Обзор методов и средств генерации тестовых программ для микропроцессоров

А.Д. Татарников <andrewt@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В работе дается обзор существующих методов и средств генерации тестовых программ для микропроцессоров. Генерация тестовых программ и анализ результатов их выполнения являются основным подходом к функциональной верификации микропроцессоров. Этот подход также принято называть тестированием. Несмотря на то, что методы генерации тестовых программ непрерывно совершенствуются, тестирование остается крайне трудоемким процессом. Одна из основных причин состоит в том, что средства генерации не успевают адаптироваться к изменениям. В большинстве случаев они созданы под конкретные типы микропроцессоров и предназначены для решения конкретных задач. Поэтому поддержка новых типов микропроцессоров и новых методов генерации требует значительных усилий. Часто в таких случаях приходится создавать новую реализацию с нуля. Невозможность повторного использования имеющейся реализации затрудняет развитие средств генерации и, как следствие, препятствует улучшению качества тестирования. Текущее положение дел создает мотивацию для поиска решений, позволяющих создавать более гибкие средства генерации, которые могли бы быть с минимальными усилиями адаптированы для тестирования новых типов микропроцессоров и применения новых методов генерации. Цель данной работы – обобщить имеющийся опыт генерации тестовых программ, который мог бы послужить основой для создания таких средств. В работе рассматриваются сильные и слабые стороны распространенных методов генерации, области их применения и варианты их совместного использования. Также делается сравнительный анализ возможностей существующих средств генерации, реализующих эти методы. На основе данного анализа даются рекомендации по созданию универсальной методологии разработки средств генерации тестовых программ для микропроцессоров.

Ключевые слова: микропроцессоры; цифровая аппаратура; функциональная верификация; тестирование; генерация тестовых программ.

DOI: 10.15514/ISPRAS-2017-29(1)-11

Для цитирования: Татарников А.Д. Обзор методов и средств генерации тестовых программ для микропроцессоров. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 167-194. DOI: 10.15514/ISPRAS-2017-29(1)-11

1. Введение

Микропроцессор [1] является центральным элементом любого электронного устройства. В то время как другие компоненты отвечают за ввод и вывод данных, роль микропроцессора состоит в обработке этих данных: он определяет, какие операции должны быть выполнены над данными и контролирует процесс их выполнения. Микропроцессоры состоят из множества транзисторов, объединенных в единый вычислительный элемент на полупроводниковом кристалле. Число транзисторов постоянно увеличивается и в настоящее время достигает нескольких миллиардов [2].

Микропроцессоры являются сложными устройствами, поэтому их производство требует тщательного проектирования. В настоящее время для проектирования применяются специализированные языки, известные как *языки описания аппаратуры* (*HDL, Hardware Description Languages*), которые с предельной точностью позволяют описывать структуру и поведение микропроцессора. Результатом проектирования является *модель уровня регистровых передач* (*RTL, Register Transfer Level*), также называемая *HDL-моделью*, которая затем посредством логического и физического синтеза преобразуется в представление, используемое для производства интегральных схем.

Высокая сложность современных микропроцессоров приводит к неизбежности существования ошибок. Главной причиной этого является большое число комбинаций состояний микропроцессора и их взаимозависимостей. Дальнейшее усложнение, продиктованное погоней за повышением производительности, влечет за собой рост числа ошибок. Например, по данным на август 2008-го года в микропроцессоре Intel Pentium 4 было найдено 104 ошибки, из которых 43 не исправлено и их исправление не запланировано [3]. В то же время в микропроцессоре Intel Core i7-900 по данным на февраль 2015-го года обнаружено 167 ошибок, из которых исправлено только 16 и еще для 2 запланировано исправление [4]. Следует понимать, что в обоих случаях речь идет лишь об известных проблемах, в то время как реальное число ошибок может быть гораздо выше.

Цена ошибок в микропроцессорах очень высока. В отличие от дефектов в программах, которые устраняются сравнительно просто, ошибки в микропроцессорах не могут быть исправлены и для их устранения потребуется повторный выпуск и замена микросхемы или целого блока. Например, в 1994-м году замена продукции из-за ошибки в реализации инструкции FDIV микропроцессора Pentium обошлась компании Intel в 475 миллионов долларов [5].

Обеспечение функциональной корректности микропроцессоров является фундаментальной проблемой, для решения которой применяется комплекс мер, известный как *функциональная верификация* [6]. Верификация выполняется параллельно с проектированием, и ее задача заключается в проверке соответствия результатов, полученных на текущем этапе, заданным

требованиям и ограничениям. Верификация может осуществляться как на уровне отдельных модулей микропроцессора (*модульная верификация*), так и для устройства в целом (*системная верификация*).

Существуют два основных подхода к функциональной верификации: (1) *имитационная верификация* (simulation-based verification), также называемая *тестированием*, и (2) *формальная верификация* (formal method-based verification) [7]. Первый заключается в проверке корректности реакции проектируемого устройства, работа которого имитируется при помощи программного или аппаратного симулятора, на тестовые воздействия. Второй основан на построении математической (формальной) модели и проверке выполнимости ее свойств. Формальная верификация позволяет осуществить исчерпывающую проверку всех возможных вариантов поведения, заданных моделью. Однако она имеет ряд ограничений (высокая трудоемкость и вычислительная сложность, а также трудности формализации), которые затрудняют ее использование в промышленных проектах. По этой причине на практике основной акцент делается на тестирование [8].

На системном уровне тестирование осуществляется путем генерации *тестовых программ* на языке ассемблера, которые представляют собой последовательности команд (инструкций) микропроцессора, и анализа трасс их выполнения с целью убедиться, что их поведение соответствует спецификации. Такой подход видится наиболее естественным, т.к. система команд определяет функциональность микропроцессора и является единственным доступным интерфейсом, через который пользователи могут с ним взаимодействовать.

Тестовые программы создаются при помощи специальных программных средств, известных как *генераторы тестовых программ*, которые используют широкий набор методов генерации для обеспечения максимального покрытия тестируемой функциональности. Методы генерации эволюционировали по мере усложнения микропроцессоров от случайной генерации до нацеленной генерации, основанной на формальных методах (известны как *гибридные* методы [9]). Применяемые в настоящий момент методы можно разделить на следующие основные группы [10]: (1) *случайная генерация*; (2) *комбинаторная генерация*; (3) *генерация на основе ограничений* и (4) *генерация на основе моделей*. Важно понимать, что каждый метод имеет свою область применения и ни один из них не является универсальным решением для всех задач верификации. На практике применяется целый набор методов, взаимно дополняющих друг друга. Этот набор может пополняться по мере изобретения новых методов.

В общем случае генерация осуществляется на основе *тестовых шаблонов*, абстрактных описаний тестовых программ, которые задают используемые инструкции, их порядок и входные данные. Способы построения последовательностей инструкций и генерации тестовых данных зависят от методов генерации и используемых ими настроек.

Реализация инструментов генерации, основанных на различных методах, может существенно отличаться. Методы, позволяющие сгенерировать более точные тестовые воздействия, требуют большего количества информации о тестируемом микропроцессоре. Например, если методам случайной генерации, как правило, нужна только информация о синтаксисе команд, то для генерации тестов для подсистемы памяти необходимо знание о компонентах подсистемы памяти и семантике команд чтения и записи [11]. Это затрудняет интеграцию методов в единый инструмент. В результате, на практике для тестирования одного микропроцессора может применяться несколько инструментов, реализующих различные методы и использующих описания тестируемого микропроцессора разной степени детализации. Это отрицательно влияет на эффективность тестирования, т.к. поддержка нескольких инструментов, использующих различные форматы описания конфигурации, требует дополнительных усилий. Кроме того, совместное использование различных методов для решения общей задачи генерации часто оказывается невозможным.

Другая проблема, которая часто возникает, – это адаптация инструмента к изменениям в архитектуре тестируемого микропроцессора и поддержка новых архитектур. Многие инструменты генерации разработаны под конкретный микропроцессор и не предусматривают возможность подобных изменений (логика генерации тесно связана с конфигурацией). В таких случаях требуется внесение серьезных изменений в реализацию инструмента, что влечет за собой значительные трудозатраты.

Цель данной работы – проанализировать задачи, решаемые существующими генераторами тестовых программ, и присущие им проблемы. На основании этой информации будут сформулированы основные требования, которым должен удовлетворять универсальный инструмент генерации.

Оставшаяся часть статьи организована следующим образом. В разд. 2 рассказывается о том, как осуществляется верификации при помощи тестовых программ. В разд. 3 дается обзор существующих методов генерации. Раздел 4 содержит описание подхода к тестированию и инструмента генерации, используемого в НИИСИ РАН. Разд. 5 посвящен описанию инструментов, разрабатываемых и применяемых в компании ARM. В разд. 6 дается обзор инструментов IBM Research. Разд. 7 содержит краткий анализ методов и инструментов генерации, созданных в других компаниях. В разд. 8 делаются выводы и формулируется требования, которым должен удовлетворять универсальный инструмент генерации. Разд. 9 завершает статью.

2. Верификация при помощи тестовых программ

Верификация при помощи тестовых программ предполагает их выполнение на HDL-модели и последующий анализ результатов. Существуют два основных подхода к проверке корректности поведения HDL-модели [6]: (1) сравнение

трасс выполнения с эталонными трассами и (2) использование *встроенных проверок* (*self-checks*).

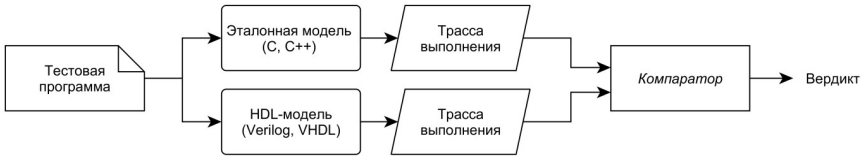

Рис. 1. Тестирование посредством сравнения трасс

Fig 1. Testing by comparing the traces

В первом случае (см. Рис. 1) при выполнении программы на HDL-модели создается трасса выполнения, фиксирующая события, которые происходят микропроцессоре. Полученная трасса сравнивается с эталонной трассой, полученной в результате выполнения той же программы на *эталонной модели (reference model)*. Для сравнения трасс используются специальные программы, называемые *компараторами*. Эталонная модель создается независимо от HDL-модели на языке высокого уровня (например, C или C++) и является более абстрактной. Часто такая модель представляет собой *симулятор уровня инструкций (instruction-level simulator)*. Более абстрактная модель, как правило, содержит меньшее количество ошибок, а тот факт, что она разрабатывается независимо, уменьшает вероятность повторения ошибок, допущенных при создании HDL-модели. Данный подход позволяет универсальным образом осуществлять проверку корректности поведения микропроцессора при выполнении различных тестовых программ. Таким образом, отпадает необходимость реализовывать проверки для отдельных тестов. К недостаткам подхода, можно отнести то, что трассы, полученные при помощи упрощенной эталонной модели, могут не отражать все события, которые происходят в HDL-модели (и реальном процессоре). Это приводит к трудностям при сопоставлении трасс.

Второй подход предполагает, что код тестовой программы содержит проверки, которые необходимо осуществить в процессе ее выполнения. Такие программы можно выполнять не только на HDL-моделях, но и на аппаратных ускорителях, ПЛИС-прототипах и экспериментальных образцах микросхем. Это позволяет повысить производительность, так как время выполнения программы на аппаратных симуляторах существенно меньше, чем на программных. Однако при таком подходе снижается точность проверки, так как множество событий, которые может фиксировать тестовая программа, ограничено.

Решение о завершении верификации принимается на основе набора критериев, который включает в себя следующее [12]:

- выполнение плана верификации;
- отсутствие ошибок при прогоне регрессионных тестов;

- отсутствие ошибок на 10^5 псевдослучайных тестов для каждого из имеющихся тестовых шаблонов;
- достижение заданного уровня покрытия HDL-кода и функционального покрытия;
- отсутствие изменений в HDL-коде в течение длительного времени (3-4 недели);
- отсутствие новых ошибок в течение определенного времени (2-3 недели);
- истечение отведенного согласно плану времени на разработку и верификацию.

Особого внимания заслуживают критерии, основанные на достижении требуемого уровня покрытия. Достигнутый уровень покрытия оценивается при помощи *метрик тестового покрытия* [13]. Они представляют собой количественные характеристики полноты покрытия выбранной *модели тестового покрытия,* которая описывается как конечный набор *тестовых ситуаций*. Числовое значение метрики покрытия представлено как отношение числа достигнутых при выполнении теста ситуаций к общему числу ситуаций в модели. Выделяют два типа моделей покрытия: (1) основанные на реализации и (2) основанные на функциональных требованиях.

В моделях первого типа тестовые ситуации связаны непосредственно с кодом HDL-модели или с производными от нее структурами. Т.е. метрики покрытия на основе реализации позволяют оценить, в какой мере код RTL-модели был задействован при выполнении набора тестов. Примерами таких метрик являются: 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 [14]. Большинство современных средств автоматизированного проектирования предоставляют инструментарий для сбора и анализа данных о достигнутом покрытии реализации.

В моделях второго типа, известных как *модели функционального покрытия*, тестовые ситуации задаются в терминах абстракций микроархитектуры, архитектуры микропроцессора или абстракций более высокого уровня. Такие модели создаются на основе спецификаций функциональных требований к микропроцессору различного уровня. Они могут создаваться вручную или посредством анализа *формальных спецификаций* [15].

Важно отметить, что ни одна из метрик тестового покрытия, взятая в отдельности не даёт ответа на вопрос о полноте набора тестов. Поэтому на практике используются комбинации различных метрик, основанных как на реализации, так и на функциональных требований.

Для генерации тестов, предназначенных для достижения требуемого уровня покрытия для выбранных метрик, применяются разнообразные методы генерации тестовых программ, реализованные различными инструментами.

3. Методы генерации тестовых программ

Большинство современных инструментов в той или иной степени полагаются *случайную генерацию* [16]. Однако применяемые методы генерации могут существенно отличаться степенью *нацеленности*. Под нацеленностью следует понимать ориентированность на покрытие конкретных тестовых ситуаций или классов тестовых ситуаций. Следует заметить, что т.к. генерация тестовых программ предполагает повторяемость результатов, алгоритмы случайной генерации, как правило, основаны на детерминированных генераторах псевдослучайных чисел.

В простейшем случае инструкции и их входные значения выбираются случайным образом [17]. Инструмент, решающий подобную задачу, можно разработать достаточно быстро, и он позволит сгенерировать набор тестов, обеспечивающих базовый уровень покрытия. К сожалению, такой подход не является систематическим и не позволит достичь полного покрытия. Подобные инструменты не располагают сведениями о семантике инструкций и особенностях реализации микроархитектуры, поэтому не гарантируют достижения всех интересных с точки зрения тестирования ситуаций, а также корректности построенной программы (отсутствие зацикливаний, переходов на секцию данных, нарушений инвариантов инструкций, и т.д.).

Первый шаг в направлении повышения нацеленности случайных тестов – ограничение области допустимых значений для случайного выбора и присвоение *весов* выбираем вариантам [18]. Такой подход предполагает описание задач генерации в виде шаблонов, которые задают списки инструкций для выбора (в том числе иерархические), веса отдельных элементов и области значений их входных аргументов. Это позволит более тщательно протестировать отдельные инструкции или классы инструкций. Веса и области значений подбираются на основе полученных значений метрик тестового покрытия.

Следующий шаг в сторону улучшения покрытия предполагает использование методов *комбинаторной генерации* [19]. Такой подход позволит нацелить генератор на тестирование ситуаций, связанных с совместным выполнением инструкций на конвейере. Суть метода заключается в систематическом переборе коротких последовательностей инструкций (включая зависимости между ними). Помимо этого, можно получать входные аргументы инструкций путем перебора данных из некоторого предопределенного набора. Это позволит достичь лучшего покрытия *пограничных случаев (corner cases)*. Путем совместного использования методов случайной и комбинаторной генерации можно построить более сложные тесты, которые помогут покрыть многие маловероятные и трудновообразимые ситуации в работе микропроцессора.

Для проверки поведения микропроцессора в конкретных ситуациях часто прибегают к *детерминированной симуляции (deterministic simulation)* [20]. Идея заключается в создании и симуляции тестовых программ с детерминированным поведением, включающих встроенные проверки. Изначально такие программы разрабатывались вручную, что делало их создание крайне трудоемкой задачей. Автоматическая генерация таких программ предполагает симуляцию инструкций в процессе генерации и использование результатов для создания проверок. Симуляция в процессе генерации также позволяет гарантировать корректность построенных программ. При данном подходе программы стоятся по шаблонам, которые описывают тестовый сценарий. Симуляцию генератор осуществляет самостоятельно или использует для этого внешние эталонные модели. Построенные таким образом тесты можно считать полностью нацеленными. Основным недостатком данного подхода остается трудоемкость создания тестов, которая сопоставима с ручной разработкой. Как и при ручной разработке, чтобы создать код, покрывающий нетривиальные ситуации в работе микропроцессора, от разработчика требуется глубокое знание особенностей микроархитектуры. Кроме того, есть риск, что некоторые ситуации будут пропущены. С целью улучшения покрытия и уменьшения трудозатрат шаблоны могут использовать рандомизацию или комбинаторный перебор. Например, построение тестовых программ может осуществляться путем перебора разнообразных графов потока управления (*структур переходов*), описанных в шаблоне, и маршрутов в них (*трасс выполнения*) [21].

Построение нацеленных тестов можно упростить при помощи *генерации на основе ограничений* [22]. При таком походе в шаблонах указываются ограничения, которым должны удовлетворять операнды инструкций. Ограничения соответствуют тестовым ситуациям, основанным на особенностях реализации микроархитектуры или на функциональных требованиях. В процессе обработки шаблона генератор, используя некоторый механизм разрешения ограничений, который зависит от их типа, строит случайные значения, удовлетворяющие заданным ограничениям. Использование ограничений позволяет значительно сократить трудозатраты, связанные с подбором значений операндов, требуемых для покрытия тестовых ситуаций. Однако, т.к. тестовые шаблоны по-прежнему разрабатываются вручную, остается риск упустить важную для тестирования ситуацию.

Систематическое тестирование может быть обеспечено путем создания *формальных моделей* тестируемого микропроцессора [23] и применения к ним методов проверки моделей для построения тестов, охватывающих все пространство состояний. Такой подход называют *генерацией на основе моделей*. Т.к. современные микропроцессоры имеют огромное количество состояний и переходов между ними, у этого метода будут высокие вычислительные затраты. Для их сокращения можно использовать тестовые шаблоны, которые будут задавать направление тестирования. К недостаткам данного метода относятся высокая трудоемкость его реализации (пока ведутся

исследования, но нет промышленных инструментов), а также трудности создания формальных моделей.

Как можно заметить, перечисленным методам требуется разное количество входных данных. В табл. 1 приводится список методов и требуемых для них входных данных в порядке увеличения их количества.

Табл. 1. Методы генерации и требуемые входные данные

Table 1. Methods of generation and required input data

Метод генерации	**Входные данные**
Случайная генерация	Ассемблерный формат инструкций; Тестовые шаблоны (списки инструкций, веса)
Комбинаторная генерация	Все выше перечисленное; Тестовые шаблоны (правила комбинирования)
Генерация детерминированных тестов со встроенными проверками	Все выше перечисленное; Тестовые шаблоны (описание тестовых сценариев); Семантика инструкций (эталонная модель)
Генерация на основе ограничений	Все выше перечисленное; Тестовые шаблоны (задание ограничений); Модели покрытия (наборы ограничений) различного типа
Генерация на основе моделей	Все выше перечисленное; Формальные модели микропроцессора

Перечисленные методы по-разному реализованы в различных инструментах генерации. Отдельный инструмент может поддерживать один или несколько методов. Особенности реализации методов в конкретных инструментах будут рассмотрены в последующих разделах.

4. Инструменты НИИСИ РАН

В НИИСИ РАН разрабатывается система INTEG [18][24], предназначенная для тестирования микропроцессоров MIPS64 [25]. Используемый ею подход к тестированию называется *стохастическим*. Он предполагает генерацию тестовых программ по заданному шаблону с параметризированным случайным выбором инструкций и их аргументов. После этого осуществляется симуляция сгенерированных программ на HDL-модели и на эталонной модели, в качестве которой выступает симулятор VMIPS [26], и сравнение полученных результатов.

Генератор тестовых программ системы INTEG поддерживает случайные и комбинаторные методы генерации. Тестовые шаблоны для него разрабатываются на специализированном языке, конструкции которого основаны на синтаксисе языка C. Система предоставляет графический интерфейс, который упрощает их разработку. Шаблоны задают последовательность фрагментов кода в тестовой программе, состав входящих в них инструкций и аргументы этих инструкций. Все параметры, оставленные свободными, генератор выбирает случайно, в соответствии с заданными для

них вероятностями. Основные возможности, предоставляемые языком описания тестовых шаблонов, включают: (1) задание областей памяти для кода и для данных; (2) описание последовательностей инструкций (в том числе конструкции для описания циклов); (3) выбор инструкций; (4) выбор регистров, используемых в качестве аргументов инструкций; (5) задание значений аргументов инструкций; (6) задание адресов данных и адресов передачи управления; (7) описание правил генерации случайных значений; (8) создание макросов для повторного использования шаблонного кода. При разработке тестовых шаблонов для системы INTEG руководствуются планом тестирования и метриками покрытия. Для устранения пробелов в покрытии изменяются настройки шаблонов, такие как степень случайности выбора, область допустимых значений, т.д.

В процессе генерации система INTEG отслеживает состояние микропроцессора, используя упрощенный подход. При таком подходе симуляция инструкций не осуществляется, а значение регистров в некоторой точке выполнения считается известным или неопределенным. При необходимости генератор обновляет значения регистров и обеспечивает защиту от записи в регистры, часто используемые для чтения.

Тестовые шаблоны, сгенерированные тестовые программы и отчеты о тестировании можно хранить в специальной базе данных. Это позволяет объединять информацию о работе нескольких экземпляров системы INTEG, запущенных на разных компьютерах.

Система INTEG предоставляет достаточно мощный и удобный инструментарий для создания случайных тестов. Средства разработки тестовых шаблонов, генерации тестовых программ, выполнения тестирования, а также анализа и хранения его результатов интегрированы в единую систему. Однако данная система имеет ряд ограничений:

- Поддерживаются только случайная и комбинаторная генерация. Остается неясно, какой уровень покрытия можно обеспечить этими методами и насколько сложно разработать шаблоны, удовлетворяющие требованиям по тестовому покрытию.
- Не предусмотрено добавление пользовательских методов генерации и пользовательских конструкций в язык описания тестовых шаблонов.
- В процессе генерации построенные инструкции не симулируется. Таким образом, генератор не может отслеживать состояния микропроцессора в любой точке выполнения тестовой программы. Эта информация необходима для контроля корректности сгенерированных программ и создания встроенных проверок. Кроме того она требуется методам генерации, основанным на разрешении ограничений. Таким образом, реализация перечисленных возможностей столкнется со значительными трудностями.

- Поддерживается только архитектура MIPS64. В работе [24] есть упоминание о возможности настройки под другие архитектуры. Однако неясно, как такая настройка будет осуществляться и каких трудозатрат она потребует.

5. Инструменты ARM

5.1 Инструменты RIS

В компании ARM [27] разработано семейство инструментов генерации тестовых программ, получившее название RIS (Random Instruction Sequence) [28][29]. Это узкоспециализированные инструменты, предназначенные для решения различных задач тестирования микропроцессоров с архитектурой ARM. Решаемые ими задачи включают тестирование таких механизмов, как многоядерность [30] и управление памятью [31]. Настройка инструментов RIS осуществляется при помощи конфигурационных файлов, которые задают используемые инструкции, их веса, ограничения на их операнды, способы размещения кода и данных в памяти, а также цепочки инструкций, решающих специальные задачи (вытеснение данных из кэш-памяти и т.п.). Инструменты RIS, описанные в работах [30] и [31], не используют эталонные модели и не осуществляют симуляцию инструкций в процессе генерации. В тех случаях, когда требуются встроенные проверки, тесты выполняются дважды и полученные результаты сравниваются (применимо только для детерминированных результатов). Такой подход упрощает разработку инструмента и делает возможным его использование непосредственно на ПЛИС-прототипе или экспериментальном образце микропроцессора. К сожалению, доступная информация об инструментах RIS не позволяет в полной мере оценить их функциональные возможности. Из того, что известно можно сделать вывод, что основным ограничением инструментов RIS является то, что они жестко привязаны к архитектуре ARM и ориентированы на решение конкретных задач. Поддержка других архитектур и методов генерации в них не предусмотрена.

5.2 Инструмент RAVEN

Другим инструментом, используемым в компании ARM, является RAVEN (Random Architecture Verification Machine) [32][33][34], разработанный в компании Obsidian Software, поглощенной ARM в 2011 году. Это универсальное средство, применимое для широкого спектра архитектур, в котором ядро, реализующее логику генерации, отделено от конфигурации для конкретной архитектуры. RAVEN позволяет создавать как случайные, так и нацеленные тесты. В процессе работы инструмент отслеживает состояние микропроцессора путем симуляции построенных программ на внешней эталонной модели. Это позволяет гарантировать корректность построенных

программ и использовать информацию о текущем состоянии микропроцессора для генерации тестов.

Конфигурация тестируемого микропроцессора задается в виде XML-файлов. Данные файлы содержат следующую информацию об архитектуре тестируемого микропроцессора: (1) перечень поддерживаемых инструкций и их групп, организованный в виде дерева; (2) сигнатуры инструкций (ассемблерный синтаксис, двоичная кодировка); (3) операнды инструкций, их свойства, используемые ими режимы адресации и правила их группировки; (4) семантика инструкций, представленная в виде формул; (5) ресурсы микропроцессора, к которым обращаются инструкции. Кроме того там также описываются специальные условия такие, как исключения, ситуации в работе конвейера инструкций и подсистемы памяти. Эти описания используются генератором для построения тестовых программ.

Другой важный аспект конфигурации – это внешняя эталонная модель, используемая генератором для отслеживания состояния микропроцессора. Она интегрируется в ядро инструмента при помощи специальных библиотек на языке C++. Как правило, модель разрабатывают производители микропроцессора, а интеграцию осуществляют разработчики инструмента. Это требует скоординированных совместных усилий т.к. для успешной интеграции необходимо, чтобы модель удовлетворяла требованиям, предъявляемым инструментом.

Задачи тестирования описываются при помощи шаблонов, которые разрабатывается вручную или создаются при помощи графического интерфейса. В шаблонах задаются используемые инструкции, вероятности их появления, правила генерации входных значений, целевые тестовые ситуации т.д. Тестовые шаблоны разрабатываются в соответствии с таблицами тестового покрытия, которые формулируют цели тестирования. На основе метрик покрытия, полученных в результате выполнения построенных тестов, в набор шаблонов вносятся изменения до тех пор, пока требуемый уровень покрытия не будет достигнут.

Тестовые ситуации описываются в виде правил, основанных на ограничениях и различных эвристиках, которые хранятся в специальной базе знаний. Набор правил может пополняться. Это позволяет накаливать знания, используемые для построения тестов, и повторно применять их для тестирования других микропроцессоров.

Список тестовых ситуаций, покрытие которых может быть обеспечено при помощи инструмента RAVEN включает: (1) комбинации следующих друг за другом инструкций; (2) ситуации в работе операций с плавающей точкой; (3) исключения в работе инструкций; (4) конвейерные конфликты; (5) различные сценарии обработки запросов к иерархии памяти; (6) ситуации, связанные с совместным использованием памяти несколькими ядрами многоядерного процессора.

К сожалению, информации об инструменте RAVEN, имеющаяся в открытом доступе, недостаточно подробна. Остается неясно, какой уровень нацеленности он позволяет достичь, насколько трудоемко писание тестовых ситуаций и предусматривается ли поддержка новых типов тестовых ситуаций. Также к недостаткам инструмента можно отнести то, что описание микропроцессора, используемое для генерации тестов, и эталонная модель являются дублирующимися представлениями одного и того же знания. Это усложняет поддержку особенно инструмента, если эти представления разрабатываются разными командами.

6. Инструменты IBM Research

В IBM Research разрабатывается несколько инструментов генерации тестовых программ, которые имеют различное назначение и используются в различных промышленных проектах. Изначально эти инструменты создавались для верификации микропроцессоров семейства PowerPC [35], а в дальнейшем применялись и для других архитектур (таких, как ARM и x86).

6.1 Инструмент Genesys-Pro

В настоящее время основным средством генерации тестовых программ, используемым в IBM, является Genesys-Pro [8]. Это универсальный инструмент, позволяющий создавать случайные и нацеленные тесты для различных типов микропроцессоров. Данный инструмент разделен на *ядро*, которое реализует методы генерации, применимые для любых архитектур, и *модель*, которая содержит всю информацию о тестируемом микропроцессоре. Задачи тестирования формулируются при помощи тестовых *шаблонов*. Сгенерированные инструкции выполняются на внешнем симуляторе с целью отслеживания состояния микропроцессора и контроля корректности построенной программы.

Инструмент Genesys-Pro включает в себя специальную среду моделирования, позволяющую создавать модели микропроцессоров на основе набора высокоуровневых блоков. Модели содержат информацию двух видов: (1) декларативное описание микропроцессора, которое включает в себя инструкции, ресурсы (регистры, память) и некоторые механизмы (например, трансляция адресов) и (2) тестовое знание для данного микропроцессора, которое представляет собой набор эвристик, позволяющих повысить уровень покрытия. Семантика инструкций описывается в виде ограничений на их входные аргументы. Описание инструкций также включает их сигнатуры, используемые ими ресурсы, типы данных аргументов и допустимые входные значения. Инструкции могут объединяться в группы. Среда моделирования имеет некоторые ограничения, не позволяющие описывать семантику некоторых типов инструкций. В частности для инструкций арифметики с плавающей точкой и инструкций доступа к памяти используются дополнительные инструменты, о которых будет рассказано в следующих

разделах. Кроме того, семантику некоторых сложных инструкций приходится описывать на языке C++.

Шаблоны создаются на специализированном языке [36], предоставляющем конструкции для описания последовательностей инструкций, распределений вероятностей и ограничений. Последовательности инструкций могут строиться при помощи методов случайной и комбинаторной генерации. Также язык предоставляет средства для описания последовательностей инструкций, которые будут выполняться в разных потоках (или разных ядрах). Входные аргументы инструкций генерируются случайным образом (с заданной вероятностью) или путем решения ограничений. Ограничения, задающие те или иные аспекты поведения инструкций, берутся из модели. Кроме этого существуют универсальные ограничения, которые можно разделить на следующие типы: (1) ограничения на выравнивание адресов; (2) зависимости по ресурсам для инструкций; (3) события, связанные с работой подсистемы памяти (промахи и попадания в различные буферы). Ограничения могут быть обязательными (hard) и необязательными (soft). Первые имеют более высокий вес при решении систем ограничений и, как правило, основаны на семантике инструкций. Вторые могут быть проигнорированы, если система ограничений не имеет решения, и обычно основаны на тестовом знании. Для ограничений можно задавать вероятности, с которыми они должны быть применены для выбранных инструкций. Также язык описания тестовых шаблонов поддерживает условную генерацию. Т.е. можно задавать пред- и постусловия, которые должны выполняться для того, чтобы фрагмент кода был добавлен в тестовую программу.

Генерация тестовых программ включает следующие стадии: (1) построение последовательностей инструкций; (2) решение ограничений (или генерация случайных значений) для каждой из инструкций; (3) симуляция инструкций на эталонной модели. В случае если не удается решить ограничения, инструмент может вносить коррективы в последовательность инструкций (добавлять дополнительные инструкции, повторно генерировать предыдущие инструкции).

Инструмент Genesys-Pro позволяет достичь достаточно высокого уровня нацеленности. Степень случайности тестов задается тестовыми шаблонами, которые разрабатываются в соответствии с планом верификации. Решение относительно полноты набора тестовых шаблонов принимается на основе метрик покрытия, основанных на реализации и функциональных требованиях.

К недостаткам Genesys-Pro можно отнести ограничения среды моделирования, которые требуют использования дополнительных средств для моделирования инструкций арифметики с плавающей точкой и доступа к памяти. Другим слабым местом является использование внешней эталонной модели для симуляции инструкций. Это требует дополнительных трудозатрат на интеграцию и поддержку данной модели. Также непонятно насколько

расширяемым является инструмент (возможность добавления новых методов генерации).

6.2 Инструмент FPGen

Для верификации модулей арифметики с плавающей точкой IBM Research был разработан инструмент FPGen [37]. Этот инструмент расширяет Genesys-Pro средствами генерации тестовых данных для покрытия всевозможных ситуаций в работе операций с плавающей точкой [38] (требования определены в стандарте IEEE 754 [39]).

Инструмент FPGen генерирует значения входных аргументов инструкций, разрешая специализированные ограничения. Ограничения определяют значения отдельных входных аргументов, отношения между значениями входных аргументов, результат промежуточных вычислений или конечный результат выполнения инструкции. Для описания ограничений используется язык XML. Генератор FPGen не привязан к какой-либо конкретной архитектуре, он генерирует наборы данных, которые сохраняются в специальном формате [40]. Эти данные используются Genesys-Pro при генерации тестовых программ, использующих инструкции арифметики с плавающей точкой. В процессе генерации Genesys-Pro комбинирует результаты решения ограничений, которые он разрешает самостоятельно и которые он разрешает при помощи FPGen.

6.3 Инструмент DeepTrans

Еще одно известное расширение Genesys-Pro – это инструмент DeepTrans [41], предназначенный для тестирования механизмов трансляции адресов. Этот инструмент использует спецификации подсистемы памяти, разработанные на специализированном языке моделирования. Данный язык позволяет представить процесс преобразования адреса в виде ориентированного ациклического графа, вершины которого соответствуют стадиям процесса, а дуги — переходам между стадиями. Множество путей от истока до стока задает конкретные варианты преобразования адреса и соответствует тестовым ситуациям. Тестовые ситуации представляются в виде ограничений, на которые можно ссылаться из тестовых шаблонов. За обработку шаблонов отвечает инструмент Genesys-Pro, который разрешает ограничения и трансформирует результаты в последовательности инструкций. К достоинствам DeepTrans следует отнести развитый язык моделирования механизмов трансляции. Он позволяет достаточно быстро осуществить настройку инструмента для тестирования механизмов трансляции адресов произвольного микропроцессора. Известный недостаток состоит в том, что не поддерживается автоматическое извлечение зависимостей между инструкциями (конфликтов использования устройств); их приходится дополнительно указывать в тестовых шаблонах.

7. Другие разработки

7.1 Среда RDG

В компании Samsung разработана среда RDG (Random Diagnostics Generator) [42], предназначенная для случайной генерации тестовых программ для реконфигурируемых микропроцессоров (Samsung Reconfigurable Processor, SRP). Эта среда использует в качестве входных данных описание синтаксиса инструкций тестируемого микропроцессора и тестовые шаблоны на языке C++. Тестовые шаблоны задают, какие инструкции будут использованы в тестовой программе, и описывают ограничения, накладываемые на входные значения этих инструкций. Среда RDG не владеет информацией о семантике инструкций и не осуществляет симуляцию тестовых программ с целью предсказания результатов. Такой подход был выбран из-за особенностей тестирования реконфигурируемых микропроцессоров (набор инструкций зависит от конфигурации, и их реализация может отличаться). Кроме того, это позволяет с минимальными усилиями добавлять поддержку новых инструкций и обеспечить высокую скорость генерации. Недостаток такого подхода в том, что для генерации нацеленных тестов требуется описывать ограничения вручную и отсутствует возможность отслеживать состояние микропроцессора. Стоит сказать, что инструмент RDG позволяет эффективно решать задачу тестирования реконфигурируемых микропроцессоров Samsung, но он не является универсальным. Поддержка новых типов микропроцессоров и методов генерации в него не заложена.

7.2 Генератор MA^{2}TG

Исследования в области генерации тестовых программ для функциональной верификации микропроцессоров проводились в Оборонном научно-техническом университете Китая (National University of Defense Technology) [43]. В рамках этих исследований был разработан прототип генератора тестовых программ MA^{2}TG. Данный генератор применим для различных типов микропроцессоров и поддерживает случайную генерацию и генерацию на основе ограничений. В качестве входных данных для генератора MA^{2}TG используются спецификации на языке EXPRESSION [44], описывающие архитектуру тестируемого микропроцессора, и шаблоны на специализированном языке, формулирующие задачи генерации. Инструмент транслирует входные в файлы в программу-генератор на языке C++, которая генерирует тесты. В процессе генерации симуляция построенных последовательностей инструкций не осуществляется.

Спецификации на языке EXPRESSION содержат информацию о структуре и поведенческих свойствах микропроцессора, а также о связи между ними. Для генерации тестовых программ в первую очередь необходима информация о поддерживаемых инструкциях, которая включает текстовый и бинарный

формат инструкций, списки их операндов и семантику инструкций (условия возникновения тестовых ситуаций). MA^{2}TG строит на основе спецификаций библиотеку шаблонов инструкций (Instruction Template Library, ITL), которая используется при генерации. Каждая инструкция описывается классом на языке C++, который содержит методы для ее печати, получения списка аргументов и доступа к ассоциированным с ней ограничениям. Использование формальных спецификаций позволяет относительно просто сконфигурировать инструмент для тестирования микропроцессора с новой архитектурой. Однако подход MA^{2}TG имеет некоторые недостатки. Информация о структуре микропроцессора, которая описывается в спецификациях на языке EXPRESSION, чаще всего не требуется для генерации тестовых программ. Эта информация может быть достаточно объемной, и ее специфицирование требует дополнительных трудозатрат. Кроме того, она может быть недоступна на ранних стадиях работы над проектом. А в случаях, когда требуется тестирование микропроцессоров, по-разному реализующих одну и ту же архитектуру, она может препятствовать повторному использованию спецификаций. Все это увеличивает трудоемкость разработки и поддержки спецификаций. Использование более простого языка, описывающего только поведенческие свойства, могло бы упростить эту задачу. Также, следует заметить, что MA^{2}TG не строит эталонный симулятор на основе спецификаций, хотя язык EXPRESSION предоставляет для этого достаточное количество информации. Эталонный симулятор был бы полезен для проверки корректности построенных программ, создания встроенных проверок и разрешения ограничений.

Задачи генерации описываются на специализированном языке. Такие описания называют ограничениями (constraints), хотя, по сути, они являются вариантом тестовых шаблонов. На их основе MA^{2}TG создает программы на языке C++, которые при помощи ITL и внешних библиотек разрешения ограничений, генерируют тесты. Язык описания тестовых шаблонов позволяет задавать следующие параметры: (1) используемые инструкции, их количество, порядок и вероятность появления; (2) ограничения на значения операндов инструкций; (3) зависимости по операндам между инструкциями. Данный язык ориентирован на случайную выборку инструкций и разрешения ограничений для выбранных инструкций. Это позволяет быстро создавать большое количество небольших тестов для верификации определенных инструкций. Однако он не позволяет описывать сценарии со сложной структурой переходов. Также не предоставляется возможностей для описания многопоточных сценариев. Т.к. инструмент не осуществляет симуляцию на эталонной модели, не поддерживается создание встроенных проверок. Кроме того, отсутствие информации о состоянии микропроцессора усложняет решение ограничений. Из описания инструмента неясно, какие типы ограничений он поддерживает и насколько сложно в него добавить поддержку новых типов.

Инструмент MA^{2}TG реализован в виде прототипа. Несмотря на то, что его подход имеет ряд преимуществ, его функциональные возможности ограничены. Остается непонятно насколько он соответствует требованиям, предъявляемым инструментам, используемым в промышленных проектах. Из очевидных недостатков можно перечислить излишне детальные спецификации, отсутствие контроля состояния микропроцессора в процессе генерации, ограниченные возможности языка тестовых шаблонов и ограниченный набор поддерживаемых методов генерации.

7.3 Исследования Университета Флориды и Калифорнийского университета в Ирвайне

В Университете Флориды (UFL) и Калифорнийском университет в Ирвайне (UCI) был разработан метод генерации тестовых программ, нацеленных на проверку корректности работы конвейера команд микропроцессора [15]. Данный метод использует спецификации на языке EXPRESSION [44], на основе которых строится модель, описывающая архитектуру микропроцессора в виде графа. Кроме этого разрабатывается модель ошибок, которая описывает типичные ошибки проектирования. На основе модели ошибок для модели микропроцессора строятся формулы, которые задают условия возникновения конкретных ошибок для данного микропроцессора. Для этих формул при помощи инструмента SMV (Symbolic Model Verifier) [45], который использует метод проверки моделей, стоятся тестовые примеры (контрпримеры для отрицания формул), на основе которых генерируются тестовые программы. По мнению авторов, метод не масштабируется на сложные микропроцессоры, поэтому предлагается дополнительно использовать тестовые шаблоны. Они создаются вручную и содержат описания цепочек инструкций, которые вызывают определенные ситуации в поведении микропроцессора (прежде всего, конвейерные конфликты). К достоинствам данного метода можно отнести то, что он применим для различных типов микропроцессоров и позволяет обеспечить покрытие ситуаций в работе конвейера команд, используя минимальное количество тестов. Главный недостаток метода - сложность разработки детальных спецификаций и модели ошибок. Следует также отметить, что данные исследования носили академический характер и на их основе не были разработаны инструменты, которые могли бы использоваться в промышленных проектах.

7.4 Инструмент μGP

Исследователями Туринского политехнического университета (Politecnico di Torino) был предложен интересный подход к генерации тестовых программ, основанный на использовании генетических алгоритмов [46][47]. Данный подход был реализован в прототипе инструмента μGP. Основная идея подхода состоит в следующем. Для генератора тестовых программ создается библиотека

инструкций, которая описывает синтаксис языка ассемблера целевого микропроцессора. Задачи генерации описываются в виде ациклического графа, который описывает поток выполнения тестовой программы. Каждая вершина графа содержит ссылку на описание инструкции в библиотеке инструкций и значения ее операндов. Библиотека инструкций и задачи генерации описываются на языке XML. Генерация тестовых программ осуществляется путем мутации структуры графа и значений операндов инструкций внутри отдельных вершин. Генератор пытается построить тестовую программу, которое обеспечит наилучшее покрытие для заданной метрики. Значение метрики получается путем выполнения построенных программ на симуляторе RTL модели.

Достоинством данного подхода является его гибкость и универсальность. Он позволяет достичь высокого уровня тестового покрытия для различных типов микропроцессоров, используя различные метрики покрытия. Главный недостаток – высокая вычислительная сложность, из-за которой скорость генерации получается достаточно низкой. Инструмент μGP реализует только один метод генерации и не предусматривает поддержку новых методов. Еще один возможный недостаток – это то, что входные форматы конфигурационных файлов не являются интуитивно понятными.

8. Анализ возможностей инструментов генерации тестовых программ

В предыдущих разделах были рассмотрены основные из существующих инструментов генерации тестовых программ для микропроцессоров. Они отличаются набором поддерживаемых методов генерации и способами реализации этих методов. В данном разделе выводятся их общие свойства, и формулируется концепция универсального инструмента генерации, сочетающего в себе данные свойства.

Для всех рассмотренных инструментов, независимо от поддерживаемых ими методов, можно выделить два свойства: (1) *реконфигурируемость* и (2) *расширяемость*. Оба этих свойства показывают, какое количество усилий требуется для адаптации инструмента к решению новых задач. Под реконфигурируемостью понимается возможность поддержки тестирования микропроцессоров с новой конфигурацией, а под расширяемостью – возможность интеграции компонентов, которые реализуют новые методы генерации.

Другим важным свойством инструмента генерации является возможность *контроля корректности* построенных программ. Корректные программы не должны содержать бесконечные циклы и приводить к переходу микропроцессора в состояния, в которых его поведение не определено. Как правило, подобный контроль осуществляется путем симуляции программ на эталонной модели в процессе их построения. Эталонные модели разрабатываются отдельно от инструмента генерации и интегрируются в него

при помощи специальных библиотек. По сути, они являются частью конфигурации инструмента т.к. используют знание о синтаксисе и семантике инструкций для их интерпретации.

Основным свойством, характеризующим методы генерации, является *нацеленность*. Методы, основанные на случайной генерации, не позволяют систематическим образом обеспечить покрытие функциональных требований. Современные микропроцессоры с конвейерной архитектурой и сложной организацией памяти имеют огромное пространство возможных состояний, которые с малой вероятностью будут достигнуты при использовании случайных тестов. Поэтому для проверки конкретных ситуаций или классов ситуаций требуются специализированные методы, основанные на разрешении ограничений и техниках проверки моделей.

Еще одним важным свойством современных инструментов верификации является возможность построения тестовых программ для многоядерных микропроцессоров. На базовом уровне она присутствует практически во всех инструментах, т.к. для этого достаточно включить в построенные программы инструкции, обеспечивающие разветвление потока выполнения на несколько ядер. Однако, методы нацеленной генерации, использующие информацию о текущем состоянии микропроцессора, должны учитывать тот факт, что каждое из ядер имеет свое состояние. Кроме того, инструменты должны обеспечивать покрытие ситуаций, связанных с параллельным выполнением программ.

В табл. 2 сравниваются функциональные возможности рассмотренных ранее инструментов по перечисленным критериям.

Табл. 2. Сравнение возможностей рассмотренных инструментов генерации

Table 2. Comparison of the considered generation tools

Инструмент	**Реконфигурируемость**	**Расширяемость**	**Контроль Корректности**	**Уровень Нацеленности**	**Поддержка Многоядерности**
INTEG	Нет	Нет	Нет	Средняя	Нет
RIS	Нет	Нет	Нет	Средняя	Да
RAVEN	Да	Нет	Да	Средняя	Да
Genesys-Pro, FPGen, DeepTrans	Да	Да (частично)	Да	Высокая	Да
RDG	Да	Нет	Нет	Низкая	Нет
MA^2TG	Да	Нет	Нет	Высокая	Нет
UFL/UCI	Да	Нет	Нет	Высокая	Нет
μGP	Да	Нет	Нет	Средняя	Нет

Как можно заметить, большинство инструментов предназначено для решения конкретных задач и не предполагают расширяемость. Это объясняется тем, что коммерческие компании занимаются верификацией выпускаемых ими микропроцессоров и создают инструменты для решения текущих задач верификации, а исследовательские институты изобретают новые методы верификации и создают инструменты для их апробации. При этом задача интеграции методов, которые не используются в данный момент, не ставится.

Ситуация с реконфигурируемостью несколько сложнее. Несмотря на то, что некоторые инструменты позволяют конфигурирование под различные архитектуры, интеграция внешней эталонной модели сопряжена с трудностями. Большинство из перечисленных реконфигурируемых инструментов не используют эталонные модели и не осуществляют контроль корректности построенных инструкций. Однако даже без необходимости интеграции эталонной модели создание пользовательских конфигураций может иметь значительную трудоемкость.

Уровень нацеленности и поддержка многоядерности характеризуют методы генерации и зависят от области применения инструмента. При этом они в той или иной степени связаны с расширяемостью и реконфигурируемость. Более гибкие инструменты реализуют большее количество методов генерации и предоставляют больше возможностей.

Т.к. применимость отдельных инструментов ограничена, актуальной задачей является создание *универсального инструмента*, который позволил бы объединить различные методы генерации, и был бы применим для различных типов микропроцессоров. Подобный инструмент должен сочетать все перечисленные выше свойства. Его характеристики могут быть сформулированы следующим образом.

Реконфигурируемость предполагает описание конфигурации тестируемого микропроцессора в простом, удобном и понятном инженеру-верификатору формате. Расширяемость предполагает, что описания, созданные для одних методов генерации, могут быть использованы другими методами. Т.к. методы генерации требуют разное количество информации, то для более сложных методов будут требоваться описания, которые дополнят информацию, представленную в описаниях, используемых более простыми методами. Информацию о конфигурации микропроцессора, используемую рассмотренными инструментами, можно разделить на четыре уровня: (1) формат инструкций (текстовый и бинарный), (2) семантика инструкций, (3) организация подсистемы памяти и (4) организация конвейера инструкций. Один из возможных способов представления этой информации – применение формальных спецификаций. Это позволит использовать единое описание для извлечения ограничений, построения формальных моделей и эталонных симуляторов. Для решения данной задачи требуется формальный язык, на котором можно в зависимости от потребности описывать различные аспекты

конфигурации микропроцессора. Другой вариант решения – использование нескольких языков, расширяющих возможности друг друга.

9. Заключение

В статье сделан обзор распространенных подходов к генерации тестовых программ для функциональной верификации микропроцессоров. Рассмотренные подходы различаются областью применения и сложностью реализации. Наиболее простые основаны на случайной генерации, наиболее сложные используют техники проверки моделей для построения тестов, нацеленных на покрытие определенных ситуаций в работе микропроцессора. Инструментам, использующим различные методы генерации, требуется разное количество информации о тестируемом микропроцессоре. Эта информация или является частью инструмента или поставляется в виде конфигурационных файлов. Кроме этого некоторые инструменты используют внешние эталонные модели для контроля корректности построенных программ и предсказания состояния микропроцессора. Эти модели требуется интегрировать в инструмент.

Основная проблема современных инструментов генерации тестовых программ для микропроцессоров – это то, что они, как правило, созданы для конкретных типов микропроцессоров и предназначены для решения конкретных задач верификации. Возможность поддержки новых типов микропроцессоров и новых методов генерации или не предусмотрена, или ограничена ввиду трудоемкости конфигурирования инструмента и трудностей интеграции эталонной модели. Т.к. ни один метод или инструмент не является универсальным решением для всех задач верификации, интеграция методов генерации является актуальной задачей. При этом важно чтобы имеющийся инструментарий методов генерации был применим к различным типам микропроцессоров.

К сожалению, в настоящее время не существует универсального инструмента, который позволил бы применять широкий набор методов генерации для тестирования различных типов микропроцессоров. Создание такого инструмента имело бы большое практическое значение. Это позволило бы снизить затраты на поддержку инструментов и повысить качество тестирования.

Список литературы

[1]. Grant McFarland. Microprocessor Design: A Practical Guide from Design Planning to Manufacturing Professional Engineering. McGraw Hill Professional, 2006, 408 p.

[2]. Статистика числа транзисторов в микропроцессорах — http://en.wikipedia.org/wiki/Transistor_count

[3]. Intel® Pentium® 4 Processor. Specification Update, August 2008 (http://download.intel.com/design/intarch/specupdt/24919969.pdf)

[4]. Intel® Core™ i7-900 Desktop Processor Extreme Edition Series and Intel® Core™ i7-900 Desktop Processor Series. Specification Update, February 2015. (http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/core-i7-900-ee-and-desktop-processor-series-spec-update.pdf)
[5]. Beizer B. The Pentium Bug – An Industry Watershed // Testing Techniques Newsletter, TTN Online Edition, September 1995.
[6]. А.С. Камкин, А.М. Коцыняк, С.А. Смолов, А.А. Сортов, А.Д. Татарников, М.М. Чупилко. Средства функциональной верификации микропроцессоров, Труды ИСП РАН, том 26, выпуск 1, 2014, с. 149-200. DOI: 10.15514/ISPRAS-2014-26(1)-5.
[7]. W.K. Lam. Hardware Design Verification: Simulation and Formal Method-Based Approaches. Prentice Hall, 2005. 624 p.
[8]. 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. IEEE Design & Test of Computers, 21(2), 2004, pp. 84-93.
[9]. J. Bhadra, M. Abadir, S. Ray, L. Wang. A Survey of Hybrid Techniques for Functional Verification. IEEE Design & Test of Computers, 24(22), 2007, pp. 112-122.
[10]. Камкин А.С., Сергеева Т.И., Смолов С.А., Татарников А.Д., Чупилко М.М. Расширяемая среда генерации тестовых программ для микропроцессоров. Программирование, № 1, 2014, стр. 3-14.
[11]. A.Kamkin, A.Protsenko, A.Tatarnikov. An Approach to Test Program Generation Based on Formal Specifications of Caching and Address Translation Mechanisms. Trudy ISP RAN / Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 125-138. DOI: 10.15514/ISPRAS-2015-27(3)-9.
[12]. Бобков С.Г., Чибисов П.А. Повышение качества тестирования высокопроизводительных микропроцессоров методами встречного тестирования с анализом функционального тестового покрытия выделенных приложений. Информационные технологии, No 8, 2013, с. 26-33.
[13]. 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, стр. 63-68.
[14]. Piziali A. Functional Verification Coverage Measurement and Analysis. New York: Kluwer Academic Publishers. 2004, 216 p.
[15]. P. Mishra, N. Dutt. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design Automation of Electronic Systems, 13(3), 2008, pp. 1-36.
[16]. E.A. Poe. Introduction to random test generation for processor verification. Technical report. Obsidian Software, 2002, 7 p.
[17]. Генератор тестовых программ RISU для тестирования симулятора QEMU - https://git.linaro.org/people/peter.maydell/risu.git/about/.
[18]. Грибков И.В., Захаров А.В., Кольцов П.П. и др. Стохастическое тестирование в системе INTEG. Программные продукты и системы. 2007. № 2. с. 22-26.
[19]. Камкин А.С. Генерация тестовых программ для микропроцессоров. Труды ИСП РАН, том 14, часть 2, 2008, с. 23–63.
[20]. N. Sharma, B. Dickman, Verifying an ARM Core, EE Times, 2001, 7 p.

[21]. А.С. Камкин. Некоторые вопросы автоматизации построения тестовых программ для модулей обработки переходов микропроцессоров. Труды ИСП РАН, 18, 2010, стр. 129-149.
[22]. Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. Marcus, G. Shurek. Constraint-Based Random Stimuli Generation for Hardware Verification. AI Magazine, 28(3), 2007, pp. 13-30.
[23]. P. Mishra and N. Dutt. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 13, Issue 3, 2008, pp. 1–36.
[24]. Грибков И.В., Захаров А.В., Кольцов П.П. и др. Развитие системы стохастического тестирования микропроцессоров INTEG. Программные продукты и системы. 2010, № 2, стр. 14–23.
[25]. MIPS64™ Architecture For Programmers. Volume 1: Introduction to the MIPS64™ Architecture. Revision 6.01. MIPS Technologies Inc. 2014. 148 p.
[26]. Программный симулятор VMIPS – http://www.dgate.org/vmips/.
[27]. Сайт компании ARM — http://www.arm.com.
[28]. N. Sharma and B. Dickman. Verifying an ARM Core. EE Times. 2001, p. 7.
[29]. Hrishikesh M.S., Rajagopalan M., Sriram S., Mantri R. System Validation at ARM — Enabling our Partners to Build Better Systems. White Paper. April 2016 (http://www.arm.com/files/pdf/System_Validation_at_ARM_Enabling_our_partners_to_build_better_systems.pdf).
[30]. Venkatesan D., Nagarajan P. A Case Study of Multiprocessor Bugs Found Using RIS Generators and Memory Usage Techniques. Workshop on Microprocessor Test and Verification, 2014, pp. 4-9. DOI: 10.1109/MTV.2014.28.
[31]. Hudson J., Kurucheti G. A Configurable Random Instruction Sequence (RIS) Tool for Memory Coherence in Multi-processor Systems. Workshop on Microprocessor Test and Verification, 2014, pp. 98-101. DOI: 10.1109/MTV.2014.26.
[32]. Генератор тестовых программ RAVEN - http://www.slideshare.net/DVClub/introducing-obsidian-software-andravengcs-for-powerpc.
[33]. Obsidian Software Inc. “Raven: Product datasheet”. 6 p.
[34]. R.N. Mahapatra, P. Bhojwani, J. Lee, and Y. Kim. Microprocessor Evaluations for Safety-Critical, Real-Time Applications: Authority for Expenditure, No.43, Phase 3 Report, 2009, 43 p.
[35]. Архитектура PowerPC - https://en.wikipedia.org/wiki/PowerPC
[36]. M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, M. Vinov. Industrial Experience with Test Generation Languages for Processor Verification. Proceedings of the Design Automation Conference, 2004, pp. 36-40.
[37]. M. Aharoni, S. Asaf, L. Fournier, A. Koifman and R. Nagel. FPgen – A Test Generation Framework for Datapath Floating-Point Verification. Proceedings of the Eighth IEEE International Workshop on High-Level Design Validation and Test Workshop (HLDVT'03), 2003, pp. 17–22.
[38]. M. Aharony, E. Gofman, E. Guralnik, A. Koyfman. Injecting floating-point testing knowledge into test generators. Proceedings of the 7th international Haifa Verification conference on Hardware and Software: Verification and Testing (HVC’11), 2011, pp. 234-241, ISBN 978-3-642-34187-8.
[39]. IEEE standard for binary FP arithmetic. An American National Standard, ANSI/IEEEE Std. 754-2008, 58 p.

[40]. IBM Floating-Point Test Suite for IEEE 754R Standard - https://www.research.ibm.com/haifa/projects/verification/fpgen/ieeets.html
[41]. Adir A., Fournier L., Katz Y., Koyfman A. DeepTrans – Extending the Model-based Approach to Functional Verification of Address Translation Mechanisms. High-Level Design Validation and Test Workshop, 2006, pp. 102-110.
[42]. Seonghun Jeong, Youngchul Cho, Daeyong Shin, Changyeon Jo, Yenjo Han, Soojung Ryu, Jeongwook Kim, and Bernhard Egger. Random Test Program Generation for Reconfigurable Architectures. Proceedings of 13th International Workshop on Microprocessor Test and Verification (MTV), 2012, 6 p.
[43]. T.Li, D.Zhu, Y.Guo, G.Liu, S.Li. MA2TG: A Functional Test Program Generator for Microprocessor Verification. Euromicro Conference on Digital System Design, 2005, pp.176-183.
[44]. P. Grun, A. Halambi, A. Khare, V. Ganesh, N. Dutt and A. Nicolau. EXPRESSION: An ADL for System Level Design Exploration. Technical Report 1998-29, University of California, Irvine, 1998, 26 p.
[45]. Инструмент - http://www.cs.cmu.edu/~modelcheck/smv.html
[46]. F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero. Efficient Machine-Code TestProgram Induction. CEC'2002: Congress on Evolutionary Computation, Honolulu, Hawaii, USA, 2002.
[47]. F. Corno et al., "Fully Automatic Test Program Generation for Microprocessor Cores," Proc. IEEE Design, Automation and Test in Europe (DATE 03), IEEE CS Press, 2003, pp. 1006-1011.
[48]. F. Corno, E. Sanchez, M. Sonza Reorda, G. Squillero. Automatic Test Program Generation – A Case Study. IEEE Design and Test, Special Issue on Functional Verification and Testbench Generation, Volume 21, Issue 2, 2004, pp. 102-109.

A Survey of Methods and Tools for Test Program Generation for Microprocessors

A.D. Tatarnikov <andrewt@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. This paper gives a survey of existing methods and tools for test program generation for microprocessors. Test program generation and analysis of their execution traces is the main approach to functional verification of microprocessors. This approach is also known as testing. Despite continuous progress in test program generation methods, testing remains an extremely laborious process. One of the main reasons is that test program generation tools are unable to quickly enough adapt to changes. In the majority of cases, they are created for specific microprocessor types and are designed to solve specific tasks. For this reason, support for new microprocessors types and generation methods requires a significant effort. Often, in such situations, tools have to be implemented from scratch. Inability to reuse existing implementations of generation methods complicates evolution of test generation tools and, consequently, prevents improvement of testing quality. The present situation creates motivation to search for solutions to developing more flexible tools which could be easily adapted to testing new microprocessor types and applying new generation methods. The goal of the present work is to summarize existing experience in test program generation, which

could serve as a basis for creating such tools. The paper considers strengths and weaknesses of popular generation methods, their application domains and cases of their combined use. It also makes a comparative analysis of facilities of existing generation tools implementing these methods. Based on the analysis, it gives recommendations on creating a unified methodology to develop tools for test program generation for microprocessors.

Keywords: microprocessors; hardware; functional verification; testing; test program generation.

DOI: 10.15514/ISPRAS-2017-29(1)-11

For citation: Tatarnikov A.D. A survey of methods and tools of test program generation for microprocessors. *Trudy ISP RAN / Proc. ISP RAS*, vol. 29, issue 1, 2017. pp. 167-194 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-11

References

[1]. Grant McFarland. Microprocessor Design: A Practical Guide from Design Planning to Manufacturing Professional Engineering. McGraw Hill Professional, 2006, 408 p.
[2]. Transistor count in microprocessors — http://en.wikipedia.org/wiki/Transistor_count
[3]. Intel® Pentium® 4 Processor. Specification Update, August 2008 (http://download.intel.com/design/intarch/specupdt/24919969.pdf)
[4]. Intel® Core™ i7-900 Desktop Processor Extreme Edition Series and Intel® Core™ i7-900 Desktop Processor Series. Specification Update, February 2015. (http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/core-i7-900-ee-and-desktop-processor-series-spec-update.pdf)
[5]. Beizer B. The Pentium Bug – An Industry Watershed // Testing Techniques Newsletter, TTN Online Edition, September 1995.
[6]. A. Kamkin, A. Kotsynyak, S. Smolov, A. Sortov, A. Tatarnikov, M. Chupilko. Tools for Functional Verification of Microprocessors. Trudy ISP RAN / Proc. ISP RAS, vol. 26, issue. 1, 2014. pp. 149-200 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-5.
[7]. W.K. Lam. Hardware Design Verification: Simulation and Formal Method-Based Approaches. Prentice Hall, 2005, 624 p.
[8]. 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. IEEE Design & Test of Computers, 21(2), 2004, pp. 84-93.
[9]. J. Bhadra, M. Abadir, S. Ray, L. Wang. A Survey of Hybrid Techniques for Functional Verification. IEEE Design & Test of Computers, 24(22), 2007, pp. 112-122.
[10]. A.S. Kamkin, T.I. Sergeeva, S.A. Smolov, A.D. Tatarnikov, M.M. Chupilko. Extensible Environment for Test Program Generation for Microprocessors. Programming and Computer Software, 40(1), 2014, pp. 1-9. DOI: 10.1134/S0361768814010046.
[11]. A.Kamkin, A.Protsenko, A.Tatarnikov. An Approach to Test Program Generation Based on Formal Specifications of Caching and Address Translation Mechanisms. Trudy ISP RAN / Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 125-138. DOI: 10.15514/ISPRAS-2015-27(3)-9.
[12]. Bobkov S.G., Chibisov P.A. Improving of Quality of High-performance Microprocessors Testing by Counter-Testing Methods with Analysis of the Functional Coverage of the

Selected User Control Tasks. Informacionnye tehnologii [Information Technology], No. 8, 2013, pp. 26-33 (in Russian)
[13]. Khisambeev I.Sh., Chibisov P.A. On one method of defining functional coverage metrics for microprocessor testing. Proceedings of the conference on Problems of Perspective Micro- and Nanoelectronic Systems Development (MES-2014), Part II, 2014, pp. 63-68 (in Russian).
[14]. Piziali A. Functional Verification Coverage Measurement and Analysis. New York: Kluwer Academic Publishers. 2004. 216 p.
[15]. P. Mishra, N. Dutt. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design Automation of Electronic Systems, 13(3), 2008, pp. 1-36.
[16]. E.A. Poe. Introduction to random test generation for processor verification. Technical report. Obsidian Software, 2002, 7 p.
[17]. RISU test program generator for testing QEMU simulator - https://git.linaro.org/people/peter.maydell/risu.git/about/.
[18]. Gribkov I.V., Zaharov A.V., Kol'cov P.P., et al. Stochastic testing in the INTEG system. Programmnye Produkty i Sistemy [Programming Products and Systems], 2007, no. 2, pp. 22–26 (in Russian).
[19]. Kamkin A.S. Test Program Generation for Microprocessors. Trudy ISP RAN / Proc. ISP RAS, vol. 14, issue. 2, 200, pp. 23-63 (in Russian).
[20]. N. Sharma, B. Dickman, Verifying an ARM Core, EE Times, 2001, 7 p.
[21]. A. Kamkin. Some issues of automation of test program generation for branch processing units of microprocessors. Trudy ISP RAN / Proc. ISP RAS, vol. 18, 2010, pp. 129-149 (in Russian).
[22]. Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. Marcus, G. Shurek. Constraint-Based Random Stimuli Generation for Hardware Verification. AI Magazine, 28(3), 2007, pp. 13-30.
[23]. P. Mishra and N. Dutt. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 13, Issue 3, 2008, pp. 1–36.
[24]. Gribkov I.V., Zaharov A.V., Kol'cov P.P., et al. Evolution of stochastic testing system of microprocessors INTEG. Programmnye Produkty i Sistemy [Programming Products and Systems], 2010, no. 2, pp. 14–23 (in Russian).
[25]. MIPS64™ Architecture For Programmers. Volume 1: Introduction to the MIPS64™ Architecture. Revision 6.01. MIPS Technologies Inc. 2014, 148 p.
[26]. VMIPS virtual machine simulator – http://www.dgate.org/vmips/.
[27]. ARM web site — http://www.arm.com.
[28]. N. Sharma and B. Dickman. Verifying an ARM Core. EE Times. 2001, p. 7.
[29]. Hrishikesh M.S., Rajagopalan M., Sriram S., Mantri R. System Validation at ARM — Enabling our Partners to Build Better Systems. White Paper. April 2016 (http://www.arm.com/files/pdf/System_Validation_at_ARM_Enabling_our_partners_to_build_better_systems.pdf).
[30]. Venkatesan D., Nagarajan P. A Case Study of Multiprocessor Bugs Found Using RIS Generators and Memory Usage Techniques. Workshop on Microprocessor Test and Verification, 2014, pp. 4-9. DOI: 10.1109/MTV.2014.28.
[31]. Hudson J., Kurucheti G. A Configurable Random Instruction Sequence (RIS) Tool for Memory Coherence in Multi-processor Systems. Workshop on Microprocessor Test and Verification, 2014, pp. 98-101. DOI: 10.1109/MTV.2014.26.

[32]. RAVEN test program generatort -http://www.slideshare.net/DVClub/introducing-obsidian-software-andravengcs-for-powerpc.
[33]. Obsidian Software Inc. “Raven: Product datasheet”. 6 p.
[34]. R.N. Mahapatra, P. Bhojwani, J. Lee, and Y. Kim. Microprocessor Evaluations for Safety-Critical. Real-Time Applications: Authority for Expenditure, no.43, Phase 3 Report, 2009, 43 p.
[35]. PowerPC architecture - https://en.wikipedia.org/wiki/PowerPC
[36]. M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, M. Vinov. Industrial Experience with Test Generation Languages for Processor Verification. Proceedings of the Design Automation Conference, 2004, pp. 36-40.
[37]. M. Aharoni, S. Asaf, L. Fournier, A. Koifman and R. Nagel. FPgen – A Test Generation Framework for Datapath Floating-Point Verification. Proceedings of the Eighth IEEE International Workshop on High-Level Design Validation and Test Workshop (HLDVT'03), 2003, pp. 17–22.
[38]. M. Aharony, E. Gofman, E. Guralnik, A. Koyfman. Injecting floating-point testing knowledge into test generators. Proceedings of the 7th international Haifa Verification conference on Hardware and Software: Verification and Testing (HVC’11), 2011, pp. 234-241.
[39]. IEEE standard for binary FP arithmetic. An American National Standard, ANSI/IEEEE Std. 754-2008, 58 p.
[40]. IBM Floating-Point Test Suite for IEEE 754R Standard - https://www.research.ibm.com/haifa/projects/verification/fpgen/ieeets.html
[41]. Adir A., Fournier L., Katz Y., Koyfman A. DeepTrans – Extending the Model-based Approach to Functional Verification of Address Translation Mechanisms. High-Level Design Validation and Test Workshop, 2006, pp. 102-110.
[42]. Seonghun Jeong, Youngchul Cho, Daeyong Shin, Changyeon Jo, Yenjo Han, Soojung Ryu, Jeongwook Kim, and Bernhard Egger. Random Test Program Generation for Reconfigurable Architectures. Proceedings of 13th International Workshop on Microprocessor Test and Verification (MTV), 2012, 6 p.
[43]. T.Li, D.Zhu, Y.Guo, G.Liu, S.Li. MA2TG: A Functional Test Program Generator for Microprocessor Verification. Euromicro Conference on Digital System Design, 2005, pp.176-183.
[44]. P. Grun, A. Halambi, A. Khare, V. Ganesh, N. Dutt and A. Nicolau. EXPRESSION: An ADL for System Level Design Exploration. Technical Report 1998-29, University of California, Irvine, 1998, 26 p.
[45]. SMV model checker - http://www.cs.cmu.edu/~modelcheck/smv.html
[46]. F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero. Efficient Machine-Code TestProgram Induction. CEC’2002: Congress on Evolutionary Computation, Honolulu, Hawaii, USA, 2002.
[47]. F. Corno et al., “Fully Automatic Test Program Generation for Microprocessor Cores,” Proc. IEEE Design, Automation and Test in Europe (DATE 03), IEEE CS Press, 2003, pp. 1006-1011.
[48]. F. Corno, E. Sanchez, M. Sonza Reorda, G. Squillero. Automatic Test Program Generation – A Case Study. IEEE Design and Test, Special Issue on Functional Verification and Testbench Generation, Volume 21, Issue 2, 2004, pp. 102-109.