УДК 681.326.7

МЕТОДЫ ОПТИМИЗАЦИИ МИКРОКОДА ВСТРОЕННОЙ АППАРАТУРЫ САМОТЕСТИРОВАНИЯ ОЗУ

А.А. ИВАНЮК, А.А. АВТУШКО

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

Поступила в редакцию 7 сентября 2009

Рассматриваются способы проектирования встроенной программируемой аппаратуры самотестирования оперативных запоминающих устройств, основанной на разрушающих маршевых тестах, а также методы оптимизации микрокода, кодирующего маршевые тесты, с целью его минимизации для сокращения аппаратурных затрат на память микропрограмм и увеличения скорости передачи тестов по последовательным интерфейсам.

Ключевые слова: оперативное запоминающее устройство, маршевый тест, встроенное самотестирование, конечный цифровой автомат, микропрограммное управление.

Введение

Одной из основных проблем современной цифровой электроники является проектирование надежных вычислительных систем. Это особенно актуально для оперативных запоминающих устройств, которые широко применяются в различных цифровых системах для хранения данных и управляющий инструкций.

С увеличением сложности цифровых устройств наиболее актуальным и распространенным способом тестирования стала встроенная аппаратура самотестирования (BIST — Built-In Self-Test). Ядро BIST передает тестовые последовательности тестируемому устройству, считывает результаты и сравнивает их с ожидаемыми, и таким образом определяет, исправно ли тестируемое устройство.

Существуют различные подходы к проектированию BIST. BIST может реализовывать один определенный алгоритм тестирования. В таком случае аппаратные затраты на BIST минимальны и скорость применения теста является максимальной, но данный поход является наименее гибким, так как он определяет единственный алгоритм тестирования, способный обнаруживать фиксированный набор неисправностей. Изменение алгоритма тестирования невозможно без необходимости перепроектирования BIST. Альтернативным подходом является BIST с микропрограммным управлением P-MBIST (Programmable Memory BIST), в которой алгоритм тестирования определяется микрокодом, хранящимся в памяти микропрограмм. P-MBIST позволяет изменять алгоритм тестирования во время жизненного цикла тестируемого устройства.

Для тестирования оперативных запоминающих устройств существует множество различных тестов, способных обнаруживать определенные классы ошибок. Примерами таких тестов являются "Шахматная доска" (checkerboard), "Бегущая 1/0" (walking 1/0), "Галоп" (Galloping 1/0). Но данные традиционные тесты имеют большую сложность, что ограничивает их применение в современной цифровых системах. В настоящее время наиболее применимыми являются тесты сложности $O(n)$, где n — емкость ОЗУ, так называемые маршевые тесты.

В данной работе рассматриваются способы минимизации микрокода, кодирующего произвольные маршевые тесты, с целью их хранения в памяти микропрограмм P-MBIST. Оптимальный способ представления алгоритмических тестов определяет архитектуру встроенной

аппаратуры самотестирования ОЗУ, аппаратные затраты и скорость передачи тестов по последовательным интерфейсам.

Минимизация микрокода маршевых тестов P-MBIST

Маршевый тест представляет собой совокупность фаз, состоящих из последовательно выполняемых операций чтения/записи над текущим запоминающим элементом памяти. Каждая фаза выполняется для всего адресного пространства, причем направление обхода адресного пространства (от старших адресов к младшим или наоборот) также определяется тестом. Например, маршевый тест, описанный в нотации MTL [1] как $\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)$, состоит из трех фаз:

1) запись 0 во все элементы памяти, причем направление обхода адресного пространства не имеет значения;

2) для каждого запоминающего элемента, начиная с первого и заканчивая последним, прочитать хранимое значение (оно должно быть 0) и записать 1;

3) для каждого элемента памяти, начиная с последнего и заканчивая первым, прочитать значение, которое должно быть равным 1, и записать 0.

Наиболее популярные маршевые тесты на сегодняшний день представлены в табл. 1.

Таблица 1. **Наиболее распространенные маршевые тесты**

Название	MTL описание
MATS	$\Updownarrow(w0);\Uparrow(r0,w1);\Updownarrow(r1)$
MATS+	$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)$
MATS++	$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)$
Marching 1/0	$\Updownarrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1;w0;r0);\Uparrow(w1);\Uparrow(r1,w0,r0);\Downarrow(r0,w1,r1)$
March X	$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$
March Y	$\Updownarrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1,w0,r0);\Updownarrow(r0)$
March C	$\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Updownarrow(r0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$
March C-	$\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$
March A	$\Updownarrow(w0);\Uparrow(r0,w1,w0,w1);\Uparrow(r1,w0,w1);\Downarrow(r1,w0,w1,w0);\Downarrow(r0,w1,w0)$
March B	$\Updownarrow(w0);\Uparrow(r0,w1,r1,w0,r0,w1);\Uparrow(r1,w0,w1);\Downarrow(r1,w0,w1,w0);\Downarrow(r0,w1,w0)$
Algorithm B	$\Updownarrow(w0);\Uparrow(r0,w1,w0,w1);\Uparrow(r1,w0,r0,w1);\Downarrow(r1,w0,w1,w0);\Downarrow(r0,w1,r1,w0)$

В простейшем случае для кодирования маршевого теста с целью его хранения в памяти P-MBIST необходимо кодировать количество фаз теста, количество базовых операций в фазе теста или маркер окончания фазы теста, тип базовой операции (чтение или запись) и направление обхода адресного пространства. Данный подход описан в работе [2]. Пример кодирования маршевого теста MATS+ представлен на рис. 1.

AO	RW	D	EE	AO	RW	D	EE	RW	D	EE	AO	RW	D	EE	RW	D	EE
1	1	0	1	1	0	0	0	1	1	1	0	0	1	0	1	0	1

Рис. 1. Пример кодирования теста MATS+: AO — Address Order, направление обхода адресного пространства; RW — Read/Write, базовая операция чтения или записи; D — Data, бит данных; EE — End of Element, маркер конца фазы теста

При таком способе кодирования маршевого теста размер микрокода зависит от количества фаз теста и количества базовых операций. Для кодирования любого маршевого теста потребуется V бит.

$$V = (3n_o + n_e), \quad (1)$$

где n_0 — количество базовых операций; $n_в$ — количество фаз теста (здесь и далее не учитываются затраты памяти для хранения количества фаз теста, которое необходимо для определения момента завершения выполнения теста памяти).

Анализ существующего множества маршевых тестов показывает, что данные, используемые в базовых операциях чтения/записи изменяются закономерно, в зависимости от типа базовой операции [3]:

$$(w \to r) \Rightarrow (dt \to dt), \tag{2}$$

$$(r \to w) \Rightarrow (dt \to \overline{dt}), \tag{3}$$

$$(r \to r) \Rightarrow (dt \to dt), \tag{4}$$

$$(w \to w) \Rightarrow (dt \to \overline{dt}). \tag{5}$$

Это значит, что бит данных, используемый в базовой операции записи, представляет собой инверсию бита данных, используемого в предыдущей базовой операции, независимо от ее типа и фазы теста, в которой базовая операция находится. Эта закономерность позволяет исключить данные из микрокода и реализовать формирование данных для маршевого теста в самой аппаратуре P-MBIST. Пример кодирования маршевого теста MATS+ указанным способом представлен на рис. 2.

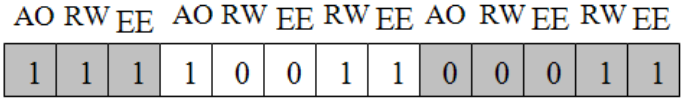

Рис. 2. Пример кодирования теста MATS+

Для кодирования любого маршевого теста данным способом необходимо V' бит.

$$V' = (2n_o + n_e). \tag{6}$$

Дальнейший анализ существующих маршевых тестов показывает, что маршевый тест можно представить как совокупность предопределенных компонентов, представляющих собой целую фазу теста [4]. В маршевых тестах из табл. 1 можно выделить следующие компоненты: $SM_0 = (wd)$, $SM_1 = (rd, w\overline{d})$, $SM_2 = (rd, w\overline{d}, r\overline{d}, wd)$, $SM_3 = (rd, w\overline{d}, wd)$, $SM_4 = (rd, w\overline{d}, r\overline{d}, wd, rd, w\overline{d})$, $SM_5 = (rd)$, $SM_6 = (rd, w\overline{d}, wd, w\overline{d})$, $SM_7 = (rd, w\overline{d}, r\overline{d})$. В табл. 2 описаны те же тесты, построенные на основе предопределенных компонентов.

Таблица 2. **Маршевые тесты, построенные на основе предопределенных компонентов**

Название	Описание
MATS	$\Updownarrow SM_0; \Uparrow SM_1; \Updownarrow SM_5$
MATS+	$\Updownarrow SM_0; \Uparrow SM_1; \Downarrow SM_1$
MATS++	$\Updownarrow SM_0; \Uparrow SM_1; \Downarrow SM_7$
Marching 1/0	$\Updownarrow SM_0; \Uparrow SM_7; \Downarrow SM_7; \Uparrow SM_0; \Uparrow SM_7; \Downarrow SM_7$
March X	$\Updownarrow SM_0; \Uparrow SM_1; \Downarrow SM_1; \Updownarrow SM_5$
March Y	$\Updownarrow SM_0; \Uparrow SM_7; \Downarrow SM_7; \Updownarrow SM_5$
March C	$\Updownarrow SM_0; \Uparrow SM_1; \Uparrow SM_1; \Updownarrow SM_5; \Downarrow SM_1; \Downarrow SM_1; \Updownarrow SM_5$
March C-	$\Updownarrow SM_0; \Uparrow SM_1; \Uparrow SM_1; \Downarrow SM_1; \Downarrow SM_1; \Updownarrow SM_5$
March A	$\Updownarrow SM_0; \Uparrow SM_6; \Uparrow SM_3; \Downarrow SM_6; \Downarrow SM_3$
March B	$\Updownarrow SM_0; \Uparrow SM_4; \Uparrow SM_3; \Downarrow SM_6; \Downarrow SM_3$
Algorithm B	$\Updownarrow SM_0; \Uparrow SM_6; \Uparrow SM_2; \Downarrow SM_6; \Downarrow SM_2$

Для минимизации микрокода маршевого теста могут быть использованы данные предопределенные компоненты, реализованные аппаратно. Для кодирования маршевого теста необходимы следующие данные:

1) направление обхода AO адресного пространства, кодируемое одним битом;

2) номер компонента CN, кодируемый $\log_2 8=3$ битами, при этом каждый из предопределенных компонентов должен быть реализован в аппаратуре.

Данные из микрокода могут быть исключены, как и описано в [3]. Тест MATS+, кодируемый указанным способом, представлен на рис. 3.

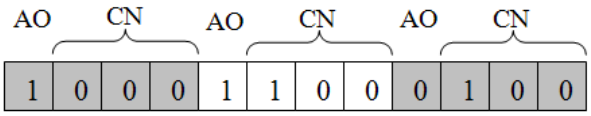

Рис. 3. Пример кодирования теста MATS+

Для кодирования любого маршевого теста данным способом потребуется V'' бит. При таком способе кодирования количество используемых компонентов всегда равняется количеству фаз маршевого теста.

$$V'' = (4n_e). \qquad (7)$$

Дадим новое понятие маршевому тесту, внеся в него свойство симметричности [5]: маршевый тест — это совокупность простых и составных примитивов, причем для каждого примитива обход адресного пространства осуществляется в направлении, противоположном предыдущему. Под простым примитивом будем понимать одну фазу теста. Под составным примитивом будем понимать примитив, состоящий из нескольких простых, для каждого из которых обход адресного пространства осуществляется в одном и том же направлении.

В рассмотренных выше маршевых тестах первым примитивом всегда является инициализирующий $\Updownarrow(w0)$, поэтому данный примитив может быть исключен из микрокода, но должен быть реализован в аппаратуре, и с его выполнения должен начинаться процесс самотестирования ОЗУ.

Исходя из приведенного определения, направления обхода ОЗУ являются предопределенными, и могут быть исключены из микрокода. Основная задача при проектировании P-MBIST — это определение примитивов. Сгруппируем примитивы так, чтобы направления обхода памяти изменялись закономерно, что позволит убрать AO из микрокода, таким образом упростив аппаратную реализацию устройства управления. Полученные результаты представлены в табл. 3.

Таблица 3. **Выделение составных примитивов из маршевых тестов**

Название	Описание
MATS	$\Uparrow SM_1; \Downarrow SM_5$
MATS+	$\Uparrow SM_1; \Downarrow SM_1$
MATS++	$\Uparrow SM_1; \Downarrow SM_7$
Marching 1/0	$\Uparrow SM_7; \Downarrow SM_7; \Uparrow (SM_0, SM_7); \Downarrow SM_7$
March X	$\Uparrow SM_1; \Downarrow (SM_1, SM_5)$
March Y	$\Uparrow SM_7; \Downarrow (SM_7, SM_5)$
March C	$\Uparrow (SM_1, SM_1, SM_5); \Downarrow (SM_1, SM_1, SM_5)$
March C-	$\Uparrow (SM_1, SM_1); \Downarrow (SM_1, SM_1, SM_5)$
March A	$\Uparrow (SM_6, SM_3); \Downarrow (SM_6, SM_3)$
March B	$\Uparrow (SM_4, SM_3); \Downarrow (SM_6, SM_3)$
Algorithm B	$\Uparrow (SM_6, SM_2) \Downarrow (SM_6, SM_2)$

Выделим следующие составные примитивы, основываясь на группировке фаз, показанной в табл. 3: $SM_8 = (SM_1, SM_5)$, $SM_9 = (SM_1, SM_1, SM_5)$, $SM_{10} = (SM_6, SM_3)$, $SM_{11} = (SM_6, SM_2)$, $SM_{12} = (SM_4, SM_3)$, $SM_{13} = (SM_7, SM_5)$, $SM_{14} = (SM_0, SM_7)$, $SM_{15} = (SM_1, SM_1)$. Всего выделено 16 примитивов, для кодирования каждого из которых потребуется $\log_2 16 = 4$ бита.

Таблица 4. **Маршевые тесты, построенные на основе простых и составных примитивов и размер микрокода, необходимого для их кодирования**

Название	Описание	Размер микрокода, бит
MATS	$SM_1; SM_5$	8
MATS+	$SM_1; SM_1$	8
MATS++	$SM_1; SM_7$	8
Marching 1/0	$SM_7; SM_7; SM_{14}; SM_7$	16
March X	$SM_1; SM_8$	8
March Y	$SM_7; SM_{13}$	8
March C	$SM_9; SM_9$	8
March C-	$SM_{15}; SM_9$	8
March A	$SM_{10}; SM_{10}$	8
March B	$SM_{12}; SM_{10}$	8
Algorithm B	$SM_{11}; SM_{11}$	8

Так как способ изменения данных и направления обхода в маршевом тесте предопределены, микрокод представляет собой множество слов фиксированной длины, каждое из которых декодируется в предопределенный простой или составной примитив. Простые примитивы кодируются числами от 0 до 7, а составные — от 8 до 15. Таким образом, для определения типа примитива в устройстве управления P-MBIST необходимо проверить старший бит инструкции. Тест MATS+, кодируемый данным способом, представлен на рис. 4, где PN — номер примитива.

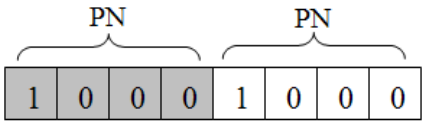

Рис. 4. Пример кодирования теста MATS+

Для кодирования любого маршевого теста данным способом потребуется V''' бит.

$$V''' = (4n_p), \quad (8)$$

где n_p — количество простых и составных примитивов, из которых состоит тест.

Предложенный способ кодирования маршевых тестов позволяет сократить аппаратурные затраты на память микрокода и увеличить скорость передачи тестов по последовательным интерфейсам благодаря уменьшенному объему микрокода. Для кодирования любого из рассмотренных выше маршевых тестов достаточно 16 бит, что на 71% меньше чем необходимо для метода, описанного в [2] и на 59% меньше, чем для [3].

P-MBIST с предложенным способом кодирования маршевых тестов была спроектирована на языке VHDL и синтезирована с использованием программного автоматизированного средства проектирования цифровой аппаратуры Xilinx ISE WebPack 9.2i [6]. Диаграмма состояний спроектированной P-MBIST изображена на рис. 5.

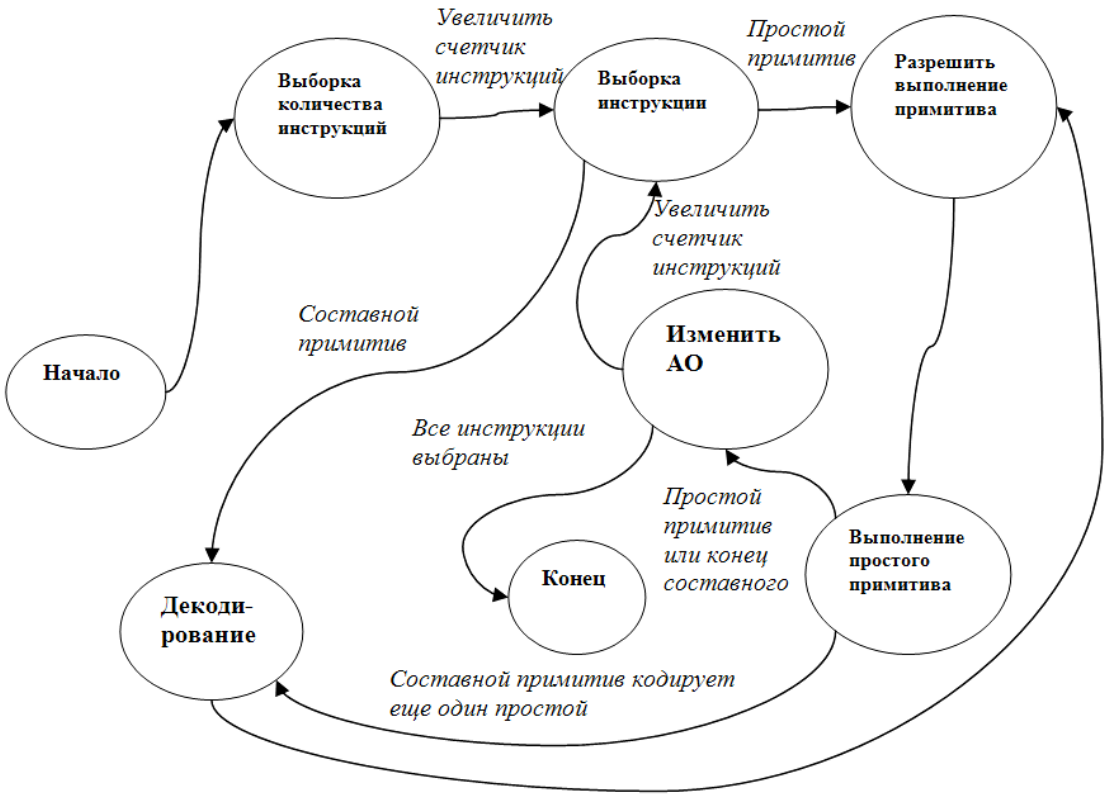

Рис. 5. Диаграмма состояний P-MBIST с оптимальным способом кодирования тестов

На диаграмме, изображенной на рис. 6, показаны отношения аппаратурных затрат на реализацию P-MBIST для каждого из рассмотренных способов кодирования тестов (для оценки аппаратурных затрат использовалось значение *Total equivalent gate count for design*, генерируемое в результате синтеза проекта средством Xilinx ISE WebPack 9.2i). P-MBIST с микрокодом, состоящим из простых и составных примитивов, сравним по аппаратурным затратам с остальными методами благодаря более простому устройству управления, что позволяет скомпенсировать издержки, вносимые необходимостью реализовывать примитивы аппаратно.

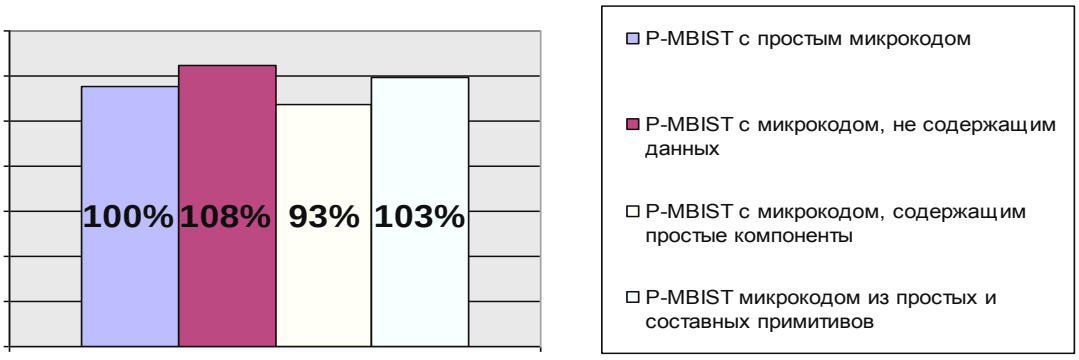

Рис. 6. Отношения аппаратурных затрат для P-MBIST с различными способами кодирования тестов

Заключение

Предложен способ кодирования существующих маршевых тестов ОЗУ с целью минимизации аппаратурных затрат на памяти микропрограмм P-MBIST, необходимой для их хранения. Предложенный способ является промежуточным между аппаратным (с предопределенными тестами) и программным (позволяющим кодировать произвольные тесты), как по скорости функционирования, так и по множеству поддерживаемых тестов и аппаратным затратам.

Данный способ позволяет кодировать предопределенный набор маршевых тестов, а также произвольные маршевые тесты, строящиеся на основе того же набора примитивов, что и базовые тесты.

Возможным улучшением аппаратуры, реализующей предложенный способ кодирования, с точки зрения поддержки будущих маршевых тестов является возможность программирования набора составных примитивов.

METHODS OF MICROCODE OPTIMIZATION FOR EMBEDDED MEMORY BUILD-IN SELF-TEST

A.A. IVANIUK, A.A. AVTUSHKO

Abstract

Methods of designing programmable build-in self-test architectures for embedded memories based on March tests are described. A new method of March test coding for programmable memory build-in self-test architectures allowing reducing hardware overheads and speed-up tests transferring over serial interfaces is offered.

Литература

1. *Goor A., Offerman A., Schanstra H.* // European Design and Test Conference (EDTC'96): Proc. of IEEE Int. Conf. Paris, France, 11–14 March 1996. Washington, DC, USA. 1996. P. 420–427.
2. *McEvoy P., Farrell R.* // Proc. of IEEE IC Test Workshop, Limerick, Ireland. 13–14 Sept., 2004, P. 15–21.
3. *Иванюк А.А., Ярмолик В.Н.* // Автоматика и вычислительная техника. 2008. № 4. С. 5–13.
4. *Zarrineh K. Upadhyaya S.J.* // Design, Automation and Test in Europe (DATE'99): proc. of IEEE Int. Conf. Munich, Germany. Mar. 9–12, 1999. P. 709–713.
5. *Hellebrand S., Wundelich Y.-J., Yarmolik V.N.* // IEEE Design, Automation and Test in Europe Conference (DATE'99): Proc. Int. Conf. Munich, Germany, 9–12 March 1999. P. 702–707.
6. ISE WebPACK 9.2i [Electronic resource]. 2008. Mode of access: http://www.xilinx.com.