УДК 004.054 DOI: 10.31114/2078-7707-2020-3-172-179

Методика тестирования процессорного ядра системы на кристалле с x86-совместимым микропроцессором

М.И. Дябин[1], А.В. Решетников[2], Е.А. Саксонов[3]

[1]ООО «Каскад», г. Москва, dyabin@mail.ru

[2]ООО «Аккорд», г. Москва, a_reshetnikov@hush.com

[3]Московский технический университет связи и информатики, г. Москва, saksmiem@mail.ru

Аннотация — **Описана практическая методика тестирования системы команд микропроцессора Intel 80186. Рассмотрены принципы генерации тестов для логических и арифметических инструкций микропроцессоров. Предложен алгоритм работы начального загрузчика микроконтроллера с Intel 80186-совместимым процессором.**

Ключевые слова — **тестирование, система команд, микропроцессорное ядро, Intel 80186.**

I. Введение

При разработке микропроцессорных систем одной из важнейших задач является тестирование логики работы микропроцессорных ядер. В настоящее время данная тематика проработана весьма глубоко и развивается очень активно ввиду непрерывного нарастания сложности микропроцессорных модулей и непрерывного увеличения сферы использования встраиваемых систем. Среди многочисленных публикаций, посвящённых функциональному тестированию микросхем, многие работы так или иначе затрагивают проблему тестирования логики работы микропроцессоров. Теоретические основы тестирования микросхем подробно рассмотрены в статье [1]. Общая методика тестирования микропроцессоров приведена в работе [2]. Отметим также работу [3], в которой обсуждается проблема формальной верификации микропроцессора ARM. Имеются и другие публикации.

В данной статье мы опишем методику тестирования системы команд микропроцессора, ориентированную на быстрое написание тестов. Рассматриваемая методика была опробована на практике в процессе разработки системы на кристалле «Каскад-1». Программное обеспечение, написанное в ходе тестирования данной системы, позволило не только выявить имевшиеся на тот момент ошибки микропроцессора на самых ранних этапах его разработки, но и отладить постоянно добавляемые в процессорное ядро улучшения, проверяя логику его работы по мере внесения в него различного рода усовершенствований.

Основное назначение СНК «Каскад-1» – обеспечить возможность передачи данных по высокоскоростному радиоканалу через входящий в состав системы OFDM-модем [4]. В качестве сетевого процессора данная система на кристалле использует отечественный микропроцессор собственной разработки с архитектурой ядра x86. В процессе разработки системы по мере того, как ужесточались требования к радиоканалу, который данная система обязана была обеспечить, её процессор подвергался непрерывной модернизации: повышалась тактовая частота микропроцессора, совершенствовался его конвейер команд. Было необходимо постоянно тестировать новые версии микропроцессора. Возникла задача разработки тестов настолько быстрых, чтобы их можно было бы запускать на этапе синтеза микросхемы в компьютерной среде, моделирующей логику её работы. Так были разработаны тесты микропроцессора, которые авторы назвали *сигнатурными*. Поскольку моделирование микросхемы – процесс исключительно медленный, в основном тестирование процессора проводилось на FPGA с помощью тестов, которые были названы *исследовательскими*.

II. Общие принципы тестирования микропроцессоров, входящих в состав систем на кристалле

Разрабатывая тестовые программы для СНК «Каскад-1», авторы исходили из следующего принципа: прежде, чем приступать к написанию каких-либо тестов, необходимо проанализировать тестируемую систему и получить некоторое представление о том, какие уязвимости угрожают тестируемой системе, и каким образом разрабатываемые тесты могут её от выявленных уязвимостей защитить. Поясним данный принцип несколько подробнее. Какие у тестируемой системы слабые стороны? Какого рода ошибки в принципе могут в ней возникать? Какие из ошибок наиболее вероятны? Какие ошибки приведут к наиболее тяжёлым последствиям? Мы полагаем, что эти и другие подобные вопросы должны быть рассмотрены до того, как будет начато написание тестирующих программ для тестируемой системы. Кроме того, мы утверждаем, что *при написании тестов необходимо регулярно возвращаться к перечисленным выше вопросам* по мере того, как в тестируемую систему вносятся те или иные изменения. Тесты для системы должны разрабатываться параллельно с её развитием. В первую очередь необходимо тестировать то, что наиболее уязвимо в тестируемой системе в данный момент времени.

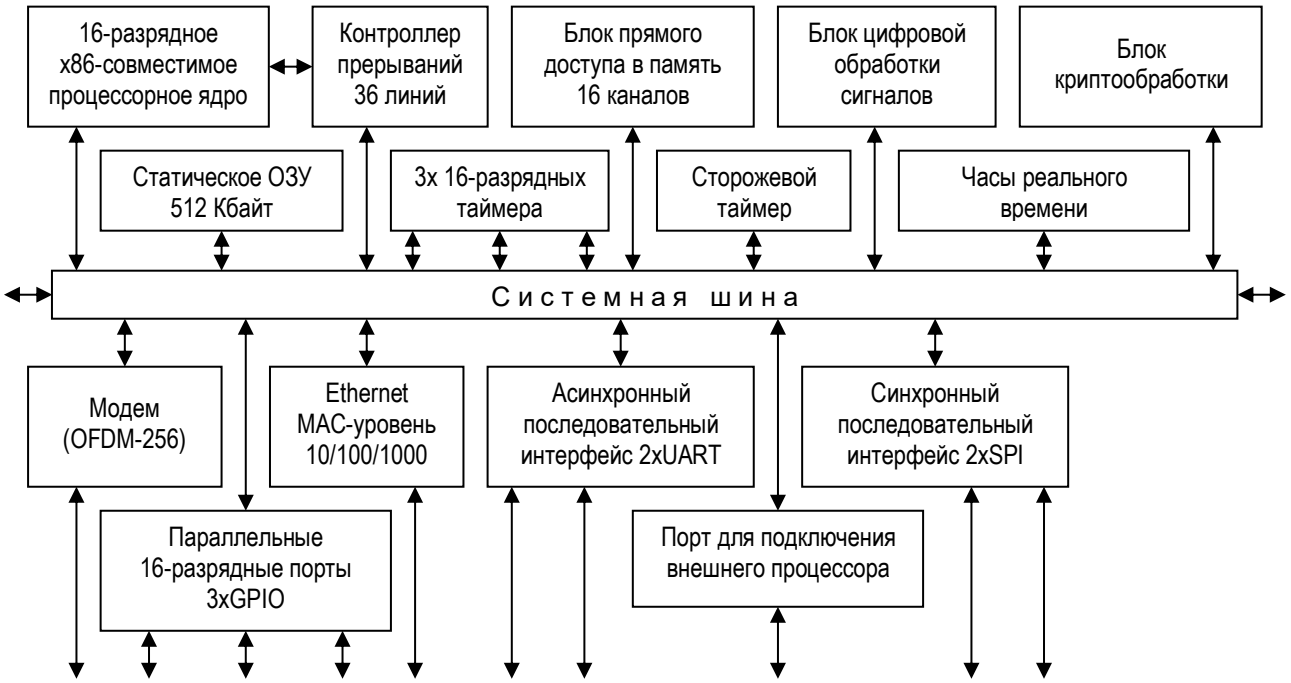

Рис. 1. Структурная схема СНК «Каскад-1»

Рассмотрим с этой точки зрения систему на кристалле «Каскад-1» (Рис. 1). В тех проектах, где данная система реально используется, от процессора в первую очередь требуется, чтобы он корректно взаимодействовал с OFDM-модемом, являющимся составной частью СНК. При этом программный код, реально выполняемый данным микропроцессором, занимается в основном *трансляцией* данных; почти вся *обработка* переданной информации производится на других процессорных модулях. Данные между процессором и модемом в системе «Каскад-1» передаются в основном по каналам DMA. По тем же каналам параллельно могут передаваться данные от других периферийных устройств. Основная функция соответствующего программного обеспечения – правильное координирование работы всех периферийных устройств системы. Такие программы активнее всего используют весьма ограниченное число инструкций микропроцессора; в то же время при взаимодействии с устройствами в этих программах могут возникать самые разнообразные и неожиданные ситуации, которые все должны быть обработаны правильно без сбоев.

Следовательно, при функциональном тестировании процессорного ядра системы «Каскад-1» основное внимание следует уделять не отдельным инструкциям микропроцессора, что было бы ожидаемо в случае тестирования микропроцессора широкого назначения, а работе микропроцессора в целом в условиях интенсивного обмена данными между различными устройствами, входящими в состав СНК. Тесты микропроцессора СНК «Каскад-1» должны покрывать как можно больше различных ситуаций, которые могут возникать в реальных, встречающихся на практике программах, взаимодействующих одновременно с несколькими периферийными устройствами системы.

В тестовых программах, действительно используемых для тестирования рассматриваемой системы, предусмотрена возможность при запуске тестов параллельно активировать различные таймеры, входящие в состав СНК, по каналам DMA непрерывно выполнять пересылку данных между различными устройствами, изменять тактовую частоту микропроцессора и т. д. Таким образом, при использовании данного программного обеспечения одна и та же тестовая программа – к примеру, программа тестирования инструкций условного перехода – может быть запущена параллельно с несколькими фоновыми процессами, в зависимости от конкретной конфигурации программы. Тестовые программы конфигурируются на этапе их компиляции путём установки значений тех или иных переменных компилятора. Перебор различных конфигураций тестовых программ мог бы осуществлять специальный скрипт, в задачи которого входили бы также сборка программы, её загрузка в FPGA, запуск и проверка результатов работы[1].

Ясно, что тесты такого рода должны отрабатывать не одну, а несколько итераций, поскольку ошибки, связанные с обменом данными между устройствами, могут проявляться не сразу – ввиду очень специфических условий их возникновения.

После того, как в микропроцессорном ядре обнаружена ошибка, она должна быть *локализована*, то есть код программы, обнаруживающей ошибку, следует упростить таким образом, чтобы в коде можно было точно указать то место, где команды микропроцессора

[1] В действительности для тестирования СНК «Каскад-1» написание подобного скрипта не понадобилось.

заведомо исполняются некорректно. Локализованная тестовая программа в итоге должна получиться достаточно ясной: должно быть полностью понятно, что ошибка, которую она обнаруживает, возникает именно в тестируемом микропроцессоре, а не заложена в саму тестирующую программу. Но как достичь такой ясности кода? Обсуждение данного вопроса выходит за рамки настоящей статьи. Отметим лишь, что если код тестирующей программы переусложнить, при его использовании локализация ошибок микропроцессора может оказаться весьма нетривиальной задачей.

Строго говоря, локализовать ошибку микропроцессора – значит упростить тестирующую его программу настолько, чтобы эту программу можно было запустить в компьютерной среде, моделирующей логику работы микросхемы с тестируемым процессором, и после моделирования на временной диаграмме было бы чётко видно неправильное поведение микропроцессора в какой-либо момент времени.

Упрощая код тестирующей программы с целью локализации ошибки, очень легко внести в него новую чисто программную ошибку, поэтому *тестовые программы следует писать таким образом, чтобы процесс локализации ошибок, в случае их обнаружения, был бы по возможности не слишком сложным, а риск внесения программных ошибок в код, в результате его упрощения, был бы по возможности невелик.*

III. Особенности тестирования системы команд x86-совместимого микропроцессора

Теперь, после того как, мы рассмотрели общие принципы тестирования микропроцессорных ядер, рассмотрим более подробно основные принципы тестирования системы команд. Безусловно, при тестировании процессорного ядра проверка правильности работы отдельных инструкций микропроцессора представляет исключительную важность. Хорошие тесты должны охватывать все инструкции микропроцессора и все режимы адресации памяти. Для каждой инструкции, помимо входных данных из условного «чаще всего встречающегося на практике» диапазона, тесты должны охватывать все возможные корректные «крайние случаи», которые пусть редко, но всё же могут встретиться в аргументах команд. Если команда микропроцессора модифицирует его флаги, то для каждого флага, на который оказывает воздействие тестируемая инструкция, необходимо убедиться в том, что она действительно изменяет состояние флага правильным образом. Наконец, если команда анализирует данные, расположенные в оперативной памяти (как, например, команда «Leave», введённая в архитектуру x86, начиная с процессоров Intel 80186 и Intel 80188), то для этих данных также необходимо рассмотреть как можно больше принципиально различных вариантов.

Основная сложность, с которой приходится сталкиваться в процессе тестирования системы команд любого микропроцессора, состоит в том, что порой непонятно, какие наборы входных данных следует считать принципиально различными настолько, что оба этих набора желательно включить в разрабатываемый тест, а какие наборы данных следует, наоборот, считать похожими друг на друга, так что корректное выполнение инструкции на одном из этих наборов скорее всего означает корректное исполнение той же инструкции и на другом похожем на него наборе.

Полный перебор всех вариантов входных данных при написании тестов системы команд осуществить практически невозможно. Но зачастую полный перебор и не требуется. Например, пусть поставлена задача протестировать команду пересылки данных из одного регистра микропроцессора в другой; в архитектуре x86 такое действие выполняет инструкция «Mov». Зададимся вопросом: какие наборы входных данных целесообразно включить в тест, чтобы в случае его успешного прохождения на каком-либо устройстве можно было бы констатировать, что на данном конкретном экземпляре протестированного устройства пересылка из регистра в регистр, скорее всего, выполняется правильно?

Напомним, что мы *не ставим* перед собой задачу проведения формальной верификации процессорного ядра. Наоборот, в рассматриваемом вопросе требуется написать такую программу, чтобы с её помощью можно было бы, во-первых, выполнять отладку схемы микропроцессора на этапе её разработки, а во-вторых, проверять функциональность отдельных экземпляров тестируемого микропроцессора после изготовления конечных устройств.

Какие циклы в требуемой программе имеет смысл делать более подробными? Какие циклы, наоборот, можно укоротить, считая маловероятными ошибки на прочих значениях, не вошедших в цикл? При разработке функциональных тестов постоянно приходится сталкиваться с подобными вопросами. В случае тестирования системы команд необходимо постоянно осуществлять выбор между различными параметрами, от которых зависит результат выполнения тестируемой инструкции, наделяя те или иные параметры команды большей значимостью, чем остальные её параметры; каким-то образом приходится отказываться от одних наборов входных данных в пользу других наборов, жертвуя тем, что представляется менее существенным, в пользу того, что следовало бы протестировать более тщательно. В предыдущем разделе были даны некоторые общие рекомендации о том, как, по мнению авторов, следует искать решения проблем такого рода. Сейчас хотелось бы перейти к более детальному изложению основных положений, которыми авторы руководствовались при разработке тестов для СНК «Каскад-1».

Итак, пусть требуется разработать тест для инструкции «Mov» x86-совместимого микропроцессора в случае, когда пересылка данных осуществляется из регистра в регистр. На самом деле задачу о составлении такого теста можно разбить на несколько подзадач.

Для начала необходимо убедиться, что данные записываются именно в тот регистр, в который требуется произвести запись значений в соответствии с заданными аргументами команды «Mov». *Можно считать, что код, осуществляющий соответствующую проверку, на самом деле тестирует не инструкцию как таковую, а общий*

механизм адресации памяти. Проверка адресации – это отдельная задача; мы ещё вернёмся к ней в следующем разделе. Сейчас лишь отметим, что в зависимости от разных факторов (среди которых, например, допустимое оценочное время работы тестового программного обеспечения, допустимый размер тестовой программы, допустимый объём занимаемой ею оперативной памяти и т. д.) может быть принято решение либо проверять адресацию памяти в широком классе инструкций микропроцессора, включающем в себя и инструкцию «Mov», либо можно провести разработку отдельного теста, целенаправленно занимающегося проверкой механизма адресации, в котором задействовано лишь ограниченное количество инструкций микропроцессора. Во втором случае[2] полагается, что если такой тест успешно завершает работу на некотором экземпляре тестируемого устройства, то, скорее всего, механизм адресации памяти на протестированном устройстве корректно работает во всех командах микропроцессора, включая те, которые не были задействованы непосредственно во время тестирования данного механизма.

Далее, необходимо убедиться, что в случае пересылки данных из регистра-источника в регистр-приёмник всякий раз, когда процессор завершает выполнение соответствующей команды, в регистре-приёмнике оказывается именно то значение, которое находилось в регистре-источнике в момент начала выполнения команды пересылки данных. Здравый смысл подсказывает, что если пересылка какого-то одного значения из некоторого регистра-источника в некоторый регистр-приёмник была успешно проведена, то другие значения также будут корректно пересылаться из того же регистра-источника в тот же регистр-приёмник. Поэтому при проверке содержимого регистра-приёмника после передачи данных из регистра-источника важно убедиться лишь в том, что пересылка данных действительно была осуществлена, то есть не должно в тестирующей программе возникать такой ситуации, что в регистре-приёмнике уже до начала пересылки данных находилось то же самое значение, что и в регистре-источнике.

На самом деле, проверять необходимо *каждый бит*, осуществляется ли его пересылка из регистра в регистр корректно или нет: дело в том, что при изготовлении экземпляров устройств могут возникать ошибки, проявляющиеся в неправильной установке отдельных битов переменных, используемых при выполнении тех или иных инструкций микропроцессора. Если бы тестировалась не аппаратная реализация процессора, а его реализации в виде, например, эмулятора, то ясно, что в таком случае проверка каждого бита была бы, скорее всего, излишней. Но поскольку мы ставим задачу тестирования аппаратной реализации, то для составления достаточно качественного теста нам необходимо покрыть наборами входных данных все значения каждого бита, от которого может зависеть результат выполнения тестируемой инструкции.

Каким образом проще всего решить обозначенную проблему? Общее правило, которым авторы пользовались при тестировании системы «Каскад-1», таково: *какой бы ни была инструкция микропроцессора, для каждого из её параметров, независимо от значений других параметров данной инструкции, в её тест (если только это в принципе возможно) должны быть включены все значения 0x00, 0x55, 0xAA и 0xFF – в случае, если параметр имеет размер 1 байт, или все значения 0x0000, 0x5555, 0xAAAA и 0xFFFF – в случае, если размер данного параметра равен 2 байтам*. При этом, говоря о параметрах команды, мы употребляем термин «параметр» в самом широком смысле: мы имеем ввиду не только аргументы, указанные в самой команде, но и вообще значения любых переменных, способных теоретически повлиять на результат выполнения команды: ими могут быть флаги процессора, неявно используемые ячейки памяти, смещение команды в памяти и т. д.

Конечно, при тестировании таких команд, как «Leave», строгое следование указанному правилу невозможно. Есть и другие случаи, в которых количество всех параметров, влияющих на результат, столь велико, что даже ограничив перебор предложенным правилом, мы получим чрезмерно большой объём вычислений. В таких случаях авторы предлагают не отказываться от правила целиком, а ориентироваться на него и придерживаться его в разумных пределах: отдельные параметры это правило всё ещё позволяет перебирать предложенным образом.

При тестировании арифметических команд, конечно же, необходимо включать в тест дополнительные наборы входных данных. Авторы пользовались следующим правилом: *независимо от того, со знаком или без знака производятся вычисления, для каждого параметра арифметической команды в её тест должны быть включены либо значения 0x01, 0x7F, 0x80, 0x81 и 0xFF – в случае, если размер рассматриваемого параметра равен 1 байту, либо значения 0x0001, 0x7FFF, 0x8000, 0x8001 и 0xFFFF – в случае, когда размер данного параметра равен 2 байтам*. Перечисленные значения являются «крайними» значениями определённых ключевых диапазонов, поэтому их имеет смысл включать в тесты как можно большего количества инструкций. Но также необходимо помнить, что некоторые инструкции имеют и другие, специфические для них «крайние» значения; все они тоже должны быть включены в тест по мере возможности. Продолжив дальнейшие рассуждения в этом направлении, мы уйдём в общую хорошо известную теорию составления тестов и выйдем за рамки настоящей статьи.

Вернёмся к задаче составления теста для инструкции «Mov». Есть ещё один блок переменных, значения которых также необходимо проверить, прежде чем делать вывод о работоспособности пересылки данных из регистра в регистр. Речь идёт о флагах микропроцессора. Инструкция «Mov» должна оставлять все флаги процессора без изменений [5, приложение C]. По крайней мере, для каждого из так называемых флагов общего назначения (а

[2] В действительности именно этот подход был применён при разработке тестов для СНК «Каскад-1».

лучше – вообще для каждого флага микропроцессора) необходимо убедиться, что если до выполнения команды пересылки данных его состояние было установлено в 1, то после завершения выполнения команды данный флаг сохраняет своё состояние, и если до выполнения пересылки состояние данного флага было установлено в 0, то также для данного значения флага необходимо убедиться, что пересылка данных это значение оставляет без изменений. Все эти случаи хотя бы по одному разу должны быть включены в тест инструкции «Mov».

Лишь после того, как все три типа проверок (адресация, значения, флаги) будут выполнены тестирующей программой на некотором экземпляре тестируемого микропроцессора, можно будет сделать заключение, что сама по себе инструкция пересылки данных на конкретном экземпляре протестированного устройства почти наверняка реализована без ошибок. Но такое заключение вовсе не означает, что операция пересылки какого-либо значения из регистра-источника в регистр-приёмник всегда будет успешно выполняться в программе: могут быть разные причины, по которым команда пересылки из регистра в регистр неправильно отрабатывается в какой-либо редко возникающей очень специфической ситуации; успешное прохождение описанного выше теста означает всего лишь, что в случае возникновения ошибки причину следует искать не в реализации команды пересылки данных, а в каких-либо других модулях микропроцессорного ядра.

Напоследок сделаем ещё одно важное замечание: *при написании тестов для системы команд крайне важно следить, чтобы результат работы каждого шага тестовой программы был однозначно определён* в соответствии с каким-либо документом, официально описывающим тестируемую архитектуру. Например, открывая приложение C руководства пользователя [5] микропроцессора Intel 80C186EC, мы замечаем, что инструкция «Test» после своего исполнения оказывает следующее воздействие на флаги микропроцессора:

– определённая часть флагов изменяется в соответствии с получившимся результатом;

– другая часть флагов данная инструкция оставляет без изменения;

– флаг «AF» после выполнения инструкции «Test» оказывается в неопределённом состоянии.

Следовательно, если программный код проверяет правильность работы инструкции «Test» на каком-либо микропроцессоре с x86-совместимой архитектурой, на этапе проверки состояния флагов тестовая программа может либо проверить каждый флаг по отдельности (что относительно долго и потому нецелесообразно), либо она может перевести флаг «AF» в некоторое определённое состояние, после чего – сразу проверить состояние всех флагов процессора одновременно. Последний вариант реализовать в тестовой программе очень легко: например, её код после выполнения инструкции «Test» может сохранить значение регистра флагов в стек и применить к сохранённому в стеке значению операцию логической конъюнкции, подобрав аргументы таким образом, чтобы, с одной стороны, все биты, значения которых не могут однозначно быть определены, обнулились после выполнения операции, а с другой стороны, значения всех битов, в которых сохранены состояния проверяемых флагов, остались бы после выполнения операции конъюнкции без изменений. Модифицированное указанным образом значение регистра флагов оказывается однозначно определённым, это значение тестовая программа уже непосредственно может проверить, правильное оно или нет.

Завершим на этом рассмотрение важнейших, по мнению авторов, принципов тестирования системы команд и перейдём к обсуждению вопросов, возникающих при реализации тестовых программ вообще, независимо от того, работают ли они в соответствии с рассмотренными принципами или же построены каким-либо иным образом.

IV. О некоторых особенностях тестовых программ, выполняемых на микроконтроллере

В предыдущем разделе мы подробно рассмотрели вопрос о том, *что* тестовая программа может сделать, для того чтобы проверить правильность работы отдельных инструкций микропроцессора. Рассмотрим теперь вопрос о том, *как* программный код может осуществить проверку такого рода и сообщить пользователю о результатах проверки.

Один из простейших и наиболее очевидных способов проверить значение той или иной переменной – сохранить это значение в фиксированной ячейке памяти, адрес которой известен пользователю, и предложить ему каким-либо способом прочесть содержимое данной ячейки и сравнить полученный результат с эталонным значением. Преимущество данного подхода понятно – это простота написания программного кода и, как следствие, повышенная надёжность тестовой программы. Однако, если впасть в заблуждение относительно этой простоты и положиться на неё, не принимая во внимание тот факт, что данный подход обладает целым рядом очень серьёзных недостатков, можно столкнуться с такими трудностями, что для их преодоления проще будет целиком отказаться от метода, чем продолжать искать решения в его рамках: дело в том, что *написанием одной только тестовой программы, запускаемой на микроконтроллере, работа по данному методу не ограничивается*. Разберёмся подробнее, какие проблемы могут возникнуть при использовании рассматриваемого метода, поскольку в том или ином виде аналогичные проблемы возникают и при использовании других методов.

Во-первых, пользователю почти наверняка необходимо будет проверить не одно, а несколько значений различных переменных. «Простейший» метод предлагает все эти значения записать в некоторый блок оперативной памяти и на том закончить работу программы. Но тогда, чтобы прочитать содержимое данного блока, у пользователя, прежде всего, должна быть возможность получения дампа оперативной памяти с микроконтроллера. Чтобы обеспечить пользователю такую возможность, необходимо проделать дополнительную работу: возможно, для этого придётся написать специальную программу, запускаемую на персональном компьютере; может быть,

придётся реализовать на микроконтроллере поддержку какого-либо хорошо известного протокола, такого как XMODEM – так или иначе, вопрос о передаче дампа памяти с микроконтроллера на персональный компьютер каким-либо образом необходимо будет разрешить.

Во-вторых, тестовая программа по завершении своей работы на микроконтроллере обязана предоставить пользователю возможность получить дамп. Это означает, что программа не может ни приостановить процессор (командой «Hlt», если речь идёт об x86-совместимом микропроцессоре), ни войти в вечный цикл – если только в ней не предусмотрены дополнительные механизмы обработки запросов от пользователя, такие как параллельно исполняемые процессы или предустановленные обработчики прерываний. Но реализация любых механизмов для установления интерфейса с пользователем – это дополнительный программный код, сводящий на нет единственное преимущество рассматриваемого метода, а именно – его простоту. В то же время, если программа после окончания процесса получения тестовых данных продолжает активную работу, она рискует испортить накопленные данные (ведь мы предполагаем, что программа ищет ошибки микропроцессора – следовательно, мы предполагаем, что такие ошибки в микроконтроллере есть; следовательно, продолжая активную работу, программа может вызвать какие-то из этих ошибок), и тогда у тестировщика вновь возникнет отнюдь не простая работа по локализации ошибок – мы говорили о такой работе в предыдущем разделе.

Метод записи проверяемых значений в оперативную память с последующим получением её дампа может оправдать себя при тестировании периферийных устройств; но при тестировании самого микропроцессора данный метод следует использовать с осторожностью. В любом случае, перечисленными проблемами список недостатков такого подхода не исчерпывается. Например, фиксируя адрес той или иной ячейки памяти, очень важно проработать механизм изменения данного адреса в случае необходимости – а это ещё одна трудность, которая тоже способна нивелировать первоначальную простоту метода. Или, например, неожиданными трудностями может обернуться разработка сопутствующего программного обеспечения для пользователя.

На самом деле, способ записи значений для проверки в оперативную память не самый простой. Ещё проще записывать проверяемые значения в регистр микропроцессора. Тогда, если тестовая программа запускается на микроконтроллере, в неё добавляется вызов несложного макроса каждый раз, когда требуется проверить содержимое выбранного регистра, и макрос распечатывает содержимое данного регистра в стандартный поток вывода; а если программа запускается в среде, моделирующей логику работы микросхемы, то не требуется даже вызывать макрос: такие среды обычно позволяют отслеживать изменения, происходящие в регистрах микропроцессора, и на основе этих изменений формировать короткую последовательность чисел, называемую *сигнатурой* теста. Запуская один и тот же тест всякий раз после того, как в архитектуру микропроцессора внесены те или иные изменения, можно сравнивать сигнатуру теста с его предыдущей сигнатурой – обычно она не меняется, если в процессор не были случайно внесены какие-либо аппаратные ошибки. Как уже говорилось ранее, соответствующие тесты, которые авторы называют *сигнатурными*, пишутся крайне просто, но исполняются крайне медленно. Такие тесты важны, поскольку позволяют видеть ошибки в микропроцессоре на этапе синтеза микросхемы. Но, если вспомнить требования к тестовым программам, которые мы поставили, начиная обсуждение принципов их работы, то становится ясно, что одних лишь сигнатурных тестов для функционального тестирования процессорного ядра явно недостаточно.

Сигнатурные тесты подсказывают идею, которой, по мнению авторов, следует руководствоваться при написании *исследовательских* тестов – то есть основных тестов, запускаемых на FPGA и проверяющих логику работы микропроцессорного ядра. Мы утверждаем, что *оптимальный способ проверки значений тех или иных переменных тестовой программы состоит в том, чтобы распечатывать тестируемые значения в стандартный поток вывода, затем – при запуске тестовой программы – записывать её стандартный вывод в файл, и наконец – после того, как тестовая программа закончит свою работу – побайтно сравнить содержимое полученного файла с содержимым заранее подготовленного эталонного файла*. Данный подход, конечно же (так же, как и все остальные подходы) не лишён недостатков: разумеется, тестовые программы, использующие стандартный вывод, работают медленнее, чем программы, подготавливающие блоки данных для дампа. Ясно также, что эталонный файл предварительно требуется каким-либо способом подготовить.

Однако, мы полагаем, что основные проблемы, с которыми приходится сталкиваться при разработке тестового программного обеспечения, проще всего решить именно путём максимально активного использования стандартного потока вывода данных. Например, для получения эталонных значений переменных имеет смысл адаптировать тестовую программу для запуска на эмуляторе микропроцессора; но программу, которая результат своей работы отправляет на распечатку в стандартный вывод, адаптировать под новую платформу гораздо проще, чем программу, которая выводит те же самые данные каким-либо другим способом.

Необязательно в стандартный поток вывода распечатывать сами переменные, требующие проверки: зачастую имеет смысл вычислить на микроконтроллере некоторую контрольную сумму проверяемых значений и предоставить пользователю лишь короткое значение вычисленной суммы. Например, пусть требуется протестировать механизм адресации памяти. Мы *не говорим* сейчас о том, как тестировать память: это отдельная задача; по данной теме имеется немало различной информации. Речь идёт о том, чтобы после того, как тот или иной тест памяти закончит работу, каким-либо образом проверить результат его работы и сообщить пользователю о результатах проверки. Нет необходимости в том, чтобы передавать пользователю целиком содержимое блока тестируемой области памяти: мы уже говорили о том, каким образом кажущаяся простота такого подхода может при реальном тестировании обернуться непростыми проблемами. Гораздо

лучше на микроконтроллере вычислить контрольную сумму проверяемой области и распечатать её в стандартный вывод. Пользователю останется лишь принять значение, распечатанное тестовой программой, и каким-либо образом выяснить, совпадает оно с правильным значением или нет (например – сравнить с эталоном).

Другой пример – запуск тестов, генерирующих псевдослучайные инструкции микропроцессора (в x86-совместимом микропроцессоре такие тесты могут быть написаны с использованием специального флага TF: передавая управление на псевдослучайно сгенерированную инструкцию, тестовая программа может установить значение флага TF в 1; тогда процессор, заканчивая выполнение очередной инструкции, сгенерирует специальное прерывание с номером 0x01, и управление будет передано обратно соответствующему обработчику прерывания). Псевдослучайные программы имеет смысл выполнять в отдельной области памяти. Такие программы важны на этапе функционального тестирования, поскольку могут обнаружить ошибки, неучтённые при разработке систематических тестов. Чтобы следить за тем, что происходит в области памяти псевдослучайной программы, тестовая программа может время от времени вычислять контрольную сумму данной области и отправлять в стандартный поток вывода вычисляемые значения[3].

Какой именно алгоритм подсчёта контрольной суммы использовать, зависит от того, что требуется протестировать и каким образом те или иные аппаратные ошибки могут повлиять на результат вычисления контрольной суммы. Играют роль и другие факторы: например, алгоритмы MD5 и SHA256, как известно, превосходно подходят для того, чтобы снабжать файлы контрольными суммами при передаче информации по сети. Но указанные алгоритмы работают относительно медленно; их использование может оказаться весьма нерациональным, например, для контроля работы псевдослучайных программ – в данном случае достаточно простого суммирования байтов, поскольку если ошибка возникнет в псевдослучайной программе, суммирование байтов рано или поздно выявит её почти наверняка. В то же время, простого суммирования байтов недостаточно, если программа тестирует, например, механизм адресации памяти. В общем случае для проведения контрольного суммирования в исследовательских тестах имеет смысл (на наш взгляд) применять *алгоритм Флетчера*. Более подробное обсуждение данного вопроса уже выходит за рамки настоящей статьи.

Обычно набор тестовых программ приходится запускать не один раз, а несколько: каждый раз в различных условиях (например, при работе микропроцессора на различных тактовых частотах). Тогда естественным образом возникает задача автоматизации тестирования. Если с микроконтроллера в том или ином виде поступают «сырые» данные (речь идёт не обязательно о дампе памяти: возможно, тестовая программа распечатывает значения для проверки в стандартный поток вывода), то на персональном компьютере принятый поток данных необходимо каким-либо образом обрабатывать. Вновь мы сталкиваемся с необходимостью писать дополнительный код. Так не проще ли обработку данных проводить прямо в тестовой программе, исполняемой на микроконтроллере? Если программа с микроконтроллера отправляет пользователю текстовые данные, снабжая их такими пояснениями, как «OK» при успешном прохождении определённой контрольной точки и «FAILED» в случае, если какое-то из контрольных значений не прошло проверку, обработка данных сводится к тому, чтобы убедиться, что стандартный поток вывода содержит требуемое значение строк вида «OK» (например – ровно одну такую строку) и не содержит ни одной строки вида «FAILED»... Авторы не берутся утверждать, что последний из описанных подходов является оптимальным, но при тестировании СНК «Каскад-1» целый ряд тестов был написан именно с использованием такого подхода.

Последнее замечание, которое хотелось бы сделать, состоит в том, что тестовую программу для запуска на микроконтроллере совсем не обязательно разрабатывать для запуска из его ПЗУ. Гораздо эффективнее прошить в ПЗУ некоторый код, выполняющий минимальную инициализацию с последующим входом в бесконечный цикл ожидания пользовательских команд. В системе на кристалле «Каскад-1» *начальный загрузчик* – как называют данную программу её авторы – после минимальной инициализации оборудования проверяет наличие программы во встроенной в систему Flash-памяти, затем либо загружает обнаруженную программу в память и передаёт ей управление, либо, если Flash-память отсутствует или не содержит программу для запуска, инициализирует последовательный порт и отправляет в него символ «*» – то есть, звёздочку. *Звёздочка сигнализирует о готовности микроконтроллера к выполнению пользовательских команд, что очень важно при тестировании конечных устройств*: получив звёздочку от устройства, пользователь видит, что данный экземпляр системы в принципе может выполнять некоторый программный код. После чего, с помощью особого набора команд, пользователь может загрузить тестовую программу в оперативную память микроконтроллера и потребовать от начального загрузчика передать управление по соответствующему адресу. Основное требование к начальному загрузчику состоит в том, чтобы его код был как можно более простым

[3] При разработке тестов с использованием псевдослучайных инструкций очень важно следить, чтобы каждая сгенерированная инструкция была корректной с точки зрения тестируемой архитектуры. Различные процессоры, полностью совместимые с одной и той же архитектурой, могут совершенно по-разному реагировать на одни и те же некорректные процессорные команды. В случае x86-совместимого микропроцессора, если в программе встречается несуществующая команда, конечно же процессор обязан сгенерировать специальное прерывание 6. Но если правильная команда снабжена префиксом, не предусмотренным для использования с данной командой – поведение процессора при обработке такой команды оказывается неопределённым. Различные примеры такого рода рассмотрены в статье Криса Касперски «Тонкости дизассемблирования. Дизассемблирование в уме».

и надёжным: он должен позволять проводить отладку системы даже в том случае, если её микропроцессор содержит какие-либо серьёзные аппаратные ошибки.

V. Заключение

Подведём итог. В зависимости от того, насколько надёжной должна оказаться итоговая система, для разных микросхем предъявляются разные требования к тестирующему их программному обеспечению. В тех случаях, когда процессор планируется использовать в системах, сбой в которых может привести к критическим последствиям, необходимо использовать глубоко проработанные методы тестирования, имеющие прочное теоретическое обоснование. Если же о встраиваемой системе известно, что сбой в работе её процессора не приведёт к крупным последствиям, зачастую оказывается достаточным убедиться, что процессор корректно выполняет программный код, реально возникающий при его практическом использовании; в этом случае такие процедуры, как формальная верификация логики работы процессорного ядра, оказываются не вполне оправданными. В данной работе мы рассмотрели общие принципы, которыми, по мнению авторов, следует руководствоваться при написании тестов микропроцессора именно для таких систем.

Литература

[1] Чегис И.А. Яблонский С.В. Логические способы контроля работы электрических схем // Тр. МИАН СССР. 1958. Т. 51. С. 270-360.

[2] Бобков С.Г. Методика тестирования микросхем для компьютеров серии «Багет» // Программные продукты и системы. 2007. № 3. С. 2-5.

[3] Patankar V.A., Jain A., Bryant R.E. Formal verification of an ARM processor // Proc. Twelfth International Conference on VLSI Design. Goa, India, 1999. P. 282-287.

[4] Архипкин А.В. Аппаратура передачи данных для комплекса с БЛА легкого класса // Тр. II научно-практической конференции «Перспективы развития и применения комплексов с беспилотными летательными аппаратами». Конгресс-центр парка «Патриот», Московская область, 14 апреля 2017. С. 29–34.

[5] Intel Corporation. 80C186EC/80C188EC microprocessor user's manual. Intel Corp., 1995. 515 p.

Methodology for Testing Microprocessor Core of System on Chip with x86-Compatible Microprocessor

M.I. Dyabin[1], A.V. Reshetnikov[2], E.A. Saksonov

[1]"Kaskad" Ltd., Moscow, dyabin@mail.ru

[2]"Accord" Ltd., Moscow, a_reshetnikov@hush.com

[3]Moscow Technical University of Communications and Informatics, Moscow, saksmiem@mail.ru

***Abstract* — This paper discusses the problem of functional testing of embedded microprocessors. A practical methodology for testing the instruction set of Intel 80186 microprocessor is offered. The main principles, adopted by the authors for development of testing software for the system-on-a-chip called "Kaskad-1", are given. Some difficulties facing the process of testing embedded microprocessor cores are noted.**

While testing the embedded system, the requirements for its testing software depend on the requirements for the embedded system's reliability. In the case when the microprocessor is intended to use as a part of a mission-critical system, i.e. such system that failure may result in serious consequences, it is necessary to use well elaborated methods for testing, based on deep theoretical justifications. But it is often case when an embedded system's processor failure does not result in any dangerous consequences. For such systems the difficult procedures like formal verification of their processor core are unnecessary.

The methodology presented in this paper for testing microprocessors is focused on writing tests quickly, concurrently with the development of the hardware being tested. The core question of testing processor instruction sets is which input data sets should be included in tests? Consider two different input data sets for testing the same processor instruction. When should they be regarded as so similar as successful completion of a test for one of these data sets will most probably mean the same instruction works correctly with the other input data set? In this paper we show how it is possible to find answers for such questions by considering a specific example.

***Keywords* — testing, instruction set, microprocessor core, Intel 80186.**

References

[1] Chegis I.A. Yablonsky S.V. Logicheskie sposoby kontrolya raboty electricheskih skhem (Logical methods of control of work of electrical schemes) // Trudy MIAN SSSR. 1958. Tom 51. S. 270-360 (in Russian).

[2] Bobkov S.G. Metodika testirovaniya mikroskhem dlya kompyuterov serii "Baget" (A methodology for testing circuits for the "Baget" series of computers) // Programmnyye producty I sistemy. 2007. № 3. S. 2-5 (in Russian).

[3] Patankar V.A., Jain A., Bryant R.E. Formal verification of an ARM processor // Proc. Twelfth International Conference on VLSI Design. Goa, India, 1999. P. 282-287.

[4] Arkhipkin A.V. Apparatura peredachi dannykh dlya kompleksa s BLA legkogo klassa (Data communications equipment for a complex with a light class UAV) // Trudy. II nauchno-prakticheskoy konferentsii "Perspektivy raz-vitiya I primeneniya kompleksov s bespilotnymi letatelnymi apparatami". Kongress-tsentr parka "Patriot", Moskovskaya oblast, 14 aprelya 2017. P. 29-34 (in Russian).

[5] Intel Corporation. 80C186EC/80C188EC microprocessor user's manual. Intel Corp., 1995. 515 p.