DOI: 10.15514/ISPRAS-2021-33(6)-9

Обзор методов функционального онлайн-тестирования микропроцессоров

[1]Н.Д. Чертоκ, ORCID: 0000-0002-7286-7098 <chertoknd@ispras.ru>
[1,2]М.М. Чупилко, ORCID: 0000-0002-8772-5631 <chupilko@ispras.ru>

[1] Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] Российский экономический университет им. Г.В. Плеханова,
117997, Россия, г. Москва, Стремянный пер., д 36

Аннотация. Функциональным онлайн-тестированием называется верификация опытных образцов микропроцессоров или их ПЛИС-прототипов, т.е. пост-производственная верификация (post-silicon verification). Такой вид тестирования отличается как от производственного тестирования, нацеленного на проверку работоспособности произведенных микросхем (отсутствие дефектов производства, допустимость значений физических характеристик), так и от функциональной верификации моделей микропроцессоров, проводимой в симуляторе (где можно наблюдать за внутренними сигналами микропроцессора и контролировать процесс исполнения). Пост-производственная верификация позволяет на высокой скорости испытывать огромные массивы тестов и обнаруживать ошибки, пропущенные при функциональной верификации на до-производственном этапе. Тесты для микропроцессоров обычно имеют вид программ; соответственно, основными задачами онлайн-тестирования микропроцессоров являются высокопроизводительная генерация тестовых программ в заданной системе команд и создание тестового окружения, отвечающего за запуск программ, оценку корректности их исполнения микропроцессором, диагностику ошибок и взаимодействие с внешним миром. В данной статье рассматриваются проблемы, возникающие при разработке систем онлайн-тестирования (онлайн-генераторов тестовых программ), делается обзор существующих решений в этой области и на их основе предлагается перспективный подход к организации онлайн-тестирования.

Ключевые слова: микропроцессоры; онлайн-тестирование; функциональное тестирование; пост-производственная верификация; валидация; генерация тестовых программ

Для цитирования: Черток Н.Д., Чупилко М.М. Обзор методов онлайн-генерации тестовых программ для функциональной верификации микропроцессоров. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 131-148. DOI: 10.15514/ISPRAS–2021–33(6)–9

Survey of Methods for Functional Online Testing of Microprocessors

[1]N.D. Chertok, ORCID: 0000-0002-7286-7098 <chertoknd@ispras.ru>
[1,2]M.M. Chupilko, ORCID: 0000-0002-8772-5631 <chupilko@ispras.ru>

[1] Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] Plekhanov Russian University of Economics,
36, Stremyanny lane, Moscow, 117997, Russia

Abstract. Online testing is a process of functional verification of microprocessors produced in silicon or their FPGA-prototypes, i.e. post-silicon verification. This type of testing differs both from the manufacturing testing,

aimed at checking the workability of manufactured chips (e.g., absence of physical defects, admissibility of physical characteristics) and from simulation-based pre-silicon functional verification of microprocessors models (where internal microprocessor signals are available for observing, and the execution process can be controlled). Post-silicon verification enables to rapidly run huge numbers of tests and detect bugs missed during pre-silicon functional verification. Tests for microprocessors are usually represented by executable programs. Accordingly, the main tasks of online testing are high-performance generation of test programs in the given ISA and creation of a test environment responsible for launching programs, assessing the correctness of their execution by a microprocessor, diagnosing errors, and interacting with the outside world. This paper examines the problems arising in the development of online testing systems (online test program generators), reviews existing solutions in this area, and, on the base on them, proposes a promising approach to organizing online testing.

Keywords: microprocessors; online testing; functional testing; post-silicon verification; validation; test program generation

For citation: Chertok N.D., Chupilko M.M. Survey of Methods for Functional Online Testing of Microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 6, 2021, pp. 131-148 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–9

1. Введение

Долгое время возможности функциональной верификации микропроцессоров, предоставляемые с одной стороны генераторами тестовых программ, а с другой стороны симуляторами RTL-моделей аппаратуры, реализованных на HDL-языках (Register Transfer Level – уровень регистровых передач и Hardware Description Languages – языки описания аппаратуры), были достаточны для проведения функциональной верификации за приемлемое время (до 60% от общего времени разработки [1]). Скорость создания простых тестовых воздействий генераторами тестовых программ можно оценить в десятки-сотни тысяч инструкций в секунду. Если же речь идет о генерации нацеленных (directed) тестов, например, для подсистемы памяти, когда используются сложные ограничения на поток управления и поток данных, скорость создания тестовых воздействий падает до сотен инструкций в секунду. Скорость исполнения тестов на RTL-модели микропроцессора, запущенной в симуляторе, составляет также сотни-тысячи инструкций в секунду. При этом тактовая частота изготовленных микропроцессоров позволяет им работать на несколько порядков быстрее.

Столь существенная разница между скоростью тестирования и реальной производительностью микропроцессоров приводит к необходимости развития средств тестирования, использующих опытные образцы микропроцессоров и их прототипы как для исполнения, так и для генерации тестовых программ. Такого рода средства называются инструментами *онлайн-тестирования* (*online testing*), *пост-производственной валидации* (*post-silicon validation*) [2], или *пост-производственной верификации* (*post-silicon verification*). Процесс проведения онлайн-тестирования обычно осуществляется путем многократного запуска множества разнородных тестов, покрывающих различные функциональные и архитектурные особенности устройства. Поскольку основной целью данного тестирования является поиск функциональных ошибок, допущенных при проектировании, то исполняемые тесты могут быть заимствованы с этапа до-производственного тестирования. Кроме того, тесты можно получить путем автоматизированной генерации на основе *тестовых шаблонов* – параметризованных описаний последовательностей команд, допускающих получение однотипных тестов, обращающихся к различным ресурсам тестируемой системы.

Отметим, что применяющимся на том же этапе изготовления микропроцессоров, но преследующим другие цели, является *производственное тестирование* (*manufacturing*

testing), которое проводится для того, чтобы оценить качество изготовления конкретного микропроцессора и выявить физические ошибки, удовлетворяющие определенному шаблону (замыкание линии на логический 0 или 1, короткое замыкание линий между собой, отказы транзисторов и др.). В производственном тестировании обычной практикой является автоматизированное получение тестов из списка соединений (*netlist*), соответствующего синтезированной RTL-модели тестируемого микропроцессора.

Проведение пост-производственной верификации и разработка соответствующих средств ее проведения сопровождается трудностями в следующих аспектах:

- идентификация и диагностика некорректного поведения;
- оценка полноты тестирования (традиционные техники на основе исходного кода и моделей здесь неприменимы);
- ограничение на сложность инструментов онлайн-тестирования для возможности запуска на ранних этапах пост-производственной верификации;
- необходимость адаптации инструментов онлайн-тестирования под целевую архитектуру;
- применимость к сложным многоядерным и многопоточным микропроцессорам, где необходимо тестировать реализацию механизма поддержки консистентности памяти.

Так, в работе [3] среди проблем, возникающих в процессе онлайн-тестирования, отмечены:

- сложность воспроизведения ошибок (на работу микропроцессора влияет множество условий, включая физический фактор);
- получение эталонных значений (необходима разработка эталонной модели, время работы которой оказывается на порядки больше времени работы тестируемого устройства);
- оценка полноты покрытия (выбор эффективной метрики полноты покрытия является нетривиальной задачей);
- особенности разработки тестовых шаблонов (отсутствие наблюдаемости значений внутренних сигналов тестируемой аппаратуры накладывает дополнительные ограничения на структуру самих тестов и формат результата их выполнения).

Данная работа включает в себя обзор существующих методов проведения онлайн-тестирования и реализующих их средств, а затем на основе обзора предлагает технологию проведения пост-производственного тестирования, которая бы удовлетворяла требованиям архитектурной независимости, скорости работы и позволяла осуществлять проверку корректности наблюдаемого поведения, диагностику ошибок и оценку полноты тестирования.

Далее статья организована следующим образом. Второй раздел посвящен подходам, используемым в различных областях онлайн-тестирования. В третьем разделе рассматриваются наиболее развитые на данный момент инструменты онлайн-тестирования микропроцессоров и проводится сравнительный анализ этих инструментов. В четвертом разделе предлагается архитектурное и технологическое решение для пост-производственной верификации. В заключении приводятся основные тезисы данной работы.

2. Проблемы онлайн-тестирования и методы их решения

В данном разделе приводится обзор основных методологических вопросов проведения пост-производственного тестирования, распределенных по следующим темам:

1) идентификация и диагностика некорректного поведения;
2) оценка полноты тестирования;

3) получение эталонных значений;
4) генерация тестовых данных;
5) применимость к многопоточным микропроцессорам.

2.1. Идентификация и диагностика некорректного поведения

Проблема идентификации и диагностики ошибок является ключевой для пост-производственного тестирования. По сравнению с до-производственной верификацией, осуществляемой с помощью симулятора, где можно в любой момент времени приостановить симуляцию для проверки выполнения заданных свойств, возможности онлайн-тестирования сильно ограничены. Рассмотрим две группы подходов, применяемых для решения данного класса проблем: *использование буферов трасс* (*trace buffers*) и *QED-тесты*.

Начнем рассмотрение проблемы поиска и диагностики ошибок с буферов трасс, как наиболее очевидного решения, изначально использовавшегося только для производственного тестирования. Метод буферов трасс подразумевает добавление в модель аппаратуры специальных накопителей информации, которые сохраняют изменения значений выбранных сигналов для последующего анализа. Использование буферов связано с несколькими направлениями исследований [4]:

- способы выбора сигналов моделей аппаратуры;
- уменьшение количества хранимых данных;
- методы передачи данных в анализирующее программное обеспечение.

Итак, поскольку использование буферов трасс сопряжено с необходимостью хранения и пересылки большого количества информации для ее последующего анализа, очень важным вопросом является уменьшение этого количества за счет выбора адекватного множества отслеживаемых сигналов, а также использования особого способа хранения такой информации. Выбор множества сигналов может быть осуществлен с помощью динамического или статического анализа исходной RTL-модели. Так в работе [5] для выбора сигналов используется симуляция RTL-модели устройства со случайно внедренными ошибками, которая подсказывает какие сигналы наиболее характерны для диагностики некорректного поведения, а в работе [6] предлагается автоматический анализ извлеченного из RTL-кода *графа потока управления и данных* (*Control Data Flow Graph*, *CDFG*).

В целях уменьшения количества хранимой информации в буферы вместо полных значений сигналов направляется лишь некоторая «*сигнатура*» (*footprint*) [4] – фрагментарная информация об изменениях, как это делается, например, в работе [7] в методе, названном *записью и анализом сигнатур инструкций* (Instruction Footprint Recording and Analysis, IFRA), который мы рассмотрим несколько подробнее. IFRA требует создания дополнительных элементов на синтезируемой схеме (порядка 1% занимаемой площади) для хранения собираемых сигнатур. Структура хранения включает в себя *идентификатор инструкции* (*Instruction ID*) – значение, устанавливающее принадлежность сигнатуры конкретной исполняемой инструкции и зависящее от состояния конвейера целевого микропроцессора: стадии выбора инструкций (fetch) или опустошения конвейера (flush) в результате неверно предсказанного перехода или исключения. Таким образом, для идентификации наблюдаемого поведения используются системные события, связанные с конвейером обработки инструкций. Обнаружение возникающих в процессе работы микропроцессора ошибок осуществляется за счет *пост-триггеров* (*post-triggers*) – дополнительно синтезированных элементов, определяющих возникновение ошибки по микроархитектурному состоянию микропроцессора. При срабатывании пост-триггера текущее состояние сигнатур инструкций сохраняется и используется для воспроизведения ситуации, приведшей к наблюдаемой ошибке, и созданию короткой трассы,

воспроизводящей ошибку. Таким образом, метод IFRA включает в себя подходы как к организации буферов, так и к анализу трасс.

Касаясь темы анализа трасс, рассмотрим еще две работы. Так, проведение анализа ошибочного поведения через воспроизведение ошибок описывается в работе [8], где каждой сохраняемой транзакции в последовательности присваивается метка времени, что позволяет реконструировать осуществляемое взаимодействие в тестируемой аппаратуре, приведшее к ошибке. Кроме того, анализ трасс можно проводить с использованием машинного обучения: в работе [9] используется метод кластеризации для поиска редко случающихся ситуаций в группе однородных трасс, полученных при запуске тестов. Трассы разбиваются на фрагменты, которые группируются в кластеры таким образом, что невозможность отнести новый фрагмент трассы к существующему кластеру сигнализирует о наличии в нем ошибок.

Итак, использование буферов трасс на данный момент является опробованным решением, которое может быть использовано для идентификации и локализации ошибочного поведения, но требует определенного искусства [4] выбора подмножества сигналов, которые будут использоваться при анализе поведения.

Альтернативным решением является метод под названием *локализация ошибок с использованием QED-тестов* (Quick Error Detection, «быстрое обнаружение ошибки»), получаемых в результате применения особого вида трансформаций к существующим тестовым программам с целью уменьшения задержки обнаружения ошибок. Так, можно выделить два родственных подхода: «быстрое обнаружение ошибок электроники» (Electrical Quick Error Detection, E-QED) [10] и «символическое быстрое обнаружение ошибок» (Symbolic Quick Error Detection, S-QED) [11]. Подход E-QED ориентирован на поиск ошибок в микросхемах и включает в себя следующие шаги.

1) Исходный HDL-код автоматически разбивается на блоки, размер которых удовлетворяет ограничениям для запуска в среде инструмента формального анализа. Для каждого блока определяются его интерфейсы. К каждому интерфейсу присоединяется E-QED-блок, записывающий изменения логических значений сигналов в сжатом виде в процессе работы устройства.
2) Запускается набор QED-тестов. При обнаружении ошибки выполнение теста останавливается и считывается содержимое блоков сигнатур.
3) Запускается *BMC-инструмент* (Bounded Model Checking, ограниченная проверка модели) для проверки соответствия сигнатур формальной модели блока. В случае несоответствия блок считается содержащим ошибку.
4) Для каждого содержащего ошибку блока производится серия последовательных запусков BMC-инструмента. При этом в триггеры в составе блока вносятся одиночные однотактовые ошибки («single-cycle FF error model») и проверяется их совпадение с записанной сигнатурой. В случае совпадения данные триггеры считаются содержащими ошибку.
5) Для каждого триггера, содержащего ошибку, проверяется соответствие возникновения ошибки сигнатурам, записанным в соседних блоках. Если соответствие не подтверждается, то триггер исключается из списка содержащих ошибку.

Подход S-QED ориентирован на поиск логических ошибок в работе устройства, а на структуру QED-тестов накладываются дополнительные ограничения, препятствующие генерации тривиальных контр-примеров, не обнаруживающих реальные ошибки. Основной мотивацией создания метода была сложность диагностики ошибок, возникающих при пост-производственной верификации, с помощью традиционных методов, таких как использование буферов трасс. Этот метод включает в себя следующие шаги.

1) В RTL-модель интегрируется модуль, трансформирующий входной поток тестовых инструкций путем применения набора QED-преобразований, обеспечивая выполнение дополнительных ограничений для BMC-инструмента. Интеграция данного модуля необходима только для запуска инструмента и не требуется при синтезе итоговой схемы микропроцессора.
2) Если размер тестируемой схемы превосходит вычислительные возможности BMC-инструмента, то применяется метод моделирования частичного набора компонентов микропроцессора.
3) Система приводится в QED-консистентное состояние – состояние регистров и памяти микропроцессора, при котором исполнение QED-тестов не приведет к генерации ложных контр-примеров BMC-инструментом.
4) Запускается BMC-инструмент для проверки корректности выполнения инструкций путем разрешения ограничений, наложенных на порядок изменения значений регистров, состояние памяти и состояний контрольных модулей. При нарушении какого-либо проверяемого свойства инструмент строит контр-пример, который содержит минимальное количество инструкций, вызывающих его нарушение.

Основанные на QED-тестах подходы видятся перспективными в силу формального характера результатов их работы, но при этом они ограничены возможностями используемого BMC-инструмента, поэтому они являются менее универсальными, чем буферы трасс.

2.2 Оценка полноты тестирования

Полнота тестирования – это метрика, которая позволяет оценить качество проверки RTL-модели микропроцессора, а также качество проверки аппаратной реализации этой модели.

В случае традиционного до-производственного тестирования существует несколько вариантов оценки полноты тестирования, среди которых можно выделить *структурные метрики*, такие как покрытие кода, ветвей, условий и т. д. Однако структурные метрики никак не выражают покрытие функциональных требований спецификации RTL-модели, поэтому для более тщательного тестирования необходимо использовать *поведенческие* (*функциональные*) *метрики*, оценивающие покрытие на основе:

- состояний и переходов конечных автоматов;
- последовательностей переключений уровней сигналов;
- комбинаций входных значений комбинаторных схем.

Структурные метрики не применимы к пост-производственному случаю [12]: при отображении RTL-модели в ее низкоуровневую реализацию изменяется объект тестирования: происходит переход от синтаксических конструкций RTL-модели к соединениям и элементам электронной схемы. А вот функциональные метрики остаются применимыми в силу их более высокой абстракции.

Среди используемых на практике функциональных метрик, наиболее часто встречается проверка *темпоральных* (*временных*) *утверждений*. Для использования этих утверждений в пост-производственном тестировании необходимо иметь способ построения компонентов, отвечающих за непосредственную оценку покрытия. Например, на этапе до-производственного тестирования можно использовать автоматизированное построение компонентов-сборщиков покрытия RTL-модели, их внедрение в RTL-модель, проведение пост-производственного тестирования и анализ полученных результатов на инструментальной ЭВМ. Однако отметим, что синтез и размещение компонента сборки и оценки покрытия вместе с проверяемой схемой может увеличить накладные расходы на занимаемую площадь, потребовать дополнительного питания и изменить временные характеристики схемы.

В работах [13-15] предлагаются различные усовершенствования данного базового метода:

1) в работе [13] для тестирования покрытия требований, накладываемых на протоколы взаимодействия RTL-моделей, предлагается использовать настроенный на тестируемые протоколы анализатор потоков данных, оформленный в виде стороннего IP-блока – необходимость синтезировать компонент-сборщик покрытия отсутствует;
2) в работе [14] предлагается использовать метрику, основанную на покрытии темпоральных утверждений SystemVerilog: при анализе RTL-модели происходит выбор множества задействованных в темпоральных утверждениях сигналов для буферов трасс, а затем информация об изменении значений этих сигналов в процессе онлайн-тестирования используется для реконструкции покрытия утверждений.
3) в работе [15] так же предлагается оценивать полноту тестирования по покрытию темпоральных утверждений, однако утверждения автоматически формируются в процессе синтеза RTL-модели, а факт их покрытия определяется на основании анализа трасс методами работы с большими данными (кластеризация).

Из описания приведенных работ видно, что помимо компонентов, создаваемых специально для анализа покрытия, можно воспользоваться также другими компонентами инфраструктуры пост-производственной верификации: например, буферами трасс и мониторами производительности. Эти идеи озвучены в работе [16] и в книге [17].

Таким образом, можно сделать вывод, что для получения информации о функциональном покрытии RTL-модели в зависимости от сложности этой модели используются:

- средства получения и доставки информации о покрытии (буферы трасс или специально создаваемые компоненты);
- набор темпоральных утверждений, получаемых на каком-либо этапе преобразования RTL-модели;
- инструментальная ЭВМ для анализа покрытия.

2.3 Получение эталонных значений

В связи со значительно более высокой скоростью исполнения тестов в процессе пост-производственного тестирования по сравнению с до-производственной верификацией возможно также значительно увеличить и количество тестовых последовательностей. Последнее теоретически позволяет исследовать поведение тестируемой системы в более сложных внутренних состояниях. Однако, если проведение тестирования означает сравнение получаемых значений с эталонными, узким местом становится производительность программных симуляторов, обычно используемых для получения требуемых эталонных значений. В качестве решения можно предложить следующие пути: использование симуляторов, осуществляющих моделирование на более абстрактном уровне, задействование *условно-эталонные значений,* отказ от эталонных значений путем обратных преобразований.

Начнем рассмотрение данных подходов с использования симуляторов, моделирующих поведение системы на более высоких уровнях абстракции (по сравнению с полносистемным моделированием RTL-модели системы). Так, например, в работе [18] TLM-модель тестируемой системы (Transaction-Level Modeling – моделирование на уровне транзакций) используется для высокоуровневой генерации тестов и для сравнения результатов тестирования.

При невозможности получения эталонных значений альтернативой является использование условно-эталонных значений – значений, корректность которых не подтверждена, но которые позволяют сигнализировать о наличии расхождения с текущими результатами тестирования. Одним из часто используемых способов получения условно-эталонных значений является многократный прогон одних и тех же тестов (или модифицированных

эквивалентным образом) и принятие одного из результатов таких прогонов за условно-эталонное значение. Так, например, в работе [19] в качестве условно-эталонных значений используются значения, полученные при первом прогоне тестов.

Также возможно построение тестов, не требующих наличия эталонных значений, с использованием инструкций «прямой» и «обратной» обработки данных, как в подходе Reversi [20]. Суть метода заключается в том, что в случае корректного поведения системы исходные данные для теста и результирующие, полученные в результате обратных преобразований, будут совпадать.

2.4 Генерация тестовых данных

При онлайн-тестировании микропроцессоров тесты представляют собой последовательности тестовых программ, состоящих из отдельных инструкций, нацеленных на повышение достигнутого уровня покрытия. Тестовые программы могут включать в себя дополнительные инструкции подготовки тестового воздействия (пролог) и инструкции непосредственной проверки результата обработки тела тестовой программы (эпилог). Наиболее простой и часто используемый способ генерации тестовых последовательностей – случайный выбор тестовых инструкций, что, однако, не дает быстрого достижения приемлемого уровня полноты тестирования. С целью нахождения ошибок, возникающих в труднодостижимых внутренних состояниях, необходимо каким-то образом нацеливать генератор на определенную функциональность тестируемого микропроцессора путем выбора специальных последовательностей инструкций с подходящими параметрами.

Сначала рассмотрим некоторые более простые в своей организации способы генерации тестовых последовательностей:

1) генерация псевдослучайных последовательностей на основе сдвиговых регистров (linear-feedback shift registers) по аналогии с производственным тестированием, как это описывается в работе [21];
2) использование в качестве тестов компилируемых С-программ [22];
3) нацеливание тестов на покрытие ограничений (constraints) SystemVerilog [23] исходной RTL-модели микропроцессора;
4) присваивание вероятности выбора очередной инструкции на основе некоторых эвристик [24].

Более сложными методами организации работы генератора являются: *исполнение эквивалентных программ* и «*лакмусовые тесты*».

В подходе исполнения эквивалентных программ (Equivalent Program Execution, EPEX) [25] предлагается строить альтернативную тестовую программу P’ для тестовой программы P с помощью SMT-решателей и сравнивать состояние тестируемого микропроцессора после подачи программы P и программы P’. Подход напоминает S-QED, рассмотренный в разделе 2.1, однако применим для более широкого класса тестовых программ.

С целью повышения уровня покрытия можно также предложить использовать тесты, нацеленные на подсистему памяти (многоядерных) микропроцессоров: для проверки ограничений консистентности памяти можно использовать подход генерации «лакмусовых тестов» (litmus tests) [26], состоящих из небольшого числа инструкций, создающих особые ситуации в работе памяти. Создание тестов в упомянутой работе осуществляется на основе аксиоматической модели памяти микропроцессора, которая работает существенно быстрее обычных операционных моделей памяти.

Подытоживая, можно сказать, что существует несколько различных методов генерации тестовых данных, часть из которых позволяет получить тесты быстро, но с достижением определенного уровня покрытия. Одновременно можно разрабатывать усложненные

генераторы с помощью нацеленных методов для повышения достигнутого покрытия до приемлемого уровня.

2.5 Применимость к многопоточным микропроцессорам

Поддержка многопоточных микропроцессоров в генераторах тестовых программ для онлайн-тестирования обычно включает загрузку теста на каждое ядро и запуск их таким образом, чтобы максимизировать обращения в общую для всех процессов/ядер память [19].

Приведем некоторые работы, касающиеся вопросов тестирования памяти в многоядерных системах. В статье [27] предлагается метод оценки качества проводимого тестирования общей кэш-памяти инструментом Dacota, который сохраняет трассу запросов в кэш-память от каждого ядра, а также имеет возможность проверки корректности порядка операций работы с памятью. В статье [28] предлагается метод проверки ограничений консистентности памяти многоядерных микропроцессоров. Для этого предлагается генерация случайных тестов на основе разрешения ограничений, которые инструментируются для возможности обнаружения результата тестирования: для каждого теста генерируется набор компактных сигнатур, отражающих уникальные паттерны упорядочивания инструкций запросов в память, которые могут быть обнаружены даже после многократного исполнения теста. Уникальность и наблюдаемость сигнатур позволяют качественно оценить покрытие тестируемой памяти различными вариантами доступа.

3. Существующие инструменты онлайн тестирования

Разработка генераторов для проведения онлайн-тестирования обычно выполняется в рамках процесса разработки микропроцессоров, в котором также имеется стадия традиционной верификации. Более того, позволить себе проводить такую верификацию могут лишь достаточно крупные разработчики микропроцессоров, остальные ограничиваются традиционными средствами. Поэтому неудивительным выглядит факт, что все существующие на данный момент разработки в области онлайн-генераторов тестовых программ носят коммерческий характер. Далее мы рассмотрим инструменты онлайн-генерации, разрабатываемые ведущими поставщиками микропроцессоров: Threadmill от компании IBM [19, 29], RIS от компании ARM [30] и RITG от компании Intel [31, 32].

3.1. Threadmill

Инструмент Threadmill разрабатывается в компании IBM и используется для верификации микропроцессоров с архитектурой POWER. Инструмент возник в рамках продолжения работы над инструментом до-производственного тестирования Genesys-Pro [33] и использует его информацию о тестируемой архитектуре. Он ориентирован на тестирование многопоточных многоядерных микропроцессоров. Для генерации тестов инструментом компонуется программа-генератор на основе библиотечных средств, которая затем компилируется под целевую архитектуру и загружается на тестируемый микропроцессор. В рамках данного подхода сконфигурированный двоичный образ принято называть *исполнителем тестов* (*exerciser image*), а механизм его создания – *построителем* (*builder*). Исполнитель тестов включает базовые сервисы операционной системы, тестовые шаблоны, информацию об особенностях микроархитектуры тестируемой системы, модель архитектуры и некоторые тестовые данные (например, набор тестовых данных для арифметики с плавающей точкой). Исполнитель обеспечивает выполнение в бесконечном режиме следующих действий:

1) генерация тестовых программ на основе шаблонов, соответствующих целям тестирования, и информации об архитектуре тестируемого микропроцессора;

2) исполнение тестовых программ;
3) проверка результатов.

Далее на примере Threadmill будут рассмотрены основные проблемы проведения онлайн-тестирования.

3.1.1 Методика оценки корректности и эталонные значения

Характерной особенностью метода построения исполнителя является отсутствие в нем эталонной модели тестируемого микропроцессора. Это сделано для ускорения работы исполнителя, так как моделирование деталей работы тестируемой системы снизило бы объем вычислительных ресурсов, доступных для непосредственного процесса генерации новых тестов и их запуска. Отчасти это также связано с возможной нестабильностью работы целевой системы, что может привести к некорректным вычислениям по эталонной модели. Исполнитель тестов (см. рис. 1) состоит из следующих компонентов: спецификации (а именно, шаблонов тестовых программ и некоторой микроархитектурной информации) и компонентов, обеспечивающих генерацию тестовых последовательностей, их выполнение и проверку результатов их работы.

Рис. 1. Архитектура исполнителя тестов Threadmill
Fig. 1. Architecture of Threadmill's Test Executor

Из-за отсутствия эталонной модели возникают следующие две трудности проведения тестирования: необходимость получения эталонных значений некоторым альтернативным стандартному подходу способом, а также организация вычисления адресов инструкций, следующих за инструкциями условного перехода. Для решения первой проблемы, а именно получения значений, используемых при проверке корректности результатов, в инструменте используется подход многократного исполнения тестовых программ, однако размещенных по разным адресам. Причем заметим, что даже один и тот же адрес размещения многократно исполняемой тестовой программы не гарантирует одинаковых условий исполнения: нельзя обеспечить одинаковые начальные значения системных регистров, при отсутствии возможности их определения по эталонной модели. Для решения второй проблемы, связанной с вычислением адресов перехода, задействуется механизм исключений: в тех местах, где необходимо осуществить переход на следующий блок кода в зависимости от результата выполнения инструкции условного перехода (факт осуществления перехода нельзя определить в момент генерации теста, поскольку отсутствие эталонной модели не позволяет определить значение аргументов инструкции), в код тестовой программы вставляется инструкция, вызывающая прерывание. В обработчике прерывания происходит обращение к генератору тестовых программ, который опираясь на текущие наблюдаемые

значения системных регистров генерирует корректную инструкцию, осуществляющую переход по желаемому адресу.

3.1.2 Многоядерность и многопоточность

Ориентация генератора тестовых программ на многопоточные и многоядерные микропроцессоры означает необходимость запуска копий образа исполнителя тестов на каждом ядре и их синхронизацию друг с другом для создания разнообразных ситуаций одновременного доступа к общим ресурсам (в частности, оперативной памяти и когерентной кэш-памяти). В Threadmill для задачи синхронизации используется рассылка одновременно работающим исполнителям тестов инициализационного числа (*seed*), которое необходимо для определения их дальнейших действий. Далее исполнители тестов взаимодействуют между собой на уровне одноранговой сети.

3.1.4 Работа с исключениями

В случае Threadmill исключения используются для замены адреса на динамически вычисленные (в зависимости от расположения) в сгенерированном ассемблерном коде.

3.1.5 Выводы

Инструмент Threadmill является зрелым инструментом пост-производственной верификации микропроцессоров фирмы IBM, ориентированным на многопоточное тестирование. Инструмент связан с инструментом до-производственного тестирования Genesys-Pro, используя его информацию о тестируемой архитектуре. Основным недостатком подхода является как ограниченность способов проверки корректности, так и в целом закрытость инструмента.

3.2 RIS (MEMRIS)

В ARM разрабатывается генератор случайных тестов RIS (Random Instruction Sequences) [30], предназначенный для тестирования подсистемы памяти микропроцессоров с архитектурой ARM. С технологической точки зрения, подход, лежащий в его основе, аналогичен подходу, применяемому в Threadmill: генератор является двоичным образом, загружаемым на тестируемый микропроцессор, включает в себя информацию о целевой архитектуре, и производит оценку корректности на основе условно-эталонных значений.

Генератор, загружаемый на тестируемый микропроцессор, обеспечивает непрерывную генерацию и исполнение тестовых примеров на основе конфигурационных файлов. Конфигурационные файлы включают в себя информацию об используемых регистрах микропроцессора, свойствах областей памяти и регистров, веса инструкций для генерации тестов, количество инструкций в тесте, количество тестов и число запусков каждого сгенерированного теста. Генерация направленных тестов осуществляется за счет внедрения функций, описываемых на языке C++, вызов которых будет интегрирован генератором в последовательность инструкций случайного теста. Поскольку вызов функции имеет дополнительные накладные расходы, связанные с сохранением состояния процессора, инструмент поддерживает альтернативный механизм описания желаемых последовательностей инструкций при помощи специальных макросов, которые очень близки по внешнему виду с ассемблером ARM. Для проверки корректности результатов тестирования тестовые примеры исполняются дважды. Инструмент ориентирован на тестирование многоядерных процессоров и имеет встроенные механизмы проверки общего доступа в память, для чего включает специализированный алгоритм расположения данных в общей памяти.

Схема проведения верификации с помощью RIS выглядит следующим образом. Сначала создается тестовый план (test plan) на основе сценария тестирования. Тестовый план является отправной точкой для разработки конфигурационных файлов для настройки RIS-генератора. Сгенерированные тестовые программы подаются на тестируемое устройство для обнаружения ошибок его реализации.

Как и Treadmill, данный подход является коммерческим, и его исходный код закрыт.

3.3. RITG

Еще одним коммерческим инструментом пост-производственного тестирования микропроцессоров является *RITG* (*Random Instruction Test Generator*) [32], разработанный и используемый в компании Intel. Данный инструмент генерирует и исполняет случайные последовательности команд. Цели генерации задаются при помощи специальных конфигурационных файлов и тестовых шаблонов (*test patterns*). Инструмент включает в себя также компонент-упаковщик, который создает компактное представление результатов для отправки на инструментальную машину с запущенным симулятором тестируемой архитектуры для проверки корректности результатов. Отдельные копии RITG работают под наблюдением компонента, называемого *Лабораторным менеджером* (*Laboratory manager*), который контролирует: типы создаваемых генераторами тестов, связь тестируемых систем с тестовыми генераторами, мониторинг возникающих ошибок, перезапуск и воспроизведение тестов. Каждая отдельная тестовая последовательность маркируется уникальным индексом, который позволяет повторять конкретный тест в случае нахождения в нем ошибки. Модель памяти для загрузки ядра генератора включает в себя следующие элементы: обработчик исключений, множество тестов в двоичном виде, размеченное пространство для результатов тестов. RITG генерирует множество случайных тестов, каждый из которых начинается со сброса (reset) и включает:

1) уникальную в зависимости от инициализационного числа (seed) мини-операционную систему с высоким уровнем случайности параметров компонентов;
2) несколько тысяч случайных инструкций пользовательского кода;
3) уникальную в зависимости от инициализационного числа (seed) настройку внешних интерфейсов и специальную обвязку для проведения верификации.

Подытоживая, можно отметить, что инструмент RITG является сильно автоматизированным генератором случайных тестовых последовательностей, ориентированным на архитектуру микропроцессоров фирмы Intel, использующим инструментальную машину для оценки корректности наблюдаемого поведения.

3.4 Промежуточные выводы

Общепринятым подходом к пост-производственному тестированию микропроцессоров является использование инструментов онлайн-генерации тестовых программ. Такие генераторы загружаются непосредственно в память, доступную микропроцессору, и осуществляют его тестирование путем порождения и исполнения разнообразных последовательностей команд. Построение команд осуществляется на основе некоторого вида шаблонов, задающих их структурные и поведенческие свойства.

Все рассмотренные инструменты (см. табл. 1) являются коммерческими и закрытыми. По этой причине их функциональные возможности и характеристики сложно точно оценить. Общий недостаток данных инструментов заключается в том, что они применимы для тестирования микропроцессоров со строго определенной архитектурой. Поэтому актуальной видится задача разработки методов, позволяющих создавать инструменты онлайн генерации, применимые для широкого спектра микропроцессорных архитектур.

Из сведений, находящихся в табл. 1, можно сделать следующие выводы:

1) в части спецификаций инструменты пост-производственной верификации обычно имеют связь с инструментами до-производственного тестирования;
2) для генерации используется набор тестовых сценариев, конфигурационных файлов, соответствующих целям тестирования, и описываемых на некотором макроязыке;
3) для оценки результатов тестирования обычно используется неоднократный прогон, либо инструментальная машина с запущенным симулятором;
4) тестовым покрытием считается покрытие сценариев тестирования;
5) многопоточное тестирование подразумевает тиражирование загружаемого двоичного образа с генератором, который нацеливается на создание ситуации одновременного доступа к памяти, а синхронизация между генераторами осуществляется либо на их уровне, либо с помощью отдельного компонента управления.

Табл. 1. Рассмотренные инструменты тестирования
Table 1. Testing Toolkits Reviewed

Название инстру-мента	Спецификация и настройка генерации	Проверка корректности	Оценка полноты тестирования	Поддержка многопоточности
Threadmill	Повторно используются спецификации из инструмента Genesys-Pro. Для генерации используются сценарии тестирования.	Многократ-ный прогон одних и тех же тестов, размещенных по разным адресам	Покрытие сценариев тестирования	Одноранговое взаимодействие единых двоичных образов, настроенных с помощью инициа-лизационного числа
RIS	Спецификация явно не описывается, однако тестовые сценарии описываются на языке, близком к ассемблеру ARM. Для генерации используются сценарии (планы) тестирования.	Двукратный прогон	Покрытие сценариев тестирования, инструкций и их комбинаций	Одноранговый единый двоичный образ и использование инициализационного числа
RITG	Спецификация содержится на инструментальной машине в виде симулятора целевой архитектуры. Для генерации используются тестовые шаблоны и конфигурационные файлы.	Инструмен-тальная машина с запущенным симулятором	Покрытие сценариев тестирования	Используется Лабораторный менеджер для запуска и синхронизации отдельных копий генераторов

4. Поиск перспективного решения

Итак, как уже было отмечено, основным недостатком существующих зрелых решений пост-производственной верификации является закрытость инструментов и их ориентация на определенную архитектуру микропроцессоров. При этом существует ряд методов осуществления тех или иных аспектов верификации, рассмотренных в разделе 2, но не претендующих на полное решение.

Целью данного раздела является поиск перспективного решения пост-производственной верификации, которое должно быть открытым и должно учитывать сильные стороны перечисленных в разделе 3 инструментов. А это значит, что:

1) должна обеспечиваться высокая скорость генерации тестов для разных целевых архитектур;
2) в подходе необходима реализация оценки корректности наблюдаемого поведения либо путем многократного прогона тестов (возможно, видоизмененных), либо использованием инструментальной машины;
3) должна обеспечиваться диагностика ошибочного поведения.

Разработку перспективного инструмента предлагается вести на основе среды разработки генераторов тестовых программ MicroTESK [34], исходный код которой открыт. По аналогии со связкой инструментов Genesys-Pro и Threadmill предлагается добиться большой степени повторного использования компонентов, разработанных в рамках генератора тестов для функциональной верификации RTL-моделей микропроцессоров. Так, предлагается повторно использовать спецификации системы команд на языке nML: они включают в себя двоичное представление каждой команды, поэтому отсутствует необходимость запускать компилятор на исполняемом микропроцессоре. Аналогичным образом предлагается повторно использовать шаблоны тестовых программ на языке Ruby. Однако для непосредственного запуска генерации тестов эти шаблоны должны быть транслированы в более простое представление. Предполагается, что спецификации и шаблоны подаются на вход среды, создающей программу в двоичном коде, поддерживаемом целевым микропроцессором, которая включает в себя как возможности *генерации*, *запуска* и *проверки* тестовых программ, так и необходимые сервисы, обычно предоставляемые операционной системой.

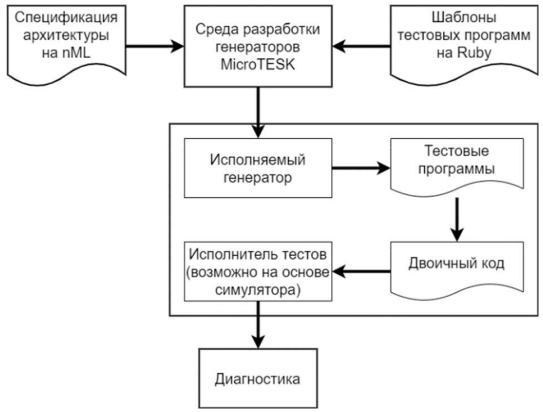

Рис. 2. Предлагаемый подход к онлайн генерации
Fig. 2. The suggested approach to online generation

Предлагаемое архитектурное решение изображено на рис. 2. Исполняемый генератор, вызываемый при работе тестовой системы на целевом микропроцессоре, создает тестовые программы, которые необходимо разместить в памяти для получения готового к запуску двоичного кода. *Исполнитель тестов* размещает эти программы, передает управление на соответствующие адреса и оценивает корректность результатов работы тестовых программ на основе какой-либо методики. Простейший вариант такой методики заключается в повторном многократном запуске одного и того же кода, где между различными случайными инструкциями вставляются инструкции, которые при их корректной работе не должны приводить к каким-либо изменениям в наблюдаемом поведении (т.е. nop, которых в микропроцессорах гораздо больше одного). Если же наблюдаемое поведение изменяется от прогона к прогону тестов, то фиксируется ошибочное поведение.

Более сложным вариантом является создание *мутантов*, т.е. таких тестовых последовательностей, которые были бы похожи на исходные, были им функционально эквивалентны и при этом имели определенные не влияющие на поведение отличия, например, измененные номера регистров. Заметим, что рассмотренный первым простейший случай является частным случаем создания мутанта. Наконец, исполнитель тестов может самостоятельно вычислять эталонные значения с помощью симулятора, созданного, например, на основе тех же nML-спецификаций. Во всех трех случаях при обнаружении ошибок сигнатуры инструкций исполненных тестовых последовательностей передаются в компонент *диагностики*, который располагается на инструментальной ЭВМ, где для анализа ошибочного поведения может быть задействован тот же MicroTESK.

Для адаптации MicroTESK к пост-производственной верификации необходимо решить еще несколько технических моментов:

1) поддержать обработку исключений, диагностика которых в пост-производственном случае крайне затруднена;
2) поддержать подсистему работы с памятью;
3) иметь поддержку многоядерности и многопоточности.

Отвечая на эти вопросы, отметим, что для каждого из них существуют как простые решения, так и более сложные, позволяющие проводить более детальные исследования целевого микропроцессора. При этом открытость инструмента тестирования позволяет при необходимости разрабатывать и добавлять отсутствующие в нем возможности.

Простыми решениями могли бы, например, быть:

1) использование обработчика исключений, возвращающего исполнение на следующую инструкцию, за той, что привела к исключению;
2) использование неоптимизированного по занимаемой памяти компонента распределения областей виртуальной памяти, однако учитывающего информацию о целевых сегментах и их отображение на физическую память;
3) создание одной и той же программы для всех ядер (потоков) с путем исполнения этой программы, определяемым номером исполняющего ее ядра, как это и делается в MicroTESK для до-производственного случая.

5. Заключение

В данной работе были рассмотрены различные подходы к созданию пост-производственных генераторов тестовых программ для микропроцессоров. Из их анализа следует, что на данный момент не существует такого решения, которое одновременно полностью бы удовлетворяло следующим выбранным критериям: скорость работы, открытый исходный код, возможность использования спецификаций, разработанных для генерации тестовых программ на этапе разработки RTL-моделей, поддержка многоядерных микропроцессоров. При этом перспективной видится как раз реализация принципа повторного использования спецификаций, так как это позволяет обрести возможность настроить генератор на разные целевые архитектуры, что повышает его ценность. Открытость исходного кода также является важным моментом.

Таким образом, в качестве основы для создания онлайн-генератора может быть использована среда генерации тестовых программ MicroTESK, которая включает формальные спецификации тестируемой архитектуры микропроцессора на языке nML, допускающие повторное использование в пост-производственном случае. В качестве метода генерации предлагается использовать генерацию случайных тестовых последовательностей на начальном этапе, а затем – шаблоны тестов по аналогии с до-производственным случаем. Методом оценки корректности может служить как многократный запуск функционально

эквивалентных тестов, так и сравнение с результатами работы симулятора. Для анализа ошибок предлагается использовать сохранение сигнатур инструкций тестовых последовательностей, приведших к наблюдаемому ошибочному поведению, для запуска аналогичных последовательностей в рамках среды MicroTESK, развернутой на инструментальной ЭВМ.

Список литературы / References

[1]. H. Foster. Trends in functional verification: a 2014 industry study. In Proc. of the Design Automation Conference, 2015, pp. 1-6.

[2]. P. Mishra, F. Farahmandi. Post-Silicon Validation and Debug. Springer, 2019, 394 p.

[3]. S. Mitra, S. A. Seshia, N. Nicolici. Post-Silicon Validation Opportunities, Challenges and Recent Advances. In Proc. of the Design Automation Conference, 2010, pp. 12-17.

[4]. Q. Xu, X. Liu. On signal tracing in post-silicon validation. Proceedings of Asia and South Pacific Design Automation Conference, 2010, pp. 262–267. DOI: 10.1109/ASPDAC.2010.5419883

[5]. B. Kumar, A. Jindal et al. A Methodology for Trace Signal Selection to Improve Error Detection in Post-Silicon Validation. In Proc. of the International Conference on VLSI Design and International Conference on Embedded Systems, 2017, pp. 147-152.

[6]. K. Basu, P. Mishra. RATS: Restoration-Aware Trace Signal Selection for Post-Silicon Validation. IEEE Transactions on Very Large Scale Integration Systems, vol. 21, issue 4, 2013, pp. 605-613.

[7]. S. B. Park, S. Mitra. Post-silicon bug localization for processors using IFRA. Communications of the ACM, vol. 53, issue 2, 2010, pp. 106-113.

[8]. S. Chen, M. Hsiao et al. A configurable bus-tracer for error reproduction in post-silicon validation. In Proc. of the International Symposium on VLSI Design, Automation, and Test, 2013, pp. 1-4.

[9]. E. El Mandouh, A. G. Wassal. Application of Machine Learning Techniques in Post-Silicon Debugging and Bug Localization. Journal of Electronic Testing, vol. 34, issue 2, 2018, pp. 163-181.

[10]. E. Singh, C. Barrett, S. Mitra. E-QED: Electrical Bug Localization During Post-silicon Validation Enabled by Quick Error Detection and Formal Methods. In Proc. of the International Conference on Computer-Aided Verification, 2017, pp. 104-125.

[11]. E. Singh, D. Lin et al. Logic Bug Detection and Localization Using Symbolic Quick Error Detection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, pp. 1-14.

[12]. S. Ma, D. Pal et al. Can't see the forest for the trees: State restoration's limitations in post-silicon trace signal selection. In Proc. of the International Conference on Computer-Aided Design (ICCAD), 2015, pp. 1-8.

[13]. S. Aslan, G. K. R. Chandrai, V. Siddaiah. Unified Coverage Methodology for SoC Post-Silicon Validation. Optics and Photonics Journal, vol. 6, issue 10, 2016, pp. 261-268.

[14]. B. Kumar, K. Basu et al. RTL level trace signal selection and coverage estimation during post-silicon validation. In Proc. of the International High Level Design Validation and Test Workshop, 2017, pp. 59-66.

[15]. E. E. Mandouh, A. Gamal et al. Construction of coverage data for post-silicon validation using big data techniques. In Proc. of the International Conference on Electronics, Circuits and Systems, 2017, pp. 46-49.

[16]. F. Farahmandi, R. Morad et al. Cost-effective analysis of post-silicon functional coverage events. In Proc. of the Design, Automation and Test in Europe Conference and Exhibition, 2017, pp. 392-397.

[17]. F. Farahmandi, P. Mishra. Utilization of Debug Infrastructure for Post-Silicon Coverage Analysis. In: Post-Silicon Validation and Debug. Springer, 2019, pp. 307-321.

[18]. F. Farahmandi, P. Mishra, S. Ray. Exploiting transaction level models for observability-aware post-silicon test generation. In Proc. of the Design, Automation and Test in Europe Conference and Exhibition, 2016, pp. 1477-1480.

[19]. A. Adir, M. Golubev et al. Threadmill: A post-silicon exerciser for multi-threaded processors. In Proc. of the Design Automation Conference, 2011, pp. 860-865.

[20]. I. Wagner, V. Bertacco. Post-Silicon and Runtime Verification for Modern Processors. Springer, 2011, 241 p.

[21]. N. Nicolici. On-Chip Stimuli Generation for Post-Silicon Validation. In Proc. of the International High Level Design Validation and Test Workshop, 2012, pp. 108-109.

[22]. P. Moharikar, J. Guddeti. Automated test generation for post silicon microcontroller validation. In Proc. of the International High Level Design Validation and Test Workshop, 2017, pp. 45-52.
[23]. X. Shi. Constrained-Random Stimuli Generation for Post-Silicon Validation. PhD Thesis, McMaster University, Hamilton, Ontario, Canada, 2016, 171 p.
[24]. V.M. Suryasarman, S. Biswas, A. Sahu. Automation of Test Program Synthesis for Processor Post-silicon Validation. Journal of Electronic Testing, vol. 34, 2018, pp. 83-103.
[25]. L. Klemmer, D. Große. EPEX: Processor Verification by Equivalent Program Execution. In Proc. of the Great Lakes Symposium on VLSI, 2021, pp. 33-38.
[26]. J. Alglave, L. Maranget, M. Tautschnig. Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. ACM Transactions on Programming Languages and Systems, vol. 36, issue 2, 2014, article 7, pp. 1-74.
[27]. A. DeOrio, I. Wagner, V. Bertacco. Dacota: Post-silicon validation of the memory subsystem in multi-core designs. In Proc. of the International Symposium on High Performance Computer Architecture, 2009, pp. 405-416.
[28]. D. Lee, V. Bertacco. MTraceCheck: Validating non-deterministic behavior of memory consistency models in post-silicon validation. In Proc. of the Annual International Symposium on Computer Architecture, 2017, pp. 201-213.
[29]. A. Adir, A. Nahir, A. Ziv. Concurrent Generation of Concurrent Programs for Post-Silicon Validation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 31, issue 8, 2012, pp. 1297-1302.
[30]. S. Thiruvathodi, D. Yeggina. A Random Instruction Sequence Generator for ARM Based Systems. In Proc. of the International Microprocessor Test and Verification Workshop, 2014, pp. 73-77.
[31]. B. Bentley. Validating the Intel Pentium 4 Microprocessor. In Proc. of the Design Automation Conference, 2001, pp. 244-248.
[32]. T. Bojan, F. Igor, M. Robert. Intel's Post Silicon Functional Validation Approach. In Proc. of the High Level Design Validation and Test Workshop, 2007, pp. 53-56.
[33]. A. Adir, S. Copty et al. A unified methodology for pre-silicon verification and post-silicon validation. In Proc. of the Design, Automation and Test in Europe Conference and Exhibition, 2011, pp. 1-6.
[34]. MicroTESK. URL: http://www.microtesk.org/, accessed 15.11.2021.

Информация об авторах / Information about authors

Никита Дмитриевич ЧЕРТОК – стажер-исследователь отдела технологий программирования ИСП РАН, обучается в аспирантуре ИСП РАН. Область научных интересов: автоматизация проектирования цифровой аппаратуры.

Nikita Dmitrievich CHERTOK is a researcher at the Software Engineering Department of ISP RAS. His research interests include digital hardware design automation. Nikita is a PhD student at ISP RAS.

Михаил Михайлович ЧУПИЛКО – кандидат физико-математических наук, старший научный сотрудник отдела технологий программирования ИСП РАН, старший научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Mikhail Mikhaylovich CHUPILKO is a senior researcher at the Software Engineering Department of ISP RAS, senior researcher at the Heterogeneous Computing Systems research lab of Plekhanov RUE. His research interests include digital hardware design automation, verification and testing. Mikhail has a PhD degree in Physics and Mathematics.