УДК 004.318

Разработка схем BIST для SRAM-памяти, изготовленной по технологическим нормам 28 нм

Альфонсо Д.М., Исаев М.В.

В статье описываются новые типы возможных дефектов, становящихся статистически значимыми при производстве процессорных кэш-памятей на более тонких технологиях. Был проведён анализ способов обнаружения таких дефектов для однопортовой памяти. В результате исследования создан марш-тест, позволяющий оптимальным образом определять ошибки известных типов, разработана схема коррекции ошибок.

Ключевые слова: SRAM-память, однопортовая память, BIST, обнаружение дефектов, March-тест, BLIF, ActD/DeactD.

1. Введение

Рост полупроводниковой промышленности в последние годы спровоцировал резкое уменьшение характеристического размера полупроводниковых структур со 130 нм до 28 нм. Благодаря этому повышается доступный для разработки микропроцессоров транзисторный бюджет, растёт частота процессоров и их производительность. Также увеличился доступный объём внутренних кэш-памятей процессора, выполненных по технологии SRAM (Static Random Access Memory, статическая оперативная память с произвольным доступом). Однако значительно растёт и сложность электронных устройств, тестирование которых стало представлять серьёзную техническую задачу. Уменьшение норм технологического процесса более остро ставит вопрос улучшения качества отбраковки кристаллов и повышения выхода годной продукции. При уменьшении размеров ячеек памяти увеличивается вероятность возникновения различных дефектов при их производстве, в результате чего в памяти возникают различные типы ошибок. При этом большее значения начинают приобретать те типы дефектов, которые могли крайне редко возникать на более старых техпроцессах.

2. Тестирование памяти с помощью March-тестов

В общем случае считается, что ошибка в работе кристалла происходит из-за каких-либо механических или электрических повреждений транзисторов и прилегающих элементов при изготовлении или работе устройства, что отражается на функционировании более крупных частей схемы, таких как ячейки памяти, отдельные логические элементы или крупные логические блоки. Для обнаружения ошибок в SRAM-памяти широко распространено семейство т.н. марш-тестов (March-тестов), которые характеризуются не только хорошим

покрытием различных типов дефектов, но и сравнительно простой реализацией, а также линейно зависящим от количества ячеек памяти временем тестирования. Марш-тест представляет собой последовательность марш-элементов, каждый из которых состоит из последовательности операций записи (*w0* – запись логического нуля в ячейку, *w1* – запись логической единицы) и чтения (*r0* – чтение из ячейки и сравнение результата с “0”, *r1* – чтение из ячейки и сравнение с “1”; в случае несовпадения при сравнении регистрируется ошибка) и индикатора направления ($\Uparrow$ - инкрементирование адреса, $\Downarrow$ - декрементирование адреса, $\Updownarrow$ - порядок адресации не важен). Операции выполняются последовательно с каждой ячейкой памяти, адрес изменяется в соответствии с индикатором направления. По итогам сравнения результатов чтения с ожидаемыми (эталонными) формируется сигнатура ошибок, которая может быть использована в дальнейшем для получения статистики или для компенсации неисправностей памяти.

Одним из текущих проектов ЗАО МЦСТ является микропроцессор с суммарным размером кэш-памяти более 20Мб, выполняемый по технологическим нормам 28 нм. Память в данном микропроцессоре реализована как однопортовая или двухпортовая с независимыми портами чтения и записи, то есть, она может тестироваться как однопортовая. Наиболее общим марш-тестом, покрывающим дефекты массива однопортовой памяти (SAF, SOF, TF, PF, CFin, CFid, CFst, DRF, DDRF), является модифицированный тест MarchG:

$$\{\Updownarrow(w0);\Uparrow(r0,w1,r1,w0,r0,w1);\Uparrow(r1,w0,w1);\Downarrow(r1,w0,w1,w0);$$
$$\Downarrow(r0,w1,w0);delay;\Updownarrow(r0,w1,r1,r1);delay;\Updownarrow(r1,w0,r0,r0)\}.$$

Сложность теста равна $25\cdot n+2\cdot delay$, где n — количество ячеек памяти, $delay$ — длительность фазы задержки [1]. Дефекты адресного декодера и логики при использовании этого марш-теста покрываются частично. Однако именно эти типы дефектов становятся более статистически-значимыми при уменьшении техпроцесса. Для покрытия дефектов, связанных с утечкой тока транзистора доступа к ячейке и временем активации/деактивации wordline (флага выбора заданной строки), требуется произвести модификацию марш-теста.

3. Новые статистически-значимые типы дефектов

BLIF (BitLine Imbalance Fault) — дефект транзистора доступа bitline (флага выбора заданного столбца). При наличие дефекта данного типа, ток утечки транзистора доступа bitline имеет повышенное значение, таким образом значение, записанное в ячейке, «утекает» из неё за конечное время. Наибольшая вероятность для проявления дефекта возникает в случае, когда все ячейки в соответствующей колонке содержат значения, инверсные содержащемуся в проверяемой ячейке. На этом и основывается принцип обнаружения

дефекта типа BLIF — создание наихудших (т.е. наиболее вероятных) условий для проявления дефекта. [2]

Реализация обращения к памяти по строке позволяет проверять дефекты типа BLIF для всех ячеек строки одновременно. В памяти, инициализированной нулями, в проверяемую строку записывается значение логической единицы во все ячейки, затем происходит считывание значения и сравнение его с единицей. В случае успешного прохождения проверки, значение всех ячеек строки переводится в состояние логического нуля, и управление переходит к проверке следующей строки. Для этого необходимо добавление марш-элементов, составляющих марш-тест BLIF: $\{\Updownarrow(w0);\Updownarrow(w1,r1,w0);\Updownarrow(w1);\Updownarrow(w0,r0,w1)\}$. Однако ввиду конечного времени утечки тока транзистора доступа, считывание единицы сразу же после её записи может не выявить наличие дефекта даже при создании наиболее вероятных для этого условий. В связи с этим была предложена следующая модификация BLIF-теста, названная BLIF+ [3]: $\{\Updownarrow(w0);\Updownarrow(w1,w0_{nxt},r1,w0);\Updownarrow(w1);\Updownarrow(w0,w1_{nxt},r0,w1)\}$, где операции «$w0_{nxt}$» *и* «$w1_{nxt}$» означают запись соответственно нуля и единицы в следующую ячейку, что позволяет увеличить время и вероятность для проявления дефекта.

ActD/DeactD — дефекты, вызванные наличием паразитного сопротивления на входе одного (или нескольких) бит декодера wordline (Рис.1). [3]

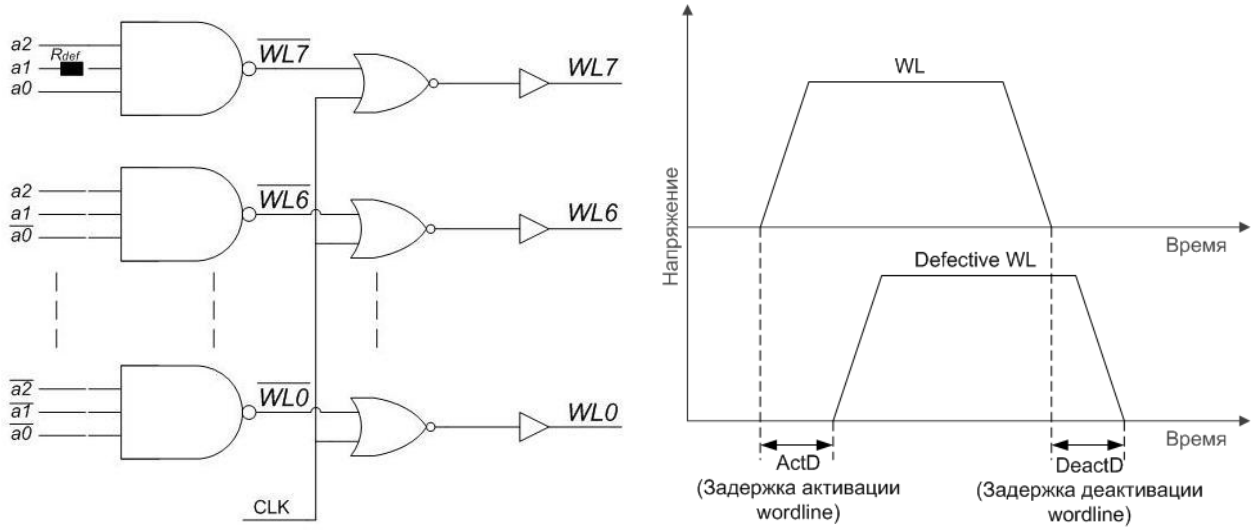

Рис.1. Декодер строки с дефектом и поведение дефектной wordline.

В случае отсутствия дефекта, переключение бит адреса wordline происходит одновременно, и требуемая строка включается сразу же после предыдущей использованной. При наличии такого дефекта, как паразитное сопротивление R_{def} хотя бы на одном из входных бит декодера строки, переключение битов адреса wordline происходит не одновременно: происходит задержка активации или деактивации wordline . Таким образом, в результате дефекта активными могут оказаться одновременно несколько wordline, либо же

наоборот, ни одна worline не будет активна в момент совершения операции записи или чтения, что вызовет некорректное поведение памяти.

Для полного обнаружения данного типа дефектов требуется проверка каждой пары ячеек, чьи адреса wordline отличаются только на 1 бит (далее будем называть такие ячейки «соседними»). При этом одна из ячеек должна содержать значение, инверсное значению другой ячейки, и переключение между ними должно производиться в обе стороны. В таком случае при работе с каждой ячейкой блока памяти должны проверяться $\log_2 n$ ячеек памяти (где n — количество всех ячеек памяти), т.е. только ячейки памяти, являющиеся «соседними» для проверяемой. Таким образом, тестовый алгоритм уже не может быть назван марш-тестом, но может быть представлен схожим образом [3]: $\left\{\Updownarrow(w0);\Uparrow\left(w1,\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_{v\wedge 2^i},r1_v,r0_{v\wedge 2^i}\right),w0\right);\Uparrow(w1);\Uparrow\left(w0,\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_{v\wedge 2^i},r0_v,r1_{v\wedge 2^i}\right),w1\right)\right\}$. Здесь нижний индекс v означает, что операция, к которой он относится, проводится по адресу текущей проверяемой ячейки, а $v\wedge 2^i$ – что операция проводится по отношению к «соседней» ячейке, адрес которой отличается от адреса проверяемой в i-ом бите; $\Uparrow_{v\wedge 2^i}^{N-1}$ означает, таким образом, проход по всем «соседним» ячейкам, $N=\log_2 n$ – разрядность адреса ячеек проверяемой памяти. Сложность такого теста равна $6\cdot n+6\cdot n\cdot\log_2 n$.

Рассмотрим более подробно переходы, проверяемые при таком тестовом алгоритме на примере трёхбитового адреса. Перебор адресов «соседних» ячеек осуществляется во втором и четвёртом марш-элементах, причем в одном случае массив памяти инициализирован нулями, а в проверяемой ячейке находится единица, а в другом случае наоборот, нуль содержится только в проверяемой ячейке. В приведённых ниже таблицах 1 и 2 представлены некоторые осуществляющиеся переходы между адресами, соответствующие элементам $\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_{v\wedge 2^i},r1_v,r0_{v\wedge 2^i}\right)$ и $\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_{v\wedge 2^i},r0_v,r1_{v\wedge 2^i}\right)$. Первой строчкой в таблице указаны значения, содержащиеся в ячейках, между которыми происходит переключение.

0 – 1 – 0	**0 – 1 – 0**	**0 – 1 – 0**	**0 – 1 – 0**
001-000-001	000-001-000	011-010-011	010-011-010
010-000-010	011-001-011	000-010-000	001-011-001

Таблица 1. Некоторые переходы, проверяемые в блоке $\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_{v\wedge 2^i},r1_v,r0_{v\wedge 2^i}\right)$.

1 – 0 – 1	**1 – 0 – 1**	**1 – 0 – 1**	**1 – 0 – 1**
001-000-001	000-001-000	011-010-011	010-011-010
010-000-010	011-001-011	000-010-000	001-011-001

Таблица 2. Некоторые переходы, проверяемые в блоке $\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_{v\wedge 2^i},r0_v,r1_{v\wedge 2^i}\right)$.

Наглядно можно видеть, что вторая таблица будет составлена из точно таких же пар переходов, что и первая, но в них изменён порядок прохождения адресов. В самом деле, если необходимо проверить правильность выполнения переходов, например, между адресами 101 и 100, то это может быть сделано двумя способами (100-101-100 и 101-100-101), в зависимости от того, с какого адреса начинать, но сами переходы при этом абсолютно одинаковые, только переставлены местами. В силу схемотехнических причин возникновения дефектов типов ActD/DeactD (Рис.1), переходы в одну и в другую стороны между парой «соседних» ячеек не имеют влияния друг на друга, поэтому можно разъединить данные цепочки и менять переходы местами между собой и различными парами «соседних» ячеек.

Данное замечание позволяет существенно снизить сложность теста и упростить его реализацию. Наиболее удобно, чтобы не нарушать симметрию тестового алгоритма, оставить и во втором, и в четвёртом марш-элементах по одной операции чтения из каждой ячейки, являющейся «соседней» для проверяемой, следующим образом: $\Uparrow\left(w1,\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_v,r0_{v\wedge 2^i}\right),w0\right)$; $\Uparrow\left(w0,\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_v,r1_{v\wedge 2^i}\right),w1\right)$. Например, проверка всех переходов в цепочке адресов 000-001-000 при всех возможных значениях в ячейках будет происходить по частям в указанных марш-элементах: в первом проверяются переходы 000-001 и 001-000 при наличии в первой ячейке единицы, а во второй нуля; во втором проверяются переходы 000-001 и 001-000 при наличии в первой ячейке нуля, а во второй единицы. Аналогично можно проверить, что так же раздельно проверяются все переходы для всех цепочек соседних адресов, а значит, обеспечивается полное покрытие переходов, требующих проверки.

4. Оптимизированный тестовый алгоритм

Таким образом оптимизированный тестовый алгоритм для обнаружения дефектов типов ActD/DeactD будет выглядеть следующим образом:
$\left\{\Updownarrow(w0);\Uparrow\left(w1,\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_v,r0_{v\wedge 2^i}\right),w0\right);\Uparrow(w1);\Uparrow\left(w0,\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_v,r1_{v\wedge 2^i}\right),w1\right)\right\}$. Его сложность равна $6\cdot n+4\cdot n\cdot\log_2 n$, что даёт выигрыш порядка 25-30% (~25% для малых памятей, ~30% для больших) по сравнению с алгоритмом без оптимизации.

Необходимо дополнить также общий тестовый алгоритм проверки памяти для более полного покрытия ошибок адресного декодера. Ввиду необходимости наличия элементов вида $\left\{\Updownarrow(w1,r1,w0)\right\}$ и $\left\{\Updownarrow(w0,r0,w1)\right\}$ для покрытия ошибок типа BLIF, был выбран симметричный вид модифицированного марш-теста MarchG [1]:

$\{\Updownarrow(w0);\Uparrow(r0,w1,r1,w0,w1);\Uparrow(r1,w0,r0,w1);\Downarrow(r1,w0,w1,w0);$
$\Downarrow(r0,w1,r1,w0);delay;\Updownarrow(r0,w1,r1,r1);delay;\Updownarrow(r1,w0,r0,r0)\}$.

Как видно, данный марш-тест включает в себя *необходимые* наихудшие условия теста BLIF в третьем и пятом марш-элементах. Для обнаружения дефектов типа ActD/DeactD предлагается дополнить данные марш-элементы, так что итоговый тест будет выглядеть следующим образом:

$$\{\Updownarrow(w0);\Uparrow(r0,w1,r1,w0,w1);\Uparrow(r1,w0,\Uparrow_{v\wedge 2^i}^{N-1}(r0_v,r1_{v\wedge 2^i}),w1);\Downarrow(r1,w0,w1,w0);$$
$$\Downarrow(r0,w1,\Uparrow_{v\wedge 2^i}^{N-1}(r1_v,r0_{v\wedge 2^i}),w0);delay;\Updownarrow(r0,w1,r1,r1);delay;\Updownarrow(r1,w0,r0,r0)\}.$$

При этом также будут созданы *достаточные* наихудшие условия для BLIF, т.к. чтение из проверяемой ячейки/строки (в которой содержится значение, инверсное содержимому остальных ячеек столбца/остальных строк) происходит неоднократно и, таким образом, проходит достаточное время для проявления дефекта утечки тока транзистора доступа проверяемой ячейки.

5. Заключение

Сложность предложенного общего тестового алгоритма равна $25\cdot n+4\cdot n\cdot\log_2 n+2\cdot delay$, что характеризует его как более долгий по сравнению с обычными марш-тестами, имеющими сложность $O(n)$, но более быстрый, чем GalRow и GalCol (сложность $O(n^{1,5})$) и значительно более быстрый, чем GalPat и GalPat- (сложность $O(n^2)$) [1].

Литература

1. Ad J. van de Goor, «Using March Tests to Test SRAMs», IEEE Design & Test of Computers, pp 8-14, 1993
2. Ad J. van de Goor, Said Hamdioui and R. Wadsworth, «Detecting Faults in the Peripheral Circuits and an Evaluation of SRAM Tests», In Proc. of the IEEE Int. Test Conf., pp. 114-123, 2004
3. Ad J. van de Goor, Said Hamdioui, Georgi N. Gaydadjiev, Zaid Al-Ars, «New Algorithms for Address Decoder Delay Faults and Bitline Imbalance Faults», In Proc. of the IEEE Int. Test Conf., pp. 391-396, 2009