УДК 004.33.054

АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ ФЛЭШ-ПАМЯТИ

С.В. ЯРМОЛИК

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

Поступила в редакцию 23 марта 2010

Рассматриваются вопросы построения современной флэш-памяти и анализируются модели неисправностей характерные для данного типа запоминающих устройств. Показывается эффективность применения маршевых тестов запоминающих устройств для обнаружения неисправностей флэш-памяти. Приводится оценка покрывающей способности маршевых тестов.

Ключевые слова: флэш-память, NOR- и NAND-архитектуры, неисправности памяти, тестирование памяти, маршевые тесты.

Введение

В последнее время широкое распространение получили мобильные, энергонезависимые запоминающие устройства, такие как флэш-память. Данный тип памяти представляет собой разновидность E^2PROM памяти и позволяет выполнять операции программирования и стирания информации. Флэш-память отличается своей компактностью, дешевизной и низким энергопотреблением, что и предопределило ее повсеместное применение и в первую очередь в цифровых портативных устройствах. Кроме того, флэш-память используется для хранения встроенного программного обеспечения в различных устройствах и системах. Наиболее известно применение флэш-памяти в USB флэш-накопителях. Сейчас активно рассматривается возможность замены жестких дисков на флэш-память. В связи с широким применением на практике флэш-памяти важным фактором является ее надежность, которая достигается применением эффективных методов тестирования флэш-памяти. Для осуществления процедуры тестирования необходимо применение тестов, которые эффективно обнаруживают неисправности флэш-памяти.

Архитектура флэш-памяти

Флэш-память состоит из массива ячеек памяти, представляющих собой транзисторы с плавающими затворами [1]. Традиционно одна ячейка такой памяти хранит один бит информации (*single-level cell — SLC*), однако некоторые современные виды флэш-памяти позволяют записать более одного бита информации в одну ячейку (*multi-level cell — MLC*).

Схема однобитовой ячейки (рис. 1) представляет собой транзистор с двумя изолированными затворами: управляющим (*control gate*) и изолированным оксидными слоями плавающим (*floating gate*), расположенным под управляющим затвором [2]. Некоторые из электронов, благодаря наличию большей энергии, преодолевают слой изолятора и попадают на плавающий затвор, который может удерживать электроны, т.е. заряд, в течение нескольких лет. При этом, когда на плавающем затворе хранится заряд, ячейка находится в состоянии логического "0", т.е. является запрограммированной. Стирание информации, изменение состояния транзистора с логического "0" на "1", происходит при помощи удаления заряда с плавающего затвора. Для этого на управляющий затвор подается высокое отрицательное напряжение и электроны с плавающего затвора переходят (туннелируют) на исток. При чтении эти состояния распознаются

путем измерения порогового напряжения транзистора. Существуют две технологии для удаления и переноса заряда из и на плавающий затвор: инжекция горячих электронов (*hot-electron injection — HEI*) и механизм туннелирования Фаулера-Нордхайма (*Fowler-Nordheim — FN — tunnelling mechanism*) [1].

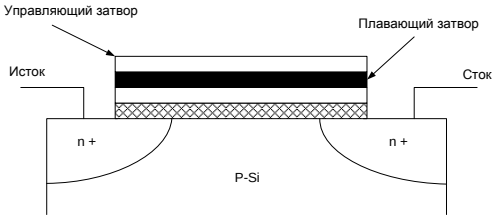

Рис. 1. Полевой транзистор с плавающим затвором

Особенностью флэш-памяти является конечное число циклов стирания/записи ячейки. Обычные флэш-карты позволяют совершить около 100 000 циклов стирания/записи. При этом флэш-память проектируется с учетом данной особенности памяти таким образом, чтобы все ячейки "снашивались" примерно одинаково и не было в памяти участков с большой разницей между количеством примененных операций записи к ячейкам.

Существует два основных вида организации транзисторов с плавающим затвором во флэш-памяти: *NOR*- (рис. 2,*а*) и *NAND*-структуры (рис. 2,*б*), отличающихся друг от друга логической схемой и интерфейсом доступа для чтения и записи памяти [1].

В *NOR*-памяти каждая ячейка памяти подключена к линии бит и к линии питания, как показано на рис. 2,*а*. При этом к линии бит *BL* подключен сток транзистора, к линии питания *SL* – исток, к линии слов *WL* – затвор. Такая структура называется "*NOR*-флэш", потому что она ведет себя соответственно логике *NOR*: когда на линию слов (*WL*) подается высокое напряжение, соответствующий транзистор устанавливает низкое напряжение на линии бит (*BL*) [3].

NOR-память требует более длительного промежутка времени на программирование ячейки и стирание ее значения по сравнению с *NAND*-памятью. Однако, так как *NOR*-память имеет полный набор линий данных и адресов, она обеспечивает доступ к любой произвольной ячейке памяти. В то же время, когда программироваться (устанавливаться в состояние логического нуля) может каждая ячейка, стирать данные (устанавливать в состояние логической единицы) можно только всю память или блок памяти одновременно. В такие блоки объединяются ячейки, подключенные к одной линии питания (*SL*).

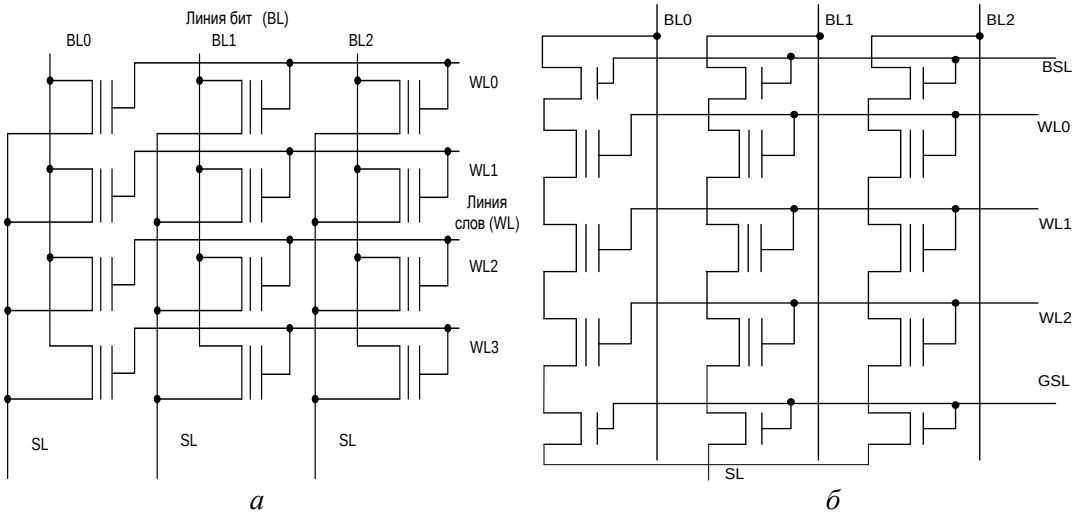

Рис. 2. *NOR*- и *NAND*-архитектуры флэш-памяти

Сразу после операции стирания над ячейками блока любую позицию в данном блоке можно запрограммировать. Однако, как только бит памяти был установлен в состояние логического нуля, он не может быть перепрограммирован и установлен в единицу, до того как операция стирания не будет применена ко всему блоку. В то же время ячейку, в которую записана

"1", можно перезаписывать до тех пор, пока в ней не установиться "0". Так, значение "11111" можно перезаписать как "10111", потом изменить на "10100" (но не на "11100") и так до значения "00000".

В *NAND*-памяти транзисторы соединены последовательно линией бит *BL* таким образом, что исток одного транзистора подключен к стоку следующего, как показано на рис. 2,*б*. При этом соединенные таким образом ячейки будут стираться одновременно. Все ячейки памяти помещены между двумя транзисторами *BSL* и *GSL*, которые управляют доступом ячеек соответственно к линии бит *BL* и линии источника питания *SL*. *NAND*-архитектура позволяет программировать и стирать данные памяти быстрее, чем *NOR*-архитектура памяти, и требует меньше места для одной ячейки, позволяя увеличить плотность расположения транзисторов и уменьшить стоимость памяти, но она не позволяет осуществлять произвольный доступ к ячейкам памяти. *NAND*-память значительно проигрывает в операциях с произвольным доступом и не позволяет напрямую работать с байтами информации. К примеру, для изменения одного байта требуется сначала считать в буфер блок информации, в котором данный байт находится, потом в буфере изменить нужный байт и записать блок с измененным байтом обратно.

NAND-флэш память нашла применение в качестве запоминающего устройства больших объемов информации и для ее переноса. Наиболее распространенные в настоящее время запоминающие устройства, основанные на этом типе памяти, это флэш-драйвы и карты памяти. Что касается *NOR*-флэша памяти, то чипы с такой организацией используются в качестве хранителей программного кода, иногда реализовываются в виде интегрированных решений (ОЗУ, ПЗУ и процессор на одной мини-плате, а то и в одном чипе).

Модели неисправностей флэш-памяти

Для флэш-памяти характерны как известные и описанные для ОЗУ модели неисправностей, такие как константные неисправности (*Stuck-At Faults — SAF*); переходные неисправности (*Transition Faults — TF*); неисправности типа обрыв (*Stuck-Open Faults - SOF*); адресные неисправности (*Address Faults — AF*) и статические неисправности взаимного влияния (*State Coupling Faults — CFst*) [4, 5], так и неисправности, присущие только флэш-памяти.

Рассмотрим неисправности, которые возникают только во флэш-памяти. Их условно можно поделить на три группы, а именно неисправности программирования, неисправности записи и неисправности чтения флэш-памяти [6, 7].

1. Неисправности программирования флэш-памяти (*Program Disturbances*).

– *Word-line program disturbance — WPD.* Данная неисправность возникает, когда при программировании одной ячейки состояние другой ячейки (ячейки-жертвы), присоединенной к той же линии слов, что и ячейка-агрессор, изменяется на логический ноль.

– *Word-line erase disturbance — WED.* В случае данной неисправности программирование одной ячейки заставляет другую ячейку на той же линии слов, что и ячейка-агрессор, стереть записанное в ней значение логического нуля.

– *Bit-line program disturbance — BPD.* Данная неисправность возникает, когда программирование ячейки-агрессора ведет к программированию другой незапрограммированной ячейки (ячейки-жертвы), присоединенной к той же линии бит.

– *Bit-line erase disturbance — BED.* Неисправность *BED* возникает, когда при программировании одной ячейки состояние ячейки-жертвы, присоединенной к той же линии бит, что и ячейка-агрессор, изменяется на логическую единицу.

– *Source-line program disturbance — SPD.* Данная неисправность возникает, когда программирование ячейки-агрессора ведет к программированию другой ячейки (ячейки-жертвы), присоединенной к той же линии питания.

Данные неисправности при всей своей схожести с описанными для традиционных ОЗУ неисправностями взаимного влияния (*coupling faults — CF*) все-таки имеют отличия от них. Так, если для неисправностей взаимного влияния ячейкой-агрессором и ячейкой-жертвой являются две определенные ячейки памяти, то в случае неисправностей, приведенных выше, активизировать неисправное состояние в ячейке-жертве может любая ячейка, которая находится на одной линии слов для *WPD* и *WED* и на одной линии бит для *BPD* и *BED* и к которой применяется операция программирования. Так, к примеру, алгоритм обнаружения неисправности ти-

па *WPD* имеет следующий вид: стирание всей памяти, программирование 1-ого столбца матрицы ячеек памяти, проверка состояния всех ячеек, не находящихся в 1-ом столбце, после чего снова стирание памяти и программирование любого столбца кроме 1-ого, затем чтение значения ячеек 1-ого столбца.

2. Неисправности стирания (*Erase Disturbances*).

– *Over Erase Disturbances — OED.* Когда возникает подобная неисправность, то ячейка не может изменить состояние с логической единицы на ноль, т.е. быть запрограммированной. Однако последовательное применение нескольких операций записи к данной ячейке может вернуть ее способность хранить заряд.

3. Неисправности чтения (*Read Disturbances*).

– *Soft-Program — SP*. Чтение содержимого ячейки может привести к тому, что на плавающий затвор транзистора данной ячейки могут попасть электроны, таким образом происходит частичное программирование ячейки.

– *Read Disturbances — RD* возникает при чтении из ячейки, когда после нескольких последовательных операций чтения ячейка не в состоянии хранить логический ноль.

– *Gate Read-Erase — GRE.* Неисправность *GRE* возникает, когда при чтении из ячейки в другой ячейке на той же линии слов происходит изменение состояния с "0" на "1" (стирание значения).

– *Channel Read-Program — CRP*. Данная неисправность *GRP* проявляется, когда при чтении из конкретной ячейки происходит программирование другой ячейки, находящейся на той же линии слов.

В [8] была предложена и описана такая модель неисправности для флэш-памяти, как *Source Line Interconnect Faults* (*SLIF*). *SLIF*-неисправность возникает из-за открытого соединения (*open connection*) на линии питания и является причиной того, что при стирании блока флэш-памяти не все ячейки будут установлены в состояние логической единицы, в то время как ячейки другого блока могут потерять хранимые значения.

Следует отметить, что возникновение данных неисправностей во флэш-памяти обусловлено ее архитектурой, типом транзисторов и другими факторами. Так, для *NOR*-архитектуры флэш-памяти рассматриваются в основном неисправности *WPD, WED, BPD, BED, OED, RD.*

Тестирование флэш-памяти

Показано [6–9], что традиционные маршевые тесты непригодны для тестирования флэш-памяти. В отличие от ОЗУ, во флэш-памяти присутствуют три вида операций: программирование, стирание и чтение ячейки. Большинство разновидностей подобных запоминающих устройств, как отмечалось выше, может осуществлять произвольный доступ к ячейкам памяти для чтения и записи "0", но стирание не может производиться для каждой ячейки — только для блока ячеек или всей памяти одновременно. Однако на основе обычных маршевых тестов разрабатываются и широко применяются маршево-подобные тесты для флэш-памяти [9–11].

К основным требованиям, предъявляемым к разрабатываемым тестам флэш-памяти, помимо высокой покрывающей способности и минимальной временной сложности, относят также минимальное число операций записи (программирования ячейки). Последнее требование обусловлено конечным числом возможных циклов записи в ячейку флэш-памяти, которое для современных типов памяти составляет примерно 100 тыс. Для достижения же минимальной временной сложности тестов следует рассмотреть время, затрачиваемое на программирование, стирание и чтение значения ячейки. Так, для обоих видов флэш-памяти (*NOR* и *NAND*) время чтения намного меньше, чем время программирования ячейки, которое в свою очередь меньше времени стирания. Время чтения составляет для *SLC NAND*- и *MLC NOR*-памяти соответственно 24 и 103 млн Б/с [12]. *NOR*-архитектура памяти характеризуется очень медленной операцией стирания значений ячеек (900 мс) и медленной операцией программирования (470 тыс. Б/с). В то время как для *NAND*-архитектуры эти значения являются более высокими (соответственно 2 мс и 8 млн Б/с) [12]. Следует также отметить, что данные значения могут зависеть от схемы обрамления флэш-памяти и времени ее эксплуатации. Так постепенно время, требуемое на выполнения описанных операций, увеличивается.

Одним из первых специально разработанных маршевых тестов для флэш-памяти был *Flash March* [9], который можно записать следующим образом: {(*f*); ↑(*r*1, *p*); ↕(*r*0); (*f*); ↓(*r*1, *p*); ↕(*r*0)}, где *f* (*flash*) обозначает операцию стирания значений в памяти, а *p* (*program*) — программирование ячейки памяти. Все остальные обозначения эквивалентны обозначениям для традиционных маршевых тестов ОЗУ. Маршевый тест состоит из конечного числа *маршевых элементов* [13–15]. В свою очередь, каждый маршевый элемент содержит символ, определяющий порядок формирования *адресной последовательности* ОЗУ: ↑ — определяет последовательный перебор адресов ОЗУ по возрастанию, ↓ — определяет последовательный перебор адресов по убыванию, ↕ — означает перебор по убыванию либо по возрастанию. Кроме этого, маршевый элемент содержит последовательность *операций чтения* и *записи*, заключенных в круглые скобки. Каждая операция представляет собой элемент из следующего набора: *r*0 — операция чтения содержимого ячейки с ожидаемым значением 0, *r*1 — операция чтения содержимого ячейки ОЗУ с ожидаемым значением 1, *w*0 — операция записи 0 в запоминающую ячейку, *w*1 — операция записи 1 в ячейку ОЗУ. В случае флэш-памяти операции *w*0 и *w*1 заменяются соответственно на операции *p* и *f*. 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 [15]. Сложность *C* теста *Flash March* можно оценить количеством циклов чтения *r*, программирования *p*, а также числом операций стирания *f* содержимого флэш-памяти. Основным аргументом функции *C* является емкость *N* флэш-памяти, тогда для случая *Flash March* теста получим, что $C(Flash\ March)=4Nr+2Np+2f(N)$.

Дальнейшим развитием маршевых тестов стало появление такого теста, как *March-FT* [10], представляющего собой усовершенствованный вариант *Flash March*: {(*f*); ↕(*r*1, *p*, *r*0); ↕(*r*0); (*f*); ↕(*r*1, *p*, *r*0); ↕(*r*0)}.

Данный тест имеет большую покрывающую способность в сравнении с тестом *Flash March*. Он позволяет покрыть большее количество неисправностей, так, в частности, данный тест 100%-но обнаруживает такие неисправности, как *SAF*, *SOF*, *TF*, *AF*, *CFst*, *WPD*, *WED*, *BPD*, *BED*, *RD*, *OE* (см. таблицу). При этом он требует по две операции стирания и программирования памяти (как и тест *March Flash*), но на две операции чтения больше по сравнению с тем же *March Flash*. Тогда его сложность будет оцениваться как $C(March\text{-}FT)=6Nr+2Np+2f(N)$.

Для уменьшения числа операций записи при тестировании памяти был предложен тест *Diagonal-FT* [11]: {(*f*); $\uparrow_{!D1}$(*r*1, *p*, *r*0); $\uparrow_{D1}$(*r*1, *p*, *r*0); $\uparrow_{!D1}$(*r*0); (*f*); $\downarrow_{D1}$(*r*1, *p*, *r*0); $\uparrow_{!D1}$(*r*1); $\downarrow_{D2}$(*r*1, *p*, *r*0); $\uparrow_{D1}$(*r*0); (*f*);}.

Обозначим, что ячейки на 1-й диагонали — это ячейки, которые находятся на диагонали, идущей из верхнего левого угла в правый нижний угол матрицы ячеек памяти, а ячейки на 2-й диагонали — это ячейки, которые находятся на диагонали из нижнего левого угла в верхний правый. Тогда $\uparrow_{!D1(2)}$ обозначает, что операции будут применяться ко всем ячейкам, кроме тех, что находятся на 1-й (2-й) диагонали, а $\uparrow_{D1(2)}$, наоборот, означает, что действия будет осуществляться только с ячейками на 1-й (2-й) диагонали. Более подробно данный тест описывается следующим образом.

Шаг 1. Стирание всей памяти (*f*), после чего все ячейки памяти будут содержать значение логической "1".

Шаг 2. Программирование всех ячеек, кроме тех, что находятся на 1-й диагонали. Данная операция активизирует *WPD* и *BPD* неисправности на 1-й диагонали. Далее операцией чтения (*r*0) выявляются неисправности *OE* для остальных ячеек.

Шаг 3. Считывая значение "1" из ячеек на 1-й диагонали, выявляются активизированные на шаге 2 неисправности типа *WPD* и *BPD*. Затем программируются ячейки на 1-й диагонали для активизации неисправности *WED* и *BED* всех остальных ячеек. Далее операцией чтения (*r*0) выявляются неисправности *OE* для ячеек на диагонали 1.

Шаг 4. Обнаруживаются неисправности *WED* и *BED*, активизированные на предыдущем шаге, чтением "0" из всех ячеек кроме тех, что находятся на 1-й диагонали.

Шаг 5. Стирание всей памяти (*f*).

Шаг 6. Программирование ячеек, находящихся на 1-й диагонали, что активизирует *WPD* и *BPD* неисправности для всех ячеек кроме программируемых. В то же время, используя

обратную последовательность адресов, обнаруживаются традиционные неисправности для ОЗУ.

Шаг 7. Считывая значение "1" из ячеек не на 1-й диагонали, выявляются активизированные на шаге 6 неисправности типа *WPD* и *BPD*.

Шаг 8. Для активизации *WED* и *BED* для ячеек на 1-й диагонали программируются ячейки на 2-й диагонали.

Шаг 9. Считывание "0" со всех ячеек на 1-й диагонали выявляет наличие в них неисправности типа *WED* и *BED*.

Сравнение количества неисправностей, обнаруживаемых тестами *Flash March, March-FT, Diagonal-FT*, %

Тип неисправности	WPD	WED	BPD	BED	OE	RD	SAF	TF	SOF	AF	CFst	Total
Flash March	100	100	100	100	100	100	100	100	100	100	100	100
March-FT	100	100	100	100	100	100	100	100	100	100	100	100
Diagonal-FT	100	100	100	100	100	100	100	100	100	81,6	89,15	97,34

Главное преимущество данного теста по сравнению с *March-FT* состоит в том, что у него число операций записи и чтения меньше соответственно на $(1N–2\sqrt{N})$ и $(2N–3\sqrt{N})$ операций. Однако число неисправностей, выявляемых описанным выше тестом, также меньше (на 2,66%) [11].

Для многократного использования теста *Diagonal-FT* можно использовать ее модификацию, *Diagonal-FT* с плавающей диагональю, при которой для каждого последующего прохода теста будет использоваться смещенная диагональ. Данная модификация позволит использовать ресурсы памяти более эффективно, и флэш-память будет "снашиваться" более равномерно.

Так при использовании такого модифицированного алгоритма вместо ячеек, расположенных на 1-й диагонали матрицы памяти, следует использовать ячейки, для которых выполняется равенство: $(i–l–1) \bmod p = j \bmod p$, $p = \min(n, m)$, где i и j — индексы соответственно столбца и строки ячейки в матрице ячеек памяти, а m и n — число столбцов и строк. Порядковый номер прохода теста при его многократном использовании обозначается l. А вместо ячеек 2-й диагонали стоит использовать ячейки, для которых $(i–l–1) \bmod p = (p–1–j) \bmod p$.

В [8] предложен свой тест, который позволяет 100% выявлять неисправности типа *SLIF*. Он состоит из двух независимых шагов:

Шаг 1. $\{(f); \uparrow(p); \uparrow_B\{(f); \uparrow(r_B 0)\}\}$

Шаг 2. $\{(f); \uparrow(r1, p); \updownarrow(r0); (f); \downarrow(r1, p); \downarrow(r0)\}$

На 1-м шаге применяется тест, позволяющий выявить 100 %-но неисправности типа *SLIF*, в то время как на шаге 2 применяется описанный выше тест *Flash March* для выявления оставшихся неисправностей. Так на первом шаге, после программирования всех ячеек, для каждого блока применятся операция стирания, после чего во всех остальных блоках проверяется состояние ячеек, которые должны быть равны логическому "0".

Заключение

Приведенный анализ состояния проблемы тестирования флэш-памяти показывает эффективность применения маршевых тестов для целей обнаружения специфических неисправностей, присущих только данному типу памяти. При этом проблема неисправности флэш-памяти является на настоящий момент мало изученной и требует дальнейших изысканий. Исследования, связанные с данной проблемой, могут быть направлены на разработку новых, более адекватных моделей неисправностей и построение более эффективных тестов для их обнаружения.

FLASH MEMORY TESTING METHODS ANALYSIS

S.V. YARMOLIK

Abstract

The problems of design and reliability of modern flash-memory are discussed in this paper. The architectures of two types of flash-memory namely the NOR- and NAND-architectures are described and the specific flash-memory faults and tests for their detection are discussed. The march-like tests, which provide the high cover possibility, are shown to be effective tests for detection specific flash-memory faults.

Литература

1. *Pavan P., Bez R., Olivo P., Zanoni E.* // Proc. IEEE. 1997. Vol. 85, № 8. P. 1248–1271.
2. *Brown W., Brewer J.* Nonvolatile Semiconductor Memory Technology: A comprehensive Guide to Understanding and Using NVSM Devices. IEEE Press, 1998.
3. *Inoue A., Wong D.* NAND Flash Applications Design Guide. Toshiba Press. 2003.
4. *Sharma A.-K.* Semiconductor Memories: Technology, Testing and Reliability. IEEE Press, 1997.
5. *Wu C.-F., Huang C.-T., Cheng K.-L., Wu C.-W.* // IEEE Transition Computer-Aided Design Integrated Circuits Systems. 2002. Vol. 21, № 4. P. 480–490
6. IEEE 1005 Standard Definitions and Characterization of Floating Gate Semiconductor Arrays. IEEE Standards Department. Piscataway, 1999.
7. *Cheng K.L., Yeh J.C., Wang C.W., Huang C.T., Wu C.W.* // Proc. of the 20th IEEE VLSI Test Symposium. 2002. P. 281–286
8. *Mohammad M.G., Saluja K.K., Yap A.* // Proc. 13th Int. Conf. VLSI Design. 2000. P. 406–411.
9. *Mohammad M.G., Saluja K.K.* // Proc. of 19th VLSI Test Symposium. 2001. P. 218–224.
10. *Yeh J.C., Wu C. F., Cheng K.L. et al.* // Proc. of 1st International Workshop on Electronic Design, Test and Applications. 2002. P. 137–141.
11. *Chiu S.K., Yeh J.C., Huang C.T., Wu C.W.* // Proc. of International Test Conference. 2002. P. 37–46.
12. NAND vs. NOR Flash Memory Technology Overview. Toshiba America Electronic Components, Inc. 2006.
13. *Goor A.J.* Testing Semiconductor Memories, Theory and Practice. UK, Chichester, John Wiley & Sons, 1991.
14. *Goor A.J.* // IEEE Design and Test of Computer. 1993. Vol. 10, № 1. P. 8–14.
15. *Goor A.J., Al–Ars. Z.* // Proc. 18th IEEE VLSI Test Symposium (VTS'00). 2000. P. 281–289.