УДК 681.326.7

ОБЗОР МЕТОДОВ НЕРАЗРУШАЮЩЕГО ТЕСТИРОВАНИЯ ОЗУ

В.Н. ЯРМОЛИК, А.П. ЗАНКОВИЧ

Белорусский государственный университет информатики и радиоэлектроники
П. Бровки, 6, Минск, 220013, Беларусь

Поступила в редакцию 26 апреля 2005

В статье представлен обзор методов периодического неразрушающего тестирования схем памяти. Основное внимание уделено неразрушающим аналогам классических маршевых алгоритмов. Рассмотрен получивший широкое распространение метод Николаидиса, а также новые методы, основанные на локальной и глобальной симметрии данных. Приведена сравнительная оценка эффективности использования рассмотренных методов.

Ключевые слова: неразрушающее тестирование, маршевые тесты, симметрия данных, условия скрытия и проявления ошибок, маскирование ошибок.

Введение

В настоящее время быстро прогрессирующие технологии производства полупроводниковых приборов позволяют создавать схемы ОЗУ с чрезвычайно высокой степенью интеграции. Нежелательным последствием использования новых технологий является увеличение вероятности возникновения неисправностей. Для обеспечения надежного функционирования цифровых систем большое внимание должно быть уделено своевременному обнаружению неисправностей памяти, поскольку результаты исследований [1, 2] свидетельствуют о том, что отказы ОЗУ могут составлять до 70% от общего числа отказов системы в целом.

Помимо традиционного заводского контроля на этапе производства в настоящее время широко применяется встроенная аппаратура самотестирования ВАСТ [3, 4]. Такой подход позволяет периодически выполнять тестирование на внутренней частоте без применения внешнего оборудования. Большинство ВАСТ реализуют тесты сложности $O(N)$ (N – емкость запоминающего устройства), называемые маршевыми тестами [5]. При выполнении стандартного маршевого тестирования со всеми ячейками памяти в определенном порядке ($\Uparrow$ — от младших адресов к старшим, $\Downarrow$ — наоборот, $\Updownarrow$ — в любом направлении) выполняются операции, задаваемые несколькими маршевыми элементами. Используются операции: $w0$, $w1$ — запись в элемент памяти значений 0 или 1; $r0$, $r1$ — чтение текущего значения из элемента памяти и сравнение его со значением 0 или 1. Достоинствами маршевых тестов являются относительно высокие скорость выполнения и покрывающая способность, а также простота реализации ВАСТ. Из недостатков отметим полную потерю хранящейся в памяти информации после выполнения тестирования, что ограничивает применение классических маршевых тестов только моментами начального тестирования устройства. Для надежной работы постоянно включенных систем, помимо методов с использованием избыточных данных (коды Хэмминга, контроль на четность), применяется методы периодического неразрушающего тестирования [6–9], основанного на классических маршевых алгоритмах. Их обзор представлен в данной статье.

При анализе покрывающей способности маршевых тестов рассматриваются не реальные дефекты в структуре ОЗУ (замыкания, обрывы и т.п.), а математические модели, описывающие функциональное проявление этих дефектов в процессе работы памяти. Приведем наиболее распространенные модели неисправностей [2].

1. Константные неисправности (*Stuck-At Fault — SAF*): логическое значение ячейки памяти всегда равно 0 (SA0) или 1 (SA1) независимо от операций, производимой с этой или другими ячейками памяти.

2. Переходная неисправность (*Transition Fault — TF*): ячейка не способна осуществлять переход из состояния логического 0 в состояние логической 1 (TF↑) либо наоборот (TF↓).

3. Неисправности взаимного влияния (*Coupling Fault — CF*): изменение логического значения одной (влияющей) ячейки отражается на значении второй (зависимой) ячейки. Ячейка с меньшим адресом может влиять на ячейку со старшим адресом (это обозначается символом $\wedge$) и наоборот (символ $\vee$). Различают три типа неисправностей CF.

3.1. Инверсная неисправность CF (*Inversion CF — CFin*). Изменение значения влияющей ячейки вызывает инвертирование значения зависимой. Возможны следующие виды неисправностей CFin: $\wedge(\uparrow,\updownarrow)$, $\wedge(\downarrow,\updownarrow)$,$\vee(\uparrow,\updownarrow)$, $\vee(\downarrow,\updownarrow)$.

3.2. Неинверсная неисправность CF (*Idempotent CF — CFid*). Изменение значения влияющей ячейки переводит зависимую ячейку в определенное состояние. Возможно восемь видов неисправностей CFid: $\wedge(\uparrow,0)$, $\wedge(\uparrow,1)$, $\wedge(\downarrow,0)$, $\wedge(\downarrow,1)$, $\vee(\uparrow,0)$, $\vee(\uparrow,1)$, $\vee(\downarrow,0)$, $\vee(\downarrow,1)$.

3.3. Константная неисправность CF (*State CF – CFst*). Переход зависимой ячейки в какое-либо состояние возможен только при определенном значении влияющей ячейки. Различают следующие неисправности CFst: $\wedge(0,0)$, $\wedge(0,1)$, $\wedge(1,0)$, $\wedge(1,1)$, $\vee(0,0)$, $\vee(0,1)$, $\vee(1,0)$, $\vee(1,1)$.

Неразрушающее тестирование памяти

Основным требованием, предъявляемым к неразрушающим тестам, является необходимость восстановления исходного состояния объекта тестирования после выполнения процедуры тестирования. Поэтому все тестовые воздействия, направленные на активизацию неисправностей и проявление их в виде ошибок на выходах схемы, должны носить обратимый характер. Простейшим способом реализации этого требования для случая неразрушающего тестирования памяти является сохранение всего ее содержимого в резервном буфере с последующим выполнением одного из классических разрушающих алгоритмов. Хотя этот способ позволяет использовать существующую ВАСТ, он потребует дублирования массива запоминающих элементов, а также дополнительного времени для сохранения и последующего восстановления содержимого памяти. Поэтому для большинства приложений он непригоден.

Другой подход был предложен в докладе Б.Конемана на семинаре Design For Testability в 1986 г. [10]. Обозначим через $s(A)$ сигнатуру упорядоченного потока данных A длинной N, а через TP — поток тестовых данных той же длины. Тогда алгоритм работы метода можно сформулировать следующим образом.

Шаг 1: Прочитать содержимое памяти (CONTENTS). Вычислить сигнатуру содержимого памяти s(CONTENTS). Записать новое содержимое памяти NEWCONTENTS = (CONTENTS $\oplus$ TP). Знак "$\oplus$" здесь и далее используется для обозначения операции исключающего ИЛИ.

Шаг 2: Прочитать новое содержимое памяти (NEWCONTENTS). Вычислить сигнатуру s(NEWCONTENTS). Восстановить в памяти прежние значения путем записи (NEWCONTENTS $\oplus$ TP)

Шаг 3: Вычислить результирующую сигнатуру RESULT как s(CONTENTS) $\oplus$ s(NEWCONTENTS)

Для работы этого метода необходимо, чтобы используемые сигнатурные анализаторы обладали свойством линейности, потому что только в этом случае для исправной памяти будет получено детерминированное значение RESULT = s(TP).

Несмотря на то что описанный алгоритм удовлетворяет требованиям неразрушающего тестирования и достаточно просто реализуется в программном или аппаратном виде, он обладает некоторыми существенными недостатками.

1. Некоторые неисправности генерируют на Шаге 1 и Шаге 2 одинаковые векторы ошибок. Такие неисправности не будут обнаружены за счет наложения ошибок друг на друга.

2. Предлагаемые тестовые последовательности состоят из двух элементарных циклов, содержащих по одной операции чтения и записи для каждой ячейки. Им соответствует простейший маршевый алгоритм $\Uparrow(r0, w1); \Uparrow(r1, w0)$. Не существует универсальных способов построения неразрушающих аналогов произвольных маршевых алгоритмов.

3. Предложенный алгоритм не гарантирует обнаружение целого ряда неисправностей памяти.

Тестирование 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, была предложена М.Николаидисом в работе [4]. На вход преобразования поступает классический маршевый алгоритм ALin, промежуточные алгоритмы, получаемые на каждом шаге *i*, обозначаются через AL*i*.

Шаг 0. В начало каждого маршевого элемента, начинающегося операцией записи, добавляется операция чтения. Это требуется для предотвращения затирания маршевым элементом текущего содержимого памяти.

Шаг 1. Если первый маршевый элемент алгоритма ALin используется только для записи в память инициализирующих значений (а это справедливо для большинства классических маршевых алгоритмов), и этот элемент не обнаруживает никаких дополнительных неисправностей, то его можно удалить из AL0.

Шаг 2. На этом шаге все операции из алгоритма AL1 заменяются неразрушающими аналогами. Ими являются операции r_a и $r_{\bar{a}}$, читающие значение текущей ячейки памяти и помещающие его в прямом (a) или обратном ($\bar{a}$) виде во временный буфер, а также w_a и $w_{\bar{a}}$, записывающие в текущую ячейку памяти прямое или проинвертированное значение из временного буфера. Пусть x — ожидаемое значение первой операции чтения алгоритма AL1. Тогда все последующие операции алгоритма, читающие значение y_j (j = 2, …, m, где m – общее количество операций в алгоритме AL1), преобразуются в соответствующие неразрушающие операции со значением, если $x = y_j$, или со значением $\bar{a}$, если $x = \bar{y}_j$. Самая первая операция чтения алгоритма AL1 преобразуется в операцию r_a.

Шаг 3. Если последняя операция записи помещает в память значения, обратные значениям, используемым на стадии инициализации алгоритма ALin, то в конец алгоритма AL2 добавляется маршевый элемент $\Updownarrow(r_a, w_{\bar{a}})$, инвертирующий содержимое памяти. Этот шаг необходим для восстановления начального состояния памяти в тех случаях, когда после выполнения AL2 оно оказалось обратным начальному. Однако дополнительный маршевый элемент может существенно увеличить время тестирования. В работе [11] предложен один из способов элиминации данного шага, основанный на изменении архитектуры схемы памяти (рис. 1). Переключение Т-триггера производится после каждого выполнения процедуры тестирования. Его выход управляет инверсией входных и выходных данных схемы памяти с помощью двух вентилей исключающего ИЛИ.

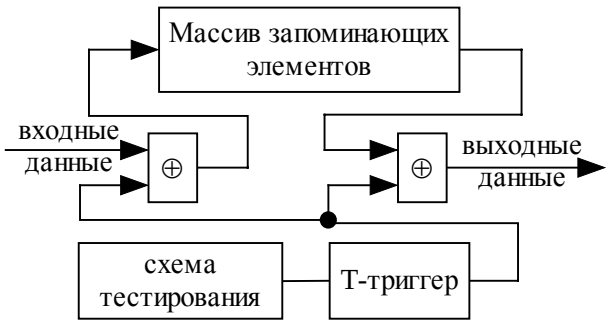

Рис. 1. Модифицированная архитектура памяти для инвертирующих тестов

Полученный на последнем шаге алгоритм AL3 представляет собой неразрушающую версию исходного маршевого алгоритма ALin. Он называется *базовым неразрушающим тестом*. Активизирующая неисправности модификация памяти в процессе его выполнения осуществляется путем инвертирования всего содержимого памяти. Для обнаружения ошибок результаты всех операций чтения сжимаются с помощью сигнатурного анализатора в *рабочую сигнатуру*. Для получения *эталонной сигнатуры*, соответствующей содержимому памяти без ошибок, перед выполнением базового алгоритма запускается *начальный алгоритм*, получаемый на шаге 4.

Шаг 4. Из алгоритма AL3 удаляются все операции записи, а операции $r_{\bar{a}}$ заменяются на r_a.

Рассмотрим применение этих правил для классического маршевого теста March C- [2].

ALin: $\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$

AL0: $\Updownarrow(r0,w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$

AL1: $\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$

AL2: $\Uparrow(r_a,w_{\bar{a}});\Uparrow(r_{\bar{a}},w_a);\Downarrow(r_a,w_{\bar{a}});\Downarrow(r_{\bar{a}},w_a);\Updownarrow(r_a)$

AL3: совпадает с AL2, обозначается tMarchC-

AL4: $\Uparrow(r_a);\Uparrow(r_a);\Downarrow(r_a);\Downarrow(r_a);\Updownarrow(r_a)$

Опишем полную процедуру тестирования памяти алгоритмом с использованием базового и начального неразрушающих алгоритмов.

1. Вычисляется эталонная сигнатура C_{REF} путем выполнения начального алгоритма AL4.

2. Вычисляется рабочая сигнатура C_{TEST} путем выполнения базового алгоритма AL3.

3. Сигнатуры C_{REF} и C_{TEST} сравниваются между собой. Их неравенство или равенство определяют соответственно наличие или отсутствие неисправностей в ОЗУ.

Описанной технологии неразрушающего тестирования также присущ ряд недостатков.

1. Существенно увеличивается сложность теста за счет добавления начального алгоритма, используемого для вычисления эталонной сигнатуры.

2. Не гарантируется 100-ная покрывающая способность даже для однократных неисправностей из-за эффекта маскирования [12].

Симметричное тестирование памяти

Для устранения основного недостатка неразрушающего тестирования по методу Николаидиса, связанного с существенным (до 30%) увеличением сложности алгоритма, был предложен новый метод [7]. Его основная идея заключается в разбиении базового неразрушающего теста на две части, считывающие симметричные последовательности данных. Неисправности схемы памяти проявляют себя в виде нарушений симметрии этих последовательностей, что легко обнаружить, вычислив и сравнив в конце алгоритма их сигнатуры.

Пусть $D=(D_0,\ldots,D_{N-1})$ — некоторая последовательность данных. Обратной назовем последовательность $D^*=(D_{N-1},\ldots,D_0)$, инверсной — последовательность $\overline{D}=(\overline{D}_0,\ldots,\overline{D}_{N-1})$, а обратной инверсной — последовательность $\overline{D}^*=(\overline{D}_{N-1},\ldots,\overline{D}_0)$. Все возможные комбинации симметричных последовательностей данных делятся на четыре типа, представленные на рис. 2.

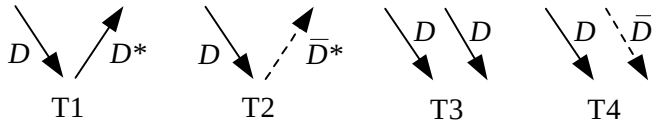

Рис. 2. Типы симметрии последовательностей данных

Для применения симметричных последовательностей данных при тестировании схем памяти требуется описать способ их сигнатурного анализа. Введем некоторые условные обозначения. Пусть $S = (s_0, ..., s_{k-1})$ представляет собой некоторое состояние сигнатурного анализатора размерности k, его обратным состоянием назовем $S^* = (s_{k-1}, ..., s_0)$. Если $H(x) = h_k x^k + h_{k-1}x^{k-1} + ... + h_1 x + h_0$ представляет собой некоторый полином степени k, то $H^*(x) = x^k H(x^{-1}) = h_k + h_{k-1}x + ... + h_1 x^{k-1} + h_0 x^k$ будет представлять полином степени k, обратный исходному полиному H(x). Через sig(D, S, H) обозначим сигнатуру, получаемую при сжатии последовательности данных D на сигнатурном анализаторе, описываемом полиномом $H(x)$, с начальным состоянием S.

Т е о р е м а 1. *Пусть при сжатии последовательности данных D на сигнатурном анализаторе, описываемом полиномом H*(*x*), *с начальным состоянием* S_0, *была получена сигнатура* S_1 = sig(D, S_0, H). *Тогда сигнатура* S_2, получаемая при сжатии последовательности данных D^* *на сигнатурном анализаторе, описываемом полиномом* $H^*(x)$, *с начальным состоянием* S_1^*, *будет определяться следующим соотношением:*

$$\text{sig}(D^*, \text{sig}(D, S_0, H)^*, H^*) = S_0^* \quad (1)$$

Д о к а з а т е л ь с т в о теоремы 1 приведено в [7]. Аналогичным образом было получено, что:

$$\text{sig}(\overline{D}^*, \text{sig}(D, S_0, H)^*, H^*) = S_0^* + \text{sig}((1, ..., 1), (0, ..., 0), H^*), \quad (2)$$

$$\text{sig}(\overline{D}, S_0, H) = \text{sig}(D, S_0, H) + \text{sig}((1, ..., 1), (0, ..., 0), H), \quad (3)$$

$$\text{sig}(D, S_0, H) \equiv \text{sig}(D, S_0, H). \quad (4)$$

Пользуясь соотношениями (1)–(4), можно построить схемы сигнатурного анализа последовательностей данных, обладающих любым из рассмотренных на рис. 2 типов симметрии. Пример сжатия двух последовательностей с симметрией типа T2 приведен на рис. 3. При отсутствии в считываемых последовательностях данных нарушений симметрии сигнатуры S_2 и S_C должны быть равны. S_C используется для обозначения константной сигнатуры из (2): S_C = = S_0^* + sig((1, ..., 1), (0, ..., 0), H^*).

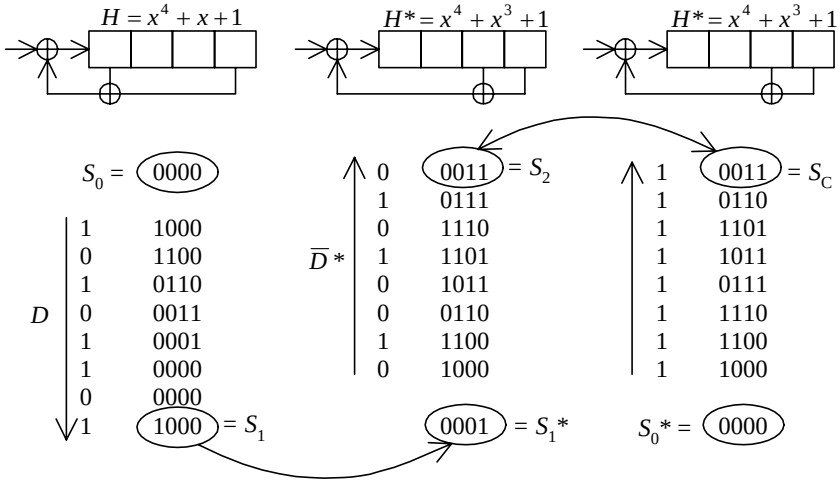

Рис. 3. Сигнатурный анализ взаимообратных инверсных последовательностей данных

Главным условием применения симметричного сигнатурного анализа является наличие симметричных последовательностей данных. Для их формирования в случае неразрушающего маршевого тестирования в [7] предлагается следующий подход. В качестве основы берется базовый неразрушающий алгоритм. Он разделяется на две половины, операции чтения которых должны генерировать взаимосимметричные последовательности данных. Этого можно добиться, вводя в маршевый алгоритм дополнительные операции чтения. Подобная возможность основывается на утверждении о том, что 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 справедливо, поскольку ни одна из рассматриваемых моделей неисправностей не активизируется при выполнении операции чтения.

Рассмотрим пример преобразования маршевого теста March C- в симметричный. Для этого создадим схему считываемых базовым неразрушающим алгоритмом tMarch C- данных (рис. 4,*a*).

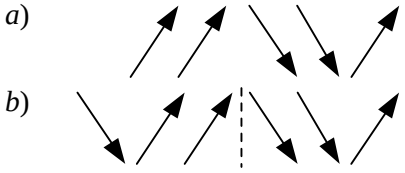

Рис. 4. Схема чтения данных тестом March C-: *a*) до добавления операции чтения; *b*) после добавления

Здесь каждая стрелка соответствует отдельной операции чтения из каждой фазы маршевого теста. Направление стрелки указывает направление движения фазы теста. Теперь задача преобразования сводится к добавлению дополнительных операций чтения к тесту таким образом, чтобы представленная выше схема стала бы обратно симметричной. В нашем случае для этого достаточно добавить в начало алгоритма маршевый элемент $\Downarrow(r_a)$ (рис. 4,*b*). Вертикальной линией помечена граница симметричных последовательностей. Окончательный вариант симметричного алгоритма будет выглядеть следующим образом (добавленная операция подчеркнута):

UDMarch C- $\underline{\Downarrow(r_a);}\Uparrow(r_a, w_{\bar{a}});\Uparrow(r_{\bar{a}}, w_a);\,|\,\Downarrow(r_a, w_{\bar{a}});\Downarrow(r_{\bar{a}}, w_a);\Uparrow(r_a).$

Использование симметрии позволило существенно уменьшить сложность неразрушающего алгоритма. Она снизилась с 14*N* для алгоритма Николаидиса tMarch C- до 10*N* для UDMarch C-.

Локально-симметричные алгоритмы тестирования памяти

Главным недостатком описанных выше алгоритмов является то, что они рассматривают покрывающую способность построенных алгоритмов только после их получения, т.е., при создании алгоритма в первую очередь рассматриваются вопросы придания ему определенной структуры (добавление начального алгоритма вычисления эталонной сигнатуры, приведение алгоритма к симметричному виду), что приводит, как правило, к его необоснованному усложнению. Авторами предложена методика анализа поведения неисправностей в процессе неразрушающего маршевого тестирования [13], которая позволяет построить алгоритмы с теоретически минимальной сложностью. Она основана на выделении условий проявления и скрытия неисправностей.

Модификация значений памяти в процессе выполнения базового неразрушающего алгоритма выполняется путем инвертирования содержимого. Инвертирование исправной ячейки означает запись в нее противоположного значения. Неисправность может проявить себя сохранением прежнего состояния или изменением другой ячейки. Наведенное в результате ошибочное значение считывается и модифицируется последующими операциями до тех пор, пока повторная активация неисправности не установит в зависимой ячейке такое значение, какое бы она имела в отсутствие неисправности, т.е. произойдет скрытие ошибки.

Введем условные обозначения, описывающие процессы инвертирования и чтения. Символом I обозначим операцию инвертирования значения ячейки. Эту операцию составляют чтение текущего значения ячейки и последовательность записей противоположных значений в ту же ячейку. Количество инвертирований ячейки k задается индексом символа I (при k=1

его можно опустить). Префиксы $\Uparrow$, $\Downarrow$ или $\Updownarrow$ используются для обозначения направления выполнения процесса инвертирования ($\Uparrow$ — от младших адресов к старшим, $\Downarrow$ — наоборот, $\Updownarrow$ — не имеет значения). Наличие префикса i говорит о том, что содержимое памяти перед началом выполнения текущего маршевого элемента должно быть проинвертировано по отношению к содержимому перед началом выполнения теста. Символом R обозначим следующую за инвертированием операцию чтения. Префикс $\wedge$, $\vee$ или = задает порядок взаимного расположения проинвертированной и читаемой ячеек ($\wedge$ — адрес первой ячейки больше адреса второй, $\vee$ — меньше, = — равен).

Условием обнаруживающего проявления константной неисправности SAF неразрушающим тестом является инверсия неисправной ячейки – следующая операция чтения вернет ошибочное значение. Прочитанное после повторного инвертирования значение совпадет со значением, которое было бы прочитано из исправной памяти. В общем случае, любое четное количество инвертирований наводит ошибку в неисправной ячейке, нечетное — скрывает ошибку. Запишем это утверждение с помощью введенных условных обозначений.

Условия обнаруживающего проявления неисправности SAF (A_{SAF}):

$\Updownarrow I_k=R, \Uparrow I_k \vee R, \Downarrow I_k \wedge R$ для k = 2i – 1 (i = 1, 2, ...);

$i\Updownarrow I_k=R, i\Uparrow I_k \vee R, i\Downarrow I_k \wedge R$ для k = 2i (i = 1, 2, ...).

Условия скрывающего проявления неисправности SAF (B_{SAF}):

$i\Updownarrow I_k=R, i\Uparrow I_k \vee R, i\Downarrow I_k \wedge R$ для k = 2i – 1 (i = 1, 2, ...);

$\Updownarrow I_k=R, \Uparrow I_k \vee R, \Downarrow I_k \wedge R$ для k = 2i (i = 1, 2, ...).

Рассуждая аналогичным образом, были построены условия проявляющего и скрывающего проявления для всех рассматриваемых классов неисправностей [9]. Следующим этапом исследования стало выделение условий проявления неисправностей в произвольном базовом неразрушающем алгоритме. Для этого были сформулированы следующие правила.

1. Последовательность вида "$d(r_a, ..., o)$" удовлетворяет условию $dI_k bR$, где o обозначает операцию записи, $d \in \{\Uparrow, \Downarrow, \Updownarrow\}$, $b \in \{\wedge, \vee\}$. Направление стрелки, обозначаемой b, совпадает с направлением стрелки, обозначаемой d.

2. Последовательность вида "$d(o_1, ..., o_l, r_a, ...)$" , где o_i (i $\in$ 1, ..., l) обозначает операцию чтения или записи, удовлетворяет условию $dI_{inv}=R$, где inv — количество инверсий, выполняемых маршевым элементом до рассматриваемой операции чтения.

3. Если текущий маршевый элемент не последний в алгоритме, то он удовлетворяет условию $dI_k bR$, где k – количество выполняемых маршевым элементом инверсий. Направление стрелки d совпадает, а направление стрелки b противоположно направлению выполнения маршевого элемента.

4. Если предыдущие маршевые элементы инвертировали содержимое памяти нечетное число раз, то все условия из текущего элемента получают префикс i.

После выделения всех условий проявления неисправностей необходимо выделить пары обнаруживающих и скрывающих ошибку условий для каждой из рассматриваемых неисправностей. Это позволяет определить, какие операции оперируют с ошибочными данными в процессе тестирования. На рис. 5 эти операции подчеркнуты темными линиями. Начало каждой линии расположено под операцией теста, соответствующей символу R из условия, обнаруживающего проявления неисправности. Заканчивается каждая линия перед операцией, соответствующей символу R из ближайшего условия, скрывающего проявления рассматриваемой неисправности.

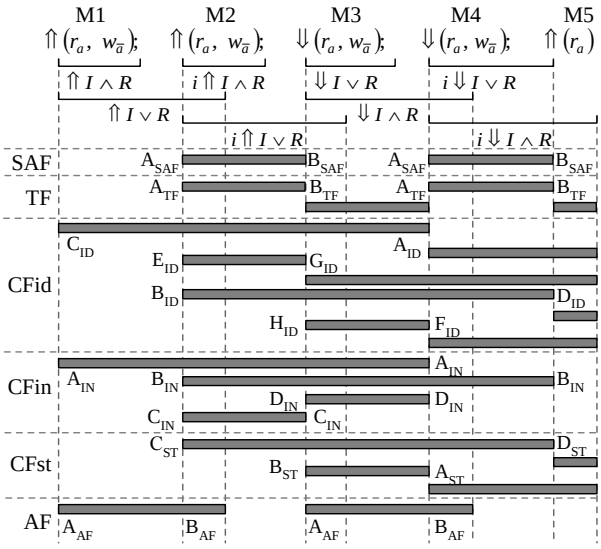

Рис. 5. Карта проявления неисправностей для неразрушающего теста March C-

Рассмотрим для теста March C- в качестве генераторов симметричных последовательностей данных операции чтения из следующих маршевых элементов:

(M1,M2) и (M3,M4) (симметрия типа T2), M4 и M5 (T2);

M1 и M3 (T1), M2 и M5 (T4);

M2 и M3 (T2), M2 и M5 (T4);

M1 и M2 (T4), M3 и M4 (T4), M4 и M5 (T2).

Анализ рис. 5 показывает, что четвертый вариант обнаруживает не все активизированные неисправности — ошибочные данные, вызванные неисправностями с условием обнаружения $\Downarrow I \vee R$, считываются элементами M3, M4 и M5 и не считываются элементами M1 и M2, а значит, симметрия данных в этом случае не нарушается. Поэтому в дальнейшем он рассматриваться не будет. Для записи оставшихся алгоритмов дополним принятые условные обозначения [2, 4] следующим образом.

1. Верхний индекс операций чтения содержит в квадратных скобках список номеров симметричных элементов, включающих данную операцию.

2. Граница первого и второго симметричного потоков отмечается вертикальной чертой. Нижний индекс задает тип симметрии образованного симметричного элемента, верхний индекс – его порядковый номер.

С помощью предложенных обозначений рассмотренные алгоритмы запишутся так:

1) $\Uparrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Uparrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Big|^{[1]}_{\mathrm{T2}}\Downarrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Downarrow\left(r_a^{[1,2]}, w_{\bar{a}}\right);\Big|^{[2]}_{\mathrm{T2}}\Uparrow\left(r_a^{[2]}\right);$

2) $\Uparrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Big|^{[1]}_{\mathrm{T1}}\Uparrow\left(r_a^{[2]}, w_{\bar{a}}\right);\Big|^{[2]}_{\mathrm{T4}}\Downarrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Downarrow\left(r_a, w_{\bar{a}}\right);\Uparrow\left(r_a^{[2]}\right);$

3) $\Uparrow\left(r_a, w_{\bar{a}}\right);\Uparrow\left(r_a^{[1,2]}, w_{\bar{a}}\right);\Big|^{[1]}_{\mathrm{T2}}\Big|^{[2]}_{\mathrm{T4}}\Downarrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Downarrow\left(r_a, w_{\bar{a}}\right);\Uparrow\left(r_a^{[2]}\right).$

Рассмотрим сложность аппаратной реализации этих алгоритмов в виде ВАСТ. Использование симметрии типов T1 и T3 более эффективно при аппаратной реализации. Инвертирование прочитанных данных перед сжатием на сигнатурном анализаторе позволяет изменить считываемые последовательности D на $\overline{D}$ и D^* на $\overline{D}^*$. Кроме того, для маршевых элементов, допускающих оба направления адресации ($\Updownarrow$), считываемые последовательности D и $\overline{D}$ можно заменить на D^* и $\overline{D}^*$ соответственно. В результате оптимизации можно получить алгоритмы, генерирующие последовательности данных, обладающих симметрией только типов T1 и T3:

1) $\Uparrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Uparrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Big|^{[1]}_{\mathrm{T1}}\Downarrow\left(r_{\bar{a}}^{[1]}, w_a\right);\Downarrow\left(r_{\bar{a}}^{[1,2]}, w_a\right);\Big|^{[2]}_{\mathrm{T1}}\Uparrow\left(r_a^{[2]}\right);$

2) $\Uparrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Big|^{[1]}_{\mathrm{T1}}\Uparrow\left(r_a^{[2]}, w_{\bar{a}}\right);\Big|^{[2]}_{\mathrm{T1}}\Downarrow\left(r_a^{[1]}, w_{\bar{a}}\right);\Downarrow\left(r_a, w_{\bar{a}}\right);\Downarrow\left(r_{\bar{a}}^{[2]}\right);$

3) $\Uparrow(r_a, w_{\bar{a}}); \Uparrow(r_a^{[1,2]}, w_{\bar{a}}); \Big|_{\text{T1}}^{[1]} \Downarrow(r_{\bar{a}}^{[1]}, w_a); \Big|_{\text{T1}}^{[2]} \Downarrow(r_a, w_{\bar{a}}); \Downarrow(r_{\bar{a}}^{[2]})$.

Таким образом, для базового неразрушающего теста MarchC- можно построить несколько вариантов локально симметричных алгоритмов без добавления операций чтения. Покрывающая способность построенных алгоритмов равна покрывающей способности соответствующего классического неразрушающего теста.

Оценка эффективности использования рассмотренных методов

На рис. 6 представлены наиболее популярные алгоритмы неразрушающего тестирования, построенные с помощью рассмотренных выше методов.

		Алгоритм
MATS	1	$\{\Uparrow(r_a)\Downarrow(r_a)\}\{\Uparrow(r_a, w_a)\Downarrow(r_a)\}$
	2	$\Uparrow(r_a, w_a)\ \ \Downarrow(r_a)$
	3	$\Downarrow(r_a^{[1]})\ {}_{\text{T1}}^{[1]}\Uparrow(r_a^{[1,2]}, w_a)\ {}_{\text{T1}}^{[2]}\Downarrow(r_a^{[2]})$
MATS++	1	$\{\Uparrow(r_a)\Downarrow(r_a, r_a)\}\{\Uparrow(r_a, w_a)\Downarrow(r_a, w_a, r_a)\}$
	2	$\Uparrow(r_a, r_a, w_a)\ \ \Downarrow(r_a, w_a, r_a)$
	3	$\Downarrow(r_a^{[1]})\ {}_{\text{T3}}^{[1]}\Uparrow(r_a^{[2]}, w_a)\ {}_{\text{T1}}^{[2]}\Downarrow(r_a^{[1]}, w_a, r_a^{[2]})$
March Y	1	$\{\Uparrow(r_a, r_a)\Downarrow(r_a r_a)\Uparrow(r_a)\}\{\Uparrow(r_a, w_a, r_a)\Downarrow(r_a, w_a, r_a)\Uparrow(r_a)\}$
	2	$\Downarrow(r_a)\Uparrow(r_a, r_a, w_a, r_a)\ \ \Downarrow(r_a, r_a, w_a, r_a)\Uparrow(r_a)$
	3	$\Uparrow(r_a^{[1,2]})\ {}_{\text{T1}}^{[1]}\Uparrow(r_a, w_a, r_a)\Downarrow(r_a^{[1]}, w_a, r_a)\ {}_{\text{T1}}^{[2]}\Downarrow(r_a^{[2]})$
March C-	1	$\{\Uparrow(r_a)\Uparrow(r_a)\Downarrow(r_a)\Downarrow(r_a)\Downarrow(r_a)\}\{\Uparrow(r_a, w_a)\Uparrow(r_a, w_a)\Downarrow(r_a, w_a)\Downarrow(r_a, w_a)\Downarrow(r_a)\}$
	2	$\Uparrow(r_a)\Uparrow(r_a, w_a)\Uparrow(r_a, w_a)\ \ \Downarrow(r_a, w_a)\Downarrow(r_a, w_a)\Downarrow(r_a)$
	3	$\Uparrow(r_a^{[1]}, w_a)\ {}_{\text{T1}}^{[1]}\Uparrow(r_a^{[2]}, w_a)\ {}_{\text{T1}}^{[2]}\Downarrow(r_a^{[1]}, w_a)\Downarrow(r_a, w_a)\Downarrow(r_a^{[2]})$

Рис. 6. Алгоритмы неразрушающего тестирования: 1 — метод Николаидиса; 2 — симметричные; 3 — локально симметричные.

Одним из важнейших параметров создаваемых алгоритмов является их сложность, оцениваемая по суммарному количеству операций в алгоритме (рис. 7, *a*).

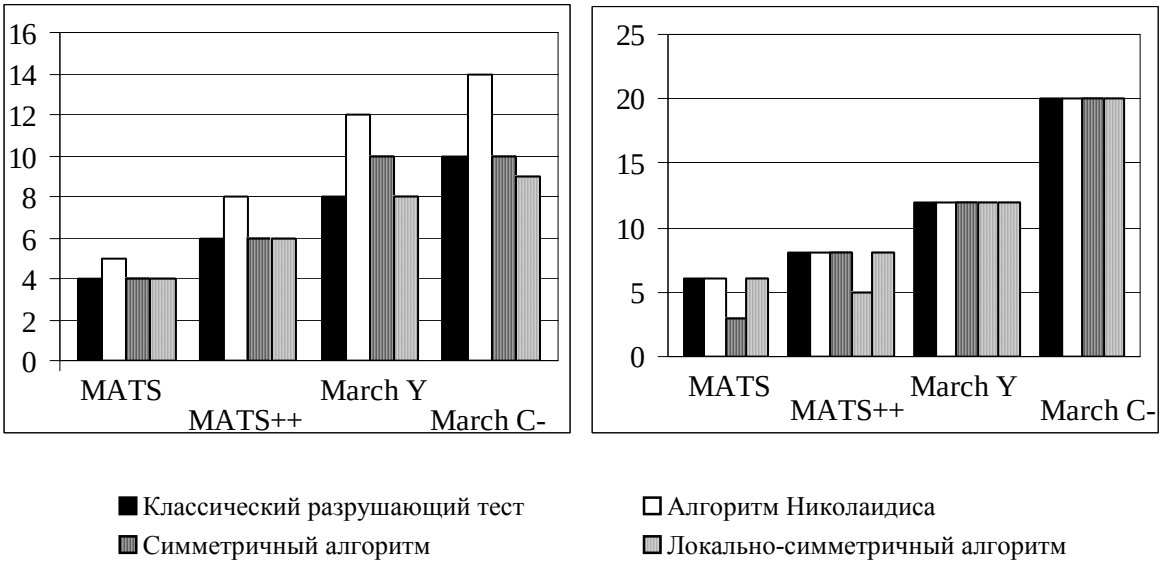

Рис. 7. Эффективность неразрушающих алгоритмов: *a*) сложность алгоритма; *b*) количество обнаруживаемых неисправностей

Еще одним критерием оценки эффективности алгоритмов тестирования является их покрывающая способность. В случае неразрушающего тестирования ее можно рассматривать с двух сторон. С одной стороны, следует рассмотреть способность алгоритма активизировать некоторый набор неисправностей – его генерирующую способность, с другой, созданный алгоритм должен однозначно определять ошибочность прочитанного значения, т.е. обладать обнаруживающей способностью.

Поскольку при тестировании по методу Николаидиса активизация неисправностей производится только в процессе работы базового неразрушающего алгоритма, то только он обладает генерирующей способностью. А так как он был получен из разрушающего маршевого алгоритма путем выполнения преобразований, не влияющих на покрывающую способность, то их покрывающие способности совпадают [6]. Это утверждение распространяется также и на алгоритмы с симметричными данными, поскольку в их основу положены базовые алгоритмы Николаидиса.

Обнаружение сгенерированных ошибок при тестировании по алгоритму Николаидиса производится при сравнении эталонной и рабочей сигнатур. Для рассматриваемых неисправностей ошибки гарантированно исказят только вторую из них, и, следовательно, все могут быть обнаружены. На обнаруживающую способность алгоритмов с глобальной симметрией влияет выбор позиций, в которые добавляются корректирующие симметрию операции чтения. Дополнительная операция в самом начале алгоритма более предпочтительна, поскольку она выполняется перед активизацией большинства неисправностей. Однако в некоторых случаях (например, для алгоритма MATS) приведение к симметричному виду требует большого количества дополнительных операций. Этого можно избежать только за счет снижения обнаруживающей способности (рис. 7,*б*). После добавления операций выполняется проверка, все ли целевые неисправности обнаруживаются созданным алгоритмом. До появления методики анализа поведения неисправностей [13] эта проверка представляла собой трудную задачу, особенно для тестов большой сложности. Указанная методика позволяет выполнить ее автоматически [14]. Способность локально симметричных алгоритмов обнаруживать ошибки проверяется в процессе выбора симметричных стадий, что также может быть автоматизировано.

На обнаруживающую способность неразрушающих алгоритмов влияет также тот факт, что используемые сигнатурные анализаторы, основанные на линейных сдвиговых регистрах с обратной связью, подвержены маскированию ошибок – их скрытию в результате наложения друг на друга. Процесс сжатия потока данных на сигнатурном анализаторе эквивалентен нахождению остатка от деления полинома данных генерирующим полиномом [15]. Маскирование проявляется, если полином ошибок делится на генерирующий без остатка. Для минимизации маскирования во всех трех случаях неразрушающего тестирования применим подход, предложенный в [12]. В его основу положен тот факт, что при выполнении маршевых тестов неисправности проявляют себя в виде детерминированных векторов ошибок, имеющих простое математическое описание.

Сигнатурный анализатор, основанный на сдвиговом линейном регистре с обратной связью, позволяет обнаруживать все одиночные и двойные ошибки в последовательности сжимаемых данных длиной 2^m-1 (m – разрядность анализатора), если для его задания используется примитивный полином. Одиночные неисправности в процессе неразрушающего тестирования по алгоритму Николаидиса tMarch C- генерируют ошибку в считываемом потоке длиной $4N < 2^m$-1 дважды, поэтому для их обнаружения требуется сигнатурный анализатор с задающим полиномом порядка $m = \lceil \log_2(N+1) \rceil + 2$. Поскольку при выполнении симметричного неразрушающего тестирования неисправности проявляют себя аналогично, то и в этом случае требуется сигнатурный анализатор, задаваемый полиномом указанного порядка. Докажем, что этот же полином позволяет обнаруживать также все двойные константные неисправности для теста tMarch C-. Проявление любых двух константных неисправностей (в ячейках с произвольными адресами i и j, $i \neq j$) при выполнении данного алгоритма может быть описано с помощью полинома

$$x^{3N-i} + x^{3N-j} + x^{N+j-1} + x^{N+i-1} = x^{N+i-1}\left(x^{2N-2i+1} + x^{2N-i-j+1} + x^{j-i} + 1\right). \quad (5)$$

Из (5) следует, что полином ошибок может быть разложен на два множителя. Первый из них не делится ни одним полиномом порядка >1, а второй можно представить в виде $x^{N+i-1}\left(x^{2N-i-j+1}+1\right)\left(x^{j-i}+1\right)=\left(x^{P}+1\right)\left(x^{Q}+1\right)$, где $P=2N-i-j+1<2^m-1$ и $Q=j-i<2^m-1$ для $m=\lceil\log_2(N+1)\rceil+2$. Следовательно, все двойные константные неисправности будут обнаружены.

Ошибки большей кратности на практике встречаются реже. Нахождение точного порядка и вида полиномов, позволяющих их обнаружить, представляет собой сложную задачу. В [12] вычислены нижние границы размерности генерирующих полиномов: $m=\lceil\log_2(2N)\rceil$ для трехкратных ошибок и $m=\left\lceil\log_2\left(N+\binom{N}{2}+1\right)\right\rceil$ — для четырехкратных. Таким образом, для обнаружения всех двойных неисправностей требуется сигнатурный анализатор в два раза меньший, чем требуемый для обнаружения неисправностей четвертой кратности.

Заключение

В статье представлен обзор методов периодического неразрушающего тестирования схем памяти маршевыми алгоритмами. Рассмотрены алгоритмы Николаидиса, симметричные и локально симметричные алгоритмы. Проведен сравнительный анализ методов с точки зрения сложности алгоритмов и их покрывающей способности. Показан способ минимизации маскирования ошибок, генерируемых в процессе тестирования.

TRANSPARENT MEMORY TESTING METHODS OVERVIEW

V.N. YARMOLIK, A.P. ZANKOVICH

Abstract

Overview of methods for periodical transparent memory testing is presented in the article. Transparent analogues of classical March algorithms receive primary attention. Wildly accepted method of Nicolaidis is considered here, as well as novel methods, that is based on local and global data symmetry. Comparison studies of considered methods are also presented.

Литература

1. *Prince B.* High-performance memories: new architecture DRAM's and SRAM's, evolution and function. Gr.Britain, John Wiley & Sons, 1996.
2. *Goor A.J. van de.* Testing Semiconductor Memories, Theory and Practice. Gr.Britain, John Wiley & Sons, 1991.
3. *McCluskey E.J.* // IEEE Design & Test of Computers. 1985. Vol. 2, No. 2. P. 21–28.
4. *Nicolaidis M.* // Proc. of Int. Test Conf., USA, 1992. P. 598–606/
5. *Suk D., Reddy S.* // IEEE Trans. on Computers. 1981. Vol. C-30, No. 12. P. 982–985.
6. *Nicolaidis M.* // IEEE Trans. on Computers. 1996. Vol. 45, No. 10. P. 1141–1155.
7. *Yarmolik V.N., Hellebrand S., Wundelich H.-J.* // Proc. of the DATE'99 Conference. 1999. P. 702–707.
8. *Ярмолик В.Н., Калоша Е.П., Быков Ю.В. и др.* Проектирование самотестируемых СБИС // Научная монография в двух томах. Мн., 2001. Т. 2.
9. *Занкович А.П., Ярмолик В.Н.* // Автоматика и телемеханика. 2003. № 9. С. 141–154.
10. *Koneman B.* // Oral presentation, DFT Workshop, USA, 1986.
11. *Yarmolik V., Klimets Yu., Demidenko S., Piuri V.* // 7th Int. Symposium on IC Technology Systems and Applications. Singapore, 1997. P. 192–195.
12. *Yarmolik V.N., Nicolaidis M., Kebichi O.* // Proc. of Int. Test Conf., 1994. P. 368–377.
13. *Yarmolik V.N., Zankovich A.P., Ivaniuk A.A.* // Proc. of Int. Conf. On MixDes. Poland. 2002. P. 545–548.
14. *Zankvich A.P., Yarmolik V.N., Sokol B.* // Proc. of the CADSM. Ukraine. 2003. P. 226–229.
15. *T.W.Williams.* // IEEE Trans. CAD. 1988. Vol. 7, No. 1. P. 75–83.