ВЕРИФИКАЦИЯ МИКРОПРОЦЕССОРОВ:
БОРЬБА С ОШИБКАМИ И УПРАВЛЕНИЕ КАЧЕСТВОМ

Развитие полупроводниковых технологий и постоянное совершенствование компьютерных архитектур сделали организацию микропроцессоров настолько сложной, что при их разработке делаются тысячи (!) ошибок. Избежать просчетов не удается даже таким крупным компаниям, как Intel и AMD. Ошибки в сложных проектах неизбежны, главное — уметь их быстро обнаруживать и не давать "распространяться" в микросхемы. Это и является основной задачей верификации.

А. Камкин, к.ф.-м.н.
kamkin@ispras.ru

ЧТО ТАКОЕ ВЕРИФИКАЦИЯ?

Верификацией называется проверка соответствия результатов, полученных на отдельных этапах проектирования, требованиям и ограничениям, установленным для них на предыдущих этапах. На начальном этапе проверяют, соответствуют ли результаты исходным требованиям (техническому заданию). Основная задача верификации — контроль качества проектирования, включая такие его аспекты, как функциональная корректность, надежность, удобство использования и многие другие [1]. В рамках этой статьи рассматривается лишь одна составляющая верификации — проверка соответствия результатов проектирования функциональным требованиям или, другими словами, функциональная верификация. В дальнейшем для краткости будем называть функциональную верификацию просто верификацией.

Поскольку функциональность микропроцессора определяется реализуемой им системой команд, то, в общих словах, задача верификации состоит в проверке того, что микропроцессор (точнее, его проектная модель) реализует все указанные в техническом задании команды, причем результат выполнения каждой из них во всех допустимых ситуациях является корректным (соответствует заданным функциональным требованиям). Если проектная модель расходится с требованиями, говорят, что имеет место ошибка проектирования. Принятие решения о том, какие части проекта (модель микропроцессора, документация и т.п.) подлежат исправлению, является отдельной задачей, не входящей в верификацию. Часто ошибочной является проверяемая модель, и именно ее после локализации ошибки исправляют.

Помимо выявления дефектов (ошибок и недоработок в проекте), верификация решает еще несколько очень важных задач, в том числе определяет наиболее критичные и наиболее подверженные ошибкам подсистемы и предоставляет всем заинтересованным лицам информацию о текущем состоянии проекта [1]. Таким образом, верификация — это не просто поиск ошибок в микропроцессоре, а сложная система обратных связей в процессе проектирования, которая позволяет своевременно оценивать качество продукта и на основе этой оценки эффективно управлять проектом.

Верификация при проектировании микропроцессоров

Напомним кратко, как устроен процесс проектирования микропроцессоров. Исходную информацию для проекта дает техническое задание, в котором описаны различные характеристики разрабатываемого устройства: функциональность, производительность, интерфейс с внешними устройствами, конструктивные ограничения и др. Микропроцессор проектируют в четыре основных этапа, результатом каждого из которых является модель на определенном уровне абстракции. Это архитектурное проектирование, проектирование на уровне регистровых передач, логическое проектирование и физическое проектирование (рис. 1).

На этапе архитектурного проектирования уточняются система команд микропроцессора и его микроархитектура. На данном этапе в качестве основного средства применяют языки программирования общего назначения (C, C++), языки системного проектирования (SystemC, SystemVerilog) и специализированные языки описания архитектуры (nML, LISA). В результате архитектурного проектирования, помимо уточнения требований к процессору и создания соответствующей документации, создается программный симулятор (эмулятор), который позволяет интерпретировать программы, написанные для целевого микропроцессора.

На следующем этапе — этапе проектирования на уровне регистровых передач (RTL, Register Transfer Level) — с помощью языка описания аппаратуры (VHDL, Verilog) предельно

точно описывается логическая структура и функционирование схемы микропроцессора. Результатом RTL-проектирования является модель уровня регистровых передач (RTL-модель), которая с потактовой точностью описывает пересылки данных, возникающие при работе модулей микропроцессора.

В рамках логического проектирования создается схема из логических вентилей в заданном технологическом базисе, которая функционально эквивалентна RTL-модели, разработанной на предыдущем этапе. Данный этап полностью автоматизирован, хотя иногда схему дорабатывают вручную в целях оптимизации или повышения уровня тестируемости (testability). Построение логической схемы для модели уровня регистровых передач называется логическим синтезом.

На этапе физического проектирования производят размещение и трассировку топологии схемы на кристалле для заданного набора ограничений (взаимное расположение элементов схемы, площадь кристалла, минимальное расстояние между проводниками, размер проводника и т. п.). Этап физического проектирования (физический синтез), также, как и предшествующий ему этап логического проектирования, автоматизирован средствами САПР.

Строго говоря, верификация присутствует на всех перечисленных этапах, но в первую очередь — на уровне регистровых передач. Это связано тем, что RTL-модель разрабатывается вручную* и фиксирует функциональность схемы, а последующие процессы логического и физического синтеза эту функциональность не изменяют (точнее, не должны изменять). Важной составляющей верификации, которая не отражена на представленной схеме, является "верификация на кристалле" (post-silicon verification), которую можно провести, только когда появляются первые опытные образцы микропроцессора (так называемые образцы A0). Использование микросхем вместо программных моделей существенно ускоряет процесс проверки, однако не позволяет контролировать все сигналы микропроцессора, что ухудшает качество верификации.

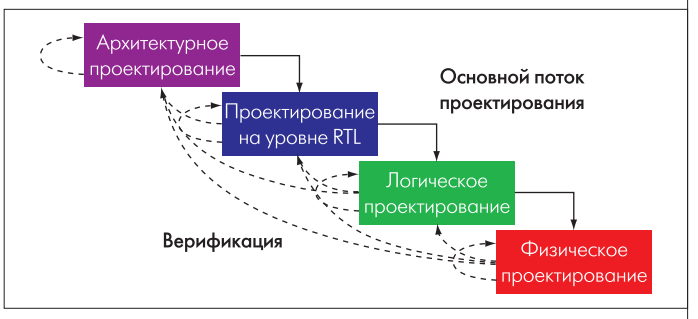

Рис. 1. Обобщенная схема процесса проектирования микропроцессора

* Сейчас ведутся исследования и уже есть промышленные разработки по так называемому поведенческому или высокоуровневому синтезу (HLS, High-Level Synthesis) — автоматическому построению RTL-моделей из высокоуровневых описаний (например, на SystemC или C/C++). Если при проектировании микропроцессора используются инструменты поведенческого синтеза (например, Cynthesizer или Synphony HLS), основным объектом верификации становятся модели, разработанные на этапе архитектурного (поведенческого) проектирования.

На ошибках учатся

Опыт проектирования микропроцессоров, накопленный десятилетиями, говорит, что верификацию нельзя игнорировать или уделять ей мало внимания — последствия ошибок проектирования могут быть катастрофическими.

Пожалуй, самой известной ошибкой в микропроцессоре является "Pentium FDIV bug" — ошибка в команде деления чисел с плавающей точкой микропроцессора Pentium, разработанного компанией Intel [2] (она была обнаружена в 1994 году). Ошибка возникает только в очень редких случаях (биты с пятого по десятый мантиссы делителя должны быть равны единице, а биты с первого по четвертый — принимать одно из пяти определенных значений) [3]. Вероятность проявления данной ошибки на случайных числах очень мала, но, тем не менее, она была найдена. Ее обнаружил профессор математики Томас Найсли, заметив неточность в научных расчетах. В результате, несмотря на то, что большинства пользователей эта ошибка не касалась, Intel пришлось заменить микропроцессоры с ошибкой, что обошлось ей в 475 млн. долл.

Описанная ошибка — наиболее известна, но она далеко не единственная в своем роде — "багов" в микропроцессорах "живет" достаточно много, и даже самая тщательная верификация (при разумных ограничениях на ресурсы) не позволяет обнаружить их все. Например, в процессоре Pentium II сначала продажи было найдено более 50 ошибок. Задача верификации, в современном понимании, является не создание безошибочной микросхемы, а выпуск работоспособного и надежного продукта.

Производители извлекают уроки из своих ошибок и стараются не повторять их. Да, после скандальной ошибки в Pentium Intel сделала несколько ошибок, касающихся арифметики с плавающей точкой (в частности, "Dan-0411 bug"), однако сейчас компания проверяет арифметические модули (FPU, Floating Point Units) всех разрабатываемых микропроцессоров с помощью формальных методов (о методах верификации будет кратко рассказано ниже). В частности, при верификации Pentium 4 были обнаружены две серьезные ошибки в FPU: в командах сложения и умножения. Боб Бентли, руководитель работ по верификации этого микропроцессора, отмечает, что если бы не формальные методы, указанные ошибки, вероятно, так и остались бы не обнаруженными, что вполне могло привести к ситуации 1994 года [4].

Сложность верификации

Верификация микропроцессоров — процесс очень трудоемкий, а, следовательно, очень дорогостоящий. Согласно Дженику Бергерону, авторитетному специалисту из компании Synopsys, затраты на верификацию составляют около 70% от всех усилий на разработку процессора; число инженеров-верификаторов обычно в двое превосходит число инженеров, проектирующих RTL-модель, а исходный код тестов, проверяющих систе-

му, составляет до 80% от общего объема разработанного кода [5]. С ростом сложности (закон Мура работает до сих пор) положение только ухудшается. Если эти цифры мало что говорят вам, скажем проще: верификация современного микропроцессора общего назначения — это сотни человек, тысячи компьютеров и миллионы долларов ежегодно.

Сложность верификации связана с огромным числом ситуаций, возможных в работе микропроцессора. На его функционирование влияют команды программы, их расположение в памяти, значения операндов, генерируемые исключения, зависимости между командами, состояния подсистем и многое другое. Помножьте это на длинные конвейеры, переупорядочивание команд, спекулятивное выполнение и другие архитектурные изыски и вы получите "практически невыполнимую задачу" [6]. Факторов, от которых зависит поведение микропроцессора, очень много, но число их комбинаций практически бесконечно.

Для иллюстрации этого положения рассмотрим ошибку в относительно простом по современным меркам микропроцессоре MIPS R4000PC/SC (1994 год) [7]. Ошибка проявляется только в следующей ситуации: команда загрузки данных из оперативной памяти в регистр вызывает промах в кэше данных; за ней через одну "пустую" команду пор (NoOPeration) следует безусловный переход по адресу, содержащемуся в загруженном регистре; команда перехода — последняя команда на странице виртуальной памяти; номер следующей страницы не содержится в буфере трансляции адресов (TLB, Translation LookasideBuffer). В микропроцессорах архитектуры MIPS имеется так называемый слот задержки перехода. Это означает, что инструкция, следующая за переходом (в данном случае — находящаяся на следующей странице), должна быть выполнена вместе с инструкцией перехода. Поэтому в рассматриваемом примере при выполнении инструкции перехода возникает исключение TLBMiss, связанное с промахом (отсутствием номера страницы) в TLB. Ошибка в микропроцессоре заключается в том, что при возникновении исключения управление передается не на предопределенный адрес обработчика исключений, а на адрес, по которому осуществлялся переход. Ниже приведен шаблон программы на языке ассемблера, вызывающей эту ошибку.

```
lw r, ...      // промах в кэше данных
nop            // одна команда NOP
jr r           // команда перехода
...            //начало страницы виртуальной памяти,
               // которой нет в TLB.
```

Пример демонстрирует, что при верификации микропроцессоров необходимо учитывать множество факторов, некоторые из которых, на первый взгляд, могут показаться никак не связанными друг с другом. Многие серьезные ошибки можно обнаружить лишь при одновременном выполнении нескольких нетривиальных условий.

Для большей эффективности обнаружения ошибок и упрощения их локализации верификация микропроцессора включает в себя верификацию отдельных модулей (это реализация принципа "разделяй и властвуй" — основного подхода в борьбе со сложностью). Для определения правильности работы модуля используется не техническое задание на микропроцессор в целом, а проектные документы, описывающие функции именно этого модуля. Таким образом, проверка микропроцессоров осуществляется на двух основных уровнях: модульном и системном. Указанное деление на уровни является относительным, например, микропроцессор может быть частью более сложной системы, а модуль представлять систему из более простых блоков.

МЕТОДЫ ВЕРИФИКАЦИИ

Методы верификации можно разделить на три основных класса: экспертизу, имитационное тестирование и формальную верификацию. Более подробную классификацию методов верификации можно найти в работе [1], мы же ограничимся указанными тремя. Помимо этих методов существуют так называемые синтетические или гибридные методы [8], использующие комбинации разных подходов. Так, в отдельный класс выделились методы тестирования, использующие элементы формальных методов — тестирование на основе моделей.

К экспертизе относятся методы верификации, в которых оценка результатов проектирования выполняется людьми путем непосредственного анализа. От других методов экспертизу отличает возможность выполнять ее, используя только результаты проектирования, а не их формальные модели (как в формальной верификации) или результаты работы (как в имитационном тестировании). Экспертиза позволяет выявлять практически любые виды ошибок, причем на самых ранних стадиях. В то же время она не может быть автоматизирована и требует активного участия людей. Различные отчеты показывают, что от 50 до 90% всех зафиксированных в жизненном цикле ошибок может быть обнаружено с помощью экспертиз, однако эффективность экспертизы существенно зависит от опыта и мотивации ее участников.

Имитационным тестированием называется тестирование моделей, полученных на разных этапах проектирования. Для краткости будем называть имитационное тестирование просто тестированием, хотя следует понимать, что под тестированием обычно понимают проверку микросхем на предмет наличия технологических ошибок (обрывов проводников, замыканий и т.п.), что, безусловно, верификацией не является. Для применения тестирования необходимо иметь работающую модель микропроцессора или хотя бы некоторых модулей, поэтому тестирование нельзя использовать на ранних фазах проектирования. Для проведения тестирования требуется дополнительная подготовка: создание тестов и разработка тестовой системы, позволяющей их выполнять. Создание тестов, позво-

ляющих получить адекватную оценку качества сложного микропроцессора, является трудоемкой задачей, требующей достаточно глубокого понимания технического задания и знания особенностей реализации процессора.

Формальные методы верификации используют формальные модели микропроцессоров и их модулей. Это могут быть логические или алгебраические модели, модели на основе графов или конечных автоматов, а также модели других типов. Анализ формальных моделей выполняется с помощью специфических техник, таких как проверка моделей (инструменты SMV, Magellan, Formality), автоматизированное доказательство теорем (HOL Light, ACL2, PVS), проверка эквивалентности (Conformal, Formality, FormalPro). Указанные техники позволяют автоматически проверить соответствие модели системы формальным спецификациям. Под формальными спецификациями понимаются машиночитаемые (допускающие автоматическую обработку) представления требований к верифицируемой системе.

Формальные методы способны обнаруживать сложные ошибки, практически не выявляемые с помощью экспертиз или тестирования. Разумеется, чтобы применять формальные методы, необходимо построить модели, что весьма трудоемко. Кроме того, формализовать требования к микропроцессорной системе можно только при его глубоком понимании, а значит — необходим тщательнейший анализ системы, выполняемый совместно специалистами по формальным методам и разработчиками.

Применение на практике

На разных этапах проектирования могут использоваться разные методы верификации. Например, на начальных стадиях, когда уточняется техническое задание и формируется документация, может применяться экспертиза; на этапах архитектурного проектирования и проектирования на уровне RTL — имитационное тестирование; на этапе логического проектирования — формальная верификация.

Самым распространенным методом, который используют все производители микропроцессоров, является тестирование. Экспертиза в той или иной степени также присутствует в большинстве проектов. Что касается формальной верификации, ее могут позволить себе лишь достаточно крупные компании (Intel, AMD, IBM, Motorola), но они применяют ее только для сравнительно небольших подсистем. Формальные методы сложно применять в условиях изменяющегося проекта, поэтому, как правило, они используются на поздних этапах проектирования. Компания Intel вложила громадные средства в развитие методов регрессионной проверки формальных доказательств [9] (при изменении структуры схемы нужно заново перестраивать систему теорем и лемм, а их число может достигать десятки тысяч).

Число найденных ошибок зависит от метода верификации (рис. 2). Видно, что большинство ошибок (74%) находится с помощью тестирования, причем около 60% из них приходится на модульное тестирование. Поскольку формальные методы применяются в ограниченных масштабах (в Pentium 4 таким образом были проверены модуль арифметики с плавающей точкой и модуль декодирования команд), процент ошибок, найденных с их использованием, невелик (6%), но среди них есть такие, которые сложно обнаружить, применяя тестирование или экспертизу. При верификации FPU Pentium 4 было найдено около 20 таких ошибок.

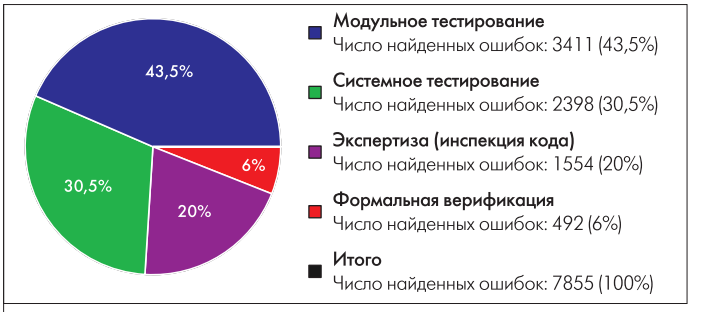

Рис. 2. Метод верификации — число найденных ошибок (Pentium 4)

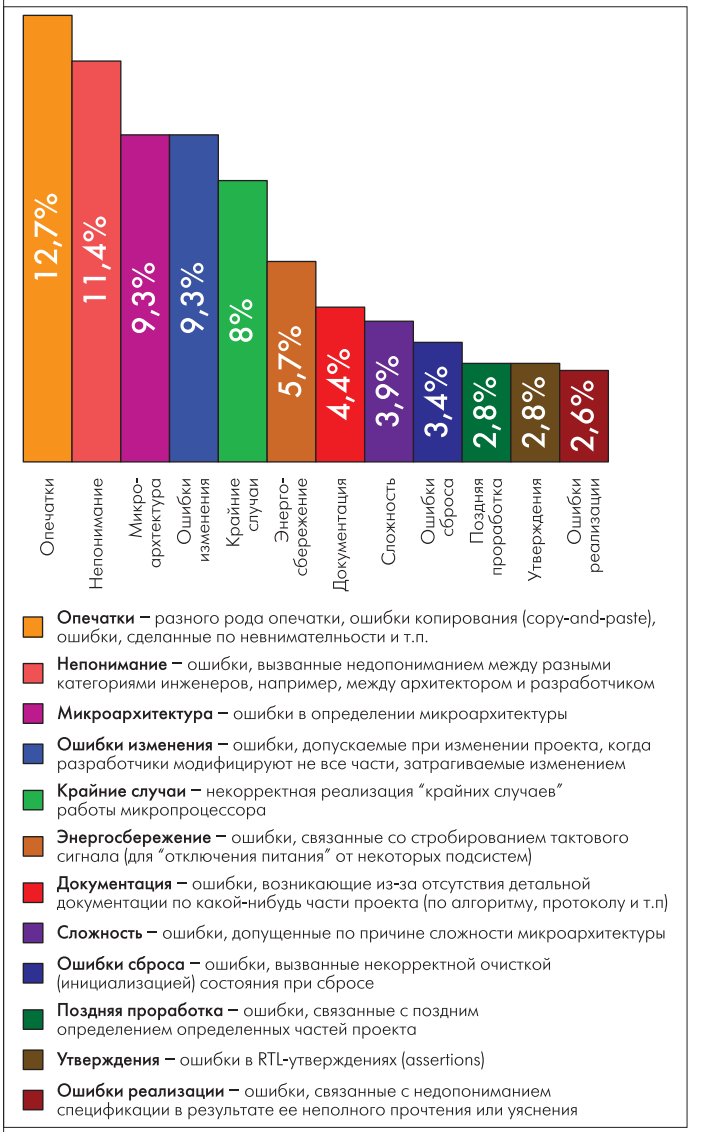

Рис. 3. Основные типы ошибок проектирования (Pentium 4)

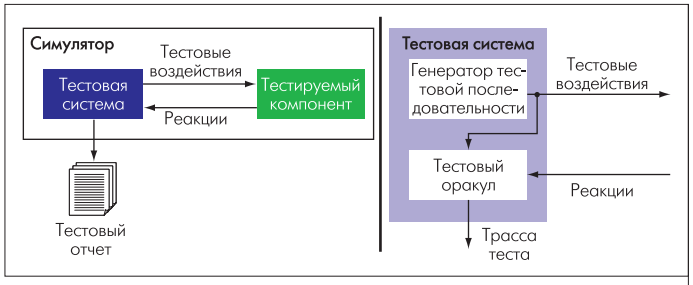

Рис. 4. Схема имитационного тестирования (а) и структура тестовой системы (б)

Типы ошибок в микропроцессорах

На наш взгляд, интерес представляет не только количество обнаруженных ошибок, но и также их типы. Согласно [9], более 75% ошибок, обнаруженных при проектировании Pentium 4, можно отнести к 12 типам (рис. 3). Отметим, что некоторые из этих типов очень похожи, поэтому четкой границы между ними нет.

Подробнее о тестировании

Поскольку имитационное тестирование является самым распространенным подходом к верификации, остановимся на нем подробнее. В дальнейшем под тестируемым компонентом будем понимать либо модель микропроцессора в целом, либо модель отдельного модуля. Для тестирования необходима тестовая система (рис. 4а) – специализированная программа, написанная на том же языке, что и тестируемый компонент (Verilog, VHDL), на языке программирования общего назначения (C, C++), либо на специализированном языке верификации аппаратуры (OpenVera, SystemVerilog). В некоторой степени как тестовую систему можно рассматривать программу, написанную на ассемблере или языке высокого уровня, которая после компиляции и загрузки в память выполняется моделью микропроцессора (тестирование с помощью тестовых программ) [10].

Работа тестовой системы и тестируемого компонента происходит в симуляторе. Тестовая система подает на тестируемый компонент тестовые воздействия (устанавливая входные сигналы или, если тестовой системой является программа, создавая с помощью команд определенные ситуации в работе микропроцессора) и осуществляет проверку правильности реакций на них (считывая выходные сигналы или проверяя значения регистров). В процессе тестирования тестовой системой создается тестовый отчет (трасса теста), который используется для анализа ошибок и оценки полноты тестирования.

В общем случае тестовая система решает три основные задачи: генерирует тестовую последовательность, проверяет правильность поведения тестируемого компонента, оценивает полноту тестирования. С учетом этого разбиения и тестирования на задачи обычно детализируют архитектуру тестовой системы, различая в ней такие компоненты, как генератор тестовой последовательности и тестовый оракул (рис. 4б).

Такая архитектура тестовой системы применима как для аппаратного, так и для программного обеспечения. Например, она используется в технологии тестирования UniTESK, разработанной в ИСП РАН [11, 12]. Современные методологии верификации аппаратуры, в частности, OVM (Open Verification Methodology), предлагают дополнительную детализацию компонентов тестовой системы [13].

Генератор тестовой последовательности создает последовательность тестовых воздействий, на которой проверяют поведение тестируемого компонента. Он разрабатывается таким образом, чтобы создавать разнообразные сценарии работы тестируемого компонента.

Тестовый оракул оценивает соответствие поведения тестируемого компонента требованиям к нему и выносит вердикт об их соответствии или несоответствии. Тестовые воздействия, реакции, а также свой вердикт оракул записывает в тестовый отчет. Если тестовая система реализована в виде программы, роль тестового оракула выполняют специальные команды этой программы, проверяющие значения регистров микропроцессора после выполнения команд тестового воздействия [10].

В настоящее время существует множество методов тестирования, которые по-разному подходят к генерации тестовой последовательности, проверке правильности поведения и оценке полноты тестирования. Рассмотрим современные подходы к решению этих основных задач тестирования.

Генерация тестовой последовательности. Качество тестирования напрямую зависит от используемых тестовых последовательностей (от того, насколько полно они охватывают ситуации, возможные в работе микропроцессора; какие в них взаимодействия между его модулями вызывают). Для построения тестовых последовательностей применяют несколько основных методов, среди которых ручная разработка, случайная генерация, генерация на основе шаблонов и генерация тестов на основе конечных автоматов.

Ручная разработка тестов до сих пор используется на практике, хотя доля тестов, разработанных вручную, постепенно снижается. "Вручную" проверяют "тонкие" ситуации в работе микропроцессора, описание и создание которых сложно формализовать.

Самым распространенным "дешевым" методом автоматического построения тестов является случайная генерация. Несмотря на простоту, "случайные тесты" позволяют быстро обнаруживать значительное число ошибок в проекте. Кроме того, они могут создавать ситуации, которые сложно предугадать заранее, но которые, тем не менее, интересны для тестирования. Наиболее известный коммерческий генератор случайных тестов для микропроцессоров – инструмент RAVEN, разработанный компанией Obsidian Software [14].

Развитием случайной генерации тестов стала генерация на основе шаблонов. Шаблоном называется абстрактное

представление теста, в котором вместо конкретных значений входных параметров (операндов команд или входов модуля) указывают ограничения, которым они должны соответствовать. Генератор, используя механизм разрешения ограничений, строит случайное значение, удовлетворяющее заданной системе условий. Подход на основе шаблонов поддерживается, например, в генераторах тестовых программ Genesys-Pro (IBM Research) [15] и MicroTESK (ИСП РАН) [12]. В последнем, кроме того, есть поддержка автоматической генерации тестовых шаблонов на основе комбинаторных техник.

Конечные автоматы являются широко используемым формализмом для представления аппаратных систем и часто используются для генерации тестовой последовательности. Как правило, тесты строятся путем обхода графа состояний автоматной модели тестируемого компонента. Автоматная модель может разрабатываться вручную, строиться на основе статического анализа кода RTL-модели или либо на основе формальных спецификаций. На использовании автоматных моделей для построения тестовых последовательностей базируется, в частности, технология UniTESK и поддерживающие ее инструменты [11].

Проверка правильности поведения. Даже если тестовые последовательности охватывают все ситуации в работе тестируемого компонента, ошибки невозможно обнаружить, не проанализировав поведение компонента. Существует три основных метода проверки правильности поведения: ко-симуляция, самопроверяющие тесты и формальные спецификации.

Для ко-симуляции отдельно от тестируемого компонента разрабатывают его исполнимую модель, которая имеет ту же функциональность, но представлена в более абстрактной форме (эталонная модель). Язык, на котором разрабатывается эталонная модель, может отличаться от языка разработки тестируемого компонента. На тестируемый компонент и эталонную модель подается одна и та же тестовая последовательность, результаты работы компонента и модели сравнивают на соответствие. Если находится несоответствие между поведением тестируемого компонента и эталонной модели, выясняют, какая модель некорректна. После исправления ошибки тестирование продолжается. Достоинство ко-симуляции в том, что проверка правильности поведения компонента производится автоматически. Таким образом, можно задействовать автоматические генераторы тестов, что повышает эффективность тестирования.

Когда тесты нужно получить быстро, вместо ко-симуляции применяют подход под названием самопроверяющие тесты. Данный способ предполагает, что каждый тест, помимо тестовой последовательности, содержит проверки, которые необходимо провести после или во время воздействия на компонент. С одной стороны, при использовании самопроверяющих тестов не нужна эталонная модель, но, с другой стороны, требуются большие усилия на разработку тестов, поскольку их необходимо снабдить соответствующими проверками. Автоматическая генерация тестовых последовательностей для самопроверяющих тестов практически невозможна. Другой существенный недостаток подхода — сложность сопровождения тестов: при изменении тестируемого компонента приходится переделывать все тесты, связанные с измененной функциональностью.

Наконец, третий способ проверки правильности поведения основан на использовании формальных спецификаций (машиночитаемого представления требований к тестируемому компоненту). Частным случаем формальных спецификаций является эталонная модель, используемая для ко-симуляции (это исполнимая спецификация). Другой разновидностью спецификаций являются декларативные спецификации, которые описывают требования в виде набора ограничений (утверждений), которым должно удовлетворять поведение тестируемого компонента. Примерами являются контрактные спецификации в форме пред- и постусловий [16] и спецификации, основанные на темпоральной логике (OpenVera Assertions, SystemVerilog Assertions, Property Specification Language). Как ко-симуляция, подход на основе формальных спецификаций сочетается с автоматическим построением тестовых последовательностей, при этом разработать декларативные спецификации чаще бывает проще, чем исполнимую модель.

Оценка полноты тестирования. Очевидно, что полный перебор всевозможных значений входных сигналов при всевозможных состояниях микропроцессора невозможен, вообще говоря, не нужен (такое тестирование является избыточным). На практике нужен метод оценки полноты тестирования — критерий, который на основе значений некоторых метрик позволяет определить, когда можно завершать проектирование и приступать к производству.

Один из наиболее "древних" методов оценки полноты тестирования основан на измерении частоты обнаружения ошибок. Когда частота становится достаточно низкой и сохраняется такой в течение длительного времени, можно заканчивать тестирование. Метод кажется устаревшим, но в сочетании с другими подходами он активно используется современными производителями микропроцессоров.

Есть и другой подход, когда метрики полноты тестирования определяют на основе кода RTL-модели микропроцессора или формальной спецификации — такие метрики называются структурными и функциональными, соответственно. Идея этого метода состоит в следующем. На основе кода определяется набор тестовых ситуаций, которые в совокупности составляют метрику тестового покрытия. Самыми известными структурными метриками являются покрытие операторов (тестовой ситуацией здесь является срабатывание того или иного оператора, заданного в коде RTL-модели), покрытие путей (каждая ситуация соответствует определенно-

му путиивыполнения)ипокрытиевыражений(здесьтестовой ситуациейявляетсяопределеннаякомбинациязначенийистинностиусловий,входящихвлогическоевыражение).Критериемполнотытестированияобычноявляетсядостижение 100%-ногопокрытияситуацийврамкахвыбраннойметрики.

Дляспецификаций,посколькуэтокоднанекоторомязыке, такжеможноиспользоватьуказанныеметрики.Висполнимых спецификацияхоцениваютпреимущественнопокрытиеграфапотокауправления(покрытиепутей),авдекларативных— покрытиеусловийвограничениях,описывающихтребования (покрытиевыражений).Важноедостоинствофункциональных метрикввтом,чтоонибазируютсянатребованияхктестируемомукомпоненту,ихможноиспользоватьдлявсехреализаций с одной и той же функциональностью.

Напрактикедляоценкиполнотытестированияиспользуютсякомбинированныеподходы,учитывающиеразныепоказатели.Например,компанияMotorolaиспользуетследующую совокупностькритериевдляпрекращениятестированияизапуска чипа в производство [4]:

- *необнаруженоошибокпривыполнении40•10⁹тактовслучайных тестов;*
- *выполнен план верификации;*
- *достигнутыцели,заданныеструктурнымиифункциональными метриками;*
- *наблюдаетсяустойчивоеуменьшениечастотыобнаружения ошибок;*
- *на календаре наступила определенная дата.*

Несмотрянато,чтометодыверификациинепрерывносовершенствуются,онидажепоспеваютзапостояннымусложнениеммикропроцессоров.Еслипервыеустройствамогпроверитьбуквальноодинчеловекпутемнепосредственного анализасхем,тосейчаскомандыинженеров-верификаторов достигаютвнушительныхразмеров.Так,основакоманды,занимавшейсяпроверкойпроцессораPentiumPro,состоялаиз 10человек,впроектеPentium4книмприсоединилосьеще 60инженеров[9].Унаснетточныхданныхотносительнотекущегосостояниядел,но,скорееевсего,сейчассчетидетна сотни.Трудоемкостьверификациимикропроцессоровпросто поражает—тольконаформальнуюпроверкуPentium4было затраченообочеловеко-лет[4],вцеломжезатратынаверификацию составляют несколько сотен человеко-лет!

Мечтамногихпроизводителеймикропроцессоров—автоматическоедоказательствокорректностипроекта.Однаковближайшейперспективепрорывавэтойобластинепредвидится.Скорееевсего,формальныеметоды,какираньше,будутприменятьсялишьдлянаиболеекритичныхподсистем(хотявполне закономерно,чточислоисложностьтакихподсистембудетпостепенноувеличиваться).Основнымспособомверификациипока остаетсятестирование,нонужноотметить,чтосовременные методытестированияногоезаимствуютизформальныхметодов(например,использованиематематическихметодовдля построениятестовыхпоследовательностейиприменениеформальныхспецификацийдляпроверкиправильностиповедения). Скорееевсего,вскоромбудущемнасожидаетещеболеетесная интеграция разных методов верификации.

ЛИТЕРАТУРА

1. Кулямин В. Методы верификации программного обеспечения, 2008 – www.ispras.ru/~kuliamin/docs/VerMethods-2008-ru.pdf.

2. Beizer B. The Pentium Bug – An Industry Watershed. – Testing Techniques Newsletter, TTN Online Edition, September 1995.

3. Coe T., Tang P.T.P. It Takes Six Ones to Reach a Flaw. – Chinese University of Hong Kong. Technical Report 95-5(61), 1995.

4. N. Mokhoff. Intel, Motorola Report Formal Verification Gains. – EE Times, 06/21/2001.

5. Bergeron J. Writing Testbenches: Functional Verification of HDL Models. – Kluwer Academic Publishers, 2000.

6. Корниленко Д.Д. Проверка на дорогах. – www.ixbt.com/cpu/cpu_validation.html.

7. MIPS R4000PC/SC Errata, Processor Revision 2.2 and 3.0. – MIPS Technologies Inc., 1994, May 10.

8. Bhadra J., Abadir M., Ray S., Wang L. A Survey of Hybrid Techniques for Functional Verification. – IEEE Design & Test, 2007, v. 24, № 22, p. 112–122.

9. Bentley B. Validating the Intel Pentium 4 Microprocessor. – DAC 2001: Design Automation Conference, 2001, p. 244–248.

10. Камкин А.С. Генерация тестовых программ для микропроцессоров. – Труды Института системного программирования РАН, 2008, т. 14, ч. 2, с. 23–63.

11. Сайт, посвященный технологии тестирования UniTESK. – www.unitesk.com.

12. Сайт, посвященный исследованиям в области верификации аппаратуры, проводимым в ИСП РАН. – hardware.ispras.ru.

13. Сайт, посвященный методологии OVM. – www.ovmworld.org.

14. Страница, посвященная генератору RAVEN. – www.obsidiansoft.com/products-and-services/inroducing-raven/.

15. Страница, посвященная генератору Genesys-Pro. – www.haifa.ibm.com/projects/verification/genesys_pro/index.shtml.

16. Иванников В.П., Камкин А.С., Косачев А.С., Кулямин В.В., Петренко А.К. Использование контрактных спецификаций для представления требований и функционального тестирования моделей аппаратуры. – Программирование, 2007, №5, с. 47–61.

32-разрядный микроконтроллер рекордно малых размеров

КомпанияNXPSemiconductorsобъявилаовыпускеопытныхобразцов32-разрядногомикроконтроллеранабазепроцессораCortex-MотипаLPC1102,монтируемыхвультракомпактныйкорпус WL-CSP размером 2,17×2,32×0,6 мм.

НоваямодельLPC1102,входящаявсемействомикроконтроллеровLPC1100компании,имеетключевыехарактеристикиэтогосемейства–низкуюпотребляемуюмощность(неболее130мА/МГцвактивномрежиме),высокуюпроизводительностьиобширнуюфункциональность.Поутверждениюразработчиков,благодарявысокойпроизводительности,простотеприменения,маломуэнергопотреблениюи,чтоособенноважно,значительномууменьшениюразмеракодадля8-/16-битприложенийновыймикроконтроллерзначительнопроще применять, чем существующие 8-/16-бит устройства.

В микросхемуLPC1102входят32-Кбайтфлеш-памятьи8-КбайтОЗУ,четырехканальный10-битныйАЦП,приемопередатчикUARTинтерфейсSPI,два32-битныхтаймера,два16-битныхтаймераиодин24-битныйсистемныйтаймер.Кромеего,вмикросхемереализованаподдержкаSWD-отладкиипрограммированиясчетырьмяточкамиприрыванияидвумяточкаминаблюдения.Дляобеспечениямаксимальнойгибкостивсеодиннадцатьфункцийввода-выводавыполняютдополнительнуюфункциювводавыводаобщегоназначения.Микроконтроллеримеетвстроенный IRC-генераторсточностьюдо ±1%впромышленныхдиапазонахтемпературинапряжений. Возможнаподачатактовыхимпульсовсвнешнегоисточника.

МикроконтроллерLPC1102поддерживаютразличныеинструментальныесредстваразработкистороннихорганизаций,атакжесобственнаяплатформаразработки–LPCXpresso,котораяимеетмощнуюинтегрированнуюсредуразработки(IDE)набазеEclipse,совершеннонового интуитивнопонятногоинтерфейса,разработанногоспециалистамикомпанииNXP,атакжекомпиляторибиблиотеки,оптимизированныеподпроцессорCortex-M0.ОтладчикLPC-Linkиплаты для разработки предоставляютпользователямвсеинструменты,необходимыедля ускорения разработки продуктов и их вывода на рынок.

Крупносерийноепроизводствоновогомикроконтроллеранамеченоначетвертыйквартал2010года.Позднеекомпанияпланируетвыпуститьновыемикросхемымикроконтроллеров семейства LPC1100.

Дополнительнуюинформациюобэтойплатформеможно найти по адресу www.nxp.com/lpcxpresso.