УДК 004.052.42

Запуск ОС Linux как этап функционального тестирования микропроцессоров

Е.В. Ровинский, П.А. Чибисов

Федеральное государственное бюджетное учреждение науки Научно-исследовательский институт системных исследований РАН,

chibisov@cs.niisi.ras.ru

***Аннотация* — В статье рассмотрен метод тестирования микропроцессоров приложениями пользователя под ОС Linux, обсуждаются способы получения тестовых программ и приложений, приведен пример репрезентативного набора тестов, предложен алгоритм действий при нахождении ошибки в микропроцессоре.**

***Ключевые слова* — тестирование микропроцессора, RTL-модель, ПЛИС-прототип, тестовый кристалл, ОС Linux.**

I. Введение

По данным различных источников трудозатраты на верификацию и валидацию микропроцессора (МП) составляют около 70% общих трудозатрат на разработку. Так как сложность современных микропроцессоров постоянно растёт, для проведения работ по верификации в заданное (ограниченное согласно плану верификации) время требуется применение разнообразных методов и средств. Процесс верификации процессора включает в себя три этапа:

1) Верификация RTL-модели и ПЛИС-прототипа микропроцессора.

2) Валидация тестового кристалла (first-pass silicon validation).

3) Валидация серийно выпускаемых экземпляров СБИС (post-silicon validation).

Время нахождения и стоимость исправления ошибок, найденных на второй и третьей стадиях, на несколько порядков выше, чем на стадии верификации RTL-модели, поэтому основные усилия инженеров-верификаторов направлены на нахождение ошибок на первой стадии.

Однако верификация RTL-модели имеет существенный недостаток – очень низкую скорость выполнения инструкций на модели, приводящий к невозможности выполнения достаточного для полноты покрытия модели количества тестов в разумное время.

Поэтому верификацию и тестирование следует продолжать на стадии ПЛИС-прототипа, тестового кристалла и даже после выпуска партии готовых СБИС. На второй и третьей стадиях верификации благодаря возросшей скорости выполнения тестов появляется возможность запустить гораздо более объемные тесты, в том числе операционную систему (ОС) и приложения пользователя, что позволяет существенно увеличить покрытие кода RTL-модели (при условии логической и функциональной эквивалентности модели и тестового кристалла). Состоятельность этого подхода подтверждает информация различных фирм-изготовителей об ошибках в уже выпущенных микропроцессорах.

В статье рассматриваются существующие методы верификации и тестирования современных микропроцессоров. Особенное внимание уделяется методу тестирования ПЛИС-прототипов и тестовых кристаллов микропроцессоров реальными пользовательскими приложениями под ОС Linux, рассматривается общий алгоритм действий при нахождении ошибки в микропроцессоре.

II. Загрузка ОС

Конечная цель верификации – дать пользователю уверенность, что микропроцессор будет корректно исполнять любую последовательность инструкций, допустимых системой команд (instruction set architecture), которая была получена компиляцией программ, либо написана вручную. Для этого проверяется корректная работа компиляторов и инструментальных программ (toolchain), приложений пользователя, ОС реального времени (real time OS) и ОС Linux. С одной стороны, запуск нескольких приложений или тестов не является достаточным для полноценного тестирования. С другой стороны, даже применение совокупности псевдослучайных, направленных, переборных методов тестирования не гарантирует отсутствия ошибок, связанных с человеческим фактором. Это связано с тем, что стопроцентное покрытие блока в рамках какой-либо выбранной метрики покрытия не дает гарантии его правильного функционирования. Поэтому на практике преследуются обе цели: выбирается репрезентативный набор тестовых приложений и проверяется, что они

корректно работают на тестируемом микропроцессоре, а кроме этого выполняется систематизированный тестовый план, реализующий тестовое покрытие блоков МП. В данной работе предлагается развитие метода запуска ОС и существующего программного обеспечения (ПО) под ОС. Этот метод обладает следующими достоинствами:

- ОС является большим программным комплексом, использующим различные особенности микропроцессорной архитектуры. Поэтому ОС – сама по себе большой архитектурный тест.
- Пользовательские пакеты программ под ОС, как правило, содержат встроенные в дистрибутивы пакетов наборы готовых самопроверяющихся тестов.
- Найденные ошибки в результате прохождения тестов ПО часто служат основой для создания новых шаблонов псевдослучайных тестов.
- Простота процесса запуска ПО и тестов под него означает невысокие накладные расходы на один тест.
- Возможность использования огромного количества уже существующих тестов под ОС, таких как LTP (Linux Test Project), SPEC CPU 2000/2006.
- Возможность изучения производительности процессора, например, анализ трасс на потактовом эмуляторе и результатов тестов, измеряющих производительность на готовом кристалле.
- Возможность решения задач на родной платформе, таких как программирование в среде ОС специализированных сопроцессоров, входящих в состав разрабатываемого микропроцессора.
- Возможность выбора из множества программ пользователя подходящего набора тестов для создания блока функционального контроля, на котором проводятся приемочные испытания изготавливаемых СБИС микропроцессоров.

Недостатки данного метода следующие:

- Множество повторяющихся массивов кода приводит к избыточности метода без значительного увеличения тестового покрытия.
- Локализация ошибки нередко затруднена сложностью ПО.
- Тестируемый объект представляется в виде черного ящика, хотя есть возможность проверки RTL-модели механизмом утверждений (assertion checkers).
- Высокая стоимость исправления ошибки, если она найдена в серийно выпускаемой СБИС.

Частный случай применения метода тестирования микропроцессоров готовыми пользовательскими приложениями описывается создателями процессора OpenRISC 1200 [1]. Авторы статьи используют для верификации лишь внутренние тесты компилятора GCC, не используя при этом ОС. Этот подход позволяет запускать в общей сложности более 80000 тестов для тестирования микропроцессора. Однако есть возможность расширить данный подход, включив в перечень тестов загрузку ОС, другое ПО и его внутренние проверки.

Наличие огромного количества свободно распространяемых программ с открытым исходным кодом, большинство из которых имеют встроенные механизмы самопроверки, делает возможным выделить запуск приложений под ОС Linux в отдельный подход к верификации. Конечно, применение описываемого метода не исключает, а только дополняет современный набор методов и средств тестирования и верификации микропроцессоров и их моделей. Стоит отметить факт создания специальной ОС для верификации аппаратного обеспечения [2].

Многие авторитетные фирмы-разработчики микропроцессоров признают полезность как можно более ранней загрузки какой-либо операционной системы (ОС) на разрабатываемой RTL-модели [3]. Успешность этой операции зачастую дает разработчикам больше уверенности в правильности уже проделанной работы, чем тысячи прошедших тестов.

III. Объекты применения методики

Существует пять различных объектов, к которым применима описываемая методика в течение проведения работ по верификации и тестированию:

- RTL-модель.
- Покомандный эмулятор (C-модель, “golden model”).
- ПЛИС-прототип.
- Тестовый кристалл микропроцессора.
- Выпущенная в серийное производство СБИС.

Для загрузки ОС Linux процессору требуется выполнить порядка 10^9-10^{10} инструкций. Скорость симуляции RTL-модели весьма низка – она составляет около 10^3 инструкций в секунду при моделировании на современном ПК (время загрузки ОС порядка 30 суток). Эта проблема решается параллельным запуском на нескольких рабочих станциях нескольких копий RTL-модели. Сначала ОС запускается на эмуляторе (скорость моделирования $(1–3)*10^6$ инструкций в секунду), при этом состояния виртуальной машины периодически сохраняются в соответствующие файлы. Таким образом, получается набор «снимков» промежуточных состояний виртуальной машины в процессе запуска ОС. Затем, на нескольких рабочих станциях запускается несколько экземпляров RTL-модели, каждому их которых передается свой «снимок» состояния виртуальной машины. Другими словами, «длинный» процесс запуска ОС из начального состояния делится на несколько коротких процессов, запускаемых независимо и параллельно с остальными [4]. При этом ОС можно загрузить за двое суток.

Критерием успешной загрузки ОС служит результат покомандного сравнения трасс выполнения кода на RTL-модели с трассами, полученными на покомандном эмуляторе. Как только ОС загружена на RTL-модели, код проекта, написанный на языке Verilog или VHDL, синтезируется в ПЛИС. ПЛИС-прототип микропроцессора, несмотря на относительно невысокую частоту функционирования (десятки мегагерц), позволяет создать отладочный стенд, представляющий полнофункциональную плату с возможностью запуска ОС Linux (время загрузки порядка 5 минут) и программ пользователя без создания промежуточных «снимков».

IV. Репрезентативный тестовый набор

В качестве тестов в рассматриваемой методике предлагается запускать реальные пользовательские пакеты программ и встроенные в дистрибутивы пакеты тестов (выполнив команду ОС «make check»). Источником большинства тестовых программ является глобальная сеть Интернет, например, пакеты набора GNU [5]. Для сборки программ требуется наличие нативного или кросс-компилятора для различных языков программирования (наиболее часто встречаются C, C++ и Fortran). Тестовый набор зависит от имеющегося на верификацию времени, скорости работы ПЛИС-прототипа или тестового кристалла. Например, он может включать в себя запуск следующего ПО.

- Операционные системы:
  - сборка ядра ОС Linux: 32/64-разрядные версии дистрибутивов Red Hat и Debian.
- Пакеты тестов:
  - пакет тестов LTP;
  - пакет вычислений простых чисел glucas;
  - тест памяти memtester;
  - тесты производительности процессора dhrystone, whetstone, coremark, Lmbench, HPL (MPI + ATLAS/GotoBLAS), CPU SPEC2000, CPU SPEC2006.
- Прикладное ПО:
  - среда рабочего стола KDE или Gnome;
  - игры kdegames, wormux;
  - пакет офисных программ koffice;
  - интернет-браузер mozilla-firefox.
- Приложения, получаемые с помощью команды apt-get в ОС Debian [6].
- Встроенные тесты (make check) пакетов пользовательского ПО:
  - компиляторы набора GNU gcc, g++, gfortran, gcj;
  - трансляторы языков программирования ruby, python, perl, php;
  - кодеры/декодеры аудио mp3/flac lame, flac и видео ffmpeg;
  - математические библиотеки точных вычислений gmp, mpfr, mpc, gappa.

Предложенный перечень тестовых задач является начальной точкой при рассмотрении тестового набора. Примером составления набора для определенной задачи является подход, представленный в работе [7]. Интерес представляет сборка исходных кодов различным образом для получения как можно большего разнообразия исполняемого кода:

- различными версиями компиляторов;
- с несколькими ключами оптимизации кода;
- под разные, но совместимые модели процессоров (так как получается различное заполнение конвейера планировщиком инструкций);
- для различных ABI (бинарный интерфейс приложений – application binary interface).

Всё описанные выше ПО можно компилировать непосредственно на тестовом кристалле, используя нативный компилятор. Еще одним перспективным направлением получения уникального кода является кросс-компиляция. Использование быстрых инструментальных серверов для компиляции тестовых приложений позволяет получать исполняемый код за более короткое время. В результате появляется возможность получить большее количество вариантов исполняемого кода. Использование кросс-компиляции имеет ряд преимуществ. Кроме того, что скорость получения кода многократно возрастает по сравнению с нативной компиляцией, есть возможность сборки образов ОС Linux по уже подготовленным заготовкам образов. К таким средствам можно отнести Buildroot [8]. Это средство включает в себя набор инструкций сборки пакетов (Makefile) и меню выбора параметров сборки системы. Buildroot позволяет выбирать параметры получаемого образа ОС и набор стандартных инструментальных программ, которые также будут включены в собираемый образ. После выбора параметров сборки запускается процесс компиляции, который занимает от получаса до двух часов в зависимости от выбранного набора пакетов на инструментальном сервере архитектуры Intel x86 с частотой процессора 3ГГц.

Также есть возможность собирать загрузочный образ ОС Linux без использования дополнительных инструментальных средств. Вместо этого можно использовать набор инструкций создания системы ОС Linux с нуля, используя только исходные коды необходимого программного обеспечения – «Linux From Scratch» («Linux с нуля») [9], или версию инструкции для кросс-компиляции Cross Linux From Scratch («Linux с нуля с помощью кросс-компиляции») [10]. Это позволяет добиться большей гибкости при сборке системы, однако требует больших трудозатрат по сравнению с системой Buildroot.

В качестве кросс-компилятора можно использовать собранный вручную кросс-компилятор GCC [11], либо набор готовых инструментов (например, cross-tools NG [12]), включающий кросс-компилятор.

V. Алгоритм поиска ошибок

С помощью описываемого метода в ходе верификации моделей двух микропроцессоров с архитектурой MIPS64 было найдено несколько десятков ошибок на стадии RTL-модели и ПЛИС-прототипа, а также 5 ошибок на стадии выпуска тестового кристалла. Предлагается следующий алгоритм действий при нахождении ошибки:

- Подробно описать условия возникновения ошибки.
- Локализовать исполняемый файл и shell-команду, запуск которой приводит к обнаружению ошибки.
- Снизить частоту работы микропроцессора; отключить суперскалярность, предсказание переходов и другие аппаратные возможности.
- Проверить, как ведет себя тестовый случай на покомандном эмуляторе.
- Если это программная ошибка, то исправить тест.
- Найти ошибку в RTL-коде и исправить, после чего убедиться в исчезновении ошибки, запустить регрессионное тестирование.

Анализ ошибок позволяет подтвердить известную истину: “то, что не было протестировано - не работает”. Если описываемый в данной статье метод позволил найти новую ошибку (в особенности, если это ошибка в тестовом кристалле или в серийно выпускаемом микропроцессоре), то такая ошибка требует тщательного рассмотрения и исправления, либо нахождения обходных путей. Как правило, исследование причин возникновения ошибок позволяет найти недостатки в реализации других групп тестов. При изучении кода ошибок следует зафиксировать такие условия как:

- трассы выполнения кода на потактной модели, набор инструкций;
- сегментация памяти;
- режим работы процессора;
- исключительные ситуации и прерывания;
- кэш-память;
- сопроцессоры.

По результатам разбора причин возникновения ошибок могут быть написаны новые направленные тесты, созданы новые шаблоны для псевдослучайного тестирования.

VI. Заключение

В статье рассматривается метод тестирования RTL-моделей, ПЛИС-прототипов, тестовых кристаллов и серийных СБИС микропроцессоров реальными пользовательскими приложениями под ОС Linux (тестирование на основе существующего ПО). В качестве тестов выступают пакеты тестов под ОС, сама ОС, встроенные механизмы диагностики и самотестирования пакетов ПО, реальные пользовательские приложения. Данный метод был успешно применен в совокупности с остальными для верификации разрабатываемых микропроцессоров и позволил найти ряд ошибок в микропроцессоре с архитектурой MIPS64.

Литература

[1] Bennett J. Processor Verification using Open Source Tools and the GCC Regression Test Suite: A Case Study // Design Verification Club Meeting at Infineon, Bristol, September 2010. URL: http://www.embecosm.com/articles/ear6/dvc-or1k-verification.pdf (дата обращения: 21.06.2012).

[2] Gong L., Lu J. Verification-Purpose Operating System for Microprocessor System-Level Functions // IEEE Design & Test of Computers. 2010. P. 76-85.

[3] Functional Verification of a Multiple-issue, Out-of-Order, Superscalar Alpha Processor - The DEC Alpha 21264 Microprocessor / Taylor S., Quinn M., Brown D., Dohm N., Hildebrandt S., Huggins J., and Ramey C. // In Proceedings of DAC. 1998. P. 638-643.

[4] Лесных А.А., Широков И.А. Покомандная модель проектируемого 64-битного процессора // Методы встречной оптимизации в задачах обработки сигналов. М.: НИИСИ РАН, 2005. С. 96-112.

[5] GNU Project Archives. URL: ftp://ftp.gnu.org/ (дата обращения: 21.06.2012).

[6] Debian – универсальная операционная система. URL: http://www.debian.org/index.ru.html (дата обращения: 21.06.2012).

[7] Benchmark and Application Selection for the Evaluation of the Microgrid Architecture and its Tool Chain / Rolls D., Penczek F., Sinkarovs A., Joslin C. URL: http://www.apple-core.info/wp-content/apple-core/2009/06/d22.pdf (дата обращения: 21.06.2012).

[8] Buildroot: making Embedded Linux easy. URL: http://buildroot.uclibc.org/ (дата обращения: 21.06.2012).

[9] Linux From Scratch. URL: http://www.linuxfromscratch.org/ (дата обращения: 21.06.2012).

[10] Cross Linux From Scratch. URL: http://cross-lfs.org/ (дата обращения: 21.06.2012).

[11] GCC, the GNU Compiler Collection. URL: http://gcc.gnu.org/ (дата обращения: 21.06.2012).

[12] Crosstool-NG. URL: http://crosstool-ng.org/ (дата обращения: 21.06.2012).