Генерация тестовых программ для подсистемы управления памятью микропроцессора

Д.Н. Воробьев, А.С. Камкин
{vorobyev, kamkin}@ispras.ru

Аннотация. В работе рассматривается методика генерации тестовых программ, предназначенных для функционального тестирования подсистемы управления памятью микропроцессора. Предлагаемый подход основан на формальной спецификации инструкций, работающих с памятью, и устройств подсистемы, включая буфер трансляции адресов и кэш-память. Использование формальных описаний позволяет автоматизировать разработку генератора тестовых программ, и, что не менее важно, систематизировать тестирование за счет четкого определения целей тестирования. Для генерации тестовых программ в предлагаемом подходе используются комбинаторные техники — тестовые воздействия на микропроцессор представляются в виде цепочек инструкций небольшой длины, составленных из различных сочетаний тестовых ситуаций.

1. Введение

Память современных компьютеров представляет собой сложную иерархию запоминающих устройств различных объемов, стоимости и времени доступа. Помимо регистров и оперативной памяти в микропроцессорах имеется, по крайней мере, одноуровневая, а чаще двухуровневая кэш-память; для ускорения преобразования виртуальных адресов в физические используются буферы трансляции адресов. Логически связанный набор модулей микропроцессора, отвечающих за организацию работы с памятью, называется *подсистемой управления памятью*.

Подсистема управления памятью является ключевым компонентом микропроцессора, и, естественно, к корректности ее функционирования предъявляются очень высокие требования. Поскольку подсистема имеет сложную, многоуровневую организацию, число различных ситуаций, возможных при ее работе, огромно, что не позволяет проверить ее вручную.

Эффективное тестирование подсистемы управления памятью возможно только при использовании методов автоматизации разработки тестов[1].

В существующей практике тестирования часто полагаются на случай — генерация тестов (как правило, в форме программ на языке ассемблера тестируемого микропроцессора) осуществляется автоматически, но случайным образом. Инженер, отвечающий за тестирование, может задавать распределение вероятностей появления тех или иных инструкций в тестовых программах и указывать события, возникающие при их выполнении. Такой подход позволяет обнаруживать многие ошибки, но не является систематическим и, соответственно, не гарантирует полноты тестирования.

В данной работе рассматривается подход к автоматизации генерации тестовых программ для подсистемы управления памятью. Подход является дополнением к «случайным тестам» и позволяет обнаруживать сложные, нетривиальные ошибки в моделях микропроцессоров. В основе предлагаемой методики лежат формальные спецификации устройств, входящих в подсистему управления памятью, спецификации инструкций, работающих с памятью, и описание тестового покрытия на уровне отдельных инструкций. Для генерации тестовых программ используются комбинаторные техники — тестовые воздействия на микропроцессор строятся как цепочки инструкций небольшой длины, составленные из различных сочетаний тестовых ситуаций.

Оставшаяся часть статьи организована следующим образом. Во втором, следующем за введением, разделе кратко описывается устройство подсистемы управления памятью современных микропроцессоров. Третий раздел рассматривает используемый подход к генерации тестовых программ. В нем также обсуждается специфика тестирования подсистемы управления памятью. Четвертый раздел представляет собой сравнение предлагаемой методики с существующими подходами. В пятом разделе рассказывается об опыте практического использования методики. Наконец, шестой раздел содержит заключение.

2. Подсистема управления памятью

Подсистемой управления памятью (*MMU*, *Memory Management Unit*) называется подсистема компьютера, отвечающая за обработку обращений к памяти. К ее функциям относятся: трансляция виртуальных адресов в физические, защита памяти, управление кэш-памятью и др. [1]. Вообще говоря, отдельные компоненты подсистемы управления памятью могут располагаться вне интегральной схемы микропроцессора, однако в данной статье рассматриваются только те составляющие MMU, которые являются

[1] Обычно функциональное тестирование микропроцессора проводится на модели, разработанной на языке описания аппаратуры (Verilog, VHDL).

частью микропроцессора: *буфер трансляции адресов* (*TLB*, *Translation Lookaside Buffer*) и *устройства кэш-памяти*.

2.1. Виртуальная память

В современных микропроцессорах общего назначения используется форма адресации, известная как *виртуальная память*. С точки зрения программиста память представляет собой непрерывную область, называемую *виртуальным адресным пространством*. Доступ к памяти осуществляется с помощью *виртуальных адресов*. Перед доступом в реальную, *физическую память* виртуальные адреса при помощи специальных механизмов транслируются в соответствующие им *физические адреса*.

В общем случае виртуальное адресное пространство разделено на сегменты. Возможность доступа к каждому сегменту определяется режимом работы микропроцессора. Если режим микропроцессора удовлетворяет уровню привилегий сегмента, при обращении к памяти осуществляется трансляция виртуального адреса, после чего происходит обращение к физической памяти; в противном случае микропроцессор генерирует специальное исключение.

Сегменты виртуального адресного пространства можно классифицировать на *отображаемые* и *неотображаемые*. В свою очередь, неотображаемые сегменты можно разделить на *кэшируемые* и *некэшируемые*. Виртуальные адреса отображаемых сегментов преобразуются в физические с помощью буфера трансляции адресов. Виртуальные адреса неотображаемых сегментов не задействуют TLB — физический адрес содержится в определенных битах виртуального адреса[2]. Кэшируемые сегменты отличаются от некэшируемых тем, что при обращении к памяти через эти сегменты задействуются механизмы кэширования микропроцессора.

2.2. Буфер трансляции адресов

Буфер трансляции адресов входит в состав большинства современных микропроцессоров и предназначен для кэширования *таблицы страниц* — таблицы операционной системы, хранящей соответствие между номерами виртуальных и физических страниц памяти. Использование такого буфера позволяет значительно увеличить скорость трансляции адресов.

В общих словах трансляция виртуального адреса с помощью TLB осуществляется следующим образом. Если буфер содержит ячейку с нужным номером виртуальной страницы, в определенном выходном регистре модуля формируется соответствующий физический адрес — номер виртуальной страницы меняется на номер физической, а смещение остается прежним; в противном случае на одном из выходов модуля устанавливается сигнал,

[2] Помимо физических адресов виртуальные адреса неотображаемых сегментов могут содержать дополнительную информацию, например, политику кэширования данных и т.п.

говорящий о промахе в буфер (в этом случае микропроцессор генерирует исключение).

2.3. Кэш-память

Кэш-память представляет собой промежуточный буфер с быстрым доступом, содержащий наиболее часто используемые данные, которые хранятся в менее быстродействующих устройствах. Кэш-память современных микропроцессоров имеет несколько уровней (обычно их два — L1 и L2). Кэш L_i буферизует обращения к кэшу L_{i+1}; кэш последнего уровня является самым крупным, и данные в него подгружаются непосредственно из оперативной памяти.

Общая идея работы кэш-памяти следующая. Когда происходит обращение к основной памяти, контроллер кэша проверяет, есть ли требуемые данные в буфере. Если данные в нем есть (попадание в кэш), они берутся из кэша. В противном случае (промах в кэш), один из блоков данных, содержащихся в буфере, замещается запрашиваемыми данными из основной памяти. Какой именно блок будет замещен, определяется *стратегией замещения*.

В общем случае кэш-память состоит из некоторого числа *множеств*, обозначим это число $S = 2^s$; каждое множество состоит из E строк; а каждая строка состоит из блока данных размера $B = 2^b$, тэга — старших разрядов физического адреса[3], используемых для вычисления признака попадания, — и бит управляющей информации. В зависимости от значений S и E различают три типа кэш-памяти:

1. *кэш-память прямого отображения* ($E = 1$);
2. *полностью ассоциативная кэш-память* ($S = 1$);
3. *частично ассоциативная кэш-память* ($E > 1$ и $S > 1$).

Величина E называется *уровнем ассоциативности* кэш-памяти.

Пусть физический адрес имеет разрядность m. В большинстве случаев при обращении к кэш-памяти с параметрами $\langle S = 2^s, B = 2^b, E\rangle$ физический адрес интерпретируется следующим образом: биты $[0, \ldots, b\text{-}1]$ определяют позицию байта внутри блока данных, $[b, \ldots, b+s-1]$ — номер множества и, наконец, $[b+s, \ldots, m]$ — тэг. Для определения, имеет место попадание или промах, по физическому адресу вычисляется номер множества, после чего для каждой строки этого множества осуществляется сравнение тэга адреса, по которому осуществляется обращение, с тэгом, хранящимся в строке. Если для одной из строк сравнение истинно, значит, имеет место кэш-попадание, и в блоке данных этой строки на соответствующей позиции находятся требуемые данные.

[3] Обычно тэг определяется старшими разрядами *физического адреса*, однако возможны реализации MMU, в которых кэширование осуществляется на основе виртуальных адресов.

Отметим, что кэш-память может поддерживать различные стратегии обновления содержимого буферов разных уровней (в случае многоуровневой организации), а также стратегии записи в оперативную память (сквозная или отложенная запись). Будем называть такие стратегии *политиками кэширования*.

3. Тестирование подсистемы управления памятью

Как отмечалось во введении подсистема управления памятью современного микропроцессора имеет очень сложную организацию, что существенно затрудняет ее ручное тестирование. Эффективная проверка MMU возможна только при использовании методов автоматизации разработки тестов.

Одним из наиболее распространенных способов тестирования микропроцессоров и их подсистем является прогон так называемых *тестовых программ*, то есть программ специально разработанных для целей тестирования. В процессе выполнения тестовые программы создают разнообразные ситуации в работе микропроцессора, результаты выполнения программ протоколируются и используются для анализа правильности функционирования микропроцессора.

Данная работа посвящена автоматизации генерации тестовых программ, предназначенных для функционального тестирования подсистемы управления памятью.

3.1. Используемый метод генерации тестовых программ

В предлагаемом подходе построение тестовых программ осуществляется автоматически на основе формальной спецификации подсистемы управления памятью. Цель генерации задается с помощью критерия тестового покрытия, выделяющего набор тестовых ситуаций для инструкций, работающих с памятью. Тестовые программы строятся путем целенаправленного перебора всевозможных сочетаний тестовых ситуаций для цепочек инструкций ограниченной длины.

Общее описание используемого метода генерации тестовых программ доступно в работе [2]. Идея метода основана на предположении, что поведение микропроцессора зависит от множества выполняемых инструкций (состояние конвейера), зависимостей между ними (через регистры или память) и событий, возникающих при выполнении инструкций (исключения, попадания/промахи в кэш и т.п.).

Генератору на вход подаются формальные спецификации инструкций – в нашем случае инструкций, работающих с памятью (типа *загрузки* и *сохранения*). Кроме того генератору даются описания возможных тестовых ситуаций и зависимостей между инструкциями, а также параметры управления генерацией, например, длина генерируемых цепочек инструкций.

Тестовая программа представляет собой последовательность *тестовых вариантов*. Каждый тестовый вариант содержит *тестовое воздействие* — специально подготовленную цепочку инструкций, предназначенную для создания определенной ситуации в работе микропроцессора. Тестовое воздействие предваряется *инициализирующими инструкциями* и может завершаться *тестовым оракулом* — инструкциями, проверяющими корректность состояния микропроцессора после выполнения тестового воздействия.

Таким образом, структуру тестовой программы можно описать с помощью формулы $test = \{\langle init_i, action_i, post_i \rangle\}_{i=1,n}$, где $init_i$ — это инициализирующие инструкции, $action_i$ — тестовое воздействие, $post_i$ — тестовый оракул.

Ниже приведен фрагмент тестовой программы для микропроцессора с системой команд MIPS64 [3], который включает один тестовый вариант.

```
///// Инициализирующие инструкции /////

// Инициализация регистров инструкции 1:
// загрузка виртуального адреса в базовый регистр a0
// a0[base]=0xffffffff81af1590
ori  a0, zero, 0xffff
dsll a0, a0, 16
ori  a0, a0, 0xffff
dsll a0, a0, 16
ori  a0, a0, 0xa1af
dsll a0, a0, 16
ori  a0, a0, 0x1590

// Инициализация памяти для инструкции 1
ori a1, zero, 0xdead
sw  a1, 0(a0)

// Инициализация регистров инструкции 2:
// загрузка виртуального адреса в базовый регистр t1
// t1[base]=0xffffffff81c49598
...

// Инициализация памяти для инструкции 2
...

// Инициализация кэша L1 для инструкции 1:
// помещение данных в кэш L1
lw t0, 0(a0)
```

```
    // Инициализация кэша L1 для инструкции 2:
    // вытеснение данных из кэша L1
    ...

    //////// Тестовое воздействие ////////

    // Зависимость между инструкциями 1 и 2:
    // L1IndexEqual=false
    lw v0, 0(a0) // L1Hit=true
    sw a2, 0(t1) // L1Hit=false

    ////////// Тестовый оракул //////////

    // Тестовый оракул для инструкции 1
    ori t1, zero, 0xdead

// Ошибочное завершение
    bne v0, t1, error_found
    nop

    // Тестовый оракул для инструкции 2
    ...
```

Тестовое воздействие состоит из двух инструкций: `lw`, осуществляющей загрузку слова, и `sw`, которая сохраняет слово по указанному адресу. Для наглядности тестовые ситуации для этих инструкций затрагивают только кэш-память первого уровня (L1): первая инструкция вызывает попадание в кэш, вторая — промах.

Несколько слов о вспомогательных инструкциях, используемых в примере. Инструкция `ori` осуществляет побитовое ИЛИ значения второго регистра с 16 битным значением, заданным в третьем операнде, результат записывается в первый регистр; инструкция `dsll` сдвигает значение второго регистра влево на заданное число разрядов и сохраняет результат в первом регистре; `bne` — это инструкция условного перехода (переход осуществляется в случае неравенства значений регистров).

3.2. Формальная спецификация подсистемы управления памятью

На вход генератору тестовых программ подаются формальные спецификации инструкций. Спецификация отдельной инструкции включает в себя *описание интерфейса инструкции*, *функцию выполнения инструкции* и *ассемблерный формат*. В интерфейсе инструкции описываются ее операнды и предусловие. Функция выполнения инструкции вычисляет значения выходных операндов

инструкции и обновляет модельное состояние микропроцессора. Ассемблерный формат определяет запись инструкции на языке ассемблера.

Для того чтобы описать семантику инструкций, работающих с памятью, необходимо смоделировать устройства MMU, в частности, буфер трансляции адресов и кэш-память. Все такие устройства могут быть описаны однотипным образом путем задания следующих параметров:

- уровень ассоциативности;
- число множеств;
- функция вычисления тэга;
- функция вычисления индекса;
- структура блока данных;
- управляющие биты;
- стратегия замещения данных при промахе.

В используемом генераторе тестовых программ для разработки формальных спецификаций используется язык программирования Java. Нами разработана библиотека абстрактных классов, позволяющая путем механизма наследования и перегрузки методов (соответствующих указанным выше параметрам) получать спецификации основных устройств MMU. Объекты полученных спецификационных классов можно использовать для определения семантики инструкций, вызывая необходимые интерфейсные методы (чтение, запись и т.п.).

Моделирование подсистемы управления памятью позволяет генератору автоматически строить цепочки инициализирующих инструкций, приводящих микропроцессор в соответствующее тестовой ситуации состояние.

3.3. Типы тестовых ситуаций для подсистемы управления памятью

Под тестовой ситуацией для инструкции понимается ограничение на значения входных операндов и состояние микропроцессора перед началом выполнения инструкции. Множество возможных тестовых ситуаций выявляется на основе анализа описания архитектуры микропроцессора.

При тестировании MMU структура тестовых ситуаций определяется функциональностью подсистемы, связанной с обработкой запросов к памяти. Например, на первом этапе обработки определяется сегмент адресного пространства, к которому относится виртуальный адрес: если сегмент является отображаемым, трансляция адреса осуществляется с помощью TLB; в противном случае вычисление физического адреса производится в обход буфера трансляции адресов. Для данного этапа определены две тестовые ситуации: *Mapped = true* и *Mapped = false*. Далее, если виртуальный адрес является отображаемым, возможны две альтернативные ситуации: *TLBHit = true*, когда страница виртуальной памяти находится в TLB (попадание), и *TLBHit = false*, когда страницы в буфере нет (промах).

Выделение ситуаций можно продолжить до тех пор, пока данные из памяти не будут загружены в регистр или, наоборот, сохранены из регистра в память (в зависимости от операции).

Для подсистемы управления памятью можно выделить следующие типы элементарных ситуаций (ситуаций для отдельных этапов обработки), на основе композиций которых строятся результирующие тестовые ситуации:

- *Mapped* — отображаемый/неотображаемый сегмент виртуальной памяти;
- *Cached* — кэшируемый/некэшируемый сегмент виртуальной памяти;
- *TLBHit* — попадание/промах в TLB;
- *Valid* — бит достоверности секции TLB;
- *L1Hit* — попадание/промах в кэш-память L1;
- *L2Hit* — попадание/промах в кэш-память L2.

На Рис. 1 элементарные ситуации структурированы в виде дерева (пунктирная дуга соответствует значению *false*, сплошная — значению *true*). Результирующая тестовая ситуация для инструкции соответствует пути от корня до листовой вершины.

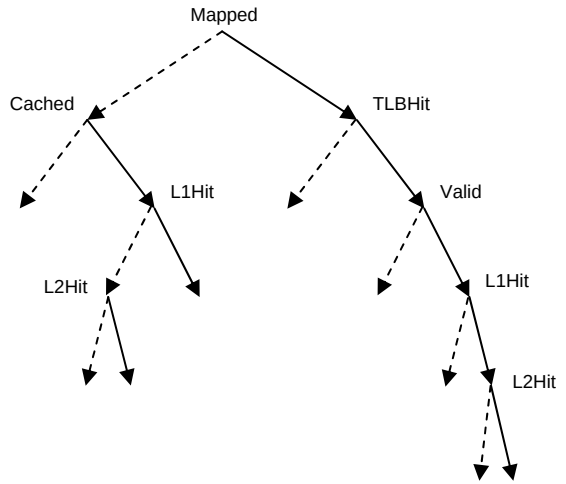

Рис. 1. Иерархия тестовых ситуаций

Отметим, что выделение тестовых ситуаций можно проводить на разных уровнях абстракции, например, тестовую ситуацию, связанную с промахом или попаданием в буфер трансляции адресов в MIPS64-совместимом

микропроцессоре, можно детализировать на основе выражения, вычисляющего признак попадания в строку TLB [3]:

$$(TLB[i]_R = va_{63..62})\ and$$

$$((TLB[i]_{VPN2}\ and\ not\ (TLB[i]_{Mask})) = (va_{\text{SEGBITS-1..13}}\ and\ not\ (TLB[\text{i}]_{Mask})))\ and$$

$$(TLB[i]_G\ or\ (TLB[\text{i}]_{ASID} = EntryHi_{ASID}))$$

При тщательном тестировании рассматриваются все комбинации значений истинности элементарных условий (exclusive condition coverage) или только те комбинации, когда значение истинности одного условия влияет на значение истинности всего выражения (MC/DC, Modified Condition/Decision Coverage) [4].

3.4. Типы зависимостей для подсистемы управления памятью

Зависимости между инструкциями бывают двух основных типов: *зависимости по регистрам* и *зависимости по адресам*. Зависимости по регистрам выражаются с помощью равенств номеров регистров, использующихся в качестве операндов инструкций тестового воздействия. Зависимости по адресам связаны с устройством подсистемы управления памятью. Примерами зависимостей по адресам являются совпадение виртуальных адресов, совпадение номеров страниц виртуальной памяти, совпадение физических адресов, совпадение используемых множеств кэш-памяти и др. Многоуровневая организация памяти приводит к сложной, многоуровневой структуре зависимостей.

Зависимости по адресам можно условно разбить на *зависимости по виртуальным адресам* и *зависимости по физическим*. Зависимости по виртуальным адресам связаны со структурой буфера трансляции адресов. Зависимости по физическим адресам определяются организацией кэш-памяти.

Мы используем следующие типы зависимостей для буферов, входящих в состав MMU:

- *IndexEqual* — совпадение используемых множеств;
  - *EntryEqual* — совпадение строк внутри множества;
    - *BlockEqual* — совпадение данных внутри строки;
  - *EntryReplace* — обращение к данным, строка с которыми была вытеснена из буфера предшествующей инструкцией;
    - *BlockEqual* — совпадение данных внутри строки.

В целом для подсистемы управления памятью можно выделить следующие типы элементарных зависимостей:

- *VAEqual* — совпадение виртуальных адресов;

- *VPNEqual* — совпадение номеров страниц виртуальной памяти;
- *PAEqual* — совпадение физических адресов;
- *PFNEqual* — совпадение номеров страниц физической памяти;
- *L1IndexEqual* — совпадение множеств кэш-памяти L1;
- *L2IndexEqual* — совпадение множеств кэш-памяти L2;
- *L1Replace* — совпадение тэга кэш-памяти L1 с тэгом данных, которые были вытеснены из кэш-памяти предшествующей инструкцией;
- *L2Replace* — совпадение тэга кэш-памяти L2 с тэгом данных, которые были вытеснены из кэш-памяти предшествующей инструкцией.

Как отмечалось в силу иерархической организации памяти зависимости по адресам имеют многоуровневую структуру. Фрагмент дерева возможных зависимостей представлен на Рис. 2 (пунктирная дуга соответствует значению *false*, сплошная — значению *true*). Результирующая зависимость между инструкциями соответствует пути от корня до листовой вершины.

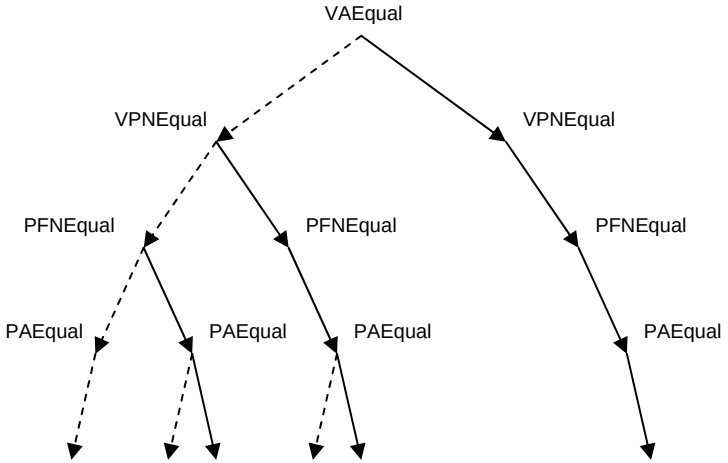

Рис. 2. Фрагмент иерархии зависимостей по адресам

3.5. Подготовка тестового воздействия

Рассмотрим подробнее, как генератор конструирует цепочки инициализирующих инструкций, осуществляющих подготовку тестовых воздействий. Для каждой тестовой ситуации описывается способ ее подготовки — последовательность инструкций, которая инициализирует нужные регистры и память. На первый взгляд, задача является тривиальной — генератору достаточно последовательно подготовить тестовые ситуации для

инструкций, входящих в состав тестового воздействия. Для подсистемы управления памятью такой способ не работает.

Для инструкций, работающих с памятью, подготовка тестовых ситуаций состоит из двух основных частей: инициализация буфера трансляции адресов и инициализация кэш-памяти. Инструкции подготовки буфера трансляции адресов могут изменить состояние кэш-памяти, поэтому при построении программы подготовки тестового воздействия целесообразно сначала осуществлять подготовку буфера трансляции адресов для всех инструкций тестового воздействия, а затем — подготовку кэш-памяти (подготовка кэш-памяти не затрагивает TLB, если использовать адреса из неотображаемых сегментов адресного пространства).

В общем случае, разработчик тестов разбивает программы подготовки тестовых ситуаций для тестируемых инструкций на несколько фрагментов. Каждый фрагмент отвечает за инициализацию определенного модуля микропроцессора. Разработчик упорядочивает фрагменты таким образом, чтобы каждый следующий фрагмент не влиял на модули, подготовленные предыдущими фрагментами.

4. Сравнение с существующими подходами

Нам известен только один подход к автоматизированному построению тестовых программ для MMU, использующий формальные спецификации подсистемы, — DeepTrans, разработка исследовательской лаборатории IBM в г. Хайфе (IBM Haifa Research Lab) [5]. Эта методика нацелена на тестирование части подсистемы, отвечающей за преобразование виртуальных адресов.

В DeepTrans используется язык спецификации, специально созданный для моделирования механизмов трансляции адресов. В спецификации процесс преобразования адреса представляется в виде ориентированного ациклического графа. Вершины графа соответствуют отдельным стадиям трансляции адреса, ребра — переходам между стадиями. Из каждой вершины может выходить несколько дуг, каждая из которых помечена своим условием перехода.

Подход DeepTrans предполагает ручную разработку шаблонов тестовых программ, но в шаблонах можно указывать тестовые ситуации, связанные с трансляцией адресов, извлеченные из спецификации. На основе шаблонов генератор Genesys-Pro путем разрешения ограничений строит набор тестовых программ [6, 7, 8].

Сравнение DeepTrans (в паре с генератором Genesys-Pro) с используемым нами подходом (MicroTESK) приводится в таблице 1.

Сравниваемая характеристика	DeepTrans (Genesys-Pro)	MicroTESK
Поддержка пользовательских шаблонов тестовых программ	Genesys-Pro предоставляет развитый язык описания шаблонов тестовых программ	MicroTESK позволяет описывать лишь сравнительно простые шаблоны тестовых программ
Поддержка автоматической генерации тестовых шаблонов	Автоматическая генерация тестовых шаблонов не поддерживается	Поддерживается автоматическая генерация несложных тестовых шаблонов
Поддержка описания тестовых ситуаций	Присутствует	Присутствует
Поддержка описания зависимостей по адресам	Специальной поддержки нет, зависимости вручную описываются в тестовом шаблоне	Поддерживается описание сложных зависимостей по адресам
Поддержка моделирования подсистемы управления памятью	DeepTrans предоставляет специальный декларативный язык описания механизмов трансляции	В MicroTESK реализована библиотека классов на языке Java, моделирующая устройства MMU
Поддержка генерации самопроверяющих тестов	Присутствует	Присутствует

Таблица 1. Сравнение DeepTrans с MicroTESK

Как видно из сравнения, в DeepTrans упор сделан на ручную разработку шаблонов тестовых программ, но при этом разработчику тестов предоставляются развитые средства описания шаблонов. В нашем подходе тестовые шаблоны генерируются автоматически, систематическим образом. Кроме того, в используемом нами подходе поддерживается описание сложных зависимостей по адресам.

5. Практическая апробация подхода

Мы применили описанный подход для тестирования подсистемы управления памятью MIPS64-совместимого микропроцессора. В качестве тестовых воздействий использовались всевозможные пары, составленные из четырех

инструкций: `lb` (загрузка байта), `ld` (загрузка двойного слова), `sb` (сохранение байта) и `sd` (сохранение двойного слова). Структура используемых тестовых ситуаций и зависимостей была близка к описанной в статье, но были дополнительно учтены управляющие биты в TLB и наличие микро-TLB — промежуточного буфера, содержащего записи, по которым были последние обращения.

В результате тестирования было найдено несколько критических ошибок в подсистеме управления кэш-памятью, которые не были обнаружены с помощью тестовых программ, сгенерированных случайно.

6. Заключение

Тестирование подсистемы управления памятью микропроцессора является нетривиальной задачей, которую практически невозможно решить без применения методов автоматизации. В статье был рассмотрен подход к автоматической генерации тестовых программ для MMU. В отличие от распространенных на практике методов, предлагаемый подход имеет высокий уровень автоматизации и является систематичным. Сгенерированные тестовые программы могут содержать встроенные проверки (тестовые оракулы), что делает их пригодными для тестирования не только моделей микропроцессоров, написанных на языках описания аппаратуры, но и готовых микросхем.

Литература

[1] http://en.wikipedia.org/wiki/Memory_management_unit.

[2] А.С. Камкин. *Генерация тестовых программ для микропроцессоров*. Труды Института системного программирования РАН, 2008.

[3] *MIPS64™ Architecture For Programmers. Revision 2.0.* MIPS Technologies Inc., June 9, 2003.

[4] K.J. Hayhurst, D.S. Veerhusen, J.J. Chilenski, and L.K. Rierson. *A Practical Tutorial on Modified Condition/Decision Coverage*. Report NASA/TM-2001-210876, NASA, USA, May 2001.

[5] A. Adir, R. Emek, Y. Katz, A. Koyfman. *DeepTrans – A Model-Based Approach to Functional Verification of Address Translation Mechanisms*. Microprocessor Test and Verification: Common Challenges and Solutions, 2003.

[6] http://www.haifa.ibm.com/projects/verification/GenesysPresent/index.htm.

[7] A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov, A. Ziv. *Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification*. Design and Test of Computers, 2004.

[8] M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, M. Vinov. *Industrial Experience with Test Generation Languages for Processor Verification*. Design Automation Conference, 2004.