УДК 004.052.42

Об одном методе построения метрик функционального покрытия в тестировании микропроцессоров

И.Ш. Хисамбеев, П.А. Чибисов

ФГБУН НИИСИ РАН, chibisov@cs.niisi.ras.ru

***Аннотация* — Функциональная верификация является на сегодняшний день неотъемлемой частью процесса разработки микроэлектронных систем, в том числе микропроцессоров. Статья посвящена исследованию и разработке способа оценки полноты тестирования. Обсуждаются такие понятия как задаваемая пользователем метрика тестового покрытия на уровне набора инструкций, тестовая ситуация, модель покрытия. Описан опыт применения предлагаемого подхода к построению метрик.**

***Ключевые слова* — Функциональная верификация, стохастическое тестирование, микропроцессор, метрики тестового покрытия, тестовые ситуации, модель покрытия, генерация случайных тестов.**

I. Введение

Функциональная верификация является на сегодняшний день неотъемлемой частью процесса разработки микроэлектронных систем, в том числе микропроцессоров (*МП*). Под функциональной корректностью МП понимается соответствие между его архитектурой набора команд (*ISA*) и микроархитектурной реализацией. Функциональная верификация МП – это сопряжённый с проектированием процесс проверки его функциональной корректности, причём уровень ISA задан в спецификации, а микроархитектурный уровень реализуется по этой спецификации в виде RTL-модели. Существует и развивается множество подходов к решению этой сложной задачи, но на текущий день основным направлением в индустрии остаётся тестирование, то есть методы поиска ошибок функциональности при динамическом моделировании (*симуляции*).

В области проектирования МП распространено применение следующего маршрута функциональной верификации (рис. 1). В качестве объекта верификации выступает либо RTL-модель ядра МП, включающая АЛУ, блок вещественной арифметики, основную память, кэш-память, блоки управления памятью и потоком выполнения, сопроцессоры и другие блоки, либо модель СнК (системы на кристалле), включающая ядро МП, системный контроллер и различные контроллеры интерфейсов и внешних устройств. Тестовым воздействием является ассемблерная программа (*тест*), сохранённая в образе памяти ПЗУ и ОЗУ, в совокупности с начальным состоянием общих и управляющих регистров, кэш-памяти и буфера трансляции адресов. Её выполнение моделируется симулятором, входящим в состав САПР. Информация о ходе выполнения теста собирается в виде *трасс* – текстовых файлов с перечнем событий определённого вида в модели. Для контроля корректности выполнения используется *поведенческая модель* процессора, написанная на языке высокого уровня (*ISS* – instruction set simulator). Она, как и RTL-модель, способна симулировать выполнение теста и выдавать аналогичные трассы. Возникновение расхождений в трассах моделей означает наличие искомой ошибки.

На этапе разработки RTL-модели до доступности ПЛИС-прототипа или тестового кристалла большая часть ошибок находится методом стохастического тестирования. Он заключается в том, что последовательность тестов генерируется автоматически по заданному шаблону с параметризованным псевдослучайным выбором инструкций тела теста и их аргументов, данных в ОЗУ и кэш-памяти, а также начальных значений регистров [1]. Ключевым преимуществом метода является возможность полной автоматизации процессов генерации и запуска тестов, а также сравнения результатов, без которой немыслимо массивное тестирование МП современной сложности.

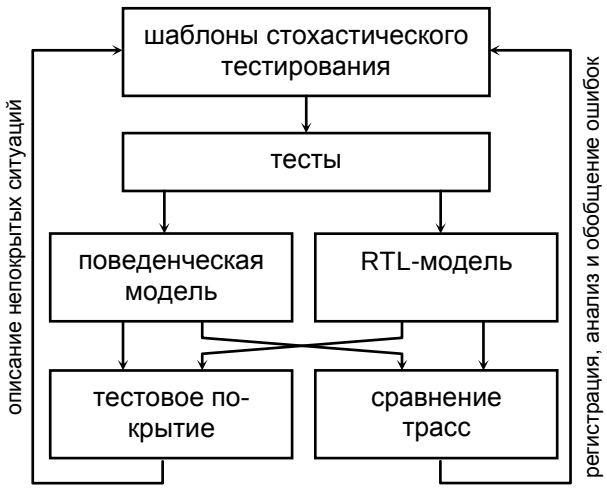

Рис. 1. Маршрут стохастического тестирования

Верификация как процесс контролируется и направляется в соответствии с *планом тестирования*, в котором динамически задаются следующие аспекты:

1) Объекты тестирования: модели МП и их модификации, связанные с основным течением разработки, с

новыми экспериментальными возможностями и с различными конфигурациями моделей.

2) Содержание тестов: способы получения новых тестовых программ и состав наборов регрессионного тестирования.

3) Критерии корректности тестов: требования к поведенческой модели, набор видов трасс для сравнения, требования поведенческой совместимости RTL-модели, методы включения самопроверок в тесты.

4) Контроль полноты тестирования: набор различных критериев и метрик, необходимых для обоснования эффективности тестирования и получения обратной связи для создания новых тестов.

При использовании тестирования как основного подхода к верификации открытым вопросом является выбор объективного и обоснованного критерия полноты тестирования. Существует множество исследований, посвящённых этому вопросу, и разработан ряд способов измерения полноты, успешно применяемых в индустрии. Тем не менее, ни один из них не является самодостаточным. В данной статье предлагается метод контроля полноты тестирования, обладающий рядом ключевых особенностей по сравнению с другими методами, известными по публикациям.

Идея предлагаемого метода – задание семейства моделей функционального покрытия и соответствующих им метрик тестирования, обладающих следующими свойствами:

1) модели не заданы a priori, а создаются и настраиваются инженером-верификатором параллельно с разработкой конкретной RTL-модели и тестов для неё;

2) модели описаны в терминах архитектуры набора команд и покомандного выполнения;

3) разработка моделей покрытия ведётся «от простого к сложному», новые модели опираются на предыдущие, а также на опыт найденных ошибок;

4) итоговое семейство моделей покрывает всю функциональность ISA и микроархитектуры.

II. Понятие метрики тестового покрытия

В тестировании сложной системы, по самой его идее, не ставится цель доказать, что система полностью функционально корректна. Напротив, предполагается, что ошибки в системе есть всегда, и ставится постоянная цель их поиска. Таким образом, в практическом применении тестирования, в частности, в разработке микроэлектронных систем, необходимы внешним образом заданные критерии и оценки степени готовности продукта. Для этого, как правило, применяются разного рода *метрики*, то есть те или иные количественные характеристики качества проведённого тестирования.

В области проектирования МП известен ряд метрик, на совокупность которых ориентируются при принятии решения о готовности проекта к передаче в изготовление. В данной работе мы рассмотрим метрики, характеризующие полноту и достаточность набора применённых тестов, а именно, разнообразные *метрики тестового покрытия*. На практике применяются и метрики другой природы, они могут быть основаны на динамике нахождения ошибок, динамике изменения кода RTL-модели, достигнутой производительности модели и выполнении конкретного перечня требований заказчика [2].

Метрика тестового покрытия – это количественная характеристика полноты (в некотором частном смысле) набора тестов. Общая идея задания метрики такова. Задаётся некоторым систематическим образом перечень утверждений, описывающих *тестовые ситуации*, обладающий двумя свойствами. Во-первых, для каждого теста и для каждого утверждения должно быть определено, выполнялось ли данное утверждение при прохождении данного теста. Во-вторых, перечень должен быть конечным и целостным (полным) в рамках описания ситуаций. Тогда этот набор ситуаций называется *моделью покрытия*. Проводится анализ, какие из тестовых ситуаций были достигнуты при выполнении данного набора тестов. Численное значение метрики покрытия – это отношение числа достигнутых (*покрытых*) ситуаций к общему числу ситуаций в модели. Тестирование считается успешным относительно данной метрики, если это значение достигло 1, то есть все ситуации в модели были протестированы [3].

Существует большое разнообразие известных подходов к описанию тестовых ситуаций и соответствующих им моделей покрытия. Их можно разделить на два типа: модели, основанные на реализации, и модели, основанные на функциональных требованиях (*функциональное покрытие*). В моделях первого типа утверждения о тестовых ситуациях связаны либо непосредственно с кодом RTL-модели МП, либо с выведенными из неё структурами, например, моделями конечных автоматов (FSM). Соответственно, метрика покрытия на основе реализации – это характеристика того, насколько код RTL-модели МП был задействован при прохождении тестов. Примеры метрик покрытия реализации: покрытие строк кода (line coverage), покрытие операторов (statement coverage), покрытие переходов управления (branch coverage), а также покрытие состояний, дуг и путей (state, arc, transition coverage) конечных автоматов [4]. Инструментарий для сбора и анализа покрытия реализации входит в большинство современных САПР и является достаточно простым в применении [5].

В моделях второго типа (модели функционального покрытия) тестовые ситуации задаются явным образом в терминах микроархитектуры МП, архитектуры системы команд или, реже, более высокого уровня абстракции. При этом получаемые модели покрытия соответствуют тому или иному аспекту функциональных требований уровня микроархитектуры или ISA. Приведём примеры.

1) Тестовая ситуация уровня ISA: «последовательность двух инструкций LD (загрузка двойного слова из памяти), использующих один регистр». Соответствующая модель функционального покрытия: «все воз-

можные пары инструкций с учётом всех возможных зависимостей по данным между ними».

2) Тестовая ситуация уровня микроархитектуры: «не выполнен предсказанный переход управления». Соответствующая модель функционального покрытия: «все возможные комбинации состояний и входных воздействий блока предсказания переходов».

Основная сложность в построении таких моделей – перевести словесные описания ситуаций на некоторый формальный язык, что необходимо, во-первых, для организации полной и точной модели покрытия, во-вторых, для автоматизации анализа покрытия этих ситуаций. Для реализации применения моделей функционального покрытия в академических исследованиях и в индустрии было разработано множество инструментов. Обзор некоторых из них приведён в следующей главе. Ясно, что при использовании метрик функционального покрытия для получения уверенности, что в целом проекте нет ошибок, необходимо, чтобы используемые модели в совокупности описывали бы все аспекты функциональных требований. Это является неформализуемой задачей в силу природы описания требований и вызывает наибольшие трудности в применении таких метрик.

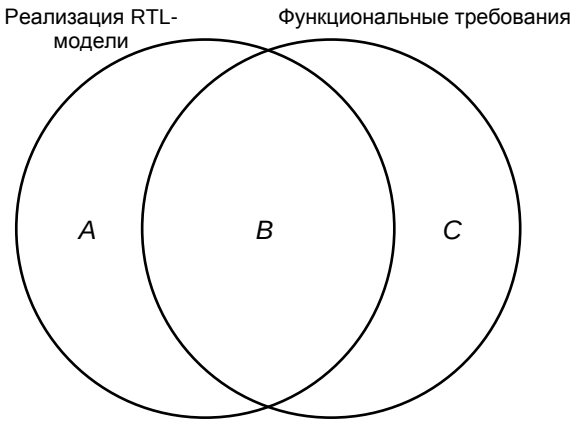

Рис. 2. Отношение реализованных и требуемых элементов функциональности

Важно отметить, что ни одна из метрик тестового покрытия, исследуемая отдельно от других, не даёт ответа на вопрос о полноте набора тестов (рис. 2). Так, например, при 100% покрытии кода модели не исключается случай, когда некоторый требуемый элемент функциональности вообще не был реализован в коде (область *C* не пуста). С другой стороны, даже при 100% покрытии функциональных требований могут оставаться ошибки, связанные с наличием избыточного конфликтующего кода (область *A* не пуста), никак не влияющего на выполнение данного набора тестов, но способного привести к ошибке в реальной работе. Нереалистичный, но простой пример: наличие в RTL-модели кода, выставляющего сигнал перезагрузки процессора по достижению счётчиком инструкций значения 10^{10}. Таким образом, одна из целей функциональной верификации МП – свести к нулю размер областей *A* и *C*, то есть обеспечить полное соответствие реализации требованиям. В силу сказанного, на практике в тестировании используется не единая универсальная метрика, а совокупность максимального числа метрик, доступных для анализа в данном проекте. В частности, метрики покрытия RTL-кода и метрики функционального покрытия используются совместно и являются взаимодополняющими инструментами функциональной верификации.

III. Известные подходы к построению метрик покрытия требований

Рассмотрим подробнее известные по публикациям подходы к созданию моделей покрытия на основе функциональных требований.

В 1990-х годах понятие тестового покрытия, ранее применяемое в разработке ПО, начинает применяться при верификации аппаратуры. Большинство работ посвящено метрикам покрытия HDL-кода и особенностям их применения. Упоминания функционального покрытия редки и сводятся к описаниям отдельных инструментов, привязанных к конкретному проекту [6]. Отдельно упоминается применение контрольных суждений (assertions) – специальных операторов в коде HDL, отслеживающих нежелательные ситуации.

В 1998-1999 годах сотрудники исследовательской лаборатории IBM в Хайфе выпускают ряд статей о верификации МП архитектуры PowerPC. В них выдвигается идея построения моделей функционального покрытия, вводятся термины, описывается универсальный инструмент, позволяющий работать с моделями покрытия, определяемыми пользователем. Создание модели проходит три стадии: определяется значимый для модели набор параметров (комбинация значений которых образует тестовую ситуацию модели), определяется вид анализируемой трассы и способ извлечения значений параметров из неё, выделяются в отдельный класс недостижимые ситуации, которые не будут учтены при подсчёте покрытия [3].

В 2000-х годах растёт популярность подходов к тестированию, в которых корректность выполнения базируется на исчерпывающем наборе контрольных суждений (assertion-based verification) [7]. Крупные производители САПР поддерживают такие подходы методологически и инструментально. Предлагаются «универсальные» методологии функциональной верификации. Модели функционального покрытия описываются на HVL (SystemVerilog, e) и также привязаны к идее контрольных суждений.

В работах [8], [9] подходы, основанные на комбинаторном покрытии, получают своё дальнейшее развитие.

IV. Постановка задачи по разработке метрик покрытия

В известных авторам маршрутах проектирования МП функциональная верификация осуществляется в основном в форме тестирования, в частности, широко используется метод стохастического тестирования. В силу вышеизложенных соображений процесс верифи-

кации представляется разумным контролировать и направлять с помощью различных метрик покрытия. И если метрики покрытия кода собираются и анализируются инструментами, встроенными в используемые САПР, требуя малых усилий, то для метрик функционального покрытия потребовалось выбрать, разработать и применить свой подход.

При выборе подхода к заданию метрик покрытия мы руководствовались следующими требованиями:

1) метрики функционального покрытия должны надёжным образом отображать полноту и тщательность проведённого тестирования;

2) механизм сбора и анализа информации для метрик должен быть максимально совместим с имеющимся маршрутом тестирования (это даст возможности для автоматизации);

3) набор метрик должен быть пригоден для повторного использования в других реализациях проектов схожей архитектуры, в частности, при усовершенствовании продуктов одной серии. Это означает независимость метрик от реализации, в частности, на инструментальном уровне механизмы получения метрик не должны затрагивать HDL-код.

Применение известных механизмов функционального покрытия на основе контрольных суждений в нашем случае затруднено в силу следующих соображений: недостаточная совместимость с имеющимся маршрутом верификации, высокие издержки при включении в маршрут, ограниченные возможности повторного использования в силу привязанности к конкретной реализации.

Рассмотрим подробнее ключевые инструменты, применяемые в маршруте стохастического тестирования: генератор стохастических тестов tergen и поведенческую модель процессора. Основное назначение генератора tergen – построение тестовой программы на основе шаблона и различных настроек случайного выбора. В шаблоне теста и настройках генератора задаются такие факторы, как последовательность команд, аргументы команд, состав, размер и расположение областей памяти теста, подпрограммы обработки прерываний и исключительных ситуаций, а также предоставляется возможность настройки привилегированных ресурсов МП через регистры управляющего сопроцессора. Таким образом, диапазон возможностей tergen охватывает весь уровень архитектуры набора команд, но и, с другой стороны, ограничен им [10].

Диапазон возможностей ISS разрабатываемых МП также, в основном, ограничен ISA. Тем не менее, для совпадения с RTL-моделью по трассам выполнения в ISS моделируются некоторые микроархитектурные аспекты. Кроме того, с точки зрения темпоральных свойств поток управления моделируется покомандно, что является значительным упрощением реализации суперскалярного конвейерного МП.

В силу вышеизложенного модели функционального покрытия разрабатываемого МП строятся на основе ISA. Заметим, что такой подход отвечает идее независимости метрик от реализации.

Основная трудность в выбранном подходе заключается в том, что функциональные требования уровня ISA – это текст, плохо поддающийся исчерпывающей формализации. Поэтому авторами выдвигается идея пользовательского задания метрик на уровне системы команд, что должно обеспечить их последовательное усложнение и усовершенствование по мере хода разработки и верификации.

V. Настраиваемые метрики функционального покрытия уровня архитектуры системы команд

Общая идея в предложенном подходе схожа с таковой в публикациях [3], [11]. Пользователем (инженером-верификатором) создаётся набор метрик путём задания параметризованных моделей.

1) За основу описания тестовой ситуации берётся подпоследовательность выполненных инструкций некоторой длины. Пользуясь терминологией, введённой в [12], назовём такую основу тестовым шаблоном.

2) Далее в модели задаётся набор значимых параметров. К ним относятся: состояние процессора (набор значений управляющих регистров) типы инструкций (фиксированные или выбранные из некоторой группы), операнды инструкций соответствующих типов, полученный результат, значения виртуального и физического адреса для адресных инструкций, параметры TLB и кэш-памяти для инструкций обмена с памятью и другие. Для параметров, диапазон значений которых недопустимо широк, этот диапазон сужается путём разбиения значений на классы.

3) Отдельно рассматриваются параметры, описывающие особые свойства тестовой ситуации. Такими свойствами могут являться зависимости между инструкциями по регистрам, данным, адресам и другим ресурсам, конкурентное выполнение, вызовы прерываний и так далее. Значения параметров такого типа, как правило, можно закодировать одним битом: встретилась данная особенность или нет.

4) Множество ситуаций, составляющих саму модель покрытия, строится как декартово произведение всех диапазонов значений всех выбранных параметров, за вычетом отдельных ситуаций, указанных как недостижимые. Построить подмножество недостижимых ситуаций можно, задав комбинацию условий, одновременное выполнение которых недопустимо.

По мере развития проекта метрики усложняются и детализируются. Численные значения этих метрик собираются в ходе верификации и указывают на её близость к завершению и, соответственно, на степень полноты набора тестов. Отдельно выдаётся информация о случаях достижения ситуаций, описанных как недостижимые. Такое может произойти из-за некорректности теста, некорректности задания модели ситуаций (излишнее ограничение) или функциональной ошибки в модели МП. Информация о непокрытых тестовых ситуациях используется при создании новых тестов и шаблонов стохастического тестирования.

Рассмотрим подробнее ситуации, потенциально являющиеся источниками искомых ошибок. Это ситуации, в которых модель МП достигает некоторого специфического микроархитектурного состояния. Выдвинем предположение: для любой микроархитектурной ситуации, значимой для возникновения специфической ошибки, найдётся достаточно детальная модель покрытия уровня ISA, содержащая ситуацию, гарантированно воспроизводящую эту ошибку. Не претендуя на доказательство, приведём следующее соображение: любая ошибка возникает из начального состояния МП (reset) при выполнении некоторой конечной последовательности инструкций. А последовательности конкретных инструкций можно описать, не выходя за ограничения предлагаемого метода. Значит, существует минимальный набор значимых ISA параметров, воспроизводящий все микроархитектурные подробности, приводящие к искомой ошибке. Практическая задача при этом подходе – построить модель, с одной стороны достаточно детальную, а с другой стороны, содержащую приемлемое число ситуаций.

VI. Примеры и практические результаты

Предлагаемый подход к построению и применению метрик был опробован при разработке 64-разрядного МП, ведущейся в НИИСИ РАН, без нарушения устоявшейся процедуры тестирования, а дополнительно к ней. До разработки универсального способа описания моделей и инструмента обработки таких описаний было предложено несколько частных примеров моделей покрытия и реализованы простые инструменты, собирающие и анализирующие такие покрытия по отдельности. Покрытия этих моделей были измерены на имеющейся базе регрессионного тестирования. Были определены непокрытые ситуации, что позволило выработать рекомендации по созданию новых тестов и доработке шаблонов стохастического тестирования.

Пример 1. Четверки типов инструкций, перебор «каждый за каждым»

В этой модели покрытия весь набор команд данной архитектуры разбивается на N типов, сообразно их близости с точки зрения управляющей логики МП. Так, к одному типу можно причислить, например, команды *add* и *sub*, поскольку при их выполнении задействуются те же блоки МП. Чем больше N, тем более детальной является модель покрытия. Тестовая ситуация – это последовательность из 4 выполненных команд. Ситуации считаются различными, если упорядоченные четвёрки соответствующих им типов различны. Общее число ситуаций равно N^4. Длина последовательности 4 была выбрана в силу того, что предвыборка в буфере инструкций выполняется по 4 инструкции. Таким образом, модель неявно привязана к микроархитектуре МП, несмотря на то, что её описание ограничено терминами ISA.

Заметим, что некоторые инструкции, например, инструкции работы с привилегированными ресурсами или вызовы системных исключений, можно опустить из рассмотрения, так как они либо не создаются компилятором, либо создаются крайне редко и в строго определенных ситуациях.

Какие-либо параметры инструкций в данной модели опускаются из рассмотрения. Кроме того, имеется одно условие, описывающее недостижимые ситуации: расположение двух инструкций ветвления подряд в четвёрке. Это связано с тем, что в рассматриваемой архитектуре каждая команда перехода имеет слот задержки: следующая команда выполняется параллельно с переходом.

Приведём результаты сбора покрытия на фиксированном наборе тестов (1201 тест, около $6{,}4 \cdot 10^7$ инструкций суммарно) для моделей с различными N.

Таблица 1

Результат покрытия четвёрок типов инструкций

N	Достигнуто	Всего	Покрытие
5	560	560	100%
7	2231	2268	98,37%
10	5927	8960	66,15%
14	13303	33880	39,27%

Пример 2. Пары инструкций загрузки/сохранения

Тестовая ситуация в этой модели – это две идущие подряд команды загрузки или сохранения, рассматриваемые вместе со следующими параметрами работы с кэш-памятью: политика кэширования при обращении к данным (*pol*), промах/попадание в кэш-память данных 1 уровня (*L1hit*) и 2 уровня (*L2hit*). Каждую команду, таким образом, можно закодировать последовательностью 5 бит, где 1 бит кодирует тип команды (загрузка или сохранение, *load/store*), 2 бита – *pol*, 1 бит – *L1hit*, 1 бит – *L2hit*. Пара команд, т.е. тестовая ситуация, кодируется последовательностью 10 бит, и, соответственно, модель покрытия состоит из 1024 ситуаций.

Отметим тестовые ситуации, которые окажутся недостижимыми при выполнении одного из следующих условий для любой из двух команд:

1) $(pol{=}0 \vee pol{=}1) \wedge L2hit$

не должно быть попаданий в кэш-память 2 уровня при политике обхода кэш-памяти 2 уровня;

2) $pol{=}2 \wedge (L1hit \vee L2hit)$

не должно быть попаданий в кэш-память при политике некэшируемых обращений;

3) $(load \wedge pol{=}3 \wedge L1hit) \wedge L2hit$

попадание команды загрузки в кэш-память 1 уровня при политике кэширования с обратной записью обращение в кэш-памяти 2 уровня не производится (а значит, не может быть L2hit).

Результат сбора покрытия на том же наборе тестов, что и в примере 1, показан на рис. 3. Каждая строка соответствует значению параметров первой команды из пары, каждый столбец – второй команде. Достигнутые ситуации обозначены 1, недостигнутые – 0, недостижимые – символом «x».

```
0:  1x1x1x1x1xxx101x1x1x1x1x1xxx1011
1:  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2:  1x1x1x1x1xxx011x1x1x1x1x1xxx1111
3:  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4:  1x1x1x1x1xxx001x1x1x1x1x1xxx0000
5:  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
6:  1x1x1x1x1xxx101x1x1x1x1x1xxx1011
7:  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
8:  1x1x1x1x1xxx111x1x1x1x1x1xxx1111
9:  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
10: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
11: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
12: 1x1x1x1x1xxx111x1x0x0x0x1xxx1111
13: 0x1x0x1x1xxx111x0x1x0x0x1xxx1111
14: 1x1x1x1x1xxx111x1x1x0x1x1xxx1111
15: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
16: 1x1x1x1x1xxx111x1x1x1x1x1xxx1101
17: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
18: 1x1x1x1x1xxx111x1x1x1x1x1xxx1011
19: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
20: 1x1x1x1x1xxx111x1x1x1x1x1xxx1000
21: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
22: 1x1x1x1x1xxx111x1x1x1x1x1xxx1011
23: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
24: 1x1x1x1x1xxx111x1x1x1x1x1xxx1111
25: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
26: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
27: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
28: 1x1x1x1x1xxx111x0x1x1x1x1xxx1111
29: 1x1x1x1x1xxx111x0x1x0x0x1xxx1111
30: 1x1x1x1x1xxx111x0x1x1x1x1xxx1111
31: 1x1x1x1x1xxx111x1x1x1x1x1xxx1111
reached 258
unreached 31
```

Рис. 3. Визуализация модели покрытия пары инструкций загрузки/сохранения

VII. Выводы и направление дальнейшей работы

Преимущества, которые даёт предложенный метод построения метрик, заключаются в следующем. Во-первых, метод позволяет получить набор метрик для повторного использования в других проектах сходной архитектуры. Во-вторых, возможность гибкого описания моделей покрытия позволяет дать подходящие критерии полноты тестирования на любой стадии разработки МП. Например, можно описывать тестовые ситуации, направленные на определённый блок. В-третьих, описание тестовых ситуаций в терминах, использующихся в других инструментах тестирования, облегчает интеграцию метода в имеющийся маршрут тестирования, а также даёт перспективы автоматизации некоторых путей в этом маршруте (рис. 1). Так, например, описание непокрытых набором тестов ситуаций является фактически готовым заданием на недостающий тест или рекомендацией к разработке шаблона стохастических тестов. С другой стороны, представление ситуации в виде трассы выполненных инструкций при анализе найденных ошибок позволяет легко дать описание тестовой ситуации, которая, возможно, ещё не задана в имеющемся наборе моделей покрытия, и, соответственно, пополнить этот набор новой практически значимой моделью на основе ошибочной ситуации.

Предложенный подход позволил дополнить маршрут тестирования, использующий ранее лишь метрики покрытия RTL-кода, что привело к увеличению эффективности тестирования МП.

Направления дальнейшего развития предложенного метода представляются следующими:

1) Обобщение описания метрик, выработка языка или табличного метода задания комбинаторики параметров.

2) Разработка универсального инструмента анализа трасс поведенческой модели (ISS) для выявления тестовых ситуаций.

3) Автоматизация обратной связи для построения шаблонов стохастического тестирования.

4) Доказательство предположения, что любая потенциальная ошибка в МП может быть описана, как тестовая ситуация в рамках некоторой модели покрытия.

5) Построение набора метрик, в котором выражается накопленное тестовое знание для верифицируемых МП.

Литература

[1] Грибков И.В., Захаров А.В., Кольцов П.П., Котович Н.В., Кравченко А.А., Куцаев А.С., Осипов А.С., Хисамбеев И.Ш., Коганов М.А. Развитие системы стохастического тестирования микропроцессоров INTEG // Программные продукты и системы. 2010. № 2. С. 14-23.

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

[3] Grinwald R., Harel E., Orgad M., Ur S., Ziv A. User defined coverage – a tool supported methodology for design verification // In Proceedings of the 35th Design Automation Conference. 1998. P. 158-163.

[4] Piziali A. Functional Verification Coverage Measurement and Analysis. New York: Kluwer Academic Publishers. 2004. 216 p.

[5] URL: http://www.verificationacademy.com/verification-methodology (дата обращения: 25.01.2014).

[6] Ho R.C., Horowitz M.A. Validation Coverage Analysis for Complex Digital Designs // In Proceedings of International Conference Computer-Aided Design (ICCAD 96). IEEE CS Press. 1996. P. 146-153.

[7] Schutten R., Fitzpatrick T. Design for verification blueprint for productivity and product quality // Synopsys Technical Papers. 2003. 13 p.

[8] Kuhn D.R., Kacker R.N., Lei Yu Combinatorial Coverage Measurement // NIST Special Publication 800-142. 2012. 10 p.

[9] Jerinic V., Langer J., Heinkel U., Müller D. New Methods and Coverage Metrics for Functional Verification // In Proceedings of Design, Automation and Test in Europe (DATE). 2006. P. 1025-1030.

[10] Хисамбеев И.Ш. Роль стохастического тестирования в функциональной верификации микропроцессоров // Программные продукты и системы. 2012. № 3. С. 107-112.

[11] Chen W., Wang Li-C., Abadir M. Simulation Knowledge Extraction and Reuse in Constrained Random Processor Verification // In Proceedings of the 35th Design Automation Conference. 2013. P. 1-6.

[12] Камкин А.С. Генерация тестовых программ для микропроцессоров // Труды ИСП РАН. 2008. С. 23-63.