Д.М. Альфонсо, М.В. Исаев, В.О. Костенко (ЗАО «МЦСТ»)

D. Alfonso, M. Isaev, V. Kostenko

РАЗРАБОТКА СИСТЕМЫ ТЕСТИРОВАНИЯ И ПОВЫШЕНИЯ ВЫХОДА ГОДНОЙ ПРОДУКЦИИ ДЛЯ КЭШ-ПАМЯТИ МИКРОПРОЦЕССОРА, ИЗГОТОВЛЕННОГО ПО ТЕХНОЛОГИЧЕСКИМ НОРМАМ 28 НМ

TEST AND IMPROVEMENT OF USABLE PRODUCT YIELD SYSTEM DESIGN FOR CACHE MEMORY OF MICROPROCESSOR, MANUFACTURED AT 28 NM PROCESS TECHNOLOGY

Описываются типы и анализируются способы обнаружения дефектов, статистически-значимых при производстве кэш-памяти по передовым технологическим нормам. Приводятся алгоритм тестирования, позволяющий оптимальным образом определять ошибки, вызванные такими дефектами, и метод их компенсации. На данной основе разработаны решения, позволяющие реализовать и проконтролировать эту компенсацию в аппаратуре.

This paper describes the types of the defects becoming more statistically significant for cache memory manufactured at advanced process technologies, and analyses methods of its detecting. Test algorithm detecting errors caused by such defects in optimal way, and method of their compensation are represented. The solution realizing and controlling the compensation in hardware, designed on this basis, is also described.

Ключевые слова: кэш-память, однопортовая память, March-тест, BLIF, ActD/DeactD, BIST, BISR, резервирование, JTAG.

Keywords: cache memory, singleport memory, March test, BLIF, ActD/DeactD, BIST, BISR, redundancy, JTAG.

Введение

Прогресс электронной индустрии обусловил резкое уменьшение характеристического размера полупроводниковых структур, свойственное и передовым разработкам ЗАО «МЦСТ», где в настоящее время создается микропроцессор «Эльбрус-4С+» на базе техно-

логических норм 28 нм. Освоение новых норм позволяет повысить заданный при проектировании транзисторный бюджет, существенно поднять тактовую частоту процессоров и их производительность. Увеличился и технологически доступный объём внутренней кэш-памяти, выполняемой в виде статической оперативной памяти с произвольным доступом (Static Random Access Memory, SRAM). Однако успешная реализация этой возможности в большой степени связана с решением проблемы, обусловленной резким сокращением размеров ячеек памяти, – увеличением вероятности производственных дефектов, в т.ч. и крайне редких в предыдущих версиях техпроцессов.

1. Тестирование памяти с помощью марш-тестов

Для обнаружения ошибок в SRAM-памяти используются встроенные системы автоматического тестирования по заданному алгоритму – BIST, Built-In Self-Test. При этом широкое распространение получило семейство так называемых марш-тестов (March-tests), которые характеризуются не только хорошим покрытием различных типов дефектов, но и сравнительно простой реализацией, а также – линейной зависимостью времени тестирования от числа ячеек памяти. Марш-тест представляет собой последовательность марш-элементов, каждый из которых состоит из ряда операций записи в ячейку ($w0$ – запись логического нуля, $w1$ – запись логической единицы) и чтения из ячейки с индикацией несовпадения при сравнении с ожидаемым результатом ($r0$ – чтение и сравнение результата с «0», $r1$ – чтение и сравнение с «1»). Эти операции выполняются с каждой ячейкой памяти, адрес изменяется в соответствии с индикатором направления ($\Uparrow$ – инкрементирование адреса, $\Downarrow$ – декрементирование адреса, χ – порядок адресации не важен). По итогам сравнения результатов чтения с ожидаемыми (эталонными) значениями формируется сигнатура ошибок, которая может быть использована в дальнейшем для получения статистики и/или компенсации неисправностей памяти.

Наиболее общим из используемых в настоящее время марш-тестов является моди-

фицированный тест MarchG:

$$\{\Updownarrow(w0);\Uparrow(r0,w1,r1,w0,r0,w1);\Uparrow(r1,w0,w1);\Downarrow(r1,w0,w1,w0);\\ \Downarrow(r0,w1,w0);delay;\Updownarrow(r0,w1,r1,r1);delay;\Updownarrow(r1,w0,r0,r0)\}. \quad (1)$$

Его сложность равна $25\cdot n+2\cdot delay$, где n – количество ячеек памяти, *delay* – длительность фазы задержки [1]. Тест обнаруживает дефекты, связанные с ошибочным значением хранящегося в ячейке бита, отсутствием соединения, нарушением динамики изменения состояния, неудовлетворительными временны́ми характеристиками, влиянием состояния одной ячейки на состояние другой (SAF, SOF, TF, PF, CFin, CFid, CFst, DRF, DDRF). Более подробная классификация приведена в [1].

2. Статистически-значимые типы дефектов, характерные для передовых технологий

Проблема полного покрытия дефектов, характерных для проектирования на базе технологических норм 28 нм, приобрела актуальность в ЗАО «МЦСТ» в связи с разработкой кэш-памяти микропроцессора «Эльбрус-4С+». Ее суммарный объем составляет более 20 Мбайт, причем более 80% этого объема представлено однопортовыми банками памяти и двухпортовыми банками с отдельными портами чтения и записи, которые без потери полноты тестирования могут быть проверены по тем же принципам, что и однопортовые.

Одной из существенных особенностей этого проекта была недостаточность традиционно используемого теста MarchG – для обнаружения новых в нашей практике типов дефектов. В публикациях на эту тему особо выделяются два из них. Первый, BLIF (BitLine Imbalance Fault) [2], характерен повышенным током утечки одного из транзисторов, расположенных на линии выбора заданного столбца (bitline), вследствие чего записанное в ячейке значение через некоторое время (до нескольких десятков наносекунд) оказывается потерянным. Наибольшая вероятность такого дефекта типична для случая, когда все ячейки в задействованном столбце содержат значения, инверсные значению проверяемой ячейки. Соответственно, принцип обнаружения дефектов типа BLIF состоит в создании

таких условий.

Построчное обращение к памяти позволяет проверять дефекты типа BLIF для всех ячеек строки одновременно. В памяти, инициализированной нулями, во все ячейки строки записываются единицы, затем происходят считывание установленных значений и сравнение их с единицей. В случае равенства значение всех ячеек строки переводится в состояние логического нуля, и тест переходит к проверке следующей строки. Такая последовательность действий описывается марш-элементами:

$$\{\Updownarrow(w0);\Updownarrow(w1,r1,w0);\Updownarrow(w1);\Updownarrow(w0,r0,w1)\}. \qquad (2)$$

Однако из-за отмеченной выше задержки в проявлении дефекта BLIF немедленное считывание единицы сразу же после её записи может не выявить дефект даже при создании соответствующих условий. В связи с этим была предложена следующая модификация BLIF-теста, названная BLIF+ [3]:

$$\{\Updownarrow(w0);\Updownarrow(w1,w0_{nxt},r1,w0);\Updownarrow(w1);\Updownarrow(w0,w1_{nxt},r0,w1)\}. \qquad (3)$$

Здесь операции «$w0_{nxt}$» и «$w1_{nxt}$» означают соответственно запись нуля и единицы в следующую ячейку, что позволяет увеличить время между записью и чтением применительно к проверяемой ячейке и, как следствие, повысить вероятность проявления дефекта.

Другим типом дефектов, который становится статистически-значимым при новых технологических нормах [3], является возникновение повышенного паразитного сопротивления на входе одного (или нескольких) битов декодера линии выбора заданной строки (wordline), известное как дефект типа ActD/DeactD (рис. 1). При отсутствии этого дефекта переключение всех разрядов адреса, поданного на вход декодера линии wordline, происходит одновременно, и требуемая строка включается сразу же после предыдущей. Если же дефект проявляется хотя бы на одном из входных разрядов в виде паразитного сопротивления R_{def}, то переключение разрядов адреса происходит не одновременно, что приводит к задержке активации или деактивации линии wordline. В результате активными могут оказаться одновременно несколько линий wordline, либо, наоборот, ни одна линия wordline не

будет активна в момент завершения операции записи или чтения.

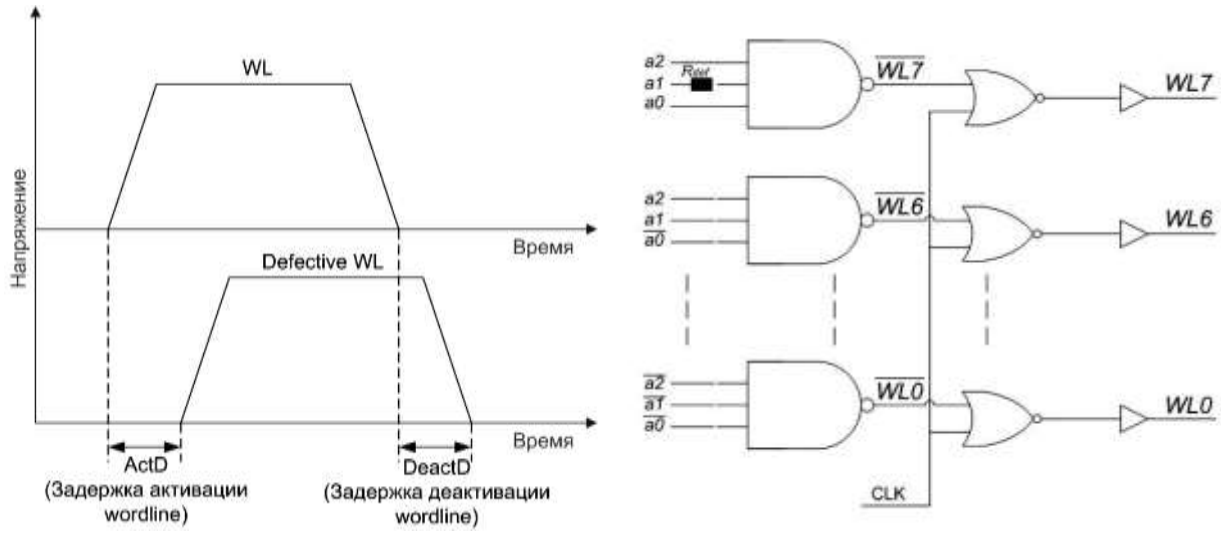

Рис. 1

Декодер строки с дефектом и поведение дефектной линии wordline

Для полного обнаружения дефектов данного типа требуется проверить все «соседние» пары строк, для которых адреса линий wordline отличаются только на единицу (переходы между парами строк, не являющихся соседними, в данном случае не представляют интереса.) При этом одна из строк должна содержать значение, инверсное значению другой строки, и переключение между ними должно производиться в обе стороны. В таком случае при тестировании каждой строки некоторого банка памяти должны также активироваться все строки, являющиеся соседними для проверяемой. Если n – количество всех строк в банке памяти, то у каждой строки в наличии $\log_2 n$ соседних строк. Из работы [3] известен следующий тестовый алгоритм, который, как будет показано далее, может быть оптимизирован:

$$\left\{\Updownarrow(w0);\Uparrow\left(w1,\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_{v\wedge 2^i},r1_v,r0_{v\wedge 2^i}\right),w0\right);\Uparrow(w1);\Uparrow\left(w0,\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_{v\wedge 2^i},r0_v,r1_{v\wedge 2^i}\right),w1\right)\right\}. \quad (4)$$

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; таким образом, $\Uparrow_{v\wedge 2^i}^{N-1}$ означает последовательный проход по всем соседним ячейкам, $N=\log_2 n$ –

разрядность адреса строк проверяемой памяти. Сложность данного теста равна $6 \cdot n + 6 \cdot n \cdot \log_2 n$.

Начнем его оптимизацию с того, что рассмотрим более подробно переходы, выполняемые в этом тестовом алгоритме, на примере трёхбитового адреса. Перебор адресов соседних строк осуществляется во втором и четвёртом тестовых элементах, причем в одном случае все строки памяти инициализированы нулями, а в проверяемой строке содержатся только единицы, в то время как в другом случае, наоборот, нули содержатся только в проверяемой строке. В табл. 1 и 2 представлены некоторые осуществляющиеся переходы между адресами, соответствующие последовательностям тестовых элементов $\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_{v\wedge 2^i}, r1_v, r0_{v\wedge 2^i}\right)$ и $\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_{v\wedge 2^i}, r0_v, r1_{v\wedge 2^i}\right)$.

Таблица 1

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

Значения, содержащиеся в строках, между которыми происходят переключения			
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 2

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

Значения, содержащиеся в строках, между которыми происходят переключения			
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

Из таблиц следует, что второй последовательности элементов соответствует точно такое же множество пар переходов, что и первой, но с изменённым порядком прохождения адресов. В самом деле, если необходимо проверить правильность выполнения переходов, например между адресами 101 и 100, то это можно сделать двумя способами (100-101-100 и 101-100-101) в зависимости от начального адреса строки. В одном случае ука-

занные последовательности элементов в первом переходе проверяют наличие задержки деактивации линии wordline, соответствующей адресу 100, и задержки активации wordline, соответствующей адресу 101; во втором переходе, соответственно, проверяется наличие задержки деактивации wordline, отвечающей адресу 101, и задержки активации wordline, отвечающей адресу 100. В другом случае проверяется то же самое, но в обратном порядке. Таким образом, переходы, представленные в обоих случаях, являются абсолютно одинаковыми, только выполняются в разном порядке. При этом совмещение проверки задержки активации wordline и проверки задержки её деактивации в одном и том же тестовом элементе не увеличивает тестовое покрытие. Поэтому можно переупорядочить рассматриваемые последовательности, а именно, избавиться от повторной проверки дефектов одних и тех же wordlines и поменять местами переходы между различными парами соседних строк.

3. Оптимизированный тестовый алгоритм

Полученный выше вывод позволяет существенно снизить сложность и упростить реализацию рассматриваемого теста. Чтобы не нарушать симметрию его алгоритма, удобно оставить и во втором, и в четвёртом тестовых элементах рассматриваемых последовательностей по одной операции чтения из каждой строки, являющейся «соседней» для проверяемой: $\Uparrow\left(w1,\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_v,r0_{v\wedge 2^i}\right),w0\right)$; $\Uparrow\left(w0,\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_v,r1_{v\wedge 2^i}\right),w1\right)$. Например, проверка всех переходов между строками с адресами 000 и 001 при всех возможных значениях этих строк будет происходить по частям в указанных элементах. В первом элементе проверяются переходы между адресами 000-001 и 001-000 при наличии в первой строке единиц, а во второй – нулей. Во втором элементе проверяются переходы между адресами 000-001 и 001-000 при наличии в первой строке нулей, а во второй – единиц. Аналогично можно показать, что так же раздельно проверяются все переходы для всех пар соседних адресов, а значит, обеспечивается полное тестовое покрытие дефектов рассматриваемого типа.

Таким образом, оптимизированный тестовый алгоритм для обнаружения дефектов типа ActD/DeactD выглядит так:

$$\left\{\Updownarrow(w0);\Uparrow\left(w1,\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_v,r0_{v\wedge 2^i}\right),w0\right);\Uparrow(w1);\Uparrow\left(w0,\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_v,r1_{v\wedge 2^i}\right),w1\right)\right\}. \qquad (5)$$

Его сложность равна $6\cdot n+4\cdot n\cdot\log_2 n$, что даёт выигрыш порядка 25-30% (~25% для банков памяти большого размера, ~30% для банков памяти маленького размера) по сравнению с исходным алгоритмом (3) без оптимизации.

На основе описанных тестовых алгоритмов (1), (2) и (5) успешно решается задача построения общего тестового алгоритма, обнаруживающего дефекты обоих рассмотренных типов. В качестве основы используется тест (1), причем, согласно [1], для более полного покрытия ошибок адресного декодера следует использовать его в симметричном виде:

$$\begin{array}{l}\{\Updownarrow(w0);\Uparrow(r0,w1,r1,w0,w1);\Uparrow(r1,w0,r0,w1);\Downarrow(r1,w0,w1,w0);\\ \Downarrow(r0,w1,r1,w0);delay;\Updownarrow(r0,w1,r1,r1);delay;\Updownarrow(r1,w0,r0,r0)\}\end{array}. \qquad (6)$$

Данный марш-тест в третьем и пятом марш-элементах включает в себя *необходимые* наихудшие условия теста (2). Для обнаружения дефектов типа ActD/DeactD следует, в соответствии с (5), дополнить тест (6) следующим образом:

$$\begin{array}{l}\left\{\Updownarrow(w0);\Uparrow(r0,w1,r1,w0,w1);\Uparrow\left(r1,w0,\Uparrow_{v\wedge 2^i}^{N-1}\left(r0_v,r1_{v\wedge 2^i}\right),w1\right);\Downarrow(r1,w0,w1,w0);\right.\\ \left.\Downarrow\left(r0,w1,\Uparrow_{v\wedge 2^i}^{N-1}\left(r1_v,r0_{v\wedge 2^i}\right),w0\right);delay;\Updownarrow(r0,w1,r1,r1);delay;\Updownarrow(r1,w0,r0,r0)\right\}\end{array}. \qquad (7)$$

При этом также будут созданы *достаточные* наихудшие условия для варианта 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 типа BLIF.

Сложность алгоритма (7) 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]. При этом он будет обеспечивать доста-

точные условия для нахождения всех статистически важных типов дефектов, свойственных производству процессоров по технологическим нормам 28 нм.

4. Активация резервных элементов памяти

Увеличивающееся число дефектов влечёт за собой снижение процента выхода годных кристаллов, т.е. таких, которые могут корректно отрабатывать все операции даже с возможным падением производительности. При производстве кристаллов большой площади увеличение количества годных чипов является одной из основных задач. Для её решения используют две основные методики: маскирование элементов, содержащих неисправности, либо их замену резервными элементами.

Маскирование используется в банках памяти, обнаруженные дефекты которых не препятствуют использованию микропроцессора, однако снижают его производительность. Для маскирования дефектных элементов кэш-памяти пользуются специальными отметками, так называемыми «don’t use» битами, помечающими дефектный блок памяти как запрещённый к использованию. Эти отметки проставляются во время тестирования памяти схемами BIST и хранятся либо в специально отведённой для этого памяти, либо в части специализированной служебной памяти, например, в памяти тегов, относящейся к соответствующей памяти данных. Соответствующие блоки памяти не используются микропроцессором при выполнении программ. Производительность при этом снижается в зависимости от размера и количества маскируемых элементов и возможной частоты обращения к ним при исправной работе. В случае значительного суммарного снижения производительности, вызванного неприемлемым количеством дефектных банков, может быть сделано заключение о неработоспособности кристалла.

За активацию резервных элементов отвечает система автоматической компенсации ошибок – BISR (Built-in Self Repair). В качестве резервных элементов обычно используются дополнительные столбцы, строки и банки памяти. Резервирование предпочтительнее

маскирования тем, что не влияет прямо на производительность процессора, хотя и снижает быстродействие памяти и увеличивает занимаемую ей площадь.

Добавление резервного банка памяти является самым простым, но и самым неэффективным видом резервирования памяти. Его недостатком является то, что при увеличении вероятности ошибок в различных банках необходимо либо увеличивать количество резервных банков, либо снижать процент выхода кристаллов. Первое решение значительно увеличивает площадь и нарушает регулярность физической структуры кэш-памяти, что повышает сложность физического проектирования и поэтому сопряжено с ухудшением ее частотных характеристик.

Так как резервные строки и столбцы вводятся в состав каждого банка памяти, то в случае, когда память набирается банками маленького размера (что повышает ее быстродействие), резервные структуры позволяют исправить большое количество ошибок, равномерно распределённых по банкам памяти. Выбирая между резервными строками и резервными столбцами, следует принимать во внимание те дефекты, которые свойственны технологии изготовления данного кристалла. Однако стоит учесть, что добавление резервных столбцов увеличивает площадь памяти менее чем на 6%, а временны́е характеристики памяти и вовсе остаются практически неизменными, тогда как добавление резервных строк существенно больше влияет на характеристики банка памяти. Это подтверждает табл. 3, составленная применительно к банкам памяти, используемым в ОКР «Эльбрус-4С+» (2048 строк, ширина данных 37 бит). Поэтому добавление резервных столбцов является фактически обязательным видом резервирования, использующегося для систем BISR. Оно было реализовано и при разработке микропроцессора «Эльбрус-4С+».

Таблица 3

Изменения характеристик памяти при добавлении резервных колонок и столбцов

	Без резервирования	**С резервными столбцами**	**С резервными строками**
Площадь банка памяти, мкм^2	17529,9	18566,5 (+5,9%)	19700,7 (+12,4%)
Тактовая частота, МГц	1255,5	1255,0	1160,5 (-7,6%)

Задержка данных чтения, нс	0,479	0,480	0,562 (+17%)

5. Сигнатура протестированной памяти

После завершения работы BIST и BISR важно проконтролировать количество и распределение по типам ошибок, обнаруженных в банках памяти (сигнатуру памяти), чтобы иметь данные для оценки реального выхода годных кристаллов. На этой основе составляется классификация кристаллов по следующим группам:

1) кристаллы без дефектов;
2) кристаллы с единичными дефектами, которые скомпенсированы активацией резервных элементов;
3) кристаллы с единичными дефектами, которые не могут быть скомпенсированы, что приводит к некоторому падению производительности;
4) кристаллы с множественными дефектами, которые должны быть признаны негодными.

В ОКР «Эльбрус-4С+» сигнатура тестирования банков кэш-памяти, содержащая большое количество данных, сохраняется во вспомогательных регистрах для выдачи через JTAG-интерфейс, а также записывается в пространстве программно-доступных конфигурационных регистров микропроцессора. Достоинствами JTAG-интерфейса являются возможность вывода необходимого объёма информации и работоспособность при широком спектре неисправностей микропроцессора, но для работы с ним нужно использовать специализированное отладочное оборудование. Конфигурационные регистры, несмотря на ограниченность их количества, предоставляют возможность считать достаточно информации для программного контроля работоспособности и производительности процессора из BIOS или средствами операционной системы. Сочетание этих двух способов обеспечивает необходимую гибкость отладки, возможность сбора статистики и отбраковки систем на кристалле.

В общем случае, рассматривая проблему хранения и вывода сигнатуры, следует учи-

тывать, что важнейшей задачей тестирования памяти на предмет наличия дефектов является увеличение точности их обнаружения. Для этого следует использовать более сложные и длительные тестовые последовательности и/или тестировать память при различных условиях, таких как температура, напряжение, частота работы, состояние после интенсивной нагрузки. Поскольку подобное тестирование занимает значительное время и не может выполняться при каждой инициализации процессора, для хранения сигнатур необходимо применять энергонезависимую память, инициализируемую в процессе технологического тестирования микропроцессора его изготовителем. При каждом включении питания микропроцессора эта память должна считываться схемами BIST/BISR, которые на основе хранящихся в ней сигнатур активируют резервные элементы банков памяти и устанавливают «don’t use» биты.

Определение оптимального типа энергонезависимой памяти и доработка встроенной системы тестирования памяти для её использования в более сложном, возможно многоступенчатом цикле тестирования, являются приоритетными направлениями для дальнейших исследований и разработок.

Заключение

С переходом на передовые нормы техпроцесса тестирование кристаллов становится всё более сложной и ответственной задачей ввиду повышения статистической значимости ранее не рассматривавшихся типов дефектов. В статье, подготовленной в процессе проектирования микропроцессора «Эльбрус-4С+» на базе технологических норм 28 нм, описаны новый алгоритм обнаружения таких дефектов системой автоматического тестирования (BIST) и схема их исправления в системе автоматического самовосстановления (BISR). Приведены решения, обеспечивающие возможность хранения и вывода сигнатуры работы BIST и BISR, достаточную при отладке, анализе качества и отбраковке микропроцессоров нового поколения.

Модули, реализующие функционал предложенных решений, были описаны на языке Verilog, успешно прошли тестирование и готовы к использованию в последующих проектах.

Литература

1. Ad J. van de Goor. Using March Tests to Test SRAMs. IEEE Design & Test of Computers, 1993, pp. 8-14.

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., 2004, pp. 114-123.

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., 2009, pp. 391-396.