© 2012 г. С.И. АРЯШЕВ, канд. техн. наук,
Н.В. НИКОЛИНА,
П.А. ЧИБИСОВ
(Федеральное государственное бюджетное учреждение науки Научно-исследовательский институт системных исследований РАН, Москва)

ОРГАНИЗАЦИЯ РЕГРЕССИОННОГО ПРОЦЕССА ТЕСТИРОВАНИЯ RTL-МОДЕЛЕЙ МИКРОПРОЦЕССОРОВ

В статье рассматривается автоматизация процесса регрессионного тестирования RTL-моделей микропроцессоров. Принцип регрессии применен как для функционального контроля, так и для оценки производительности. Для ускорения получения результата регрессионного тестования программы запускаются параллельно на грид-системе.

REGRESSION TESTING OF MICROPROCESSOR RTL MODELS / S.I. Arjashev, N.V. Nikolina, P.A. Chibisov (Scientific research Institute for system analysis of RAS, Nahimovsky prospect 36-1, Moscow 117218, Russia, E-mail: niisi@niisi.msk.ru). An automation regression testing of microprocessor RTL-models is considered. Functional control and performance evaluation are done in regression environment. For acceleration test programs are run in parallel on grid.

1. Введение

При верификации RTL-модели микропроцессора (Register-transfer level модели уровня регистровых передач) одной из ключевых задач является организация регрессионного тестирования.

Среда регрессионного тестирования может содержать тысячи тестов, которые необходимо запускать и перезапускать при любых изменениях, вносимых в проект [1]. Это объясняется сложностью модели и тем фактом, что изменения в одном блоке могут привести к проявлению скрытого дефекта в другом. Кроме того, части предыдущей версии проекта часто интегрируются в новый проект в качестве IP (Intellectual Property) блоков. Создание автоматизированной системы тестирования проекта может существенно упростить интеграцию блоков из предыдущих проектов.

Принцип регрессии должен быть применим и для контроля производительности. Часто изменения, связанные с исправлением каких-либо ошибок связаны с внесением дополнительных задержек, отрицательно сказывающихся на производительности. Крупные компании, такие как Intel, NetLogic имеют внутренние закрытые тесты для оценки производительности [2]. Таким образом, в среду регрессионного тестирования необходимо включить систему оценки и контроля производительности.

Для ускорения процесса регрессионного тестирования тестовые программы должны запускаться параллельно.

2. Особенности тестов для RTL-модели

Параллельно с разработкой RTL-модели микропроцессора создаются тестовые программы, направленные на проверку функционирования его отдельных блоков, механизмов, инструкций, оценку производительности. Такие программы могут быть написаны на языке Ассемблера, но это могут быть также программы, полученные от генератора псевдослучайных тестов, программы на C, фрагменты программного кода, полученные при загрузке ОС.

Наибольшая часть тестов построена таким образом, что положительным результатом прохождения теста на RTL-модели является совпадение работы данного теста с его работой на образцовом эмуляторе [3]. Это обусловлено сложностью написания полностью самопроверяющихся тестов из-за большого количества тестовых случаев, сложности расчета результатов арифметических операций, и так далее, но для полноценной верификации проекта осуществление проверки результатов всех инструкций является обязательным условием. Поэтому существенная часть методов верификации опирается на использование некоторых эталонных моделей процессора. Среди этих методов наиболее распространен параллельный запуск тестовых программ на RTL-модели процессора и его эталонной модели и сравнение содержимого регистров и памяти в обеих моделях после каждого такта или после каждой инструкции. В качестве эталонной модели используются, как правило, эмуляторы, реализованные на языке программирования высокого уровня. Такие эмуляторы описывают процессор на уровне его программной модели и являются относительно простыми в разработке и отладке.

Автоматизация компиляции, запуска на образцовом эмуляторе и RTL-модели, анализа полученных результатов дает возможность избавить разработчиков от выполнения рутинных операций, что, несомненно, повышает эффективность работы, позволяя сосредоточиться на исправлении найденных ошибок и дальнейшем совершенствовании модели.

3. Автоматизация регрессионного тестирования

3.1. Регрессионное тестирование для функционального контроля

Для реализации задачи автоматизированного тестирования RTL-модели была создана тестовая система, позволяющая удовлетворить следующим требованиям:

- Быть универсальным хранилищем для всех разрабатываемых тестов с единым форматом файлов;
- Работать под ОС Linux / Windows (Cygwin);
- Быть легко настраиваемой;
- Быть легко наращиваемой;
- Поддерживать разбиение тестов на группы с возможностью запуска как отдельной группы, так и всех тестов;
- Иметь возможность автоматического сравнения результатов работы теста с результатами, полученными на образцовом эмуляторе;
- Автоматически составлять таблицу-отчет, упрощающую поиск ошибок в тестах, закончившихся с ошибкой;

– Иметь возможность легко отличить тесты, закончившиеся успешно, от тестов, закончившихся неуспешно, без обращения к отчету;
– Иметь возможность временного исключения одного или нескольких тестов из тестирования.

Разработанная система автоматизированного тестирования представляет собой набор тестов, разбитых на группы, а также набор управляющих программ, позволяющих осуществить запуск выбранных групп тестов и собрать полученные результаты. Управляющие программы написаны на языках интерпретатора Bourne shell и PERL.

Схема функционирования тестовой системы приведена на рис.1. Файлы, относящиеся к одному тесту, хранятся в отдельной директории, в имени которой содержится условное название группы, к которой относится данный тест. Количество групп может быть любым, но вполне достаточно 10-15 групп, при этом признаком деления на группы может служить то, на какой блок направлены находящиеся в ней тесты (например, группа тестов, направленная на тестирование АЛУ, блока вещественной арифметики, буфера трансляции адресов и т.д.).

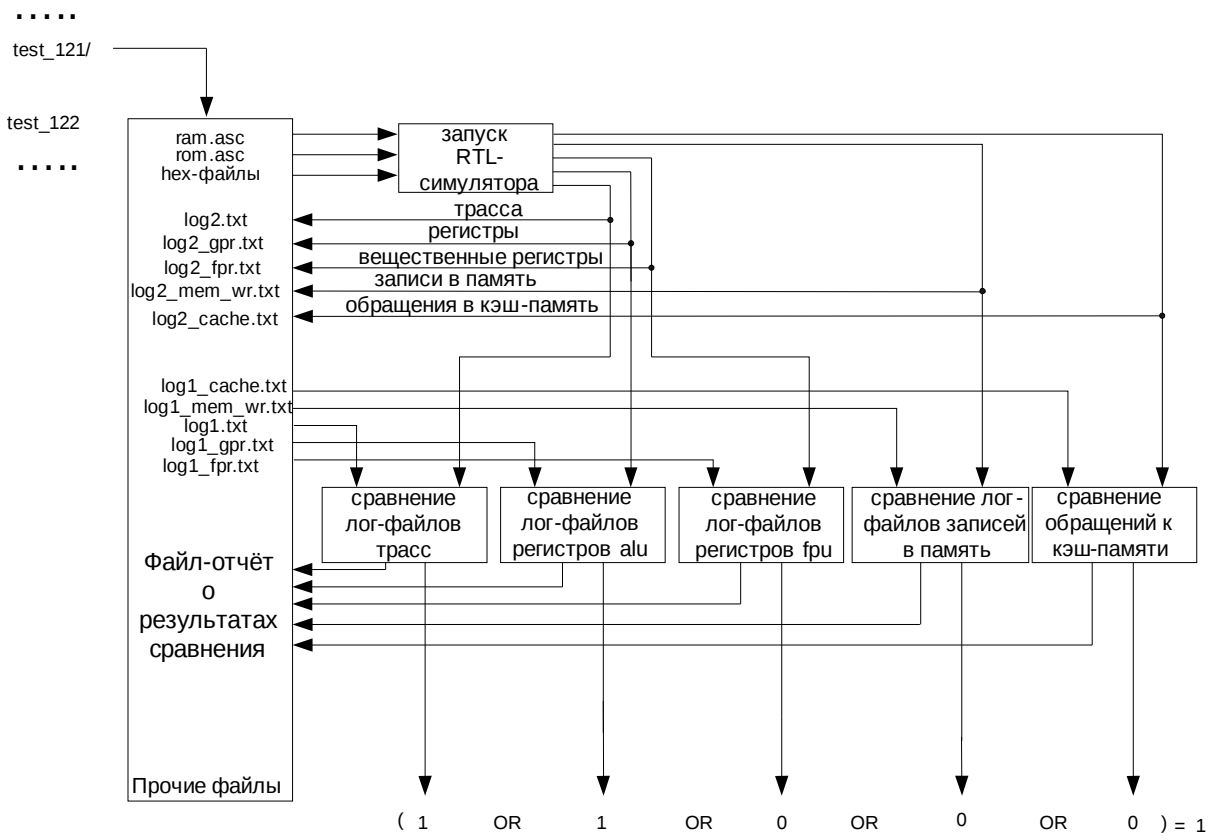

Рис. 1. Схема работы системы функционального контроля

Помимо исходного текста тестовой программы и других файлов, необходимых для сборки теста (если такие файлы имеются), каждая директория с тестом должна содержать файлы, необходимые для запуска теста на RTL-модели и образцовом эмуляторе. Это могут быть бинарные файлы либо файлы в шестнадцатеричном формате, полученные при компиляции теста, а также управляющие программы. После запуска теста каждая тестовая директория хранит файлы с трассами, изменениями содержимого регистров и памяти после каждой выполненной команды. Здесь же находятся файлы, содержащие результаты сравнения работы теста на RTL-модели и

образцовом эмуляторе, с указанием номера инструкции, на которой было обнаружено расхождение в трассах либо содержимом регистров или памяти.

При запуске управляющей программы производится подсчет числа тестов, входящих в каждую группу, а также общего количества тестов. Далее пользователю предлагаются варианты запуска одной или нескольких групп. Для выбранной группы выполняются все тесты, входящие в эту группу, при этом:

1) подготавливаются все необходимые файлы для запуска текущего теста;

2) тест запускается на RTL-модели и на образцовом эмуляторе; получившиеся в результате файлы, содержащие трассы выполненных в тесте инструкций, изменения содержимого регистров целочисленного и вещественного регистровых файлов, а также содержимое кэш-памяти и внешней памяти помещаются в директорию с тестом;

3) по очереди запускаются программы, сравнивающие полученные от эмулятора и RTL-модели файлы. С их помощью определяется номер инструкции, вызвавшей расхождение трасс, либо номер инструкции, приведшей к расхождению содержимого регистров. Результат сравнения файлов вместе с дополнительной информацией, облегчающей анализ несовпадающих инструкций или данных, сохраняется в общий файл-отчёт;

4) в окончании имени директории с данным тестом появляется символ «+» или «-», символизирующий «успех» либо «провал» этого теста. Это позволяет отличать прошедшие тесты от тестов, содержащих ошибки, не открывая файл-отчет, а также не дожидаясь окончания прохождения базы тестов на проекте.

5) пункты 1-4 выполняются для всех тестовых директорий в выбранных пользователем группах тестов;

6) составляется файл-отчет о текущем запуске тестов.

По завершении работы выбранной группы тестов, все директории с тестами обретут суффикс «+» либо «-». Этим символом обозначается текущее состояние теста («прошел» / « не прошел»). Для быстрого анализа тестов, закончившихся неуспешно, создается файл с перечнем всех ошибочно прошедших тестов с локализацией ошибок. Все тесты, закончившиеся неуспешно, передаются ответственным разработчикам вместе с файлом-отчетом.

В качестве регрессионного теста также используется загрузка операционной системы Linux. Так как для загрузки ОС Linux процессору требуется выполнить порядка 10^9 – 10^{10} инструкций, а скорость симуляции RTL-модели сложного микропроцессора весьма низка – она составляет до 10^4 инструкций в секунду при моделировании на современном ПК, то целесообразно использовать механизм среза-восстановления состояния модели микропроцессора для разбиения всей последовательности команд загрузки ОС на несколько подпоследовательностей, которые выполняются параллельно на разных вычислительных устройствах. Под срезом состояния понимается дамп ОЗУ, кэш-памяти всех имеющихся уровней, состояние регистровых файлов регистров общего назначения и сопроцессоров, TLB (translation lookaside buffer, буфер ассоциативной трансляции), а также значение PC (program counter, счетчик команд). Для быстрого получения среза состояния удобно использовать покомандный эмулятор (скорость моделирования порядка 10^6 инструкций в секунду). Критерием успешной загрузки ОС служит результат покомандного сравнения трасс выполнения кода на RTL-модели с трассами, полученными на покомандном эмуляторе.

Несмотря на удобство сравнения с образцовым эмулятором, в программной модели не всегда возможно реализовать все функции периферии. Поэтому для верификации

встроенного системного контроллера, а также систем на кристалле (СнК) целесообразно использовать отдельную базу самопроверяющихся тестов и другой критерий отсутствия ошибок. Таким критерием может быть определенное значение, занесенное в один из регистров общего назначения в случае успешного прохождения теста на модели. Еще одним аргументом в пользу разделения баз тестов для ядра процессора и системы на кристалле является увеличенное время прохождения теста на СнК, что в случае внесения изменений только в ядро процессора приведет к не целесообразно долгому ожиданию результата регрессионного тестирования.

3.2. Регрессионное тестирование для контроля производительности

В среду регрессионного тестирования также встроена автоматическая система контроля производительности. После прохождения регрессионной базы тестов на RTL-модели система выдает отчет не только о содержащихся в проекте ошибках, но и отчет о производительности, включающий название теста и значение IPC (instructions per cycle – инструкции за такт) для тех тестов, которые показали худший результат оценки производительности по сравнению с предыдущей версией RTL-модели. Система также выдает усредненное значение IPC всех тестов, позволяющее понять, насколько изменилась производительность микропроцессора в целом.

По аналогии с системой регрессионного тестирования для функциональной верификации RTL-модели микропроцессора создана среда для оценки и контроля производительности, основанная на микротестах (коротких программах, направленных на оценку производительности отдельных блоков). Для каждого блока сформированы наборы микротестов с различными тестовыми ситуациями. Изоляция каждого тестового случая в отдельном микротесте удобна тем, что разработчик имеет возможность его рассмотрения с использованием временных диаграмм и не тратит времени на поиск данной ситуации в большом и сложном тесте. На рис. 2 показана схема функционирования системы оценки производительности.

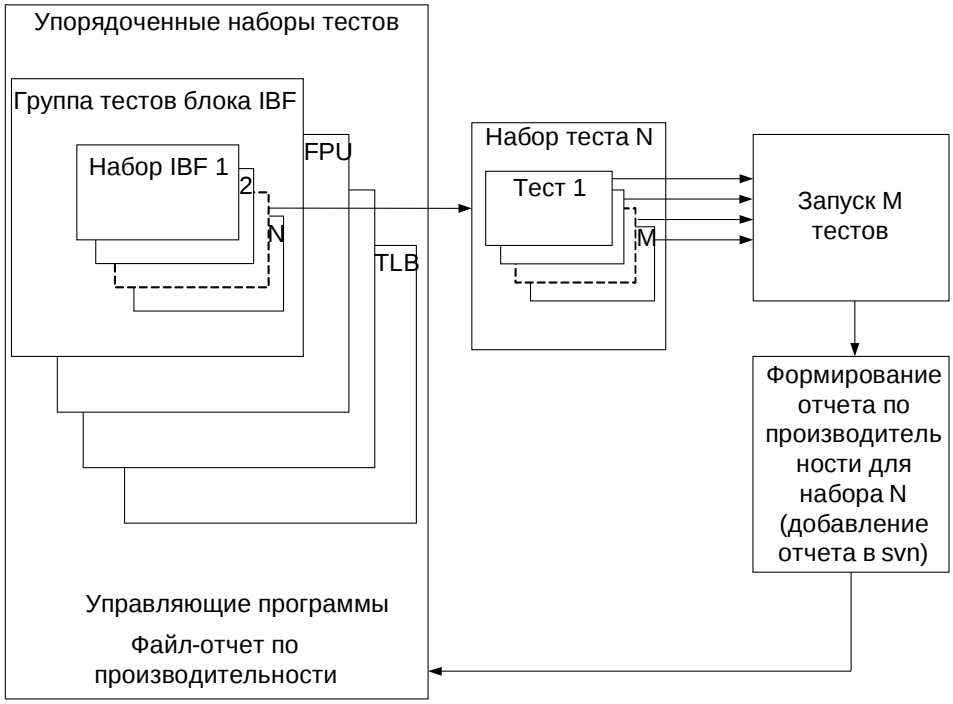

Рис. 2. Схема функционирования системы оценки производительности

Результатом регрессионного процесса измерения производительности является наглядная таблица с возможностью ее сравнения с предыдущими данными. Такая таблица содержит название теста, характеризующее реализованную в нем ситуацию, а также количество тактов, требуемое на ее выполнение. В качестве вспомогательного инструмента для сравнения результатов был использован инструмент отслеживания ошибок Trac [4]. Результаты измерения производительности выдаются в wiki-формате и сохраняются на странице в Trac со ссылкой на ревизию тестируемого проекта в системе контроля версий (svn от слова английского слова subversion). Таким образом, разработчик может увидеть итог прохождения тестов и, при необходимости сравнить текущий результат с одним из предыдущих. Благодаря наличию ссылки на проект в системе контроля версий, существует возможность увидеть, какие именно изменения в проекте привели к полученному результату. Кроме того, предусмотрен механизм, выдающий отчет об отличии полученных данных от эталонных. Эталонными данными может быть как максимально хороший результат, полученный ранее, так и желаемый результат, которого стремится добиться разработчик.

4. Запуск регрессионного тестирования на грид-системе

Для ускорения процесса регрессионного тестирования база тестов запускается параллельно на грид-системе: на узлах Linux-кластера с общими файловыми системами, единым списком пользователей на всех узлах, на которых установлен одинаковый набор ПО. На всех узлах могут запускаться пакетные задания, на некоторых узлах возможна интерактивная работа. Для организации вычислений в пакетном режиме была выбрана система SGE (Sun Grid Engine [5] – программное обеспечение для распределённых вычислений (грид), принадлежащее Sun Microsystems). Одним из главных преимуществ этой системы является ее доступность. На грид-системе НИИСИ РАН параллельно может быть запущено до 400 потоков.

Помимо запуска регрессионных баз тестов для RTL-модели на грид-системе могут быть запущены и другие виды пакетных заданий, а именно: псевдослучайное тестирование [6], моделирование NETLIST (представления СнК на библиотечных элементах) и другие задачи. Тест, проходящий на RTL-модели за несколько минут, может идти несколько часов на модели NETLIST. Таким образом, запустив 400 заданий на модели NETLIST можно на несколько дней сделать грид-систему недоступной для регрессионного тестирования. Поэтому, для того, чтобы регрессионная база тестов проходила за приемлемое время необходимо решить наиболее важную задачу – задачу разделения ресурсов между пользователями. Для этого создается несколько очередей с разным приоритетом, в которых могут быть запущены задачи пользователей. При этом, когда возникает необходимость запуска регрессионных баз тестов, они направляются в очередь с наибольшим приоритетом, приостанавливая задачи, находящиеся в других очередях. Такая организация необходима для того, чтобы разработчик мог получить результат регрессии уже за несколько часов, что позволяет отслеживать влияние любых, даже незначительных изменений, внесенных в проект. При этом загрузка ОС Linux ставится в очередь с наименьшим приоритетом, так как задача загрузки всей ОС занимает слишком много времени и не входит в состав обязательных тестов, исключением являются фрагменты кода, на которых была найдена ошибка в предыдущей версии RTL-модели.

5. Заключение

Среда регрессионного тестирования для функционального контроля позволяет избавить разработчика от рутинных операций по запуску тестовых программ. Результаты прохождения тестов на RTL-модели представляются в удобном для разработчика виде, файл-отчет содержит всю необходимую для отладки информацию.

Система регрессионного тестирования для оценки и контроля производительности позволяет в короткое время получить отчет об оценки производительности отдельных блоков микропроцессора, обнаружить «узкие места», а также сравнить результат оценки производительности либо с эталонным значением, либо с результатом, полученным на одной из предыдущих версий проекта.

Запуск на грид-системе позволяет ускорить получение результата в 300-350 раз по сравнению с запуском на одиночном вычислительном узле.

СПИСОК ЛИТЕРАТУРЫ

1. *Bergeron J.* Writing Testbenches: Functional Verification of HDL Models. Kluwer Academic Publishers, January 2000, 191p.
2. *Halfhill Tom R.* NetLogic’s XLP II Grows More CPUs. Microprocessor report, The Linley Group, October 2011, 5p.
3. *Лесных А.А., Широков И.А.* Покомандная модель проектируемого 64-битного процессора // Сб. статей под ред. академика РАН В.Ю. Бетелина. – М.: НИИСИ РАН, 2005. – С. 96–116.
4. Trac Integrated SCM & Project Management. URL: http://trac.edgewall.org/.
5. Sun Grid Engine 5.2.3 Manual. Sun Microsystems, Inc. July 2001. 456p.
6. *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.