ s \in \{3,9,15,\ldots,2^m - 1\}; \\ \\ (5\times 2^{2m} - 5\times 2^{m+1} + 3\times 2^{m+1}\times s - 16 + 12\times s - 6\times s^2)/18, \\ s \in \{2,4,8,\ldots,2^m - 2\}; \\ \\ (5\times 2^{2m} - 5\times 2^{m+1} + 3\times 2^{m+1}\times s + 2 + 12\times s - 6\times s^2)/18, \\ s \in \{1,5,7,\ldots,2^m - 3\}. \end{cases} \quad (15.12)$$

Тогда оптимальное значение s для достижения максимального значения арифметического расстояния $AD(A_j, A_k(s))$ может быть получено из следующего соотношения:

$$\frac{\partial AD(A_j, A_k(s))}{\partial s} = \frac{3\times 2^{m+1} + 12 - 12\times s}{18}, \quad s \in \{0,1,2,\ldots,2^m - 1\}.$$

В зависимости от величины s для всех четырех функций (15.12) максимальное значение будет при s, наиболее близком к $s = 2^{m-1} + 1$. Так, максимальное значение для первой функции будет при $s = 2^{m-1} - 2$, так как $s \in \{0, 6, 12, \ldots, 2^m - 4\}$. Для второй функции $s = 2^{m-1} + 1$, для третьей – $s = 2^{m-1}$ и для четвертой – $s = 2^{m-1} - 1$. После подстановки этих значений s в (15.12) получим, что наибольшее значение арифметического расстояния $AD(A_j, A_k(s))$ достигается при $s = 2^{m-1} + 1$. Например, для $m = 4$, наибольшее значение $AD(A_j, A_k(9))$ будет 100 при $s = 2^{4-1} + 1 = 9$, что соответствует данным, приведенным в таблице 15.11.

В случае нечетного значения m получаем соотношения для $AD(A_j, A_k(s))$, в зависимости от величины s, представленные в виде выражения (15.13).

Как и в предыдущем случае, значение s для четырех функций (15.12) должно быть наиболее близко к $s = 2^{m-1} + 1$. Тогда для первой функции получаем $s = 2^{m-1} + 2$ и $s = 2^{m-1}$, для второй – $s = 2^{m-1} - 1$ и $s = 2^{m-1} + 3$, для третьей – $s = 2^{m-1} - 2$ и $s = 2^{m-1} + 4$ и для четвертой – $s = 2^{m-1} + 1$. После подстановки этих значений в (15.13) получаем, что наибольшее значений арифметическое расстояние $AD(A_j, A_k(s))$ достигает при $s = 2^{m-1} + 2$, $s = 2^{m-1} - 1$, $s = 2^{m-1}$ и $s = 2^{m-1} + 3$. Действительно, $AD(A_j, A_k(3)) = AD(A_j, A_k(4)) = AD(A_j, A_k(6)) = AD(A_j, A_k(7)) = 24$ (см. таблицу 15.11):

$$AD(A_j, A_k(s)) = \begin{cases} (5\times 2^{2m} - 2^{m+1} + 3\times 2^{m+1}\times s - 16 + 12\times s - 6\times s^2)/18, \\ s \in \{0,4,6,10,\ldots,2^m - 2\}; \\ \\ (5\times 2^{2m} - 2^{m+1} + 3\times 2^{m+1}\times s + 2 + 12\times s - 6\times s^2)/18, \\ s \in \{1,3,7,9\ldots,2^m - 1\}; \\ \\ (5\times 2^{2m} - 7\times 2^{m+1} + 3\times 2^{m+1}\times s - 16 + 12\times s - 6\times s^2)/18, \\ s \in \{2,8,14,\ldots,2^m - 6\}; \\ \\ (5\times 2^{2m} - 7\times 2^{m+1} + 3\times 2^{m+1}\times s + 2 + 12\times s - 6\times s^2)/18, \\ s \in \{5,11,17,\ldots,2^m - 3\}. \end{cases} \quad (15.13)$$

Отметим, что аналитические соотношения (15.12) и (15.13) соответствуют числовым результатам, представленным в таблицах 15.11 и 15.12.

15.4. Модификации последовательностей кода Грея

Двоичный код Грея $A_G(i)$, $i \in \{0, 1, 2, \ldots, 2^m - 1\}$ перечисляет все двоичные комбинации $a_{m-1}\ a_{m-2}\ \ldots\ a_1\ a_0$, где $a_j \in \{0, 1\}$, состоящие из m-бит (m-битные кодовые слова) таким образом, что два последовательных слова отличаются только в одном разряде [67, 168]. В циклической последовательности Грея последняя двоичная комбинация отличается от первой также в одном разряде [67, 168].

Последовательность Грея часто используется в качестве адресной последовательности при тестировании и диагностировании запоминающих устройств [73, 93]. Для последовательности Грея так же, как и для счетчиковой последовательности, справедливы ранее сформулированные свойства 15.1 – 15.4, а также утверждение 15.1 [233].

Как и для счетчиковой тестовой последовательности, в случае кода Грея эффективной является модификация в соответствии с (15.4), позволяющая получить 2^m различных последовательностей тестовых наборов.

Последовательности кода Грея весьма многообразны и могут формироваться различными способами [168]. При формировании подобных последовательностей, с целью минимизации аппаратурных затрат, в дальнейшем будем рассматривать *отраженный код Грея* (*binary-reflected Gray codes*) [93, 168, 233]. Для оценки эффективности применения кода Грея в качестве генератора тестовой последовательности используем ранее рассмотренное в подразделе 15.3 арифметическое расстояние $AD(A_G, A_{Gm})$ как меру различия между оригинальной последовательностью кода Грея A_G и его модификацией A_{Gm}. Как было показано ранее, вне зависимости от структуры оригинальной исчерпывающей тестовой последовательности ее модификации в соответ-

ствии с выражением (15.4) могут быть описаны и оценены на основании теоремы 5.1.

Сложной является задача оценки эффективности модифицированных последовательностей, полученных с применением более сложных преобразований [93]. В [233] была предложена модификация последовательности кода Грея $A_G = a_{m-1}\, a_{m-2}\, \ldots\, a_1\, a_0$ с использованием операции циклического сдвига $A_{Gm} = \overline{a_0}\, a_{m-1}\, a_{m-2}\, \ldots\, a_2\, \overline{a_1}$. Для последовательностей A_G и A_{Gm} справедлива следующая теорема [233].

Теорема 15.3. Арифметическое расстояние $AD(A_G, A_{Gm})$ для $A_G = a_{m-1}\, a_{m-2} \ldots a_1\, a_0$ и $A_{Gm} = \overline{a_0}\, a_{m-1}\, a_{m-2} \ldots a_2\, \overline{a_1}$ вычисляется как:

$$AD(A_G, A_{Gm}) = 2^{m-1}(2^{m-1}+1) \tag{15.14}$$

Доказательство теоремы 15.3. Величина $A_G(i) - A_{Gm}(i)$ для любого i будет вычисляться как $(a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \ldots + a_1 2^1 + a_0 2^0) - (\overline{a_0}\, 2^{m-1} + a_{m-1}\, 2^{m-2} + a_{m-2}2^{m-3} + \ldots + a_2 2^1 + \overline{a_1}\, 2^0)$. Используя замену $\overline{a_0}$ и $\overline{a_1}$ на булево выражение $a_j \oplus 1$ и далее на арифметическое соотношение $\overline{a_j} = 1 - a_j$, окончательно получим:

$$\begin{aligned} AD(A_G, A_{Gm}) &= \sum_{i=0}^{2^m-1} \left| A_G(i) - A_{Gm}(i) \right| = \\ &= \sum_{i=0}^{2^m-1} \left| \left(\sum_{j=2}^{m-1} 2^{j-1} a_j \right) + 3a_1 + (2^{m-1}+1)a_0 - (2^{m-1}+1) \right|. \end{aligned} \tag{15.15}$$

Принимая во внимание, что для любых i:

$$\left(\sum_{j=2}^{m-1} 2^{j-1} a_j\right) + 3a_1 \le 2^{m-1} + 1,$$

выражение (15.15) может быть представлено как сумма двух слагаемых, как сумма четных и как сумма нечетных значений двоичных комбинаций последовательности Грея:

$$AD(A_G, A_{Gm}) = \sum_{i=0}^{2^m-1} \left| A_G(i) - A_{Gm}(i) \right| = \sum_{i=0,3,4,\ldots,2^m-1} \left| A_G(i) - A_{Gm}(i) \right| + \sum_{i=1,2,5,\ldots,2^m-2} \left| A_G(i) - A_{Gm}(i) \right|.$$

Первое слагаемое для $a_0 = 0$ вычисляется как:

$$\sum_{i=0,3,4,\ldots,2^m-1} \left| A_G(i) - A_{Gm}(i) \right| = + \sum_{i=0,3,4,\ldots,2^m-1} ((2^{m-1}+1) - (\sum_{j=2}^{m-1} 2^{j-1} a_j) - 3a_1) =$$

$$= 2^{m-1}(2^{m-1}+1) - (\sum_{j=2}^{m-1} 2^{j-1} 2^{m-2}) - 3 \times 2^{m-2} = 2^{2m-3} + 2^{m-2}.$$

Второе слагаемое для $a_0 = 1$ принимает вид:

$$\sum_{i=1,2,5,\ldots,2^m-2} \left| A_G(i) - A_{Gm}(i) \right| = + \sum_{i=1,2,5,\ldots,2^m-2} (\sum_{j=2}^{m-1} 2^{j-1} a_j) + 3a_1 =$$

$$(\sum_{j=2}^{m-1} 2^{j-1} 2^{m-2}) + 3 \times 2^{m-2} = 2^{2m-3} + 2^{m-2}.$$

Окончательно получим:

$$AD(A_G, A_{Gm}) = 2(2^{2m-3} + 2^{m-2}) = 2^{m-1}(2^{m-1}+1),$$

что и требовалось доказать.

Последняя теорема позволяет сделать вывод, что арифметическое расстояние между двумя тестовыми последовательностями $A_G = a_{m-1}\, a_{m-2} \ldots a_1\, a_0$ и $A_{Gm} = \overline{a_0}\, a_{m-1}\, a_{m-2} \ldots a_2\, \overline{a_1}$ вычисляется как: $2^{m-1}(2^{m-1}+1)$ вне зависимости от вида последовательности A_G. Для заданного m и отраженного кода Грея A_G существует $m - 1$ модифицированых последовательностей Грея A_{Gm}. В качестве примера в таблице 15.13 для $m = 4$ разрядного кода Грея A_G приведены всевозможные его модификации.

Таблица 15.13 – *Модифицированные последовательности Грея*

i	$A_{G0}(i)=$ $=a_3a_2a_1a_0$	$A_{G1}(i)=$ $=\overline{a_0}\, a_3a_2\, \overline{a_1}$	$A_{G2}(i)=$ $=a_1\, \overline{a_0}\, a_3\, \overline{a_2}$	$A_{G3}(i)=$ $=a_2a_1\, \overline{a_0}\, \overline{a_3}$
0	0000 (0)	1001 (9)	0101 (5)	0011 (3)
1	0001 (1)	0001 (1)	0001 (1)	0001 (1)
2	0011 (3)	0000 (0)	1001 (9)	0101 (5)
3	0010 (2)	1000 (8)	1101 (13)	0111 (7)
4	0110 (6)	1010 (10)	1100 (12)	1111 (15)
5	0111 (7)	0010 (2)	1000 (8)	1101 (13)
6	0101 (5)	0011 (3)	0000 (0)	1001 (9)
7	0100 (4)	1011 (11)	0100 (4)	1011 (11)
8	1100 (12)	1111 (15)	0110 (6)	1010 (10)
9	1101 (13)	0111 (7)	0010 (2)	1000 (8)
10	1111 (15)	0110 (6)	1010 (10)	1100 (12)
11	1110 (14)	1110 (14)	1110 (14)	1110 (14)
12	1010 (10)	1100 (12)	1111 (15)	0110 (6)
13	1011 (11)	0100 (4)	1011 (11)	0100 (4)
14	1001 (9)	0101 (5)	0011 (3)	0000 (0)
15	1000 (8)	1101 (13)	0111 (7)	0010 (2)

Арифметические расстояния для всевозможных пар тестовых последовательностей, представленных в таблице 15.13 для m = 4, приведены в таблице 15.14.

Таблица 15.14 – *Расстояния* $AD(A_{Gk}, A_{Gj})$

	$k = 0$	$k = 1$	$k = 2$	$k = 3$
$j = 0$	0	72	68	72
$j = 1$	72	0	72	68
$j = 2$	68	72	0	72
$j = 3$	72	68	72	0

Таким образом, в качестве модификаций детерминированных тестовых последовательностей возможно использование как модификации счетчиковых последовательностей, описанных в разделах (15.2 и 15.3) [331], так и модифицированных последовательностей Грея, рассмотренных в настоящем разделе. Как была показано в [331], численные оценки эффективности применения как счетчиковых тестовых последовательностей, так и последовательностей Грея позволяют достигать высокой полноты покрытия *PNPSFk* неисправностей ЗУ [331].

15.5. Последовательности анти-Грея

Последовательности анти-Грея подробно рассматривались в рамках раздела 6.5. Неслучайные тестовые последовательности, как результат модификации последовательностей Соболя [229, 345]. Далее рассмотрим данное множество тестовых последовательностей с более общих позиций, как подмножество исчерпывающих детерминированных последовательностей.

С целью получения тестовых детерминированных последовательностей с максимальным хэммингoвым расстоянием между соседними тестовыми наборами за основу может быть взята *последовательность переключений кода Грея* $t(A_G) = t_1, t_2, t_3, \ldots, t_{N-1}$, которая имеет инверсную интерпретацию (см. главу 6) [231]. А именно $t(A_{G^*}) = \bar{t}\,(A_G) = \bar{t}_1, \bar{t}_2, \bar{t}_3, \ldots, \overline{t_{N-1}}$, где $\bar{t}_l \in \{m - 1, m - 2, \ldots, 2, 1, 0\}$ определяет индекс неизменяемого бита двух последовательных m-битовых слов кода длиной $N = 2^m$. В этом случае все биты, кроме одного, при последовательном переходе будут изменяться. Соответственно хэммингово расстояние между двумя последовательными m-разрядными тестовыми наборами будет равняться $m - 1$. В дальнейшем, как и ранее, будем называть такие последовательности последовательностями анти-Грея. Результатом применения подобного метода генерирования будут последовательности, приведенные в таблице 6.10.

Как видно из приведенной таблицы, всевозможные комбинации из двух (m = 2) и четырех (m = 4) бит сформированы и, таким образом, получены последовательности анти-Грэя G^* с хэммингoвым расстоянием, равным $m - 1$. Для случая m = 3 получена вырожденная последовательность. Соответствен-

но предложенный метод генерирования исчерпывающих тестовых последовательностей для $m = 3$ оказался неработоспособным.

Определим область применения метода генерирования последовательностей анти-Грея. Для этого докажем следующее утверждение [231, 331].

Утверждение 15.2. Для четных значений величины m последовательность непереключений $t(A_{G*}) = \bar{t}(A_G) = \bar{t}_1, \bar{t}_2, \bar{t}_3, \ldots, \overline{t_{N-1}}$, где $\bar{t}_l \in \{m-1, m-2, \ldots, 2, 1, 0\}$ и $N = 2^m$ генерирует последовательность из 2^m m-разрядных тестовых наборов кода Грея с Хэммингловым расстоянием между двумя последовательными словами, равным $m - 1$.

Доказательство утверждения 15.2. Рассмотрим два случая, когда количество последовательных m-разрядных наборов $A_{G*}(k + 1)$, $A_{G*}(k + 2)$, $A_{G*}(k + 3)$, …, $A_{G*}(k + r)$ состоит из четного и нечетного числа $r < 2^m$ слов.

Согласно утверждению Гильберта [67] можно заключить, что для любого четного числа r, $r \in \{2, 4, 6, \ldots, 2^m\}$, значения последовательности переключений $t_{k+1}\, t_{k+2}\, t_{k+3} \ldots t_{k+r}$ содержат хотя бы один элемент из множества возможных элементов $\{m - 1, m - 2, \ldots, 2, 1, 0\}$ нечетное число раз. Соответственно хотя бы один из разрядов m-разрядных наборов в последовательности из r слов в классическом коде Грея будет проинвертирован нечетное число раз. Тогда в силу четности величины r этот же разряд m-битовых слов в последовательности $A_{G*}(k + 1)$, $A_{G*}(k + 2)$, $A_{G*}(k + 3)$, …, $A_{G*}(k+r)$ из r слов будет проинвертирован также нечетное число раз. Это следует из того, что $\overline{t_{k+1}}$, $\overline{t_{k+2}}$, $\overline{t_{k+3}}$, …, $\overline{t_{k+r}}$, где $\bar{t}_l \in \{m - 1, m - 2, \ldots, 2, 1, 0\}$ определяет индекс неизменяемого бита при переходе от двух последовательных m-разрядных наборов, а также из тривиального утверждения из теории чисел о том, что сумма нечетного числа только с нечетным числом дает в результате четное число. Отметим, что в случае, когда r четно, независимо от значения m всегда выполняется неравенство $A_{G*}(k) \neq A_{G*}(k + r)$ для любого $r \in \{2, 4, 6, \ldots, 2^m\}$.

Утверждение Гильберта в равной мере справедливо и для любого нечетного числа r, $r \in \{1, 3, 5, \ldots, 2^m - 1\}$, для которого значения последовательности переключений $t_{k+1}\, t_{k+2}\, t_{k+3} \ldots t_{k+r}$ также содержат хотя бы один элемент из множества возможных элементов $\{m - 1, m - 2, \ldots, 2, 1, 0\}$ нечетное число раз. Отсюда следует, что последовательность, определяющая отсутствие переключений $\overline{t_{k+1}}$, $\overline{t_{k+2}}$, $\overline{t_{k+3}}$, …, $\overline{t_{k+r}}$, задает хотя бы один разряд m-разрядных наборов в последовательности $A_{G*}(k + 1)$, $A_{G*}(k + 2)$, $A_{G*}(k + 3)$, …, $A_{G*}(k + r)$, который будет проинвертирован четное число раз. Это может привести к выполнению равенства $A_{G*}(k) = A_{G*}(k + r)$ и соответственно к невозможности получения последовательности максимальной длины.

Для $r < m$ выполнение равенства $A_{G*}(k) = A_{G*}(k + r)$ невозможно в силу того, что при последовательном переходе между словами A_{G*} во множестве из r слов, неизменяемым разрядом является только один разряд, по отношению к разрядам предыдущего слова. Соответственно, при $r < m$ будут существовать разряды слов, которые при каждом переходе будут изменять свое значение на противоположное. Общее количество подобных переключений

будет нечетным в силу нечетности величины r, что обеспечивает выполнение $A_{G*}(k) \neq A_{G*}(k + r)$.

Для случая, когда $r \geq m$ хотя бы для одного разряда в последовательности $A_{G*}(k + 1)$, $A_{G*}(k + 2)$, $A_{G*}(k + 3)$, …, $A_{G*}(k + r)$, будет выполнено четное число инверсий. Общее количество переключений для A_{G*} в r последовательных словах будет равняться $r(m - 1)$ и является величиной нечетной для m четного. Таким образом, общее количество переключений для A_{G*}, состоящее из суммы числа переключений в каждом разряде последовательных r слов, будет величиной нечетной в случае, когда хотя бы для одного разряда число переключений также будет нечетно. В результате получим, что для четного значения m всегда выполняется неравенство $A_{G*}(k) \neq A_{G*}(k + r)$ для любого $r \in \{1, 3, 5, \ldots, 2^m - 1\}$. Что и требовалось доказать [226].

Таким образом, для последовательностей анти-Грея при переходе от текущего тестового набора к последующему набору всегда изменяется $m - 1$ бит, соответственно среднее хэммингово расстояние (4.1) для таких последовательностей определяется как $HD[A_{G*}] = m - 1$.

15.6. Взаимообратные исчерпывающие тестовые последовательности

Тестирование современных вычислительных систем и их составных компонент с использованием многократных тестов и увеличение их эффективности за счет применения различных сочетаний тестовых последовательностей является весьма актуальной проблемой. Очевидно, что для достижения максимальной полноты покрытия неисправностей объекта тестирования, в результате многократного применения различных тестовых последовательностей необходимо, чтобы тесты, применяемые для каждой итерации теста, максимально отличались друг от друга. В простейшем случае это означает, что на одной и той же позиции двух тестовых последовательностей должны быть наборы, максимально отличные друг от друга, как это показывалось и доказывалось в предыдущих разделах.

Подобное требование реализовывалось в классических маршевых тестах запоминающих устройств при формировании адресных тестовых последовательностей [73, 177, 178]. В сопряженных фазах маршевого теста памяти, как правило, применяется возрастающая $A_{inc}(k)$. = 0 0 0 … 0 0, 0 0 0 … 0 1, 0 0 0 … 1 0,… , 1 1 1 … 1 1 (в десятичной системе счисления: 0, 1, 2, …, $2^m - 1$) и убывающая последовательность адресов $A_{dec}(k)$. = 1 1 1 … 1 1, 1 1 1 … 1 0, 1 1 1 … 0 1, …, 0 0 0 … 0 0 ($2^m - 1$, $2^m - 2$, $2^m - 3$, … , 0), где $k \in \{0, 1, 2, \ldots, 2^m - 1\}$. Отметим, что в случае запоминающих устройств адресные последовательности при их тестировании играют роль тестовых данных. При этом понятия возрастающей и убывающей последовательности интерпретируются как прямая и обратная последовательности. Возрастающая (прямая) последовательность тестовых наборов может представлять собой любую тестовую последовательность. Важным является факт генерирования тестовых наборов

убывающей (обратной) последовательности в обратном порядке по отношению к порядку генерирования прямой последовательности. Доказано, что данное сочетание адресных (тестовых) последовательностей является оптимальным с точки зрения покрывающей способности неисправностей запоминающих устройств [28, 73, 74, 90, 91, 199, 331]. В общем случае выбор набора взаимообратных тестовых последовательностей требует обоснованного формального подхода. В качестве меры различия адресных тестовых последовательностей часто используется арифметическое расстояние [331, 333].

Как показывалось в предыдущем разделе классическое расстояние Минковского между двумя адресными тестовыми последовательностями $A_v(k)$ и $A_w(k)$, $k \in \{0, 1, 2, \ldots, 2^m - 1\}$, при выполнении равенстве $\lambda = 1$ для его параметра λ, принимает вид арифметического расстояния, или Манхэттенского расстояния (*Manhattan Distance*) [331, 333].

Для данной меры различия в случае двукратного тестирования запоминающих устройств, когда последовательно применяются тестовые адресные последовательности $A_{inc.}(k)$ и $A_{dec.}(k)$, значение метрики различия оказывается максимальным и определяется согласно выражению [202].

$$D_{Manh\_\max}[A_{inc.}(k), A_{dec.}(k)] = \sum_{k=0}^{2^m-1} \left| 2^m - 2k - 1 \right| = 2^{2m-1}.$$

Весьма эффективным подходом для получения различных тестовых последовательностей в терминах арифметического расстояния является инвертирование определенных разрядов $A_{inc.}(k)$ для получения модифицированной последовательности $A_{mod}(k)$ из исходной последовательности $A_{inc.}(k)$ [333]. Подробное исследование эффективности применения операции инвертирования для получения управляемых вероятностных тестов приведено в главе 5 и разделе 15.2. Результаты по методам формирования тестовых последовательностей с применением операции инвертирования, а также свойств таких последовательностей широко представлены в главе 5, посвященной построению многократных управляемых вероятностных тестов.

Проведенный анализ свидетельствует о необходимости генерирования тестовых наборов $A_{dec.}(k)$ в обратной последовательности по отношению к оригинальной тестовой последовательности $A_{inc.}(k)$ и различных их модификаций как результата инвертирования определенных разрядов $A_{inc.}(k)$. В рамках рассмотренной ранее модели генерирования тестовых последовательностей (6.1), в силу эквивалентности операций сложения и вычитая по модулю два ($\oplus$), а также симметричности последовательности переключений T_{m-1} отраженного кода Грея, формирование последовательности убывающих $A_{dec.}(k)$ тестовых наборов по отношению к $A_{inc.}(k)$ соответствует следующему утверждению.

Утверждение 15.3. Убывающая последовательность тестовых наборов $A_{dec.}(k)$, $k \in \{0, 1, 2, \ldots, 2^m - 1\}$ по отношению к возрастающей последовательности $A_{inc.}(k) = A_{dec.}(2^m - 1 - i)$, формируется с использованием соотношения

(6.1) и той же порождающей матрицы V (6.3), что и для генерирования $A_{inc.}(k)$, при начальном тестовом наборе $A_{dec.}(0)$, равном $A_{inc.}(2^m - 1)$.

В силу эквивалентность операции сложения по модулю два с единицей $(a_i \oplus 1)$ и операции отрицания $\overline{a_i}$ справедливо утверждение [331, 333].

Утверждение 15.4. Модифицированная последовательность тестовых наборов $A_{mod}(k)$, $k \in \{0, 1, 2, \ldots, 2^m - 1\}$, как результат инвертирования определенных разрядов возрастающей последовательности $A_{inc.}(k)$, формируется с использованием соотношения (6.1) и той же порождающей матрицы V (6.3), что и для генерирования $A_{inc.}(k)$, при начальном тестовом наборе $A_{inc.}(0)$, содержащем единичные значения в инвертируемых разрядах.

Отметим, что данные утверждения справедливы для последовательностей, рассмотренных в разделе 6.5 и сформированных на основании соотношения (6.1) и порождающей матрицы V (6.3). В качестве примера рассмотрим последовательность $LS1$ с предельной максимальной переключательной активностью для $m = 3$, приведенной в таблице 6.11. Рассматривая $LS1$, как возрастающую последовательность $LS_{inc.}$, в таблице 15.15 приведен ряд ее модификаций.

Таблица 15.15 –*Примеры модификаций последовательности LS*1

$m = 3$	$LS_{inc.} = a_2a_1a_0$	$LS_{dec.} = a_2a_1a_0$	$LS_{mod1} = \overline{a}_2a_1a_0$	$LS_{mod2} = a_2a_1\overline{a}_0$
V	1 1 1 0 1 1 1 0 1	1 1 1 0 1 1 1 0 1	1 1 1 0 1 1 1 0 1	1 1 1 0 1 1 1 0 1
$A(0)$	0 0 0	1 0 1	1 0 0	0 0 1
$A(1)=A(0)+v_0$	1 1 1	0 1 0	0 1 1	1 1 0
$A(2)=A(1)+v_1$	1 0 0	0 0 1	0 0 0	1 0 1
$A(3)=A(2)+v_0$	0 1 1	1 1 0	1 1 1	0 1 0
$A(4)=A(3)+v_2$	1 1 0	0 1 1	0 1 0	1 1 1
$A(5)=A(4)+v_0$	0 0 1	1 0 0	1 0 1	0 0 0
$A(6)=A(5)+v_1$	0 1 0	1 1 1	1 1 0	0 1 1
$A(7)=A(6)+v_0$	1 0 1	0 0 0	0 0 1	1 0 0

Действительно, используя начальный тестовый набор $LS_{dec.}(0) = LS_{inc.}(2^3 - 1) = 1\ 0\ 1$, для порождающей матрицы V, используемой для $LS_{inc.}$, формируется последовательность $LS_{dec,.}$ обратная по отношению к $LS_{inc.}$ (см. таблицу 15.15), что соответствует утверждению 15.3. В тоже время, LS_{mod1} и LS_{mod2}, приведенные в таблице 15.15, соответствуют утверждению 15.4 [331, 333].

Заключение

В монографии были рассмотрены классические и современные подходы, применяемые для контроля и диагностики вычислительных систем, Большое внимание уделено анализу математических моделей неисправностей вычислительных систем и методам автоматизированного построения тестов и диагностических процедур. Показана специфика и общность построения тестов для трех составных частей вычислительных систем, таких как: программное обеспечение, аппаратное обеспечение и запоминающие устройства систем. Достаточно подробно изложены вопросы, касающиеся традиционных подходов к построению тестов вычислительных систем, показаны их основные ограничения и недостатки.

В монографии обосновывается доминирующее положение вероятностного тестирования по отношению к другим методологиям, используемым на современном этапе тестового диагностирования вычислительных систем. Весь последующий материал книги посвящен вопросам развития вероятностного тестирования и его различным аппроксимациям. Показано, что одним из весьма перспективных подходов для улучшения основных свойств вероятностного тестирования является управляемое вероятностное тестирование. Особое положение среди модификаций вероятностного тестирования занимает квази-вероятностное тестирование, которое характеризуется высокой степенью равномерности распределения тестовых наборов. Для всех случаев управляемого вероятностного и квази-вероятностного тестирования в монографии приводятся формальные методики и численные метрики, позволяющие автоматизировать процедуру генерирования тестов.

Рассмотренные в монографии методы псевдослучайного тестирования, многократного тестирования, исчерпывающего, а также псевдо-исчерпывающего и почти исчерпывающего тестирования рассматриваются как аппроксимации классического вероятностного тестирования, которые позволяют уменьшить сложность тестовых процедур и увеличить их эффективность.

В монографии показано, что регулярность запоминающих устройств обусловила появление таких видов тестирования, как: неразрушающее тестирование, симметричное тестирование, тестирование с использованием адаптивного сигнатурного анализа и многократное тестирования с изменяемыми адресными последовательностями и состояниями запоминающих устройств. Для указанных видов тестирования в монографии приводятся методики их построения и оценки эффективности обнаружения сложных неисправностей запоминающих устройств.

В таком виде монография представляет собой видение автора в части развития теории и практики контроля и диагностики вычислительных систем и может быть полезна специалистам в области проектирования и диагностирования современных вычислительных систем.

Литература

1. Abadir, M.S. Functional Testing of Semiconductor Random Access Memories / M.S. Abadir, H.K. Reghbati // ACM Computer Survey. – Vol. 15. № 3. – 1983. – P. 174–198.
2. Abramovici, M. Digital Systems Testing & Testable Design 1st Edition / M. Abramovici, M.A. Breuer, A.D. Friedman. − Piscataway, New Jersey: Wiley-IEEE Press, 1994. − 653 p.
3. Agrawal, P. On Monte Carlo Testing of Logic Tree Networks / P. Agrawal, V.D. Agrawal // IEEE Transactions on Computers. – Vol. C-25. – № 6. – 1976. – P. 664–667.
4. Agrawal, P. Probabilistic Analyses of Random Test Generation Method For Irredundant Combinational Logic Networks / P. Agrawal, V.D. Agrawal // IEEE Transactions on Computers. – Vol. C-24. – № 7. – 1975. – P. 691–695.
5. Agrawal, V.D. When to Use Random Testing / V.D. Agrawal // IEEE Transactions on Computers. – Vol. C-27, № 11. – 1978. – P. 1054–1055.
6. Alshahwan, N. Automated web application testing using search based software engineering / N. Alshahwan, M. Harman // In Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11). – Washington, D.C., USA, 2011. – P. 3–12.
7. Ammar, H. A Comparative Analysis of Hardware and Software Fault Tolerance: Impact on Software Reliability Engineering / H. Ammar, [et al.] // Institute for Software Research Fairmont, WV 26554 USA [Electronic resource]. – Mode of access: www.isr.wvu.edu, January 15, 1999. – Date of access: 12.07.2012.
8. Anand, S. An Orchestrated Survey on Automated Software Test Case Generation / S. Anand, [et al.] // Journal of Systems and Software. – 2013. – Vol. 86. – № 8. – P. 1978–2001.
9. Armstrong, D.B. On finding a nearly minimal set of fault detection tests for combinatorial logic nets / D.B. Armstrong // IEEE Transactions Electronics, Computers. – 1966. – Vol. 15, № 1. – P. 66–73.
10. Arnold, T.P. Visual Test 6 Bible / T.P. Arnold. – Chichester, UK: John Wiley & Sons, 1998. – 712 p.
11. Avizienis, A. Basic Concepts and Taxonomy of Dependable and Secure Computing / A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr // IEEE Transactions on Dependable and Secure Computing. – 2004. – № 1(1). – P. 11–33.
12. Balakumar, S. A BIST Approach to fault testing using pseudo-exhaustive test pattern generation / S. Balakumar, P.R. Kumar // International Journal of Advances in Engineering, Science and Technology (IJAEST). – 2013. – Vol. 2. – № 4. – P. 348–357.
13. Bamford, R., Deibler, W.J. ISO 9001: 2000 for Software and Systems Providers: An Engineering Approach / R. Bamford, W.J. Deibler. – USA: CRC Press, 2003. – 328 p.

14. Barzilai, Z. Exhaustive Generation of Bit Pattern with Application to VLSI Self–Testing / Z. Barzilai, D. Coppersmith, A. Rozenberg // IEEE Transactions on Computers. –1983. – Vol. C–31, № 2. – P.190–194.
15. Basu, A. Software Quality Assurance, Testing and Metrics / A. Basu. – Delhi: PHI Learning Pvt. Ltd., 2015. – 288 p.
16. Bennetts, R.G. Introduction to Digital Board Testing / R.G. Bennetts. – New York: Crane, Russak, 1982. – 304 p.
17. Bernet, G. A Theory of Probabilistic Functional Testing / G. Bernet, L. Bouaziz, P. LeGall // In Proceedings of the 1997 International Conference on Software Engineering. – Boton, Massachusetts, USA. – 1997. – P. 216–226.
18. Bertolino, A. Software testing research: achievements, challenges, dreams / In Proceedings of the 1st Workshop on Future of Software Engineering (FOSE ’07) at ICSE. – 2007. – P. 85–103.
19. Bleickardt, W. Multimoding and its Suppression in Twisted Ring Counters / W. Bleickardt // The Bell System Technical Journal. – 1968. – P. 2029–2050.
20. Brownlie, R. Robust testing of AT&T PMX/StarMAIL using OATS / R. Brownlie, J. Prowse, M.S. Phadke // AT&T Technical Journal. – 1992. – Vol. 71, № 3. – P. 41–47.
21. Bushnell, M. Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits (Frontiers in Electronic Testing) / M. Bushnell, V. Agrawal. – Dordrecht, Netherlands: Springer, 2004. – 690 p.
22. Buslowska, E. Multi-run march tests for Pattern Sensitive Faults in RAM / E. Buslowska, V.N. Yarmolik // In Proceedings of the IEEE East-West Design & Test Symposium (EWDTS 2018), IEEE Computer Society. – Kazan, Russia, 2018. – P. 352–357.
23. Busschbach, P. Constructive Methods to Solve the Problem of s-surjectivity. Conflict Resolution. Coding in Defective Memories / P. Busschbach // Tech. Rep. 84D005. – Ecole Nationale Superieure des Telecom, 1984.
24. Cadar, C. Symbolic execution for software testing in practice: preliminary assessment / C. Cadar, [et al.] // In Proceedings of the International Conference on Software Engineering (ICSE’11). – Waikiki, Honolulu , HI, USA, – 2011. – P. 1066–1071.
25. Cadar, C. Unassisted and automatic generation of high-coverage tests for complex systems programs / C. Cadar, D. Dunbar, D.R. Engler // In Proceedings of the Symposium on Operating Systems Design and Implementation. – San Diego, CA, USA, 2008. – P. 209–224.
26. Cascaval, P. Efficient March Tests for a Reduced 3–Coupling and 4–Coupling Faults in Random–Access Memories / P. Cascaval, S. Bennett, C. Hutanu // Journal of Electronic Testing: Theory and Applications. – 2004. – Vol. 20. – № 1. – P. 227–243.
27. Cascaval, P. Efficient March Tests for a Reduced 3–Coupling in Random–Access Memories / P. Cascaval, S. Bennett // Journal of Microprocessors and Microsystems. – 2001. – Vol. 24. – № 10. – P. 501–509.

28. Chakraborty, K. Fault–Tolerance and Reliability Techniques for High–Density Random–Access Memory / K. Chakraborty, P. Mazumder. – New Jersey: Prentice Hall, 2002. – 448 p.
29. Chan, F.T. Proportional sampling strategy guidelines for software testing practitioner / F.T. Chan, T.Y. Chen, I.K. Mak, Y.T. Yu // Information and Software Technology – 1996. – Vol. 38. – № 12. – P. 775–782.
30. Chan, K.P. Good Random Testing / K.P. Chan, T.Y. Chen, D. Towey // In Proceedings of the 9th Ada-Europe International Conference on Reliable Software Technologies (LNCS). – 2004. – P. 200–212.
31. Chan, K.P. Normalized Restricted Random Testing / K.P. Chan, T.Y. Chen, D. Towey // In Proceedings of the 8th Ada-Europe International Conference on Reliable Software Technologies (LNCS). – 2003. – P. 368–381.
32. Chan, K.P. Restricted Random Testing / K.P. Chan, T.Y. Chen, D. Towey // In Proceedings of the 7th European Conference on Software Quality. – 2002. – P. 321–330.
33. Chandramouli, R. On Testing Stuck-open Fault / R. Chandramouli // Digest of papers FTS-13. – Milan, Italy, 1983. – P. 256–265.
34. Chen, T.Y. Adaptive Random Testing / T.Y. Chen, H. Leung, I.K. Mak // In Proceedings of the 9th Asian Computer Science Conference (ASIAN 2004). – 2004. – P. 320–329.
35. Chen, T.Y. Quasi-Random Testing / T.Y. Chen, R. Merkel // IEEE Transaction on Reliability. – 2007. – Vol. 56. – № 3. – P. 562–568.
36. Chen, W.Y. Test generation for Cross-Induced Faults: Framework and computational results / W.Y. Chen, S.K. Gupta, M.A. Breuer // Journal of Electronic Testing: Theory and Applications. – 2002. – Vol. 18. – № 1. – P. 17–28.
37. Cheng, K.L. Neighborhood pattern sensitive fault testing and diagnostics for random access memories / K.L. Cheng, M.F. Tsai, C.W. Wu // IEEE Transactions On Computer–Aided Design of Integrated Circuits and Systems. – 2002. – Vol. 21, № 11. – P. 1328–1336.
38. Cheng, K.L. Neighborhood Pattern-Sensitive Fault Testing for Semiconductor Memories / K.L. Cheng., C.W. Wu // In Proceedings of International Symposium VLSI Design/CAD. – Pingtung, Taiwan, 2000. – P. 401–404.
39. Chi, H. Computational investigations of quasi-random sequences in generating test cases for specification-based tests / H. Chi, E.I. Jones // In Proceedings of the 38th on Winter Conference, ser. WSC'06. Winter Simulation Conference. – Monterey, CA, USA, 2006. – P. 975–980.
40. Chillarege, R. Orthogonal Defect Classification. A Concept for In-Process Measurements / R. Chillarege, [et al.] // IEEE Transactions on Software Engineering. – 1992. – № 18(11). – P. 943–956.
41. Cochran, W.G. Experimental Designs, 2nd Edition / W.G. Cochran, G.M. Cox. – New York: John Wiley & Sons, Inc., 1992. – 640 p.
42. Cockburn, B.E. Synthesized Transparent BIST for Detecting Scrambled Pattern-Sensitive Faults in RAMs / B.E. Cockburn, Y.F. Sat // In Proceedings of

the IEEE International Test Conference. – Washington DC, USA, 1995. – P. 23–32.
43. Cockburn, B.E. Deterministic tests for detecting scrambled pattern–sensitive faults in RAMs / B.E. Cockburn // In Proceedings of the IEEE Workshop on Memory Technology, Design and Testing. – San Jose, CA, USA, 1995. – P. 117–122.
44. Cockburn, B.E. Deterministic Tests for Detecting Single V–Coupling Faults In RAMs / B.E. Cockburn // Journal of Electronic Testing: Theory and Applications. – 1994. – Vol. 5. – № 1. – P. 91–113.
45. Cohen, D. The combinatorial design approach to automatic test generation / D.M. Cohen, S.R. Dalal, J. Parelius, G.C. Patton // IEEE Software. – 1996. – Vol. 13. – № 5. – P. 83–88.
46. Cohen, G. Exhaustive Testing of Combinatorial Circuits / G. Cohen, P. Godlewski, M.G. Karpovsky // Traitement du signal, revue scientifique francaise publiee par le GRETSI. – 1984. – Vol. 1. – № 2-2. – P. 224–226.
47. Cohen, G. Testing of Circuits with Outputs Depending on Limited Number of Inputs / G. Cohen, M.G. Karpovsky, L. Levitin // In Proceedings of the IEEE International Information Workshop. – Caesarea, Israel, 1984.
48. Cohen, M.B. Constructing test suites for interaction testing / M.B. Cohen, C.J. Colbourn, P.B. Gibbons, W.B. Mugridge // In Proceedings of the 25th International Conference on Software Engineering (ICSE'03). – Portland, USA, 2003. – P. 38–48.
49. Colanzi, T.E. Integration test of classes and aspects with a multi-evolutionary and coupling based approach / T.E. Colanzi, W.K.G. Assuncao, S.R. Vergilio, A.T.R. Pozo // In Proceedings of the 3rd International Symposium on Search Based Software Engineering (SSBSE '11). – Szeged, Hungary, 2011. – P. 188–203.
50. Crouch, A. Design-For-Test For Digital IC's and Embedded Core Systems. − New Jersey: Prentice Hall, 1999. – 347 p.
51. Dan, H. Conformance testing from message sequence charts / H. Dan, R.M. Hierons // In Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation (ICST'11), – IEEE Computer Society, Berlin, 2011. – P. 279–288.
52. Das, D. Exhaustive and Near–Exhaustive Memory Testing Techniques and their BIST Implementations / D. Das, M.G. Karpovsky // Journal of Electronic Testing: Theory and Applications. – 1997. – Vol. 10. – P. 215–229.
53. David, R. Random Pattern Testing versus Deterministic testing of RAM's / R. David, A. Fuentes, B. Courtois // IEEE Transactions on Computer. – 1989. – Vol. 38. – No 5. – P. 637–650.
54. Dekker, R.A. Fault Modeling and Test Algorithm Development for Static Random Access Memories / R. Dekker, F. Beenker, L. Thijssen // In Proceedings IEEE International Test Conference. – Washington DC, USA, 1988. – P. 343–352.

55. Dekker, R.A. Realistic Fault Model and Test Algorithms for Static Random Access Memories / R. Dekker, [et al.] // IEEE Transactions of Computers. – Vol. C-9. – № 6. – 1990. – P. 567–572.
56. Demidenko, S. March 3N and March 4N Memory Tests / S. Demidenko, V.N. Yarmolik, Y.V. Klimets, A.J. Goor // IES Journal on Electronic and Computer Engineering. – Vol. 39. – № 1. – 1999. – P. 1–5.
57. Dubrova, E. Fault-Tolerant Design / E. Dubrova. – New York: Springer-Verlag, 2013. – 185 p.
58. Eichelberger, E.B. Random Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test / E.B. Eichelberger, E. Lindbloom // IBM Journal Research and Developments. – Vol. 27. – № 3. – 1983. – P. 265–272.
59. Eiki, H. Autonomous testing and its application to testable design of logic circuits / H. Eiki, K. Inagaki, S. Yajima // In Proceedings of 10th International Symposium on Fault-Tolerant Computing. – Pittsburgh, PA, USA. – 1980. – P. 173–178.
60. Everett, G.D. Software testing: testing across the entire software development life cycle / G.D. Everett, R. McLeod. – New York, USA: John Wiley & Sons, 2007. – 335 p.
61. Fisher, M.S. Software Verification and Validation: An Engineering and Scientific Approach / Marcus S. Fisher. – USA: Springer Science & Business Media, 2007. – 172 p.
62. Fisher, R.A. The Design of Experiments, 8th Edition / R.A. Fisher. – New York: Hafner Publishing Company, 1971. – 250 p.
63. Flajolet, P. Birthday paradox, coupon collectors, caching algorithms and self-organizing search / P. Flajolet, D. Gardy, L. Thimonier // Discrete Appl. Math. – 1992. – № 39. – P. 207–229.
64. Franklin, M. Design of a BIST RAM with Row/Column Pattern Sensitive Fault Detection Capability / M. Franklin, K.K. Saluja, K. Kinoshita // Proceedings of the International Test Conference Washington D.C., USA, 29–31 August 1989 / IEEE Computer Society. Washington D.C., USA, 1989. – P. 327–336.
65. Frohwerk, R.A. Signature analysis: A new digital field service method / R.A. Frohwerk // Hewlett-Packard Journal, 1977. – № 6. – P. 2–8.
66. Gaudel, M.C. Testing can be formal, too / M.C. Gaudel // In Proceedings of the 6th International Joint Conference on Theory and Practice of Software Development, – Springer, Berlin, 1995. – P. 82–96.
67. Gilbert, T.N. Gray codes and paths on the *n*–cube / E.N. Gilbert // Bell System Technical Journal. – 1958. – № 37. – P. 815–826.
68. Godfrey, K.R. Three-level M-sequences / K.R. Godfrey // Electronics Letters. – 1966. – Vol. 2. – № 7. – P. 241 – 243.
69. Goor, A.J. An Overview of Deterministic functional RAM chip testing / A.J. Goor, C.A. Verruijt // ACM Computing Surveys. – 1990. – Vol. 22. – № 1. – P. 226–227.

70. Goor, A.J. Functional Memory Faults: A Formal Notation and a Taxonomy / A.J. Goor, Z. Al–Ars // In Proceedings 18th International IEEE VLSI Test Symposium (VTS'00) IEEE Computer Society. – Montreal, Canada, 2000. – P. 281–289.
71. Goor, A.J. March LA: A test for Linked Memory Faults / A.J. Goor, G.N. Gaydadjiev, V.N. Yarmolik, V.G. Mikitujk // In Proceedings of the 1997 European Design and Test Conference (ED&TC'97). – Paris, France, 1997. – P. 627.
72. Goor, A.J. March LR: A test for realistic linked faults / A.J. Goor, G.N. Gaydadjiev, V.N. Yarmolik, V.G. Mikitujk // In Proceedings of the 14th VLSI Test Symposium. – Princeton, NJ, USA, 1996. – P. 272–280.
73. Goor, A.J. Testing Semiconductor Memories, Theory and Practice / A.J. Goor. – UK, Chichester: John Wiley & Sons, 1991. – 536 p.
74. Goor, A.J. Towards a uniform notation for memory tests / A.J. Goor, A. Offerman, I. Schanstra // In Proceedings of the IEEE European Design and Test Conference (ED&TC'96). – Washington, DC, USA, 1996. – P. 420–427.
75. Gray, J. Why do computers stop and what can be done about it? / Jim Gray // In Proceedings of Symposium on Reliability in Distributed Software and Database Systems (SRDS-5). – 1986. – P. 3–12.
76. Grechanik, M. Is data privacy always good for software testing / M. Grechanik, C. Csallner, C. Fu, Q. Xie // In Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE'10), – San Jose, CA, USA, 2010. – P. 368–377.
77. Grieskamp, W. Model based quality assurance of protocol documentation: tools and methodology / W. Grieskamp, N. Kicillof, K. Stobie, V. Braberman // Software Testing, Verification and Reliability (STVR). – № 21. – P. 55–71.
78. Grindal, M. Combination Testing Strategies / M.Grindal, J, Offutt, S.F. Andler // GMU Technical Report ISE-TR-04-05. – July, 2004. – 32 p.
79. Grosso, C.D. Improving network applications security: a new heuristic to generate stress testing data / G. Antoniol, M.D. Penta, P. Galinier, E. Merlo // In Proceedings of the 2005 conference on Genetic and evolutionary computation. – Washington, D.C., USA, 2005. – P. 1037–1043.
80. Grottke, M. Classification of Software Faults / M. Grottke, K.S. Trivedi // In Supplemental of the Proceedings Sixteenth International IEEE Symposium on Software Reliability Engineering. – 2005. – P. 4.19–4.20.
81. Halton, J.H. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals / J.H. Halton // Numerische Mathemati. – 1960. – Vol. 2. – № 1. – P. 84–90.
82. Hamill, M. Common Trends in Software Fault and Failure Data / M. Hamill, K. Goseva-Popstojanova // IEEE Transaction on Software Engineering. – 2009. – Vol. 35. – № 4. – P. 484–496.
83. Hamming, W.R. Error Detecting and Error Correcting Codes / W.R. Hamming // Bell System Tech. Journal. – 1950. – Vol. 29. – № 2. – P. 147–160.

84. Harikrishna, B. A survey on fault tolerance in FPGAs / B. Harikrishna, S. Ravi // In Proceedings of 7th International Conference on Intelligent Systems and Control (ISCO). – Coimbatore, India, 2013. – P. 265–270.
85. Harman, M. A theoretical and empirical study of search based testing: Local, global and hybrid search / M. Harman, P. McMinn // IEEE Transactions on Software Engineering. – 2010. – Vol. 36. – № 2, – P. 226–247.
86. Harman, M. Strong higher order mutation based test data generation / M. Harman, Y. Jia, B. Langdon // In Proceedings of the 8th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11). – Vienna, Austria, 2011. – P. 212–222.
87. Harris, I.G. Hardware-Software Co-validation: Fault Models and Test Generation / I.G. Harris // IEEE Design & Test of Computers. – 2003. – Vol. 20. – № 1. – P. 40–47.
88. Hartman, A. Problems and Algorithms for Covering Arrays / A. Hartman, L. Raskin // Discrete Mathematics. – 2004. – Vol. 284, № 1-3. – P. 149−156.
89. Hartmann, J. A UML-based approach to system testing / J. Hartmann, M. Vieira, H. Foster, A. Ruder // Innovations in Systems and Software Engineering (ISSE). – 2005. – Vol. 1. – № 1. – P. 12–24.
90. Harutunyan, G. Minimal March Test Algorithm for Detection of Linked Static Faults in Random Access Memories / G. Harutunyan, V.A. Vardanian, Y. Zorian // Proceedings of the IEEE VLSI Test Symposium (VTS'06). – Berkeley, USA, 2006. – P. 120–127.
91. Harutunyan, G. Minimal March Tests for Unlinked Static Faults in Random Access Memories / G. Harutunyan, V.A. Vardanian, Y. Zorian // In Proceedings of the IEEE VLSI Test Symposium (VTS'05). – Palm Springs, CA, USA, 2005. – P. 53–59.
92. Havel, J. A randomized pseurandom number generator / J. Havel, A.N. Morozevich, V.N. Yarmolik // Kybernetika. – 1983. – Vol. 19. – № 1. – P. 58 – 65.
93. Hayes, J.P. Detection of pattern sensitive faults in random access memories / J.P. Hayes // IEEE Transactions on Computers. – 1975. – Vol. 24. – № 2. – P. 150–157.
94. Hayes, J.P. Transition Count Testing of Combinational Logic Circuits / J.P. Hayes // IEEE Transactions on Computers. – 1976. – Vol. 25. – № 6. – P. 613–620.
95. Helke, S. Automating test case generation from Z specifications with Isabelle / S. Helke, T. Neustupny, T. Santen // In Proceedings of the 10th International Conference of Z Users on the Z Formal Specification Notation (ZUM'97), – Reading, UK, 1997. – P. 52–71.
96. Hellebrand, S. Efficient Online and Offline Testing of Embedded DRAMs / S. Hellebrand, H.-J. Wunderlich, A.A. Ivaniuk, Y.V. Klimets, V.N. Yarmolik // IEEE Transactions on Computers. – 2002. – Vol. 51. – № 7. – P. 801–809.

97. Hellebrand, S. Error Detecting Refreshment for Embedded DRAMS / S. Hellebrand, H.-J. Wunderlich, A.A. Ivaniuk, Y.V. Klimets, V.N. Yarmolik // In Proceedings of 17th IEEE VLSI Test Symposium. – Dana Point, California, USA, 1999. – P. 384–390.
98. Hellebrand, S. Exploiting Symmetries to Speed up Transparent BIST / S. Hellebrand, H.-J. Wunderlich, V.N. Yarmolik // In Proceedings of 11th Workshop Testmethoden und Zuverlassigkeit von Schaltungen und Systemen. – Potsdam, Germany, 1999. – Vom 28.
99. Hitchcock, R.B. Timing Analysis of Computer Hardware / R.B. Hitchcock, G.L. Smith, D.D. Cheng // IBM J. Res. Develop. – 1982. – Vol. 26. – № 1. – P. 100–105.
100. Huima, A. Implementing conformed Qtronic / A. Huima // In Proceedings of the Joint Conference of the 19th IFIP International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing of Software (Test Com/FATES'07) LNCS. – Springer, Berlin, 2007. – P. 1–12.
101. Jain, S.K. Statistical Fault Analyses / S.K. Jain, V.D. Agrawal // IEEE Design & Test of Computers. – Vol. 2. – № 2. – 1985. – P. 38–44.
102. Jiang, B. Adaptive Random Test Case Prioritization / B. Jiang, Z. Zhang, W.K. Chan, T.H. Tse // In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. – 2009. – P. 233–244.
103. Kachan, I.V. Berechnung der Exakten Fehlererfassunng Beim Test Mit Signature analyse / I.V. Kachan, V.G. Mikitiuk, A. Strole, V.N. Yarmolik // In Proceedings of Entwurf Integrierter Schaltungen Workshop. – Karlsruhe, Germany, 1993. – P. 269–275.
104. Kajihara, S. Stuck-open faults test generation for cmos combinational circuits / S. Kajihara, N. Itazaki, K. Kinoshita // Systems and Computers in Japan. – 1991. – Vol. 22. – № 9. – P. 33–42.
105. Karpovsky, M.G. Pseudo-Exhaustive Word-Oriented DRAM Testing / M.G. Karpovsky, V.N. Yarmolik, A.J Goor // In Proceedings of the European Test Conference. – Paris, France, 1995. – P. 126–132.
106. Karpovsky, M.G. Transparent Memory Testing for Pattern Sensitive Faults / M.G. Karpovsky, V.N. Yarmolik // In Proceedings of the IEEE International Test Conference. – Washington D.C., USA, 1994. – P. 860–869.
107. Karpovsky, M.G. Transparent Random Access Memory Testing for Pattern Sensitive Faults / M.G. Karpovsky, V.N. Yarmolik // Journal of Electronic Testing: Theory and Applications. – 1994. – Vol. 5. – № 1. – P. 91–113.
108. Karpovsky, M.G. Universal Tests for Detection of Input/Output Stuck-at and Bridging Faults / M.G. Karpovsky // IEEE Transaction on Computer. – 1973. – № 12. – P. 1194–1197.
109. Karpovsky, M.G. Yarmolik V.N. Transparent Memory BIST / M.G. Karpovsky, V.N. Yarmolik // In Proceedings of the IEEE International Workshop on Memory Technology, Design and Testing. – San Jose, USA, 1994. – P. 106–111.

110. Kastensmidt, F. Fault-Tolerance Techniques for SRAM-Based FPGAs (Frontiers in Electronic Testing) / Fernanda Kastensmidt, Ricardo Reis. – New York, USA: Springer, 2006. – 184 p.
111. Kastensmidt, F. PGAs and Parallel Architectures for Aerospace Applications: Soft Errors and Fault-Tolerant Design 1st ed. / F. Kastensmidt, P. Rech. – Cham, Switzerland: Springer, 2016. – 325 p.
112. King, J.C. A new approach to program testing / J.C. King // Part of the Lecture Notes in Computer Science book series LNC. – 1975. – Vol. 23. – P. 278–290.
113. Knaizuk, J. An Optimal Algorithm for Testing Stuck-at faults in Random Access Memories / J. Knaizuk, C.R. Hartman // IEEE Transactions on Computers. – Vol. C-26. – № 11. – 1977. – P. 1141–1144.
114. Knuth, D.E. The Art of Computer Programming: Volume 3, Sorting and Searching. Second edition / D. E. Knuth. – Reading, Massachusetts: Addison-Wesley, 1998. – 730 p.
115. Knuth, D.E. The errors of TEX / Donald E. Knuth // Software–Practice and Experience. – 1989. – № 19(7). – P. 607–685.
116. Kristis, A. Delay Fault Testing for VLSI Circuits / A. Krstic, K.-T. Cheng. – New York, NY, USA: Springer Science+Business Media, 1998. – 191 p.
117. Kroese, D.P. Why the Monte Carlo method is so important today / D.P. Kroese, T. Brereton, T. Taimre, Z.I. Botev // WIREs Computational Statistics. – 2014. – № 6. – P. 386–392.
118. Kuhn, R. D. Pseudo-Exhaustive Testing for Software / R. D. Kuhn, V. Okun // In Proceedings of 30th Annual IEEE/NASA Software Engineering Workshop. – Columbia, MD, USA – 2006. – P. 153–158.
119. Kuo, F.C. An in-depth study of mirror adaptive random testing / F.C. Kuo // In Proceedings of the 14th European Conference on Software Quality. – Jeju, Korea. – 2009. – P. 51–58.
120. Lala, P.K. Fault Tolerant and Fault Testable Design. – London: Prentice Hall International, 1985. – 320 p.
121. Levitin, L.B. Efficient Exhaustive Test Based on MDS Codes // L.B. Levitin, M.G. Karpovsky // In Proceedings of the IEEE Int. Symposium on Information Theory. – Ann Arbor, USA. – 1986. – P. 64.
122. Lewis, T.G. Generalized feedback feedback shift register pseudorando number algorithm / T.G. Lewis, W.H. Payne // Journal of ACM. – 1973. – Vol. 20. – № 3. – P. 456–468.
123. MacWilliams, F.J. The Theory of Error-Correcting Codes / F.J. MacWilliams, N.J.A. Sloane. – Amsterdam, North-Holland Publishing Company, 1977. – 762 p.
124. Majumdar, R. Symbolic robustness analysis / R. Majumdar, I. Saha // In Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS'09). – Washington DC, US, 2009. – P. 355–363.

125. Malaiya, Y.K. An examination of fault exposure ratio / Y.K. Malaiya, A. Mayrhauser, P.K. Srimani // IEEE Transactions on Software Engineering. – 1993. – Vol. 19. – № 11. – P. 1087–1094.
126. Malaiya, Y.K. Antirandom Testing: Getting the most out of Back-Box Testing / Y.K. Malaiya, S. Yang // In Proceedings of Sixth International Symposium on Software Reliability Engineering. – 1995. – P. 86–95.
127. Malaiya, Y.K. The coverage problem for random testing / Y.K. Malaiya, S. Yang // In Proceedings of the IEEE International Test Conference. – Washington DC, USA. – 1984. – P. 237–242.
128. Mandl, R. Orthogonal Latin squares: an application of experiment design to compiler testing / R. Mandl // Communications of the ACM. – 1985. – Vol. 28. – № 10. – P. 1054–1058.
129. Marinescu, M. Simple and Efficient Algorithms for Functional RAM Testing / M. Marinescu // In Proceedings of the IEEE International Test Conference. – Washington DC, USA, 1982. – P. 236–239.
130. Mayrhause, A. Fast Antirandom (FAR) Test Generation / A. Mayrhause, T. Chen, A. Hajjar, A. Bai, C. Anderson // In Proceedings of the Third IEEE International High-Assurance System Engineering Symposium. – 1998. – P. 262–269.
131. Mikitjuk, V.G. RAM testing algorithm for Detection Multiple Linked Faults / V.G. Mikitujk, V.N. Yarmolik // In Proceedings of the 1996 European Design and Test Conference (ED&TC'96). – Paris, France, 1996. – P. 435–440.
132. Mourad, S. Principles of Testing Electronic Systems / S. Mourad, Y. Zorian. – Somerset, New. Jersey: John Wiley & Sons Inc. – 2000. – 424 p.
133. Mrozek, I. Problemu Funkcjonalnego Testowania Pamieci RAM. – Bialystok, Polska: Oficyna Wydawnicza Politechniki Bialostockiej. – 2009. – 264 s.
134. Mrozek I., Iterative Antirandom Testing / I. Mrozek, V.N. Yarmolik // Journal of Electronic Testing: Theory and Applications (JETTA). – 2012. – Vol. 9. – № 3. – P. 251–266.
135. Mrozek, I. Antirandom test vectors for BIST in Hardware/Software systems / I. Mrozek, V.N. Yarmolik // Fundamenta Informaticae. – 2012. – № 119. – P. 1–23.
136. Mrozek, I. Multiple Controlled Random Testing / I. Mrozek, V.N. Yarmolik // Fundamenta Informaticae. – 2016. – Vol. 144. – № 1. – P. 23–43.
137. Mrozek, I. Konrolowane generowanie optymalnych testow losowych / I. Mrozek, V.N. Yarmolik // Elektronika. – 2015. – № 8. – S. 64–66.
138. Mrozek, I. Generowanie wielopzebiegowych kontrolowanych testow / I. Mrozek, V.N. Yarmolik // Elektronika. – 2016. – № 11. – S. 99–101.
139. Mrozek, I. Method for Generation Multiple Controlled Random Tests / I. Mrozek, V.N. Yarmolik // In Proceedings of 15th IEEE International Computer Information Systems and Industrial Management Applications (CISIM 2016). – Vilnius, Lithuania, 2016. – P. 429–440.

140. Mrozek, I. Two-Run RAM March Testing with Address Decimation / I. Mrozek, V.N. Yarmolik // IEEE Journal of Circuits, Systems and Computers. – 2017. – Vol. 26. – № 2. – P. 1–17.
141. Mrozek, I. Pseudo-exhaustive random access memory testing based on march tests with random background variation / I. Mrozek, V. Yarmolik // In Proceedings of the IEEE East-West Design & Test Symposium (EWDTS 2018). – Kazan, Russia, 2018. – P. 344–351.
142. Mrozek, I. Multi-run Memory Tests for Pattern Sensitive Faults. – Netherlands: Springer International Publishing AG, 2018. – 135 p.
143. Myers, G.J. The art of software testing / Glenford J. Myers; Revised and updated by Tom Badgett and Todd Thomas, with Corey Sandler. – 2nd ed. – New Jersey: John Wiley & Sons, Inc., 2004. – 234 p.
144. Nair C. Efficient Algorithms for Testing Semiconductor Random-Access Memories / C. Nair, S. Thatte, J. Abraham // IEEE Transaction on Computers. – 1978. – Vol. 27. – № 6. – P. 572–576.
145. Nair, R. Comments on an Optimal Algorithms for Testing Stuck-at Faults in Random Access Memories / R. Nair // IEEE Transactions on Computers. – Vol. C-28. – № 3. – 1979. – P. 258–261.
146. Nguyen, C. Evolutionary testing of autonomous software agents / C. Nguyen, A. Perini, P. Tonella, S. Miles, M. Harman, M. Luck // In Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'09). – Budapest, Hungary, 2009. – P. 521–528.
147. Nicolaidis M. Zero Aliasing ROM BIST / M. Nicolaidis, O. Kebichi, V.N. Yarmolik // Journal of Electronic Testing: Theory and Applications. – 1994. – Vol. 5. – № 4. – P. 377–381.
148. Nicolaidis, M. Theory of transparent BIST for RAMs // IEEE Transactions on Computers. – 1996. – Vol. 45. – № 10. – P. 1141–1156.
149. Nicolaidis, M. Transparent BIST for RAMs / M. Nicolaidis // In Proceedings of the IEEE International Test Conference. – Washington DC, USA, 1992. – P. 598–607.
150. Niederreiter, H. Point sets and sequences with small discrepancy / H. Niederreiter // Monatshefte fur Mathematik. – 1987. – Vol. 104. – № 4. – P. 273–337.
151. Niggemeyer, D. Diagnostic testing of embedded memories based on output tracing / D. Niggemeyer, M. Redeker, E. Rudnick // Records of the IEEE Workshop Memory Technology, Design and Testing. – Washington, DC, USA, 2000. – P. 113–118.
152. Niggemeyer, D. Integration of Non-classical Faults in Standard March Tests / D. Niggemeyer, M. Redeker, J. Otterstedt // Records of the IEEE Int. Workshop on Memory Technology, Design and Testing. – San Jose, USA, 1998. – P. 91–98.
153. Offutt, A. J. Generating tests from UML specifications / A.J. Offutt, A. Abdurazik // In Proceedings of the 2nd International Conference on The Unified

Modeling Language: Beyond the Standard (UML'99). – Springer, France, 1999. – P. 416–429.
154. Pancley, A. Reconfiguration technique for reducing test time and test data volume in Illinois Scan Architecture based designs / A. Pancley, J.H. Patel // In Proceedings of 20th IEEE VLSI Test Symposium. – Monterey, CA, USA. – 2002. – P. 9–15.
155. Pandey, P. Recursive Pseudo-Exhaustive Two-Pattern Generator / P. Pandey, V. Kapse, M.T. Hod // International Journal of Advanced Research in Computer Engineering & Technology. – 2012. – Vol. 1. – № 5 – P. 380–385.
156. Parker, K.P. Analyses of Logical Circuits with Faults using Input Signal Probability / K.P. Parker, E.J. MacCluskey // IEEE Transaction on Computers. – Vol. C-24. – № 5. – 1975. – P. 573–578.
157. Parker, K.P. Compact testing: Testing with compressed data / K.P. Parker // In Proceedings of 6th International Symposium on Fault-Tolerant Computing. – Pittsburgh, PA, USA. – 1976. – P. 93–98.
158. Parker, K.P. Probabilistic Treatment of General Combinational Networks / K.P. Parker, E.J. MacCluskey // IEEE Transaction on Computers. – Vol. C-24. – № 6. – 1975. – P. 668–670.
159. Peterson, W.W. Error-Correction Codes / W.W. Peterson, E.J. Weldon. – Cambridge, Massachusetts, London, England: The MIT Press, 1972. – 572 p.
160. Pezz`e, M. Software Testing and Analysis: Process, Principles and Techniques / M. Pezz`e, M. Young. – Hoboken, N.J.: John Wiley & Sons Inc., 2007. – 488 p.
161. Pries, K.H. Quigley, J.M. Testing Complex and Embedded Systems / K.H. Pries, J.M. Quigley. – USA: CRC Press, 2011. – 319 p.
162. Prince, B. High–performance memories: new architecture DRAM's and SRAM's, evolution and function / B. Prince – UK, Chichester: John Wiley & Sons Ltd., 1996. – 338 p.
163. Rajsuman, R. System-On-A-Chip: Design and Test / R. Rajsuman. – London: Artech House Publishers, 2000. – 303 p.
164. Reddy, S.M. Data Compression Techniques for Buit-In Self-Test / S.M. Reddy, K.K. Saluja, M.G. Karpovsky // IEEE Transaction on Computers. – 1988. – Vol. 37. – № 9. – P. 1151–1156.
165. Roth, J.P. Diagnostic of automata failure: a calculus and a method / J.P. Roth // IEEE Transaction on Computers. – 1966. – Vol. 10. – № 7. – P. 278–291.
166. Saluja, K.K. Test Pattern Generation for API Faults in RAM / K.K. Saluja, K. Kinoshita // IEEE Tansactions on Computers. – 1985. – Vol. C-34. – № 3. – P. 284–287.
167. Santelices, R.A. Test-suite augmentation for evolving software / R.A. Santelices, [et al.] // In Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08), – L'Aquila, Italy, 2008. – 218–227.
168. Savage, C. A survey of combinatorial Gray code / C. Savage. – SIAM Review. – 1997. – Vol. 39. – № 4. – P. 605–629.

169. Savir, J. Random Pattern Testability / J. Savir, G.S. Ditlow, P.Y. Bardell // IEEE Transactions on Computers. – Vol. C-33. – № 1. – 1984. – P. 79–90.
170. Savir, J. Syndrome-Testable Design of Combinational Circuits / J. Savir // IEEE Transactions on Computers. – 1980. – Vol. 29. – № 6. – P. 442–451.
171. Schneider, R.R. On necessity to examine D-chains in diagnostic test generation / R.R. Schneider // IBM Journal of Research and Development. – 1967. – Vol. 11. – № 1. – P. 114–117.
172. Schnekenburger, C. Towards the determination of typical failure patterns / C. Schnekenburger, J. Mayer // In Proceedings of Fourth Intern. Workshop on Software Quality Assurance: in conjunction with the 6th ESE/FSE joint meeting, ser. SOQUA'07. – New York, USA: ACM, 2007. – P. 90–93.
173. Segal, A. Using electrical bitmap results from embedded memory to enhance yield / A. Segal, [et al.] // IEEE Design & Test of Computers – 2001. – Vol. 15. – № 3. – P. 28–39.
174. Sellers F.F. Analyzing errors with the Boolean difference / F.F. Sellers, M.Y. Hsiao, C.L. Bearnson // IEEE Transaction on Computers. – 1968. – Vol. 17. – № 7. – P. 676–683.
175. Seth, S. A statistical theory of digital circuits testability / S. Seth, V. Agrawal, H. Farhat // IEEE Transactions on Computers. – 1990. – Vol. C–39. – № 4. – P. 582–586.
176. Shahbazi, A. Centroidal Voronoi Tessellation – a New Approach to Random Testing / A. Shahbazi, A.F. Tappenden, J. Miller // IEEE Transaction on Software Engineering. – 2013. – Vol. 39. – № 2. – P. 163–183.
177. Sharma, A.K. Advanced Semiconductor Memories: Architectures, Designs, and Applications / A.K. Sharma. – London: John Wiley & Sons, 2003. – 652 p.
178. Sharma, A.K. Semiconductor Memories: Technology, Testing, and Reliability / A.K. Sharma. – London: John Wiley & Sons, 2002. – 480 p.
179. Smith, I.F. Measures of Effectiveness of Signature Analyses / I.F. Smith // IEEE Transactions on Computers. – 1980. – Vol. 29. – № 6. – P. 510–514.
180. Sobol, I.M. Uniformly distributed sequences with an additional uniform property / I. M. Sobol // USSR Comput. Math. Phys. – 1976. – Vol. 16. – P. 236–242.
181. Sokol, B. Address Sequence for March Tests to Detect Pattern Sensitive Faults / B. Sokol, S.V. Yarmolik // In Proceedings of 3rd IEEE International Workshop on Electronic Design Test and Applications (DELTA'06). – Kuala Lumpur, Malaysia. – 2006. – P. 354–357.
182. Sokol, B. Impact of the Address Changing on the Detection of Pattern Sensitive Faults / B. Sokol, I. Mrozek, V.N. Yarmolik // In book: Information Processing and Security Systems. – London: Springer Science + Business Media, Inc., 2005. – P. 217–226.
183. Sokol, B. Optymalne adresowanie pamieci przy wykorzystaniu usterek uwarunkowanych zawartoscia (PSF) / B. Sokol, S.V. Yarmolik // Pomiary, Automatyka, Kontrola. – 2006. – № 6–bis. – P. 41–43.

184. Stokes, D. Testing Computers Systems for FDA/MHRA Compliance / David Stokes. – USA: CRC Press, 2003. – 136 p.
185. Suk, D.S. A March Test for Functional Faults in Semiconductor Random-Access Memories / D.S. Suk, S.M. Reddy // IEEE Transactions on Computers. – Vol. C–30. – № 12. – 1981. – P. 982–985.
186. Tang, D.T. Exhaustive Test Pattern Generation with Constant Weight Vectors / D.T. Tang, L.S. Woo // IEEE Transactions on Computers. –1983. – Vol. C–32, № 12. – P. 1145–1150.
187. Tang, D.T. Iterative Exhaustive Pattern Generation for Logic Testing / D.T. Tang, C.L. Chen // IBM Journal Res. Develop. – 1984. – Vol. 28. – № 2. – P. 212–219.
188. Tappenden, A.F. A Novel Evolutionary Approach for Adaptive Random Testing / A.F. Tappenden, J. Miller // IEEE Transaction on reliability. – 2009. – Vol. 58. – № 4. – P. 619–632.
189. Tausworthe, R.C. Rundom number generated by linear reccurence modulo two / R.C. Tausworthe // Mathematics of Computation. – 1965. – № 90. – P. 201–209.
190. Teller-Giron, R. Random Fault Detection in Logical Networks / R. Teller-Giron, R. David // In Proceedings of International Symposium on Discrete Systems. – Riga, USSR, 1974. – P. 232–241.
191. Thompson, A.C. Minkowski Geometry / A.C.Thompson. – NY: Cambridge U.P., 1996. – 364 p.
192. Tootill, J.P.R. An asymptotically random Tausworthe sequence / J.P.R. Tootill, W.D. Robinson, D.I. Eagle // Journal of ACM. – 1973. – Vol. 20. – № 3. – P. 469–481.
193. Tretmans, J. TorX: Automated model based testing / J. Tretmans, E. Brinksma // In Proceedings of the 1st European Conference on Model-Driven Software: Foundations and Applications (ECMDA-FA'05). – Nuremberg, Germany, 2005. – P. 31–43.
194. Tubbs, J.D. A note on binary template matching / J.D. Tubbs // Pattern Rec. – 1989. – Vol. 22. – № 4. – P. 359–366.
195. Virupakshia, A.R. A Simple Random Test Procedure for Detection of Single Intermittent Fault in Combinational Circuits / A.R. Virupakshia, V.C. Reddy // IEEE Transactions on Computers. – Vol. C-32, № 6. – 1983. – P. 594–597.
196. Walcott, K.R. Time aware test suite prioritization / K.R. Walcott, M.L. Soffa, G.M. Kapfhammer, R.S. Roos // In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'06). – Portland, USA, 2006. – P. 1–12.
197. Wang, C.W. Fault Pattern Oriented Defect Diagnosis for Memories / C.W. Wang, [et al.] // In Proceedings of Intern. Test Conference (ITC 2003). – Charlotte, NC, USA, 2003. – P. 29–38.
198. Wang, L.T. System-on-Chip Test Architectures: Nanometer Design for Testability / L.T. Wang, C. Stroud, N. Touba. – San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., – 2008. – 896 p.

199. Wang, L.T. VLSI Test Principles and Architectures: Design for Testability / L.T. Wang, C.W. Wu, X. Wen. – San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., – 2006. – 808 p.
200. Wang, W.L. A Complete Memory Address Generator for Scan Based March Algorithms / W.L. Wang, K.J. Lee // In Proceedings of the IEEE Workshop on Memory Technology, Design, and Testing (MTDT'05). – Taipei, Taiwan, 2005. – P. 83–88.
201. White, L. A domain strategy for computer program testing / L. White, E. Cohen // IEEE Transaction on Software Engineering. – 1980. – Vol. SE–6, № 3. – P. 247–257.
202. Wu, S.H. Antirandom Testing: A Distance-Based Approach / S.H. Wu, S. Jandhyala, Y.K. Malaiya, A.P. Jayasumana // VLSI Design. – 2008. – № 2. – P. 1–9.
203. Wu, S.H. Antirandom vs. Pseudorandom Testing / S.H. Wu, Y.K. Malaiya, A.P. Jayasumana // In Proceedings of the IEEE International Conference on Computer Design (ICCD'98). – Austin, Texas, USA, 1998. – P. 221 – 229.
204. Xu, S. Orderly Random Testing for Both Hardware and Software / S. Xu // In Proceedings of Pacific Rim International Symposium on Dependable Computing. – Taipei, Taiwan. – 2008. – P. 160–167.
205. Xu, S. Maximum Distance Testing / S. Xu, J. Chen // In Proceedings of the 11th IEEE Asian Test Symposium (ATS'02). – 2002. – P. 15–20.
206. Yarmolik, V.N. Generation and application of pseudorandom sequences for random testing / V.N. Yarmolik, S.N. Demidenko. – New York, NY, USA: John Wiley & Sons, Inc., 1988. – 167 c.
207. Yarmolik, V.N. Fault Diagnostics of Digital Circuits / V.N. Yarmolik. – Chichester, England and New York: John Wiley & Sons, 1990. – 205 p.
208. Yarmolik, V.N. Self-Testing VLSI Design / V.N. Yarmolik, I.V. Kachan. – Amsterdam: Elsevier Science Publishers, 1993. – 345 p.
209. Yarmolik V.N. Address sequences for multiple run March tests / V.N. Yarmolik, S.V. Yarmolik // Automatic Control and Computer Sciences. – 2006. – Vol. 47. – № 5. – P. 59–68.
210. Yarmolik, V.N. Controlled Method of Random Test Synthesis / V.N. Yarmolik, I. Mrozek, S.V. Yarmolik // Automatic Control and Computer Sciences. – 2015. – Vol. 49. – № 6. – P. 395–403.
211. Yarmolik V.N. Self-Adjusting Output Data Compression: An Efficient BIST Technique for RAMs / V.N. Yarmolik, S. Hellebrand, H.-J. Wunderlich // In Proceedings of Design, Automation & Test in Europe (DATE-98). – Paris, France, 1998. – P. 173–179.
212. Yarmolik, V.N. Aliasing-Free Signature Analysis for RAM BIST / V.N. Yarmolik, M. Nicolaidis, O. Kebichi // In Proceedings of the IEEE International Test Conference. – Washington DC, USA, 1994. – P. 368–377.
213. Yarmolik, V.N. Built-in self-test and diagnosis for RAM based on self-adjusting output data compression / V.N. Yarmolik, A.A. Ivaniuk, M. Krips // In Proceedings of 5th International Workshop on Design and Diagnostics of

Electronic Circuits and Systems (DDECS 2002). – Brno, Czech Republic, 2002. – P. 360–363.
214. Yarmolik, V.N. Contents Independent RAM Built In Self-Test and Diagnoses based on Symmetric Transparent Algorithm / V.N. Yarmolik // In Proceedings of 3rd Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'2000). – Smolenice, Slowakia, 2000. – P. 220–227.
215. Yarmolik, V.N. Contents Independent RAM Built-In Self-Test for Visual Objects Storage / V.N. Yarmolik // In book Digital Visual Objects Processing, Belarusian National Academy of Science. – Minsk, Belarus, 2000. – P. 211–220.
216. Yarmolik, V.N. Counter sequences for memory test address generation / V.N. Yarmolik, B. Sokol, S.V. Yarmolik / In Proceedings of 12th Mixed Design of Integrated Circuits and Systems (MIXDES'05). – Krakow, Poland, 2005. – P. 413–418.
217. Yarmolik, V.N. Exact Aliasing Computation AND/OR Aliasing Free Design for RAM BIST / V.N. Yarmolik, M. Nicolaidis // In Proceedings of the IEEE International Workshop on Memory Technology Design and Testing. – San Jose, CA, USA, 1993. – P. 20–25.
218. Yarmolik, V.N. March 3N and March 4N Memory Tests / V.N. Yarmolik, Y.V. Klimets, S.N. Demidenko // In Proceedings of the IEEE International Workshop on Memory Technology, Design and Testing. – San Jose, USA, 1997. – P. 99–102.
219. Yarmolik, V.N. RAM Diagnostic Strategy Based on March Tests / V.N. Yarmolik, A.A. Ivaniuk D.S. Petronenko // In Proceedings of 10th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES'2003). – Lodz, Poland, 2003. – P. 556–559.
220. Yarmolik, V.N. RAM Diagnostic Tests / V.N. Yarmolik, Y.V. Klimets, A.J. van de Goor, S.N. Demidenko // In Proceedings of the IEEE International Workshop on Memory Technology, Design and Testing. – Singapore, 1996. – P. 100–102.
221. Yarmolik, V.N. Symmetric Transparent BIST for RAMs / V.N. Yarmolik, Y.V. Klimets // In book Digital Visual Objects Processing, Belarusian National Academy of Science. – Minsk, Belarus, 1999. – Vol. 2 – P. 35–43.
222. Yarmolik, V.N. Symmetric Transparent BIST for RAMs / V.N. Yarmolik, S. Hellebrand, H.-J. Wunderlich // In Proceedings of Design, Automation & Test in Europe (DATE-99). – Munich, Germany, 1999. – P. 173–179.
223. Yarmolik, V.N. Transparent Tests for Semiconductor Memory / V.N. Yarmolik, Y.V. Klimets, S.N. Demidenko, V. Piuri // In Proceedings of International Symposium on IC Technology, Systems and Applications. – 1997. – P. 192–195.
224. Yarmolik, V.N., Ivaniuk A.A., Testing and Diagnosis Algorithms for Embedded RAM's / V.N. Yarmolik, A.A. Ivaniuk // In Proceedings of 7th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES'2000). – Gdynia, Poland. – 2000. – P. 547–550.

225. Yarmolik,V.N. March PS(23N) Test for DRAM Pattern-Sensitive Faults / V.N. Yarmolik, Y.V. Klimets, S.N. Demidenko // In Proceedings of the Seventh Asian Test Symposium. – Singapure, 1998. – P. 354–357.
226. Yarmolik, S.V. Address Sequences and Backgrounds with different Hamming Distance for Multiple run March tests / S.V. Yarmolik // IEEE International Journal of Applied Mathematics and Computer Science. – 2008. – Vol. 18. – № 3. – P. 329–339.
227. Yarmolik, S.V. The Syntheses of Probability Tests with a Small Number of Kits / S.V. Yarmolik, V.N. Yarmolik // Automatic Control and Computer Science. – 2011. – Vol. 45. № 4. – P. 133–141.
228. Yarmolik, S.V. Controlled Random Tests / S.V. Yarmolik, V.N. Yarmolik // Automation and Remote Control. – 2012. – Vol. 73. – № 10. – P. 1704 – 1714.
229. Yarmolik, S.V. Generating Modified Sobol Sequences for Multiple Run March Memory Test / V.N. Yarmolik, S.V. Yarmolik // Automatic Control and Computer Sciences. – 2013. – Vol. 47. – № 5. – P. 242–247.
230. Yarmolik, S.V. Address Sequences Generation for Multiple Run Memory Testing / S.V. Yarmolik, I. Mrozek, B. Sokol // In Proceedings of 6th IEEE International Computer Information Systems and Industrial Management Applications (CISIM 2007). – Elk, Poland, 2007. – P. 341–344.
231. Yarmolik, S.V. Address Sequences with Different Average Hamming Distance for Multiple Run Memory Tests / S.V. Yarmolik // Abstracts of the 1st International Conference for Young Researchers: Computer Science, Control, Electrical Engineering and Telecommunications (ICYR). – Zielona Gora, Poland, 2006. – P. 67–68.
232. Yarmolik, S.V. Memory Address Generation for Multiple Run March Tests with Different Average Hamming Distance / S.V. Yarmolik, V.N. Yarmolik // In Proceedings of the IEEE East–West Design and Test Workshop (EWDTW'06). – Sochi, Russia, 2006. – P. 212–216.
233. Yarmolik, S.V. Modified Gray and Counter Sequences for memory test address generation / S.V. Yarmolik, V.N. Yarmolik // In Proceedings of 13th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES'2006). – Gdynia, Poland, 2006. – P. 572–576.
234. Yarmolik, S.V. Multi–Background Memory Testing / S.V. Yarmolik, I. Mrozek // // In Proceedings of 14th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES'2007). – Ciechocinek, Poland, 2007. – P. 511–516.
235. Yarmolik, S.V. Optimal Memory Address Seeds for Pattern Sensitive Faults Detection / S.V. Yarmolik, B. Sokol // In Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS'2006). – Prague, Czech Republic, 2006 – P. 220–221.
236. Yarmolik, S.V. Multi background memory March testing / S.V. Yarmolik // In Proceedings of 6th International Conference Computer-Aided Design of

Discrete Devices (CAD DD'07). – Minsk, Belarus, 2007. – Vol. 2. – P. 213–220.
237. Zankovich, A.P. Local Symmetric Transparent Memory Testing / A.P. Zankovich, V.N. Yarmolik // In Proceedings of 6th International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS 2003). – Poznan, Poland, 2003. – P. 297–298.
238. Zankovich, A.P. Nondestructive RAM Testing by Analyzing the Output Data for Symmetry / A.P. Zankovich, V.N. Yarmolik // Automation and Remote Control. – 2003. – Vol. 64. – № 9. – P. 1488 – 1500.
239. Zhang, P. Automatic generation of load tests / P. Zhang, S.G. Elbaum, M.B. Dwyer // In Proceedings of the 26th IEEE/ACM International Conference on Auto-mated Software Engineering (ASE'11). – Lawrence, Kansas, USA, 2011. – P. 43–52.
240. Zhou, Z.Q. Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing / Z.Q. Zhou // In Proceedings of the 34th IEEE Computer Software and Applications Conference Workshops. – Seoul, Korea, 2010. – P. 208–213.
241. Zhu, H. Software unit test coverage and adequacy / H. Zhu, P.A.V. Hall, J.H.R. May // ACM Computing Surveys. – 1997. – Vol. 29. – № 4. – P. 366–427.
242. Антонов, И.А. Экономичный способ вычисления *ЛПτ*-последовательностей / И.А. Антонов, В.М. Салеев // Журнал вычислительная математика и математическая физика. 1979, – Т. 19. – № 1. – С. 243–245.
243. Баруча-Рид, А. Элементы теории Марковских процессов и их приложений / А. Баруча-Рид. – М.: Наука, 1969. – 512 с.
244. Бернштейн, М.С. Генератор псевдослучайных последовательностей испытательных сигналов / М.С. Бернштейн, Л.Ф. Карачун, А.М. Романкевич, А.М. Руккас // Сб., Механизация и автоматизация управления. – 1978. – Вып. 1. – С. 57–60.
245. Бернштейн, М.С. Метод статистического контроля логических схем / М.С. Бернштейн, А.М. Романкевич // Кибернетика. – 1974. – № 4. – С. 52–67.
246. Бернштейн, М.С. О статистическом контроле многокантакных схем / М.С. Бернштейн, А.М. Романкевич // Автоматика и вычислительная техника. – 1975. – № 1. – С. 14–20.
247. Быков, Ю.В. Выбор оптимальных многопараметрических распределений вероятностей входных переменных при вероятностном тестировании цифровых схем / Ю.В. Быков, Л.А. Закревский, В.Н. Ярмолик // Автоматика и телемеханика. – 1994. – № 4. – С. 144–150.
248. Быков, Ю.В. Синтез преобразователей псевдослучайных кодов для тестирования цифровых схем / Ю.В. Быков, В.Н. Ярмолик // Автоматика и телемеханика. – 1994. – № 10. – С. 151–157.

249. Быков, Ю.В. Синтез преобразователей псевдослучайных кодов для реализации самотестирования цифровых устройств, удовлетворяющих требованиям стандарта / Ю.В. Быков, В.Н. Ярмолик // Автоматика и телемеханика. – 1996. – № 4. – С. 148–154.
250. Быков, Ю.В. Диагностика неисправностей кмоп-схем на основе Iddq тестирования / Ю.В. Быков, А.А. Иванюк, А.И. Янушкевич, В.Н. Ярмолик // Автоматика и телемеханика. – 1999. – № 7. – С. 142–153.
251. Вейцман, И.Н. Вероятностные системы синтеза контролирующих тестов на логические схемы / И.Н. Вейцман, В.Е. Жук, А.Б. Флеров // Вопросы радиоэлектроники. Серия ЭВТ. – 1971. – Вып. 6 – С. 22–26.
252. Верниковский, Е.А. Способы подключения резервных элементов вместо дефектных с помощью плавких вставок в полупроводниковых запоминающих устройствах / Е.А. Верниковский, В.К. Конопелько // Электронная техника. – 1984. – Вып. 1. – С. 42–48.
253. Гавел, Я. Генератор случайного процесса ГЕНАП-3 / Я. Гавел // Автоматика и телемеханика. – 1975. – № 3. – С. 171–175.
254. Георгиев, Н.В. Функциональный контроль полупроводниковых запоминающих устройств / Н.В. Георгиев, Б.В. Орлов // Электронная промышленность. – 1980. – № 6. – С. 3–21.
255. Гладкий, В.С. Вероятностные вычислительные модели / В.С. Гладкий. – М.: Наука, 1973. – 300 с.
256. Гольдман, Р.С. Техническая диагностика цифровых устройств / Р.С. Гольдман, В.П. Чипулис. – М.: Энергия, 1976. – 224 с.
257. Горяшко, А.П. Проектирование легко тестируемых дискретных устройств: идеи, методы, реализация / А.П. Горяшко // Автоматика и телемеханика. – 1984. – № 7. – С. 5–33.
258. Закревский, Л.А. Метод граничного сканирования и его использование для тестирования цифровых устройств / Л.А. Закревский, Е.П. Калоша, И.В. Качан, Н.Н. Хаткевич, В.Н. Ярмолик // Автоматика и телемеханика. – 1994. – № 1. – С. 3–31.
259. Закревский, Л.А. Неразрушающее тестирование памяти с применением встроенных средств самотестирования / Л.А. Закревский, А.А. Иванюк, А.И. Янушкевич, В.Н. Ярмолик // Автоматика и телемеханика. – 1999. – № 2. – С. 120–128.
260. Занкович, А.П. Встроенная аппаратура неразрушающего самотестирования для схем ОЗУ на основе локально-симметричных тестов / А.П. Занкович, В.Н. Ярмолик // Информатика. – 2005. – № 4. – С. 124–134.
261. Занкович, А.П. Неразрушающее тестирование ОЗУ на основе анализа симметрии выходных данных / А.П. Занкович, В.Н. Ярмолик // Автоматика и телемеханика. – 2003. – № 9. – С. 141–154.
262. Иванюк, А.А. Проектирование контролепригодных цифровых устройств / А.А. Иванюк, В.Н. Ярмолик. – Минск: Бестпринт, 2006. – 296 с.

263. Иванюк, А.А. Проектирование встраиваемых цифровых устройств и систем / А.А. Иванюк. – Минск: Бестпринт, 2012. – 338 с.
264. Иванюк, А.А. IDDQ тестирование итерационных логических структур / А.А. Иванюк, А.И. Янушкевич, В.Н. Ярмолик // Автоматика и телемеханика. – 1999. – № 1. – С. 148–158.
265. Иванюк, А.А. Новый подход к проектированию встроенной аппаратуры самотестирования ОЗУ / А.А. Иванюк, В.Н. Ярмолик // Автоматика и вычислительная техника. – 2008. – № 4. – С. 5–13.
266. Идзуми, Т. Инициализация *M*-последовательности высокого порядка, как источник генерирования двоичных случайных чисел / Т. Идзуми, Х. Касигави // Кэйсоку дзидо сэйче ганкай ронбусю. – 1982. – Вып. 18. – № 9. – С. 929–935.
267. Казьмина, С.К. Компактное тестирование / С.К. Казьмина // Автоматика и телемеханика. – 1982. – № 2. – С. 173–189.
268. Калоша, Е.П. Исследование универсального модуля для организации самотестирования СБИС / Е.П. Калоша, И.В. Качан, В.Н. Ярмолик // Автоматика и телемеханика. – 1991. – № 1. – С. 105–112.
269. Калоша, Е.П. О достоверности методов компактного тестирования / Е.П. Калоша, Е.И. Кацнельсон, В.Н. Ярмолик // Автоматика и телемеханика. – 1989. – № 9. – С. 160–166.
270. Кирьянов, Б.Ф. Формирование случайных последовательностей при физическом моделировании дискретных каналов связи / Б.Ф. Кирьянов, В.И. Глова, А.Г. Леонтьев, И.С. Ризаев // Сб., Кодирование и передача дискретных сообщений в системах связи. – М.: Наука. – 1976. – С. 141–151.
271. Коробейник, А.Н. Краткие основы тестирования программного обеспечения / А.Н. Коробейник. – Киев: Директ-лайн, 2012. – 126 с.
272. Кудрявцев, В.Б. Введение в теорию конечных автоматов / В.Б. Кудрявцев, С.В. Алешин, А.С. Подколзин. – М.: Наука, 1985. – 329 с.
273. Кузьмин, Е.В. Структурированные системы переходов / Е.В. Кузьмин, В.А. Соколов. – М.: Физматлит, 2006. – 174 с.
274. Куликов, С. Тестирование программного обеспечения: Базовый курс / С. Куликов. – Минск: EPAM Systems, 2017. – 296 с.
275. Кулямин, В.В. Комбинаторика слов и построение тестовых последовательностей / В.В. Кулямин // Труды ИСП РАН. – 2004. – № 8(1). – С. 25–40.
276. Лапиньш, Я.К. Об одном методе синтеза преобразователей вероятностных распределений / Я.К. Лапиньш, Н.А. Метра // Автоматика и телемеханика. – 1973. – № 4. – С. 32–35.
277. Латыпов, Р.Х. О достоверности кольцевого тестирования линейных последовательностных машин / Р.Х. Латыпов // Автоматика и телемеханика. – 1984. – № 4. – С. 89–91.
278. Латыпов, Р.Х. Сравнение сигнатурного анализа с тривиальным способом сжатия при обнаружении отказов линейной комбинационной схемы /

Р.Х. Латыпов // Автоматика и телемеханика. – 1985. – № 2. – С. 165–167.
279. Литиков, И.П. Кольцевое тестирование комбинационных устройств / И.П. Литиков // Автоматика и телемеханика. – 1983. – № 7. – С. 145–153.
280. Макуильямс, Ф.Д. Псевдослучайные последовательности и таблицы / Ф.Д. Макуильямс, У.А. Слоан // ТИИЭР. – 1975. – Т. 64. – № 12. – С. 80 – 95.
281. Мурашко, И.А. Новый подход к проектированию цепи сканирования для встроенного самотестирования БИС / И.А. Мурашко, А.М. Шмидман, В.Н. Ярмолик // Автоматика и телемеханика. – 1998. – № 7. – С. 157–167.
282. Мурашко, И.А. Методика снижения энергопотребления генератора псевдослучайных тестовых наборов для встроенного самотестирования / И.А. Мурашко, В.Н. Ярмолик // Автоматика и телемеханика. – 2004. – № 8. – С. 102–114.
283. Мурашко, И.А. Методы минимизации энергопотребления при самотестировании цифровых устройств / И.А. Мурашко, В.Н. Ярмолик. – Минск: Бестпринт, 2004. – 188 с.
284. Мурашко, И.А. Встроенное самотестирование Методы минимизации энергопотребления / И.А. Мурашко, В.Н. Ярмолик. – Saarbrücken, Germany: LAP Lambert Academic Publishing, 2012, 348 с.
285. Новик, Г.Х. О достоверности сигнатурного анализа / Г.Х. Новик // Автоматика и телемеханика. – 1982. – № 5. – С. 110–118.
286. Пархоменко, П.П. Основы технической диагностики (Применение вычислительных машин в исследованиях и управлении производством) / П.П. Пархоменко, В.В. Карибский, Е.С. Согомонян, В.Ф. Халчев. – М.: Энергия, 1976. – 464 с.
287. Питерсон, У. Коды исправляющие ошибки: Перевод с английского / У. Питерсон. – М.: Мир, 1964. – 338 с.
288. Сагалович, Ю.Л. Синтез сигнатурного анализатора для двухуровневой комбинационной схемы / Ю.Л. Сагалович, В.Н. Ярмолик // Автоматика и телемеханика. – 1990. – № 4. – С. 159–167.
289. Сапожников, В.В. Основы технической диагностики / В.В. Сапожников, Вл.В. Сапожников. – М.: Маршрут, 2004. – 318 с.
290. Скобцов, Ю.А. Логическое моделирование и тестирование цифровых устройств / Ю.А. Скобцов, В.Ю. Скобцов. – Донецк: ИПММ НАНУ, ДонНТУ, 2005. – 436 с.
291. Скобцов, Ю.А. Построение тестов для перекрестных неисправностей типа задержка / Ю.А. Скобцов, В.Ю. Скобцов, К.М. Нассер, Ияд // Наукові праці Донецького національного технічного університету. Сер.: Інформатика, кібернетика та обчислювальна техніка. – 2011. – Вип. 14. – С. 146–150.

292. Соболь, И.М. Вычисление несобственных интегралов при помощи равномерно распределенных последовательностей / И.М. Соболь // Доклады АН СССР. – 1973. – Т. 210, № 2. – С. 278–281.
293. Соболь, И.М. Метод Монте-Карло / И.М. Соболь. – М.: Наука, 1968. – 64 с.
294. Соболь, И.М. Точки, равномерно заполняющие многомерный куб / И.М. Соболь. – М.: Знание, 1985. – 32 с.
295. Сорвате, Д.В. Взаимно корреляционные свойства псевдослучайных и родственных последовательностей / Д.В. Сорвате, М.Б. Персли // ТИИЭР. – 1980. – Т. 68. – № 5. – С. 59–90.
296. Тамомото, Х. Метод определения оптимальных и квазиоптимальных вероятностей при реализации вероятностного тестирования / Х. Тамомото, Ю. Нарита // Дэнси цусин гаккай ромбуси. – 1982. – № 8. – С. 1057–1064.
297. Федоров, Р.Ф. Стохастические преобразования информации / Р.Ф. Федоров, В.В. Яковлев, Г.В. Добрис. – Ленинград: Машиностроение, 1978. – 304 с.
298. Хамитов, Г.П. Имитация случайных процессов. – Иркутск: Изд-во Иркут. ун-та, 1983. – 184 с.
299. Хопкрофт, Д.Э. Введение в теорию автоматов языков и вычислений. 2-е издание / Д.Э. Хопкрофт, Р. Мотвани, Д.Д. Ульман. – М.: Вильямс, 2016. – 528 с.
300. Яковлев, В.В. Стохастические вычислительные машины / В.В. Яковлев, Р.Ф. Федоров. – Л.: Машиностроение, 1974. – 343 с.
301. Ярмолик, В.Н. Генерирование и применение псевдослучайных сигналов в системах испытаний и контроля / В.Н. Ярмолик, С.Н. Демиденко. – Минск: Наука и техника, 1986. – 200 с.
302. Ярмолик, В.Н. Контроль и диагностика цифровых устройств ЭВМ / В.Н. Ярмолик. – Минск: Наука и техника, 1988. – 240 с.
303. Ярмолик, В.Н. Неразрушающее тестирование запоминающих устройств / В.Н. Ярмолик, И.А. Мурашко, А. Куммерт, А.А. Иванюк. – Минск : Бестпринт, 2005. – 230 с.
304. Ярмолик, В.Н. Проектирование самотестируемых СБИС: научная монография в двух томах Том 1 / В.Н. Ярмолик, Е.П. Калоша, Ю.В. Быков, Ю.В. Климец, А.А. Иванюк. – Минск : БГУИР, 2001. – 159 с.
305. Ярмолик, В.Н. Проектирование самотестируемых СБИС: научная монография в двух томах Том 2 / В.Н. Ярмолик, Е.П. Калоша, Ю.В. Быков, Ю.В. Климец, А.А. Иванюк. – Минск: БГУИР, 2001. – 163 с.
306. Ярмолик, В.Н. Построение генераторов псевдослучайных последовательностей испытательных сигналов / В.Н. Ярмолик // Автоматика и телемеханика. – 1983. – № 6. – С. 155–167.
307. Ярмолик, В.Н. Построение многоканальных сигнатурных анализаторов / В.Н. Ярмолик // Автоматика и телемеханика. – 1985. – № 1. – С. 127–132.

308. Ярмолик, В.Н. Синтез генераторов псевдослучайных тестовых последовательностей / В.Н. Ярмолик // Автоматика и телемеханика. – 1988. – № 9. – С. 119–125.
309. Ярмолик, В.Н. Анализ сигнатурной тестируемости цифровых схем / В.Н. Ярмолик // Автоматика и телемеханика. – 1989. – № 10. – С. 159–167.
310. Ярмолик, В. Н. Вычисление сигнатур регулярных периодических последовательностей / В.Н. Ярмолик // Автоматика и телемеханика. – 1992. – № 1. – С. 146–155.
311. Ярмолик, В.Н. Новый эффективный алгоритм тестирования ОЗУ / В.Н. Ярмолик, В.Г. Микитюк // Автоматика и вычислительная техника. – 1996. – № 1. – С. 61–72.
312. Ярмолик, В.Н. Диагностические тесты ОЗУ / В.Н. Ярмолик, Ю.В. Климец, А.Д. Гор // Автоматика и вычислительная техника. – 1997. – № 2. – С. 15–22.
313. Ярмолик, В.Н. Развитие методов неразрушающего тестирования ОЗУ / В.Н. Ярмолик, Ю.В. Климец, А.Д. Гор // Автоматика и вычислительная техника. – 1997. – № 5. – С. 14–21.
314. Ярмолик, В.Н. Адресные последовательности / В.Н. Ярмолик, С.В. Ярмолик // Автоматика и Вычислительная техника. – 2014. – № 4. – С. 26–34.
315. Ярмолик, В.Н. Метод синтеза управляемых вероятностных тестов / В.Н. Ярмолик, I. Mrozek, С.В. Ярмолик // Автоматика и Вычислительная техника. – 2015. – № 6. – С. 101–113.
316. Ярмолик, В.Н. Адресные последовательности для многократного тестирования ОЗУ / В.Н. Ярмолик, С.В. Ярмолик // Информатика. – 2014. – № 3(39). – С. 92–103.
317. Ярмолик, В.Н. Многократные управляемые вероятностные тесты / В.Н. Ярмолик, В.А. Леванцевич, И. Мрозек // Информатика. – 2015. – № 2(46). – С. 63–76.
318. Ярмолик, В.Н. Псевдо-исчерпывающее тестирование ОЗУ / В.Н. Ярмолик, И. Мрозек, В.А. Леванцевич // Информатика. – 2017. – № 2(54). – С. 58–69.
319. Ярмолик, В.Н. Псевдоисчерпывающее тестирование запоминающих устройств на базе многократных маршевых тестов / В.Н. Ярмолик, В.А. Леванцевич, И. Мрозек // Информатика. – 2018. – №1(15). – С. 5–16.
320. Ярмолик, В.Н. Встроенные аппаратные средства для самотестирования больших интегральных схем / В.Н. Ярмолик // Микроэлектроника. – 1986. – Т. 15. – № 1. – С. 70–76.
321. Ярмолик, В.Н. Исчерпывающее самотестирование СБИС с LSSD структурой / В.Н. Ярмолик, Е.П. Калоша // Микроэлектроника. – 1988. – Т. 17. – № 2. – С. 99–104.

322. Ярмолик, В.Н. Исследование и развитие методов построения ремонтируемого ОЗУ / В.Н. Ярмолик, Ю.В. Климец, А.Д. Гор // Микроэлектроника. – 1997. – Том 26, № 2. – С. 266–271.
323. Ярмолик, В.Н. О генераторе тестовых последовательностей для вероятностных систем диагностирования цифровых устройств / В.Н. Ярмолик // Электронное моделирование. – 1983. – № 5. – С. 49–54.
324. Ярмолик, В.Н. О достоверности контроля двоичных последовательностей методом сигнатурного анализа / В.Н. Ярмолик // Электронное моделирование. – 1985. – Т. 7. – № 6. – С. 51–54.
325. Ярмолик, В.Н. Эффективность обнаружения кодочувствительных неисправностей запоминающих устройств при неразрушающем тестировании / В.Н. Ярмолик, А.П. Занкович // Вести Института современных знаний. – 2004. – № 4 (21). – С. 69–76.
326. Ярмолик, В.Н. Об одном подходе к синтезу параллельных ГПСЧ на регистре сдвига / В.Н. Ярмолик, А.Н. Морозевич // Сб., Радиотехника и электроника. Минск: Вышэйшая школа. – 1977. – Вып. 7. – С. 66–69.
327. Ярмолик, В.Н. Исследование свойств линейной конгруэнтной последовательности / В.Н. Ярмолик, А.Е. Леусенко, С.Н. Демиденко // Автоматизация проектирования технологических процессов. – 1981. – Вып. 5. – С. 118–123.
328. Ярмолик, В.Н., Иванюк А.А. Встроенное самотестирование памяти с использованием сигнатурного анализа / В.Н. Ярмолик, А.А. Иванюк // Сборник трудов Логическое проектирование. – Минск, АН РБ, ИТК, Беларусь. – 1997. – С. 170–180.
329. Ярмолик, В.Н. Обзор методов неразрушающего тестирования ОЗУ / В.Н. Ярмолик // Доклады БГУИР. – 2005. – № 4 (12). – С. 62–72.
330. Ярмолик, В.Н., Иванюк А.А. Тестовое диагностирование аппаратного и программного обеспечения вычислительных систем // Доклады БГУИР. – 2014. – № 2 (80). – С. 127–142.
331. Ярмолик, С.В. Маршевые тесты для самотестирования ОЗУ / С.В. Ярмолик, А.П. Занкович, А.А. Иванюк. – Минск: Изд. центр БГУ, 2009. – 270 с.
332. Ярмолик, С.В Маршевые тесты для тестирования ОЗУ / С.В. Ярмолик, А.П. Занкович, А.А. Иванюк. – Saarbrücken, Germany: LAP Lambert Academic Publishing, , 2012. – 302 с.
333. Ярмолик, С.В. Многократные неразрушающие маршевые тесты с изменяемыми адресными последовательностями / С.В. Ярмолик, В.Н. Ярмолик // Автоматика и телемеханика. – 2007. – № 2. – С. 21–30.
334. Ярмолик, С.В. Управляемые вероятностные тесты / С.В. Ярмолик, В.Н. Ярмолик // Автоматика и телемеханика. – 2012. – № 10. – С. 142–155.
335. Ярмолик, С.В. Адресные последовательности для многократных маршевых тестов / В.Н. Ярмолик, С.В. Ярмолик // Автоматика и вычислительная техника. – 2006. – № 5. – С. 59–68.

336. Ярмолик, С.В. Генерирование адресных последовательностей для эффективного тестирования ОЗУ / С.В. Ярмолик // Автоматика и вычислительная техника. – 2007. – № 6. – С. 55–62.
337. Ярмолик, С.В. Определение оптимальных начальных состояний для многократного тестирования ОЗУ / С.В. Ярмолик, А.Н. Курбацкий, В.Н. Ярмолик // Автоматика и Вычислительная техника. – 2008. – № 3. − С. 15–23.
338. Ярмолик, С.В. Неразрушающее тестирование ОЗУ, основанное на многократном сравнении сигнатур // Автоматика и Вычислительная техника. – 2009. –№ 4. − С. 5–13.
339. Ярмолик, С.В. Обобщенный адаптивный сигнатурный анализ / С.В. Ярмолик, В.Н. Ярмолик // Автоматика и Вычислительная техника. – 2010. – № 4. − С. 54–61.
340. Ярмолик, С.В. Синтез вероятностных тестов с малым числом наборов / С.В. Ярмолик, В.Н. Ярмолик // Автоматика и Вычислительная техника. – 2011. –№ 3(45). − С. 19–30.
341. Ярмолик, С.В. Генерирование модифицированных последовательностей Соболя для многократных маршевых тестов ОЗУ / В.Н. Ярмолик, С.В. Ярмолик // Автоматика и Вычислительная техника. – 2013. – № 5. – С. 25–33.
342. Ярмолик, С.В. Обнаружение кодочувствительных неисправностей запоминающих устройств с многократным использованием маршевых тестов / С.В. Ярмолик, В.Н. Ярмолик // Информатика. – 2006. – № 1(9). – С. 104–129.
343. Ярмолик, С.В. Формирование адресных последовательностей с максимальным средним Хэммингoвым расстоянием для многократного тестирования ОЗУ / С.В. Ярмолик, В.Н. Ярмолик. // Информатика. – 2006. – № 4(12). – С. 88–96.
344. Ярмолик, С.В. Анализ количественных характеристик различия при тестировании ОЗУ / С.В. Ярмолик, А.Н. Курбацкий, В.Н. Ярмолик // Информатика. – 2008. – № 3(19). – С. 90–98.
345. Ярмолик, С.В. Тестирование ОЗУ на основе Адаптивного Сжатия Выходных Данных» // Информатика. – 2009. − № 3(23). − С. 27−35.
346. Ярмолик, С.В. Итеративные почти псевдоисчерпывающие вероятностные тесты / С.В. Ярмолик, В.Н. Ярмолик // Информатика. – 2010. − № 2(26). − С. 66−75.
347. Ярмолик, С.В. Управляемое случайное тестирование / С.В. Ярмолик, В.Н. Ярмолик // Информатика. – 2011. – № 1(29). – С. 79–88.
348. Ярмолик, С.В. Квазислучайное тестирование вычислительных систем / С.В. Ярмолик, В.Н. Ярмолик // Информатика. – 2013. – № 3(39), – С. 92–103.
349. Ярмолик, С.В. Эффективность многократного применения маршевых тестов для выявления кодочувствительных неисправностей / С.В. Ярмо-

лик, В.Н. Ярмолик // Известия Белорусской инженерной академии. – 2005. – № 1(91)/1. – С. 79–83.
350. Ярмолик, С.В. Анализ характеристик подобия и различия применяемых при тестировании ОЗУ / С.В. Ярмолик, А.Н. Курбацкий // Сборник докладов и тезисов 4 международной научно-практической конференции Современные информационные компьютерные технологии. – Гродно, Беларусь, 2008. – С. 15–18.
351. Ярмолик, С.В. Обнаружение кодочувствительных неисправностей ЗУ с использованием маршевых тестов / С.В. Ярмолик // Сборник докладов и тезисов докладов III Международного молодежного форума Информационные технологии в XXI веке. – Днепропетровск, 2005. – С. 217–218.
352. Ярмолик, С.В. Генерация адресных последовательностей для тестирования памяти двух – и более кратными тестами / С.В. Ярмолик // Сборник докладов и тезисов докладов V Международной молодежной научно практической конференции Информационные технологии и кибернетика. – Днепропетровск, 2007. – С. 59.
353. Ярмолик, С.В. Оптимальные начальные состояния для многократного тестирования ОЗУ / С.В. Ярмолик // Сборник докладов и тезисов докладов международной конференции Информационные технологии в управлении сложными системами. – Днепропетровск, 2008. – С. 20–21.

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

АСА – адаптивный сигнатурный анализ
АСВД – адаптивное сжатие выходных данных
ГТП – генератор тестовой последовательности
ЗУ – запоминающее устройство
ЗЭ – запоминающий элемент
КМОП – комплементарная структура металл-оксид-полупроводник
КС – комбинационная схема
МОП – металл-оксид-полупроводник
ОЗУ – оперативное запоминающее устройство
СА – сигнатурный анализатор
ТП – тестовая маска
ЭНФ – эквивалентные нормальные формы
AD − arithmetic distance (Manhattan distance)
AF – address faults
AHD − average Hamming distance
AT − anti-random test
BIST – built-in self-test
CA – covering arrays
CD − Cartesian (Euclidean) distance
CD − Chebyshev distance
CF – coupling fault
CFid – idempotent coupling fault
CFin – inverse coupling fault
CFst – state coupling fault
CMOS – complementary metal-oxide-semiconductor
CRT − controlled random test
CS − counter sequence
D&C − divide and check
D&T − divide and test
DRF – data retention fault
ET − exhaustive test
FAR – fast anti-random
FC − fault coverage
FPGA – field-programmable gate array
FSM – finite state machine
FWTF – false write through fault
GF − Galois field
HD − Hamming distance
JS − Johnson sequence
JTAG − joint test action group
LFSR − linear feedback shift register
LTS – labeled transition systems

MATS – modified algorithmic test sequence
MC − Monte Carlo method
MCD − maximal Cartesian distance
MCDAT − maximal Cartesian distance anti-random test
MCRT − multiple controlled random test
MD − Minkowski distance
MDS − maximum distance separable code
MHD − maximal Hamming distance
MHDAT − maximal Hamming distance anti-random test
NFSR − nonlinear feedback shift register
NPSF – neighborhood pattern sensitive fault
OCRT − optimal controlled random test
PCF – pre-charge fault
PET − pseudo-exhaustive test
PNPSF – passive neighborhood pattern sensitive fault
PS − pseudorandom sequence
PSF – pattern sensitive fault
RAM – random access memory
RDF – read disturb fault
RS − Reed-Solomon code
RT − random test
SA − search area
SA − signature analyzer
SAF – stuck-at fault
SAODC – self-adjusting output data compression
SMDTS − semi-maximum distance test sequence
SOCRT − short optimal controlled random test
SOF – stuck open fault
SS − shift sequence
SS − Sobol sequence
STA − symmetric transparent algorithm
TCD − total Cartesian distance
TF – transition fault
THD − total Hamming distance
TMDTS − total maximum distance test sequence
WCF – weak cell fault

Научное издание

Ярмолик Вячеслав Николаевич

КОНТРОЛЬ И ДИАГНОСТИКА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Ответственный за выпуск С. Л. Бочкарева
Редактор М. И. Волковец

Подписано в печать 12.03.2019. Формат 60*84 1/16.
Бумага офсетная. Ризография.
Усл. печ. л. 27,10. Уч.-изд. л. 22,49.
Тираж 100 экз. Заказ 48.

Издатель и полиграфическое исполнение:
УП «Бестпринт». Свидетельство о государственной регистрации
издателя, изготовителя, распространителя печатных изданий
№ 1/160 от 27.01.2014.
Ул. Филатова, 9, 220026, г. Минск.