УДК 004.072

Алгоритмы тестирования памяти при проведении радиационных испытаний микропроцессорного модуля

Д.А. Трубицын[1], П.А. Чибисов[1], С.В. Баранов[2]

[1] Учреждение Российской академии наук Научно-исследовательский институт системных исследований РАН

[2]ОАО «ЭНПО СПЭЛС», svbar@spels.ru

Аннотация — **В статье предложены алгоритмы тестирования различных типов памяти в условиях неблагоприятных внешних факторов.**

Ключевые слова — **Тестирование, память.**

I. Введение

Для микросхем, использующихся в неблагоприятных условиях внешней среды, одним из ключевых требований является обеспечение надлежащей радиационной стойкости. Практика оценки параметров радиационной стойкости микропроцессорного модуля, основанная на поэлементных испытаниях каждой БИС, имеет известные недостатки, наиболее существенным из которых является сложность последующей обоснованной оценки стойкости электронного модуля в целом по результатам испытаний отдельных типов микросхем [1]. Микропроцессорный модуль содержит большое количество компонент. Результаты проведенных к настоящему времени расчетно-экспериментальных исследований показали, что наиболее чувствительными компонентами к воздействию ядерных частиц являются микросхемы ОЗУ, а также кэш-память микропроцессора [2]. На основе предлагаемых в данной работе алгоритмов проектируется тестовое программное обеспечение, запускаемое на исследуемом микропроцессорном модуле и позволяющее получить наиболее объективную картину поведения микропроцессора и его подсистемы памяти (кэш-память первого и второго уровня, а также наплатное ОЗУ) под воздействием радиационного излучения [3].

II. Алгоритм теста памяти

Ниже представлен общий алгоритм проверки памяти вне зависимости от ее типа:

```
печать «начало теста»;
выбор политики кэширования;
for (k=1;k≤P;k++)
{
  for (i=1;i≤N;i++)
   {
    заполнение памяти прямым кодом;
    for(addr=addr_1;addr≤addr_last;addr++)
     {
     чтение прямого кода;
     запись и чтение инверсного кода;
     запись и чтение прямого кода;
     формирование результата;
     if(data≠data_эталон)
       побитовое сравнение и печать;
     }
   }
  for (i=1;i≤M;i++)
   {
    заполнение памяти инверсным кодом;
    for(addr=addr_1;addr≤addr_last;addr++)
     {
     чтение инверсного кода;
     запись и чтение прямого кода;
     запись и чтение инверсного кода;
     формирование результата;
     if(data≠data_эталон)
       побитовое сравнение и печать;
     }
   }
}
```

Алгоритм теста является циклическим: сначала память прописывается прямым «шахматным» кодом (5555_{16}), проверяется содержимое памяти заданное количество раз (N), затем память прописывается инверсным «шахматным» кодом ($AAAA_{16}$) и ее содержимое проверяется заданное количество раз (M), после чего вышеперечисленные действия повторяются определенное количество раз (P).

Несмотря на то, что для тестирования микросхем ОЗУ преимущественно используется псевдослучайный код, для тестирования микропроцессорного модуля в целом был выбран именно «шахматный» код по следующим причинам. Во-первых, «шахматный» код не требует специального формирования и памяти для его хранения, во-вторых, этот код упрощает получение результата по мажоритарному принципу.

Особенности предлагаемых алгоритмов тестов:

1) Раздельное тестирование кэш-памяти данных первого уровня (L1), кэш-памяти данных второго уровня (L2) и наплатного ОЗУ.
2) Полностью автономная работа теста, не требующая вмешательства оператора.
3) Тесты могут быть написаны на языке ассемблера.
4) Вывод подробной информации о различных типах возникающих сбоев.
5) Мажоритарный принцип формирования результата чтения ячейки памяти.
6) Проверка контроллеров кэш-памяти и ОЗУ на сбой.
7) Применение сторожевого таймера для контроля работоспособности процессора и предотвращений «зависания» микропроцессора.
8) Возможность работы из загрузочной флэш-памяти.

III. Сторожевой таймер

При проведении радиационных испытаний тестируемая система может повести себя непредсказуемо, в частности «зависнуть». Для исключения этой ситуации в тесте памяти был применен сторожевой таймер, входящий в состав микропроцессора. Механизм его работы следующий: в начале каждого цикла проверки содержимого памяти происходит инициализация таймера, в результате которой задается значение таймаута, заведомо большее ожидаемой длительности одного цикла теста. В случае «зависания» системы по окончании заданного интервала времени таймер выставляет немаскируемое прерывание, по возникновению которого происходит перезагрузка микропроцессора.

IV. Виды отказов ячейки памяти

При радиационных испытаниях возможны различные ситуации отказов работы ячейки памяти: одиночный сбой, «залипание» бита. Одиночный сбой – это ситуация, когда ячейка памяти содержит ошибочное значение, и эта ошибка может быть устранена путем записи верного значения. В отличие от одиночного сбоя «залипание» бита памяти – это ситуация, когда значение бита памяти невозможно поменять командой записи микропроцессора. Такой тип сбоя устраним отключением питания микросхемы ОЗУ.

Чтобы корректно отслеживать и различать виды отказов работы ячейки памяти, вместо однократного чтения ячейки памяти, была предложена следующая последовательность чтения данных из ОЗУ (либо кэш-памяти):

```
первое чтение по текущему адресу;
очистка буферов;
второе чтение по текущему адресу;
очистка буферов;
третье чтение по текущему адресу;
проверка контроллера на сбой;
вычисление результата чтений 1;

запись инверсного кода;
очистка буферов;
первое чтение по текущему адресу;
очистка буферов;
второе чтение по текущему адресу;
очистка буферов;
третье чтение по текущему адресу;
проверка контроллера на сбой;
вычисление результата чтений 2;

запись прямого кода;
очистка буферов;
первое чтение по текущему адресу;
очистка буферов;
второе чтение по текущему адресу;
очистка буферов;
третье чтение по текущему адресу;
проверка контроллера на сбой;
вычисление результата чтений 3;
```

Сбой контроллера памяти – это ситуация, когда результаты трех последовательных чтений не совпадают.

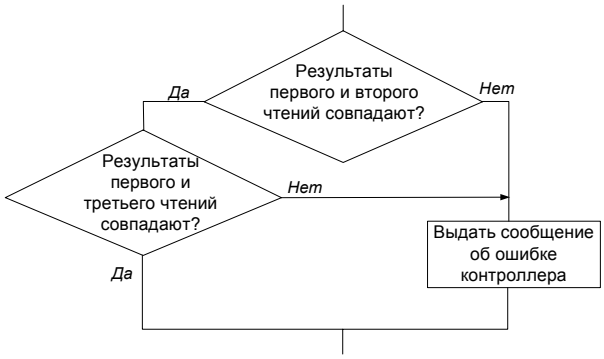

Рис. 1 Сбой контроллера памяти

Конечный результат чтения из памяти для данной ячейки формируется по мажоритарному принципу из результатов трех последовательных считываний из ОЗУ по одному адресу. Суть мажоритарного, или принципа большинства, представлена таблице 1.

Таблица 1

Формирование результата по мажоритарному принципу

Чтение 1	1	1	1	1	0	0	0	0
Чтение 2	1	1	0	0	1	1	0	0
Чтение 3	1	0	1	0	1	0	1	0
Результат	1	1	1	0	1	0	0	0

V. Алгоритмы тестирования кэш-памяти

Суть предлагаемого метода тестирования кэш-памяти состоит в записи данных в кэш-память посредством записи тестовых значений в ОЗУ. При тестировании кэш-памяти необходимо выбрать подходящую политику кэширования и произвести инициализацию кэш-памяти. Для тестирования L1(D) кэш-памяти выбирается неблокирующая, некогерентная сквозная запись с локализацией в обход кэш-памяти второго уровня. Для тестирования L2 кэш-памяти выбирается неблокирующая некогерентная сквозная запись с локализацией с обратной записью в кэш-память второго уровня [4],[5]. Предварительная подготовка к тестированию различных уровней кэш-памяти также принципиально отличается.

Для инициализации L1(D) кэш-памяти требуется выполнить следующую последовательность операций:

```
инвалидизация L1(D) кэш-памяти;
заполнение кэшируемой памяти прямым кодом;
заполнение некэшируемой памяти нулями;
```

В результате инициализации получаем кэш-память, заполненную прямым кодом, подготовленную для дальнейшего тестирования, и ОЗУ, заполненное кодом «0» для того, чтобы в случае возникшего сбоя была возможность обнаружить факт загрузки данных из ОЗУ в кэш-память.

Для инициализации L2 кэш-памяти требуется выполнить аналогичную последовательность действий, в результате выполнения которой получаем требуемые данные в L2 кэш-памяти, а в L1 кэш-памяти и ОЗУ – нулевые данные:

```
инвалидизация L1(D) кэш-памяти;
инвалидизация L2 кэш-памяти;
заполнение кэшируемой памяти прямым кодом
(подготовка данных);
инвалидизация L1(D) кэш-памяти;
заполнение некэшируемой памяти нулями;
```

После проведенной инициализации при тестировании L1-кэша предлагается использовать последовательность действий, представленную ниже:

```
первое чтение данных из памяти;
второе чтение данных из памяти;
третье чтение данных из памяти;
проверка контроллера на сбой;
вычисление результата чтений 1;

заполнение кэшируемой памяти инверсным кодом
(подготовка данных);
заполнение некэшируемой памяти нулями;
первое чтение данных из памяти;
второе чтение данных из памяти;
третье чтение данных из памяти;
проверка контроллера на сбой;
вычисление результата чтений 2;

заполнение кэшируемой памяти прямым кодом
(подготовка данных);
заполнение некэшируемой памяти нулями;
первое чтение данных из памяти;
второе чтение данных из памяти;
третье чтение данных из памяти;
проверка контроллера на сбой;
вычисление результата чтений 3;
```

При тестировании L2-кэша последовательность действий принимает следующий вид:

```
инвалидизация L1(D) кэш-памяти, L2 кэш-памяти;
заполнение кэшируемой памяти прямым кодом
(подготовка данных);
заполнение некэшируемой памяти нулями;
первое чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
второе чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
третье чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
проверка контроллера на сбой;
вычисление результата чтений 1;
заполнение кэшируемой памяти инверсным кодом
(подготовка данных);
инвалидизация L1(D) кэш-памяти;
заполнение некэшируемой памяти нулями;
первое чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
второе чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
третье чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
проверка контроллера на сбой;
вычисление результата чтений 2;
заполнение кэшируемой памяти прямым кодом
(подготовка данных);
инвалидизация L1(D) кэш-памяти;
заполнение некэшируемой памяти нулями;
первое чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
второе чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
третье чтение данных из памяти;
инвалидизация L1(D) кэш-памяти;
проверка контроллера на сбой;
вычисление результата чтений 3;
```

Следует отметить, что описанные выше алгоритмы представляется целесообразным реализовать на языке ассемблера, так как только в этом случае программист может точно определить порядок выполнения требуемых инструкций микропроцессора, а также избежать использования стека, располагающегося в оперативной памяти. Кроме того, тестовая программа должна компилироваться и выполняться из некэшируемого загрузочного ПЗУ, что позволит исключить участок памяти, требуемый для расположения кода программы, а также кэш-память инструкций.

VI. Анализ результатов тестирования

Дальнейший анализ результатов чтения прямого «шахматного» кода и инвертированного результата чтения обратного «шахматного» кода производится

поразрядно. Различные варианты получения итогового результата чтения одного двоичного разряда памяти приведены в таблице 2 для случая, когда в данный разряд записана единица (для нуля аналогично).

Таблица 2

Варианты чтения разряда памяти

№ ситуации	1	2	3	4	5	6	7	8
Эталон	1	1	1	1	1	1	1	1
Первое чтение	0	0	0	0	1	1	1	1
Второе чтение	0	0	1	1	0	0	1	1
Третье чтение	0	1	0	1	0	1	0	1
Результат	X	X	0	1	0	1	1	1
Символ	-	-	)	@	(	$	@	.

Эталон – корректное (ожидаемое) значение разряда.

В таблице используются специальные символы («.», «@», «$», «(», «)», «-») для обозначения различных видов сбоев. Эти символы были выбраны исходя из их заметности в большом массиве текстовых данных, а также из-за удобства их использования в программах последующей обработки данных, получаемых в ходе работы теста.

Расшифровка специальных символов приведена ниже:

$ – "залипший" разряд, "залипло" эталонное значение;
) – "залипший" разряд, "залипло" инвертированное значение эталона;
(– "залипший" разряд во время проверки, "залипло" инвертированное значение эталона;
@ – одиночный сбой;
- – функциональный сбой (два последовательных одиночных сбоя при чтении прямого и обратного «шахматного» кодов);
X – значение не определено;
. – чтение без ошибок.

В случае обнаружения сбоев в памяти на экран выводится строка, показывающая, в каких двоичных разрядах произошли ошибки, например:

```
# '5' check cycle 2 address: ffffffff80100000
.......@.........$.....................@.....
```

В строке указываются: специальный символ для контроля времени («#»), тип кода, номер цикла, адрес ячейки памяти, в которой произошли ошибки.

В случае отсутствия сбоев во время выполнения теста на экран периодически выводится строка, содержащая символ для контроля времени для индикации корректного функционирования системы.

Во время испытаний все выводимые тестом данные сохраняются в файл, а потом на их основе с помощью программ, написанных на языке, поддерживающем регулярные выражения, осуществляется детальный анализ радиационной стойкости исследуемой памяти.

VII. Заключение

Предложенные в статье алгоритмы были применены в экспериментальных исследованиях радиационной стойкости микросхем микропроцессора и динамической памяти в составе микропроцессорного модуля. Наблюдалась ситуация, когда в результате излучения сначала возникали единичные сбои в памяти, которые по мере накопления радиационной дозы, превращались в «залипшие» разряды.

Выходные данные тестовой программы, полученные в ходе испытаний микросхем ОЗУ и микропроцессора на радиационную стойкость, были использованы для подсчета количества и характера сбоев при заданных интенсивности облучения и времени экспозиции.

Результаты экспериментальных исследований по оценке радиационной стойкости микропроцессорного модуля при воздействии низкоинтенсивного потока нейтронов (облучение 14 МэВ нейтронами) таковы: в исследуемых СБИС имели место эффекты одиночных сбоев ($N_{ос} \approx 1000$), “залипших” разрядов ($N_{зр} \approx 10$) и функциональные сбои ($N_{зб} \approx 10$).

Предложенные алгоритмы также могут быть применены для климатических испытаний систем памяти в составе микропроцессорного модуля.

Литература

[1] Балицевич А.Ф., Смирнов С.В., Демидов А.А., Фигуров В.С. Некоторые результаты оценки стойкости ИС в составе электронного блока к воздействию стационарного излучения // Сборник трудов «Электроника, микро- и наноэлектроника». – М.: МИФИ. – 2003. – С. 219 – 222.

[2] Баранов С.В., Чумаков А.И., Яненко А.В. Исследование влияния высокоэнергетичных нейтронов на частоту одиночных сбоев в БИС ОЗУ // Сборник трудов «Электроника, микро- и наноэлектроника». – М.: МИФИ. – 2006. – С.151 – 154.

[3] Некрасов П.В. Методы тестирования микропроцессоров на наличие одиночных сбоев и тиристорного эффекта от отдельных ядерных частиц // Сборник трудов «Электроника, микро- и наноэлектроника». – М.: МИФИ. - 2005. - Ч. 2. - С. 240 – 242.

[4] Аряшев С.И., Николина Н.В., Чибисов П.А. Тесты аттестации архитектуры RTL-модели 64-разрядного суперскалярного микропроцессора // Сборник трудов «Проблемы разработки перспективных микроэлектронных систем». – ИППМ, 2005. - С. 257 – 262.

[5] Hennessy J.L., Patterson D.A Computer Architecture: A Quantitative Approach // 3rd ed., Morgan Kaufmann Publishers, - 2003. - P. 1072.