УДК 004.052.42

Исследование способов повышения эффективности стохастического тестирования моделей микропроцессоров

Н.А. Гревцев[1], П.А. Чибисов[1], И.Ш. Хисамбеев[2]

[1]ФГУ ФНЦ НИИСИ РАН, chibisov@cs.niisi.ras.ru

[2]МФТИ ГУ

***Аннотация* — В статье описана методика функционального тестирования микропроцессоров, основанная на построении моделей функционального покрытия с привлечением «тестового знания». Модели покрытия используются для измерения, наблюдения и изменения направления процесса стохастического тестирования. Согласно методике вводятся метрики тестового покрытия на уровне инструкций микропроцессора, производится анализ получаемых данных о покрытии и оценивается эффективность стохастических тестов.**

***Ключевые слова* — функциональная верификация, RTL-модель микропроцессора, стохастическое тестирование, метрики тестового покрытия, модель покрытия, генерация псевдослучайных тестов.**

I. Введение

Современные микропроцессоры - сложные программно управляемые устройства. С одной стороны, исчерпывающее их тестирование невозможно, так как попытка осуществить перебор всех возможных комбинаций внутренних состояний быстро приводит к комбинаторному взрыву, с другой стороны, такое тестирование и не является необходимым, поэтому важным вопросом в ходе функциональной верификации микропроцессоров является вопрос: как за требуемое время провести полное и эффективное тестирование всех аспектов спецификации на микропроцессор.

Современные микропроцессоры создаются с применением архитектурных и микроархитектурных усовершенствований, разрабатываемых с целью повышения их производительности и понижения электропотребления. Это приводит к значительному усложнению проекта и, следовательно, к увеличению трудозатрат на его функциональную верификацию.

Для тестирования микропроцессоров применяют направленные, псевдослучайные, комбинаторные тесты, а также программы и приложения под управлением операционной системы.

По способу получения все тесты можно разделить на рукописные, полученные с помощью автоматической генерации, а также полученные с помощью трансляции программ, написанных на языках высокого уровня. Для каждого теста должен быть задан критерий корректности его работы, а также для всего множества тестов должен быть определен критерий достаточности.

Статья является продолжением публикации [1], посвященной разработке метрик функционального покрытия на уровне инструкций микропроцессора.

II. Генерация псевдослучайных тестов

Архитектурный уровень абстракции (архитектура) микропроцессора описывает микропроцессор с точки зрения программиста. Архитектура определяется набором команд (ISA), разрядностью шин и внутренних регистров, а также компонентами, входящими в состав микропроцессора (ALU, FPU, TLB, кэш-память и т.д.), и их характеристиками (иерархия и ассоциативность кэш-памяти, режимы работы и т.п.).

В традиционном маршруте верификации важную роль при тестировании архитектурного уровня микропроцессора играет стохастическое тестирование. Как правило, на вход генератору тестов подаются тестовые шаблоны, позволяющие пользователю задать перечень инструкций, их аргументы и вероятность появления инструкций в тесте (рис. 1). Это эффективно на начальной стадии верификации и, как показывает практика, это эффективно только для тестирования архитектурного уровня и набора инструкций микропроцессора, при этом новые ошибки быстро перестают находиться. Кроме того, простая генерация псевдослучайных последовательностей инструкций малоэффективна для тестирования микроархитектуры. Как правило, качество (эффективность) псевдослучайных тестов оценивается количеством находимых ошибок для одного шаблона за единицу времени, а также покрытием (структурным и функциональным) RTL-модели.

Известны различные подходы к повышению покрытия, достижимого набором случайных тестов, например, тестирование, управляемое покрытием [2]. Во многих работах (например, [3], [4], [5]), посвящённых верификации микропроцессоров, говорится о необходимости внесения интеллекта, то есть экспертного знания инженера (*“testing knowledge”*), в процесс стохастического тестирования. Однако, даже высокое покрытие (особенно структурное) само по себе не дает гарантий отсутствия ошибок в модели микропроцессора. Как правило, для достижения особых («крайних», «*corner cases*») тестовых ситуаций [6], которые долж-

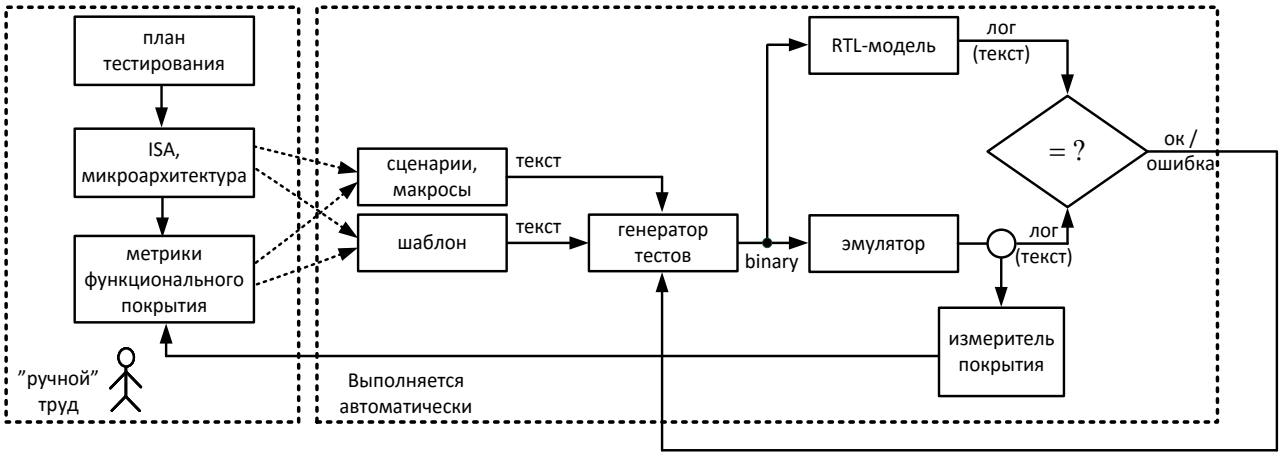

Рис. 1. Маршрут стохастического тестирования

ны потенциально найти сложно обнаруживаемые ошибки в проекте, нужны соответствующие настройки генератора.

III. Направленная генерация тестов

Микроархитектура микропроцессора - это внутренняя реализация архитектуры; она описывает, как именно в данном микропроцессоре организован конвейер инструкций, где и когда в нем возникают остановки, как расположены и соединены друг с другом различные внутренние регистры, кэш-памяти, арифметико-логические и другие устройства микропроцессора; она описывает также различные устройства буферизации потоков данных (буферы операций загрузки/сохранения из/в память), конечные автоматы, блок выборки, выдачи и переупорядочивания инструкций, блок предсказания ветвлений и другие блоки, необходимые для реализации архитектуры. В понятие микроархитектуры входят также такие понятия, как суперскалярность, переименование регистров, возможность выполнения инструкций с изменением их очерёдности. Обобщая можно сказать, что микроархитектура не видна программисту, что усложняет её тестирование.

Существуют специальные генераторы комбинаторных псевдослучайных тестов, направленных на какой-либо аспект микроархитектуры, решающие задачу удовлетворения заданных пользователем ограничений [7], [8], [9]. Однако, эти генераторы очень трудоемкие в создании и настройке, а ошибки, которые находятся в результате созданных ими тестов – в большинстве случаев крайне специфичны и мало связаны с тем, что именно требуется тестировать. Также такие генераторы тестов нацелены на очень узкий класс ситуаций. Поэтому важно понять на начальном этапе, какие аспекты спецификации и каким именно образом нужно перенести в тестовый план, чтобы создать действительно ёмкие, направленные, «интересные» тесты.

В работе [10] был предложен новый метод построения метрик функционального покрытия. Утверждается, что любую ошибку в функционировании микропроцессора можно найти с помощью некой комбинации инструкций (φ), выполненной после установления некоторого внутреннего состояния микропроцессора (к которому также можно прийти с помощью некоторой последовательности инструкций (χ)).

С абстрактной точки зрения микроархитектурная модель тестируемого ядра микропроцессора представляет собой конечный автомат, состояния которого являются произведением состояний составляющих узлов, входные воздействия на который определяются исходным содержимым модели памяти (в предложенной схеме модель памяти содержит сам тест и все данные для него в момент начала симуляции). Отсюда следует, что мощность и размерность пространства состояний тестируемой системы зависят от ее конкретной микроархитектурной реализации, в частности, от таких параметров, как глубина конвейера и размер окна предвыборки инструкций. Ошибка поведения микропроцессора с этой точки зрения представляет собой некорректный переход между состояниями. Заметим, что функция переходов задается спецификацией устройства на упорядоченной паре {состояние, входное воздействие}, а входное воздействие определяется инструкциями и данными в памяти, то есть может быть описано в терминах ISA. Тем не менее, количество инструкций в тестовом воздействии, которые одновременно формируют переход состояния, является зависимой от реализации величиной, которую требуется оценивать, учитывая такие параметры, как глубина конвейера, размер предвыборки инструкций и так далее. Формализуем понятие тестовой ситуации уровня ISA.

Тестовая ситуация – это встретившаяся в тесте комбинация значений следующих параметров:

1) Последовательность инструкций некоторой длины N.

2) Адреса их расположения в памяти (адреса влияют на значение PC (*program counter*) при их выполнении, а также на работу ряда ключевых блоков, таких как буфер инструкций).

3) Данные для инструкций (значения операндов).

4) «След» этих инструкций в иерархии памяти (регистры – кэш-память – буфер загрузки-сохранения – ОЗУ). Под «следом» инструкций будем понимать зависимость по регистрам с близкими командами, комбинацию флагов в кэш-памяти всех уровней, задействованные строки кэш-памяти и соответствующие им адреса в ОЗУ (например, это важно в ситуации, когда текущая инструкция является причиной вытеснения данных из кэш-памяти), политики кэширования, работу TLB, сегмент памяти.

5) Локальные передачи управления, в том числе: по инструкции ветвления, по прерыванию, по исключительной ситуации.

Если перечислить все возможные аспекты микроархитектуры (на уровне ISA), на которые способна повлиять отдельно взятая инструкция, применив язык описания ситуаций, можно разработать множество метрик функционального покрытия.

Доказательство теоремы о том, что любое микроархитектурное событие (точка из микроархитектурной модели покрытия) достижимо с помощью выполнения некоторой последовательности инструкций из набора ISA микропроцессора, которая и является тестом, выходит за рамки данной статьи.

Таким образом, существуют следующие варианты создания тестовых воздействий. Первый заключается в генерации тестов, направленных на достижение отдельных тестовых ситуаций (комбинаторные тесты). Второй – в использовании псевдослучайных тестов, создаваемых по шаблонам, в которые добавляются тестовые сценарии (макросы) для тестирования микроархитектуры (рис. 1) совместно с измерителями (мониторами) покрытия по требуемым метрикам для замыкания контура обратной связи (для контроля качества тестирования).

Акцент в предложенном методе делается на то, что следует использовать любые способы генерации тестов, если есть обратная связь по покрытию, организованная с помощью мониторов функционального покрытия.

Стратегия верификации модели микропроцессора, рассматриваемая в данной работе, состоит в том, чтобы осуществить как можно более полное тестирование RTL-модели с использованием знания о задачах потенциальных высокоуровневых приложений пользователя, применяя моделирование на ранних этапах цикла проектирования. Этот подход был назван «встречным» тестированием [10]. Предлагается вместо подачи произвольных (случайных) тестовых воздействий из набора всех возможных операций во всех возможных состояниях микропроцессора вслепую сфокусироваться, первую очередь, на проверке тех важных аппаратных функций, которые действительно используются реальными пользовательскими программами на данной аппаратной реализации микропроцессора. В отличие от модульного или блочного тестирования, цель рассматриваемого системного подхода – в верификации взаимодействий между отдельными блоками и, конечно, самих блоков в масштабе всей системы, как в процессе разработки модели, так и при тестировании первого кристалла СБИС микропроцессора.

IV. Выбор метрик функционального покрытия инструкций микропроцессора

Функциональное покрытие фокусируется на всех имеющихся выполняемых функциях тестируемой системы. Оно используется для того, чтобы убедиться, что все аспекты, заложенные в спецификацию, протестированы. В отличие от метрик структурного покрытия, основанных на коде реализации (код HDL-модели микропроцессора), функциональное покрытие основано на спецификации.

При использовании функционального покрытия пользователь систематично определяет модели покрытия, включающие в себя набор задач (требований к покрываемым областям, выраженным в метриках покрытия). Модели покрытия используются для измерения, наблюдения и изменения направления процесса тестирования.

Непокрытые либо недостаточно покрытые области требуют создания дополнительных тестов. Процесс тестирования с использованием метрик покрытия является итеративным, при этом качество тестирования постоянно измеряется, и по результатам анализа измерений предполагаются возможные улучшения процесса. Основная трудоемкость при разработке управляемого метриками тестирования заключается в осмысленном выборе и формализованном описании метрик, в создании инструментальной поддержки измерений и анализа данных о покрытии. Применение описанного подхода требует достаточно глубокого знания микроархитектуры тестируемого микропроцессора, а также тесной интеграции инженеров-верификаторов с проектировщиками RTL-модели для лучшего понимания того, как код приложений пользователя и тестов затрагивает внутренние структуры HDL-кода. С методическим разбором подобного примера разложения кода контрольной задачи с анализом микроархитектуры и уровня инструкций микропроцессора можно ознакомиться в работе [11].

Общая схема, описывающая алгоритм выбора метрик функционального покрытия для встречного тестирования, показана на рис. 2. Данная схема процедуры выбора метрик и составленная на ее основе методика выбора метрик функционального покрытия являются основными составляющими частями методики выбора набора тестов под определенный класс задач.

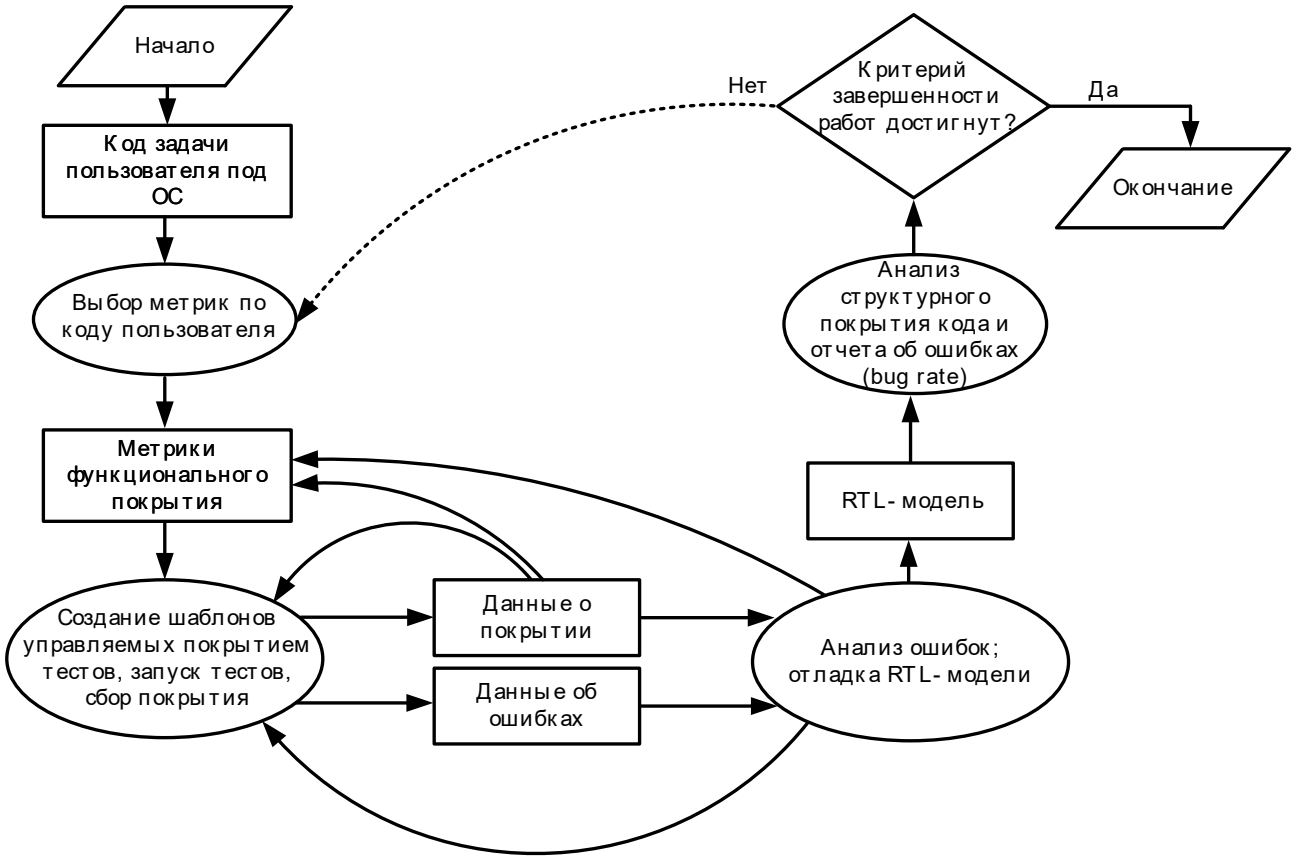

Рис. 2. Схема процедуры построения функциональных метрик

Методика выбора функциональных метрик покрытия для встречного тестирования модели микропроцессора под определенный класс задач может быть представлена в виде следующей последовательности действий:

A. Анализ миссии (назначения) комплекса, то есть области применения и типовых задач, решаемых системой на кристалле

1) Получить информацию о миссии комплекса.

2) Классифицировать задачи, решаемые с помощью комплекса в целом.

3) Оценить разделение решаемых задач на аппаратные и программные.

4) Изучить спецификацию на проектируемую аппаратуру.

5) Получить исходные коды ОС.

6) Получить исходные коды приложений пользователя под ОС, а также получить инструментальные средства.

Исходные данные: технические требования к вычислительной системе, задачи потенциальных пользователей (коды в исходном виде).

Результат: таблица с перечнем типовых задач под ОС, набор инструментальных средств, исходные коды ОС и задач пользователя.

B. Исследование встречающихся в коде инструкций ISA (и их комбинаций) и задействованных режимов работы микропроцессора

1) Изучить листинги программ и трассы исполнения на эмуляторе - оценить применяемый набор ISA.

2) Исключить редкие комбинации инструкций из рассмотрения (тестируются обычными методами).

3) Перечислить задействованные во время работы ОС и тестов по ОС особенности микроархитектурного уровня разрабатываемого микропроцессора.

4) Изучить строение кода обработчиков исключительных ситуаций и прерываний, сегментации, режимов работы микропроцессора.

5) Изучить общие особенности трасс - перечислить режимы работы кэш-памяти, TLB, FPU, DMA.

6) Зафиксировать в виде примеров характерные для трасс (листингов) задач пользователя участки кода, такие как циклы, работа со стеком (памятью) и тому подобное.

7) Составить детальный план тестирования с перечислением требований для псевдослучайного тестирования.

Исходные данные: код задач пользователя, спецификация на микропроцессор.

Результат: план тестирования в виде таблицы особенностей микроархитектурного уровня микропроцес-

сора, необходимых для тестирования; таблица с примерами характерного для задач пользователя кода.

C. Составление исходных метрик функционального покрытия

1) Выявить часто встречающиеся комбинации инструкций и их аргументов, приводящих к ошибкам.

2) Расширить такие комбинации (с учетом области допустимых значений параметров) с учетом рекомендаций разработчиков RTL-модели.

3) Составить текстовое описание таких тестовых воздействий с перечислением всех возможных комбинаций инструкций и их параметров.

4) Оценить максимальное число состояний пространства всех допустимых элементарных тестовых воздействий.

5) Задать метрику M_K, показывающую степень покрытия тестами всех допустимых состояний описанного пространства K.

Исходные данные: база данных в системе отслеживания ошибок.

Результат: таблица, содержащая описание метрик функционального покрытия; перечень требований к шаблонам и функционалу генератора (документ "свод рекомендаций для псевдослучайного тестирования").

D. Доработка системы стохастического тестирования

1) Доработать функционал генератора для выполнения требований к тестам.

2) Разработать новые шаблоны для тестов с учетом полученных в пп. B - C рекомендаций.

3) Усовершенствовать шаблоны добавлением макросов (и других технических средств) для повышения вероятности создания требуемых тестовых ситуаций.

4) Производить динамическую мутацию тестовых ситуаций в макросах для получения новых уникальных ситуаций. Получаемые ситуации отличаются от первоначально задуманных (указанных в тестовом плане), но являются производными от них (для расширения тестового покрытия) [4].

5) Провести анализ полученных тестов на предмет удовлетворения требований и рекомендаций.

Исходные данные: система стохастического тестирования; документ "свод рекомендаций для псевдослучайного тестирования".

Результат: система стохастического тестирования, доработанная для повышения качества тестирования с учетом задач потенциальных пользователей (система управляется метриками функционального покрытия).

E. Анализ возникающих ошибок в RTL-модели

1) Детально протоколировать все возникающие в ходе тестирования ошибки в RTL-модели (а также и в образцовом эмуляторе).

2) По каждой найденной ошибке получить комментарий разработчика блока RTL-модели, в котором была найдена ошибка.

3) По каждой найденной ошибке составить детальный отчёт.

4) Группировать сходные по каким-либо признакам ошибки.

5) Оценить для каждой найденной ошибки степень вероятности возникновения такой ошибки в коде пользователя, то есть провести "проверку актуальности".

Исходные данные: система стохастического тестирования; отчёты о найденных ошибках.

Результат: база данных в системе отслеживания ошибок.

F. Итерационная детализация метрик функционального покрытия, составление новых метрик

1) Выполнять пп. C - E методики с учетом накапливающихся в процессе верификации RTL-модели знаний о ее внутреннем устройстве, поведении на уровне внутренних сигналов, найденных ошибках; детализировать метрики.

2) Протоколировать значения структурных и функциональных метрик ежедневно, а также следить за частотой возникновения ошибок (*bug rate*).

3) При достижении указанного в плане верификации критерия завершения работ (критерия полноты тестирования) зафиксировать HDL-код проекта для передачи его файлов на следующую по маршруту проектирования стадию, однако продолжать тестирование.

Исходные данные: RTL-модель, подлежащая тестированию, система стохастического тестирования, управляемая встречным подходом.

Результат: RTL-модель, готовая к передаче для изготовления СБИС.

В результате включения данной методики в маршрут тестирования микропроцессора разработчики тестов получают инструмент для направления псевдослучайного тестирования в сторону увеличения значений метрик функционального покрытия кода приложений пользователя, а также возможность измерения качества тестирования с течением времени.

Модели функционального покрытия для рассматриваемой методики могут состоять из четырех компонент [12]. Основной компонент – это семантическое описание (изложение) модели. Второй компонент – это список атрибутов, относящихся к изложению. Третий компонент – это набор всех возможных значений каждого атрибута. Последний компонент – это список ограничений допустимых комбинаций в перекрестном (декартовом) произведении всех значений атрибутов. Перечисленные четыре компоненты должны присутствовать в описании каждой составляемой метрики.

Пример. Спустя полгода после начала работ по тестированию RTL-модели успешно выполнялись тысячи случайных тестов в сутки, база регрессионного тестирования успешно проходила, на ПЛИС-прототипе

была загружена ОС Linux и работали различные приложения пользователя, в том числе ресурсоёмкие. Однако простая операция копирования большого (размером более 512 Мбайт) файла с последующим подсчётом контрольной суммы (*md5sum*) свидетельствовала о потере данных. С применением описанной выше методики встречного тестирования, эта ошибка была локализована с помощью направленного усовершенствования шаблонов и макросов для псевдослучайного тестирования в течение трех дней. При этом было найдено ещё 12 других актуальных ошибок в RTL-модели.

V. Инструмент мониторинга функционального покрытия на уровне инструкций

Для эффективного управления процессом верификации необходимо явное задание измеряемых метрик, а также оценка максимального значения, которое способна принимать каждая метрика. Такое значение будет соответствовать 100% покрытия данной метрики. С помощью управляемой метриками тестового покрытия верификации можно количественно измерять степень завершенности работ по тестированию RTL-модели, а также направлять процесс генерации тестов в сторону непокрытых областей.

Для сбора и анализа данных о покрытии были разработаны скрипты на языках высокого уровня «python» и «perl». Входные данные передаются скриптам в виде текстовых файлов (листинги программ или трассы выполнения тестов), а выходными данными являются численные значения измеряемых метрик покрытия.

Далее приведены примеры с описанием микроархитектурных ситуаций и соответствующих им метрик покрытия.

Метрика 1. Четверки инструкций, сгруппированных в классы эквивалентности: перебор «каждый за каждым».

Для любой архитектуры системы команд (ISA) можно разбить весь набор инструкций по классам эквивалентности и выделить N типов. Чем больше N, тем более детальной является модель покрытия. Для данной метрики тестовой ситуацией является последовательность из 4 выполненных подряд инструкций. Общее число ситуаций равно N^4.

В предложенном примере весь набор команд был разбит на N = 10 типов: A - арифметические, B - команды ветвления, S - загрузки/сохранения и управления памятью, L - логические и сдвига, M - перемещения, F - арифметические команды вещественной арифметики, P – привилегированные инструкции, T - инструкции вызова системного исключения, N – прочие. Так же в отдельную группу были выделены события, связанные с возникновением исключительных ситуаций (E) для отслеживания последовательностей инструкций, приводящих к исключениям, так как в этом случае происходит остановка конвейера.

В рамках данной метрики были проанализированы трассы выполнения базы случайных тестов (1200 тестов), а также трассы и листинги загрузки ОС Linux и различных пользовательских приложений.

Результатом анализа данной модели покрытия является выявление наиболее часто встречающихся в реальных задачах последовательностей инструкций и получение перечня уникальных комбинаций инструкций, не встречающихся в базе случайных тестов (рис. 3) для последующей доработки шаблонов тестового генератора.

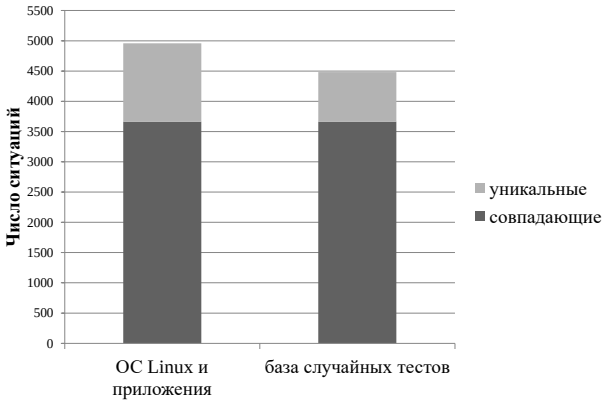

Рис. 3. Диаграмма распределения числа ситуаций

Метрика 2. Пары типов инструкций с различными зависимостями по ресурсам.

В этой модели покрытия весь набор команд архитектуры модели тестируемого микропроцессора аналогичным образом разбивается на N типов. В пределах данной метрики рассматриваются пары инструкций с информационной зависимостью в пределах n ∈ [1;5] инструкций, где n – «возраст» зависимости.

Рассматриваются 3 вида зависимостей:

RAW (read after write;

WAR (write after read);

WAW (write after write).

Зависимости могут возникать по трем ресурсам:

1) зависимость по регистрам, например:

or r1, r3, r5

sub r1, r16, r17 // RAW dependency

Следует заметить, что инструкции, использующие регистровый файл общего назначения, выполняющиеся на целочисленных арифметически-логических устройствах, не имеют доступа к регистровому файлу вещественных данных и наоборот. В таком случае зависимость отсутствует.

2) Зависимость по памяти (совпадающий адрес):

sw r2,0x1000(r1) // адрес = A, store word

add r10, r11, r12 // независимая инструкция

lw r3,0x1000(r1) // адрес = A, возраст = 2

3) Зависимость по линии кэш-памяти:

lw r1,0x1000(r10) // load word

sw r2,0x1004(r10) // store word

Учитываются обращения в кэшируемые пространства памяти в одну линию кэш-памяти, а также пересечения между линиями, принадлежащими различным секциям и уровням кэш-памяти.

Выходным файлом будет являться таблица с кортежами, в которых первые два элемента представляют собой типы инструкций, между которыми возникла зависимость, третий элемент — тип зависимости, последний - «возраст» зависимости (число инструкций между зависимыми элементами) (рис. 4).

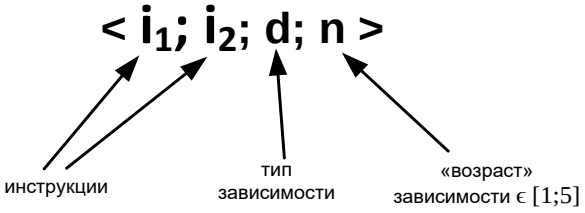

Рис. 4. Кортеж «инструкции-зависимость-возраст»

Результатом анализа модели покрытия, составленной по метрике 2, является выявление уникальных зависимостей для комбинаций инструкций, отстоящих друг от друга на различное число тактов в конвейере и разделяющих разные ресурсы. Одной из целей анализа получаемых данных является доработка языка шаблонов стохастического тестирования для расширения возможностей генерации тестов.

VI. Заключение

Для эффективного стохастического тестирования необходимо создавать псевдослучайные тесты, направленные на достижение покрытия ситуаций, указанных в тестовом плане. Для этого требуется применять экспертное тестовое знание об особенностях микроархитектурного уровня микропроцессора, перечень требований к шаблонам и функционалу генератора тестов, а также набор метрик функционального покрытия и соответствующие им мониторы покрытия. Опыт применения предлагаемого в статье маршрута стохастического тестирования позволил выявить микроархитектурные особенности разрабатываемого микропроцессора, сформулировать требования к шаблонам псевдослучайных тестов и, в итоге, найти ряд сложно обнаруживаемых ошибок в проекте.

В дальнейшем планируется расширение набора метрик, в котором выражается накопленное тестовое знание для верифицируемых микропроцессоров, разработка универсального инструмента анализа трасс поведенческой модели для выявления тестовых ситуаций.

Литература

[1] Хисамбеев И.Ш., Чибисов П.А. Об одном методе построения метрик функционального покрытия в тестировании микропроцессоров // Проблемы разработки перспективных микро- и наноэлектронных систем - 2014. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН, 2014. Часть II. С. 63-68.

[2] P. Mishra, N. Dutt. Functional Coverage Driven Test Generation for Validation of Pipelined Processors. DATE'2005: Design, Automation and Test in Europe, Volume 2, 2005.

[3] Katz, Yoav, Rimon, Michal, Ziv, Avi and Shaked, Gai. "Learning microarchitectural behaviors to improve stimuli generation quality.." Paper presented at the meeting of the DAC, 2011.

[4] Yoav Katz, Michal Rimon, Avi Ziv:A Novel Approach for Implementing Microarchitectural Verification Plans in Processor Designs. Haifa Verification Conference 2012: 148-161.

[5] O. Guzey and et al., Functional test selection based on unsupervised support vector analysis, in Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE, 2008, pp. 262 –267.

[6] Wile B., Goss J., Roesner W. Comprehensive functional verification, the complete industry cycle. Morgan Kaufmann Publishers, 2005.

[7] Yehuda Naveh, Michal Rimon, Itai Jaeger, Yoav Katz, Michael Vinov, Eitan Marcus, and Gil Shurek Constraint-based Random Stimuli Generation for Hardware Verification. In Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence - Volume 2 IAAI'06, 2006, pp. 1720—1727.

[8] Камкин А.С. Комбинаторная генерация тестовых программ для микропроцессоров на основе моделей. Препринт 21, ИСП РАН, 2008, с.23 - 63.

[9] Adir A. et al. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. IEEE Design and Test of Computers, 2004 21(2), pp. 84–93.

[10] Бобков С.Г., Чибисов П.А. Повышение качества тестирования высокопроизводительных микропроцессоров методами встречного тестирования с анализом функционального тестового покрытия выделенных приложений. // Информационные технологии, №8, 2013, с. 26-33.

[11] Ho Kim, J. E. Smith An Instruction Set and Microarchitecture for Instruction Level Distributed Processing // In Proc. of 29th Annual International Symposium on Computer Architecture, 2002, p. 71 – 81.

[12] Lachish O., Marcus E., Ur S., Ziv A. Hole analysis for functional coverage data. // In Proc. DAC – 2002, pp. 807-812.

Methods to improve efficiency of microprocessor model stochastic tests

N.A. Grevtsev[1], P.A. Chibisov[1], I.S. Hisambeev[2]

[1] Scientific Research Institute of System Analysis (SRISA RAS), chibisov@cs.niisi.ras.ru

[2]MIPT

***Keywords* — functional verification, RTL-model of microprocessor, stochastic testing, functional coverage metrics, coverage model, pseudorandom tests generation.**

ABSTRACT

This article describes the method of microprocessor functional verification based on building a functional coverage model with the involvement of testing knowledge. The coverage models are used for measuring, monitoring and changing the direction of stochastic testing process. According to the proposed procedure, coverage metrics at the level of architecture instruction set are involved, analysis of coverage data is carrying out and the efficiency of stochastic testing is evaluated.

REFERENCES

[1] Hisambeev I.Sh., Chibisov P.A. Ob odnom metode postroenija metrik funkcional'nogo pokrytija v testirovanii mikroprocessorov // Problemy razrabotki perspektivnyh mikro- i nanojelektronnyh sistem. 2014. Sbornik trudov / pod obshh. red. akademika RAN A.L. Stempkovskogo. Moscow, IPPM RAN, 2014. Part II. pp. 63-68 (in Russian).

[2] P. Mishra, N. Dutt. Functional Coverage Driven Test Generation for Validation of Pipelined Processors. DATE'2005: Design, Automation and Test in Europe, Volume 2, 2005.

[3] Katz, Yoav, Rimon, Michal, Ziv, Avi and Shaked, Gai. "Learning microarchitectural behaviors to improve stimuli generation quality." Paper presented at the meeting of the DAC, 2011.

[4] Yoav Katz, Michal Rimon, Avi Ziv:A Novel Approach for Implementing Microarchitectural Verification Plans in Processor Designs. Haifa Verification Conference 2012. pp. 148-161.

[5] O. Guzey and et al., Functional test selection based on unsupervised support vector analysis, in Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE, 2008, pp. 262-267.

[6] Wile B., Goss J., Roesner W. Comprehensive functional verification, the complete industry cycle. Morgan Kaufmann Publishers, 2005.

[7] Yehuda Naveh, Michal Rimon, Itai Jaeger, Yoav Katz, Michael Vinov, Eitan Marcus, and Gil Shurek Constraint-based Random Stimuli Generation for Hardware Verification. In Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence. Volume 2 IAAI'06, 2006, pp. 1720-1727.

[8] Kamkin A.C. Kombinatornaja generacija testovyh programm dlja mikroprocessorov na osnove modelej. Preprint 21, ISP RAN, 2008, pp.23-63. (in Russian).

[9] Adir A. et al. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. IEEE Design and Test of Computers, 2004 21(2), pp. 84-93.

[10] Bobkov S.G., Chibisov P.A. Povyshenie kachestva testirovanija vysokoproizvoditel'nyh mikroprocessorov metodami vstrechnogo testirovanija s analizom funkcional'nogo testovogo pokrytija vydelennyh prilozhenij. // Informacionnye tehnologii, No. 8, 2013, pp. 26-33 (in Russian).

[11] Ho Kim, J. E. Smith An Instruction Set and Microarchitecture for Instruction Level Distributed Processing // In Proc. of 29th Annual International Symposium on Computer Architecture, 2002, pp. 71-81.

[12] Lachish O., Marcus E., Ur S., Ziv A. Hole analysis for functional coverage data. // In Proc. DAC. 2002, pp. 807-812.