Федеральное государственное бюджетное учреждение науки
Институт системного программирования Российской академии наук

УДК 519.68

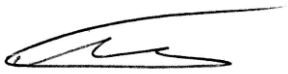
На правах рукописи

Чупилко Михаил Михайлович

ДИНАМИЧЕСКАЯ ВЕРИФИКАЦИЯ ЦИФРОВОЙ АППАРАТУРЫ НА ОСНОВЕ ФОРМАЛЬНЫХ СПЕЦИФИКАЦИЙ

Специальность 05.13.11 -
математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата физико-математических наук



Москва
2012

Работа выполнена в Институте системного программирования РАН.

Научный руководитель: профессор,
доктор физико-математических наук
Петренко Александр Константинович

Официальные оппоненты: доктор физико-математических наук
Лацис Алексей Оттович

доктор технических наук
Захаров Виктор Николаевич

Ведущая организация: **Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Московский государственный институт электроники и математики (технический университет)» (МИЭМ)**

Защита диссертации состоится « 26 » апреля 2012 г. в 16 часов на заседании диссертационного совета Д.002.087.01 при Институте системного программирования РАН по адресу:

109004, Москва, ул. А. Солженицына, д.25,
Институт системного программирования РАН, конференц-зал (комн. 110).

С диссертацией можно ознакомиться в библиотеке Института системного программирования РАН.

Автореферат разослан « 23 » марта 2012 г.

Ученый секретарь
диссертационного совета
кандидат физ.-мат. наук /Прохоров С.П./

Общая характеристика работы

Актуальность темы

Современная цифровая аппаратура на основе *интегральных схем (ИС)* представляет собой сложные устройства, проектирование и производство которых требует больших затрат ресурсов. Непосредственное изготовление ИС происходит с использованием наборов фотошаблонов, которые создаются на основе *схемы соединений* (*net list*). Схема соединений получается автоматически путем синтеза из модели, созданной на специализированном языке описания аппаратуры. В настоящее время проектирование моделей устройств осуществляется с помощью языков, называемых *языками описания аппаратуры* (*Hardware Description Language, HDL*). Такие модели, основанные на HDL-описаниях, часто называют *имитационными*, так как они допускают их выполнение в специализированной среде имитационного моделирования – *симуляторе*.

Цена ошибки в аппаратуре может оказаться очень высокой: известный случай замены микропроцессоров Intel Pentium с ошибкой деления обошелся компании приблизительно в 500 миллионов долларов (*Wolfe A. For Intel, it's a case of FPU all over again. EE Times, 1997. №5*). Так как исправление ошибок в уже готовых микросхемах невозможно, поиск и нахождение *функциональных ошибок* проводится на этапе проектирования HDL-описания устройства. Подобного рода деятельность, состоящая в проверке соответствия поведения аппаратуры, задаваемого HDL-описанием, его спецификации, называется *верификацией*. Верификация обычно требует до 70% от общих трудозатрат на создание всего устройства. Это обусловлено следующими причинами: логической сложностью HDL-описаний и ее постоянным увеличением (высокая степень параллелизма, асинхронность и большие размеры), существенной итеративностью процесса разработки HDL-описаний, требующего постоянной модификации тестов и HDL-описаний, меньшим уровнем модульности, чем в

области программного обеспечения, фрагментарностью и неактуальностью документации.

Верификация может проводиться как *статически*, то есть *формальными методами*, когда корректность HDL-описания доказывается математически, так и *динамическими*, когда поведение, задаваемое HDL-описанием, сравнивается с некоторым эталоном в процессе его выполнения в среде симулятора. Хотя формальные методы в последнее время получили большое развитие, их использование всё ещё является очень ограниченным.

Традиционный метод динамической верификации состоит в следующем. Набор входных данных для проведения верификации (*тестовая последовательность)* описывается вручную или генерируется случайным образом, но в рамках *ограничений*, наложенных на значения отдельных элементов последовательности. Собственно *проверка корректности* поведения, задаваемого HDL-описанием, оценивается на основе *утверждений (assertions)*, а для *оценки полноты* верификации используются метрики на основе *покрытия кода* HDL-описания и *покрытия комбинаций событий*, возникающих в процессе выполнения HDL-описания в симуляторе (*Open Verification Methodology User Guide [Электронный ресурс], 2011. URL: http://verificationacademy.com/file/ovm-2.1.2.zip*).

Рассмотрим этот метод в контексте проблем верификации HDL-описаний. Проблема логической сложности при использовании такого метода при создании простых тестов не является принципиальной, однако описание ограничений на значения элементов тестовой последовательности и описание утверждений в этом случае могут быть в большой степени неполными. При необходимости же более детальных проверок на доработку таких утверждений и ограничений требуется существенное время, практически равное времени разработки таких наборов с нуля. Таким образом, этот метод не ориентирован на поддержку итеративного характера разработки HDL-описаний, ввиду необходимости создания заново основных частей тестовых систем для новых версий HDL-описаний. Наконец, проблемы с документацией также приводят к

фрагментарности проверяемых свойств, от чего также страдает качество верификации.

Решению задачи тщательного тестирования HDL-описаний посвящено много работ. Многие из проблем, перечисленных выше, решены, например, в методе верификации HDL-описаний, основанном на формальных спецификациях, описывающих ожидаемое поведение тестируемой аппаратуры, в форме программных контрактов, наложенных на все операции и микрооперации (*Камкин, А.С. Метод автоматизации имитационного тестирования микропроцессоров с конвейерной архитектурой на основе формальных спецификаций, диссертация на соискание ученой степени кандидата физико-математических наук. М., 2009*). Такие спецификации описывают ожидаемое поведение с потактовой точностью. Верификация в этом случае получается очень тщательной, так как вердикт касательно корректности HDL-описания выносится на основе и проверки данных, и проверки номера такта, на котором выполняемое в симуляторе HDL-описание выдало реакцию. Детальная документация, равно как и необходимость в столь тщательных проверках, появляется, как правило, на заключительных итерациях создания HDL-описаний, поэтому наилучшее применение метод нашел именно на них. Требования метода к документации, а также сложность разработки спецификаций сделали его применение ограниченным: разработка потактовых спецификаций, которые к тому же не используют разработанные ранее на предыдущих этапах эталонные модели, занимает много ресурсов и времени, которых может и не остаться на заключительных итерациях проектирования.

Таким образом, ввиду важности и актуальности верификации HDL-описаний аппаратуры и отсутствия готовых решений, полностью удовлетворяющих промышленному применению на разных этапах проектирования, была поставлена задача исследования проблем верификации HDL-описаний. На основе анализа проблем необходимо предложить метод динамической верификации, учитывающий особенности как самих HDL-описаний, так и их процесса разработки, а именно: объективную сложность,

инкрементальный процесс разработки, неполноту требований, особенно на ранних этапах проектирования.

Цель и задачи работы

Основной целью работы является разработка метода верификации цифровой аппаратуры на основе формальных спецификаций в форме программных моделей с учетом требований промышленных процессов проектирования HDL-описаний. Для достижения цели работы были поставлены следующие задачи:

1. Провести анализ существующих методов верификации цифровой аппаратуры;
2. Разработать метод верификации HDL-описаний аппаратуры на основе использования эталонных программных моделей, допускающий:
    - применение при неполноте требований;
    - инкрементальный процесс разработки и уточнения формальных спецификаций в процессе проектирования;
    - композицию тестовых систем различных HDL-описаний;
3. Разработать инструменты, реализующие метод;
4. Оценить реализацию метода на практике.

Научная новизна работы

Научной новизной обладают следующие результаты работы:

1. Метод спецификации цифровой аппаратуры, подходящий для использования на разных уровнях абстракции;
2. Метод сопоставления реакций цифровой аппаратуры и реакций эталонной модели, позволяющий автоматизировать процедуру проверки цифровой аппаратуры.

Практическая значимость

На основе предложенного метода верификации были разработаны средства, позволяющие создавать тестовые системы на основе эталонных моделей на C++ на разных уровнях абстракции. Разработанные средства являются библиотекой классов на языке C++. Они были применены в 9 проектах верификации модулей промышленных микропроцессоров в течение 2009-2012 гг. В процессе верификации в среднем обнаруживалось 3-5 серьезных ошибок в модулях, прошедших верификацию другими методами. Продемонстрированы преимущества метода для поддержки итеративного процесса разработки.

Доклады и публикации

Основные положения работы докладывались на следующих конференциях и семинарах:

- Весенний коллоквиум молодых исследователей в области программной инженерии (SYRCoSE: Spring Young Researchers Colloquium on Software Engineering, г. Нижний Новгород, 2010 г. и г. Екатеринбург, 2011 г.);
- Международный симпозиум «Восток-Запад: проектирование и тестирование» (EWDTS: East-West Design & Test Symposium, г. Санкт-Петербург, 2010 г. и г. Севастополь, 2011 г.);
- Международная конференция «Балтийская электронная конференция» (Baltic Electronics Conference, г. Таллин, 2010 г.);
- Семинар Института системного программирования РАН (г. Москва, 2011-2012 гг.).

По теме диссертации автором опубликовано 11 работ (из них 3 в изданиях из перечня ВАК), список которых приведен в конце автореферата.

Структура и объем диссертации

Работа состоит из введения, 4 глав, заключения и списка литературы (72 наименования). Основной текст диссертации (без приложений и списка литературы) занимает 120 страниц.

Краткое содержание диссертации

Во **введении** обосновывается актуальность темы работы, определяются ее цели и задачи, раскрывается практическая значимость.

В **первой главе** приводится анализ аналогичных работ в области исследования. В главе выделяются проблемы верификации HDL-описаний аппаратуры и рассматриваются методы их решения, относящиеся как к динамической, так и к формальной верификации. Затем делается анализ существующих методов автоматизации тестирования сложных устройств аппаратуры, и делаются выводы о степени их применимости для достижения цели настоящей работы. Один из выводов, в частности, говорит об отсутствии на данный момент готовых решений, которые одновременно бы решали все поставленные задачи. В главе выдвигается тезис о необходимости поддержки повторного использования *модулей эталонных моделей аппаратуры* (обычно разрабатываемых на C++ для проведения *системной верификации*) для проведения *модульной верификации*. В конце главы уточняются задачи диссертационной работы.

В начале **второй главы** исследуются возможные пути решения поставленных задач с использованием идей, реализованных в существующих методах верификации и удовлетворяющих поставленной цели. Прежде всего, определяется необходимость использования языка C++, а не SystemVerilog, или какого-либо другого специализированного языка, в основном потому, что он используется в процессах написания эталонных моделей, а любые межъязыковые интерфейсы затрудняют поддержку системы. Далее рассмотрены проблемы верификации и возможные варианты их решения, а

именно проблемы быстрого начала верификации на ранних этапах проектирования, проблемы обеспечения инкрементальной разработки и верификации с помощью первоначально разработанных неполных спецификаций и проблема обеспечения композиции спецификаций и тестов модулей для тестирования более крупных подсистем.

Быстрое начало верификации на ранних этапах проектирования влечет за собой использование неполных спецификаций, порождающих неточность или невозможность проверок. Неточность бывает как функциональная, так и временн*а*я. Следовательно, необходимо, с одной стороны, иметь возможность проверять только те функциональные и временн*ы*е свойства, которые уже имеются в спецификациях, а с другой стороны, иметь возможность постепенного повышения точности проверки в соответствии с ходом уточнения спецификаций. Для удовлетворения этих требований можно использовать:

- расширяемые спецификации, отдельные результаты работы которых можно сравнить с работой выполняющегося в симуляторе HDL-описания,
- части эталонных моделей, временн*а*я точность которых расширяется *сопоставителями* реакций,
- спецификации не в явном исполнимом виде, а в виде более абстрактных формальных спецификаций, например, в виде контрактов.

В случае быстрого начала верификации накладываются также ограничения на генераторы тестовых последовательностей, настройка которых не должна занимать много времени. Тестовую последовательность можно конструировать вручную, но такие последовательности не могут в дальнейшем использоваться для создания более сложных последовательностей. Альтернативой ручному конструированию являются расширяемые параметризируемые генераторы тестовых последовательностей на основе некоторых стандартных компонентов.

Необходимость учета инкрементального процесса разработки после быстрого начала верификации влечет за собой необходимость уточнения

спецификаций параллельно уточнению HDL-описания, что включает две подзадачи: собственно построение спецификаций с возможностью уточнения и организацию проверок результатов выполнения в симуляторе HDL-описания с использованием таких спецификаций. Первая подзадача может быть решена особой модульной архитектурой, уточнением абстрактных спецификаций, взятых из эталонной модели, использованием формальных спецификаций в виде контрактов, или вообще отказом от явных спецификаций и описанием ограничений на результаты выполнения HDL-описания в виде темпоральных утверждений. Вторая – может решаться непосредственной сверкой результатов, вычисленных на основе формальных спецификаций и приведенных на уровень сигналов, и результатов выполнения HDL-описания; выделением промежуточных компонентов, накапливающих результаты работы эталонной модели и осуществляющих сверку, или выделением таких компонентов для каждой совокупности выходных сигналов HDL-описания. Требование учета инкрементального процесса разработки также приводит к необходимости использования расширяемых генераторов тестовой последовательности на основе стандартных компонентов.

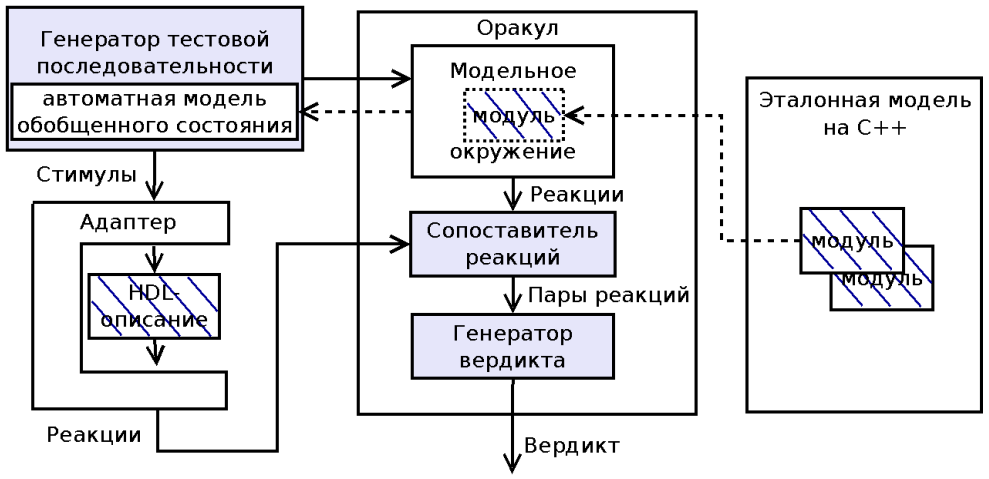
Композиция спецификаций как средство борьбы со сложностью верификации полного HDL-описания является дополнительным критерием выбора способа представления спецификаций. Существует два основных способа: выделение специализированных интерфейсов (описание конфигурации связей), связывающих спецификации между собой, которые могут быть реализованы либо на уровне сигналов, либо на некотором более абстрактном уровне, а также – встраивание спецификаций друг в друга вручную. Во втором случае одновременно модифицируются и функциональные и коммутационные части спецификаций, что достаточно затратно. Основной проблемой композиции является необходимость учета коммутации входных и выходных сигналов в HDL-описании и в спецификации согласованным образом.

Далее автором предлагается метод верификации цифровой аппаратуры, решающий поставленные задачи поддержки быстрого начала верификации,

учета итеративных изменений в проекте и композиции спецификаций. Решение этих задач осуществляется с помощью метода спецификации HDL-описаний, поддерживающего разные уровни абстракции и композицию, и метода сопоставления реакций, позволяющего автоматизировать использование таких спецификаций. Оба метода приводятся в разделе описания архитектуры тестовой системы.

Предлагаемая общая архитектура тестовых систем (см. рисунок 1) включает следующие основные компоненты, типичные для динамического тестирования: *генератор тестовой последовательности* (включающий *автоматную модель обобщенного состояния*), *оракул* (включающий *модельное окружение*, *сопоставитель реакций* и *генератор вердикта*), *адаптер* между тестовой системой и тестируемым HDL-описанием.

Создаваемый оракул ориентирован на использование модулей эталонной модели на C++, обычно разрабатываемой для аппаратуры в целях проведения системной верификации. При отсутствии таких модулей в эталонной модели их разрабатывают в соответствии с интерфейсом модельного окружения.



Серый цвет – *полностью автоматически создаваемые компоненты*

Штриховка – *сторонние компоненты*

Без заполнения – *создаваемые с использованием библиотечных средств компоненты*

Рисунок 1 – Обобщенная структура тестовой системы

Рассмотрим кратко архитектуру тестовой системы: компоненты и обмен данными между ними. Генератор тестовой последовательности на основе

модели обобщенного состояния создает *поток стимулов*. Этот поток поступает в тестируемое HDL-описание через адаптер и в модуль эталонной модели через модельное окружение. В ответ на стимулы HDL-описание и модуль эталонной модели создают *реакции*, которые поступают в сопоставитель. Сопоставитель группирует пришедшие реакции в пары, и они передаются в генератор вердикта, который на каждом такте выносит решение о корректности текущего поведения HDL-описания. Тестирование заканчивается, когда находится ошибка или достигается критерий полноты, зависящий от метрик покрытия, вычисленных на основе кода HDL-описания и модуля эталонной модели. Рассмотрим архитектуру более детально.

Генератор создает *поток стимулов* с использованием автоматной модели обобщенного состояния, создаваемой вручную на основе модуля эталонной модели. Модель обобщенного состояния задается с помощью *списка стимулов*, поддерживаемых модулем эталонной модели, и *функции получения текущего состояния* модуля эталонной модели. Генератор выбирает стимулы из списка либо случайным образом, либо на основе обхода *конечного автомата*, соответствующего автоматной модели (*Кулямин В. В., Петренко А. К., Косачев А.С., Бурдонов И. Б. Подход UniTesK к разработке тестов. Программирование, 29(6):25–43, 2003*). Использование случайного потока стимулов позволяет быстро начать верификацию, в то время как обход конечных автоматов ориентирован на подачу всех возможных стимулов во всех возможных состояниях, его отличает тщательность проверок. Достоинствами генерации случайного потока стимулов являются низкие временные затраты на разработку генератора, а также метод проверки «вширь», когда поток содержит как можно большее количество операций в разных комбинациях. Недостатком случайных генераторов принято считать невысокую вероятность обнаружения сложных ошибок, так как для этого обычно необходимо привести выполняемое в симуляторе HDL-описание в некоторое особое состояние, что практически невозможно при случайной генерации тестов. Генерация тестов при обходе конечного автомата решает проблему приведения выполняемого в симуляторе

HDL-описания в особые состояния, если используемый для обхода автомат достаточно точно отражает состояние выполняемого в симуляторе HDL-описания. Требование к адекватности используемого автомата является слабым местом этого метода генерации тестов, так как требует значительных усилий по разработке и поддержке потактовой эталонной модели, которая наиболее адекватно отражает состояние выполняемого в симуляторе HDL-описания, и на основе которой строится модель обобщенного состояния.

Эталонная модель создает *модельные реакции*, которые считаются корректными и необходимы для проверки реакций выполняемого в симуляторе HDL-описания (*реакций реализации*). Модельные реакции, посредством функций обратного вызова, передаются через модельное окружение в сопоставитель реакций. Если эталонная модель – это программа, описывающая функциональность тестируемой аппаратуры в целом, то модельное окружение необходимо для учета временных характеристик, которые могут и отсутствовать в эталонной модели, а также преобразования форматов сообщений генератора, сопоставителя реакций и эталонной модели, если они различаются. Окружение строится в виде множества параллельных процессов, моделирующих операции, имеющиеся в HDL-описании, и является определяющим в **выносимом на защиту методе спецификации цифровой аппаратуры**. Моделирование времени исполнения процессов делается с помощью специальной конструкции, означающей задержку данного процесса до следующего такта работы модельного окружения: варьируя количество и условия временных задержек можно изменять детальность учета времени для запросов в эталонную модель и получения из нее результатов. На поступающие из генератора стимулы выполняемое в симуляторе HDL-описание отвечает реакциями реализации, которые посредством адаптера поступают в сопоставитель реакций.

Оракул, как уже было замечено ранее, состоит из модельного окружения, сопоставителя и генератора вердикта. Реакции эталонной модели и HDL-описания приводят к одному из трех вариантов работы сопоставителя. Если

соответствующая реакции реализации модельная реакция была найдена, генератор вердикта покажет результат сравнения их данных. Отсутствие соответствующей модельной реакции приведет к фиксации *неожиданной реакции*. Если же время ожидания сопоставления имеющейся модельной реакции и отсутствующей реакции реализации превышает некоторый порог, фиксируется ошибка *пропущенной реакции*. Оракул настраивается на разные уровни проверки временных свойств HDL-описания: *без учета времени*, *с уточненным временем* (известна информация о последовательностях событий) или *потактовую точность*. Возможность использования модельных реакций для сопоставления с реакциями реализации независимо от уровня абстракции эталонной модели достигается за счет **выносимого на защиту метода построения сопоставителя реакций**, который работает в одном из следующих режимов:

1. пытается сопоставлять модельные реакции и реакции реализации на том же такте, когда в сопоставитель приходят модельные реакции (потактовая точность);
2. выстраивает модельные реакции в определенном порядке, ожидает реакции реализации в пределах установленного временного лимита и сопоставляет реакции согласно порядку, в котором находятся модельные реакции (уточненное время);
3. сопоставляет все модельные реакции и все пришедшие на данном такте реакции реализации согласно подмножеству данных реакций либо согласно всем данным реакций и выделяет пары реакций на основании этого сравнения без учета порядка генерации реакций модулем эталонной модели (без учета времени).

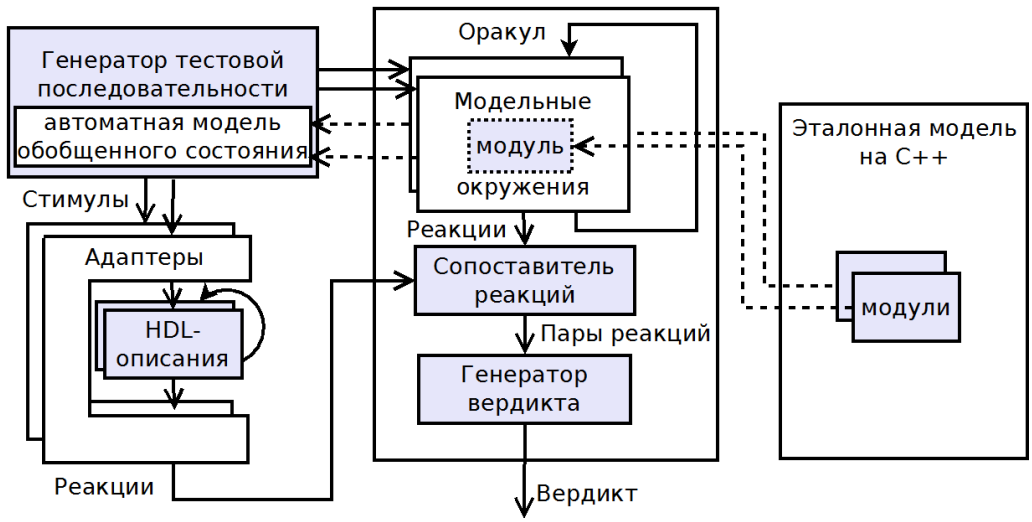
Таким образом, используемая архитектура тестовых систем обеспечивает возможность быстрого начала верификации посредством построения абстрактных тестов на основе абстрактных моделей без учета времени на ранних этапах проектирования и случайных тестовых последовательностей, а затем – доработки этих моделей и их окружений до потактового уровня для тех

модулей, где это необходимо для тщательных проверок. Для той же цели генераторы тестовых последовательностей переключаются в режим генерации стимулов на основе обхода конечных автоматов. Предлагаемая архитектура поддерживает итерационное повышение степени учета временных свойств тестируемой аппаратуры, качества проверок, что необходимо для применения тестовой системы в рамках стандартного процесса итерационной разработки HDL-описаний аппаратуры.

Следующая последовательность действий инженера, проводящего верификацию, также ориентирована на работу со сложными HDL-описаниями:

1. декомпозиция HDL-описаний на слабо связанные с функциональной точки зрения части;
2. создание отдельных тестовых систем для этих частей;
3. композиция полученных тестовых систем и HDL-описаний.

Архитектура тестовых систем для отдельных модулей, была разработана с учетом поддержки композиции тестовых систем, в основном благодаря методу построения модельных окружений. При объединении тестовых систем (см. рисунок 2) необходимо создать объединенный оракул, адаптер и генератор тестовой последовательности.



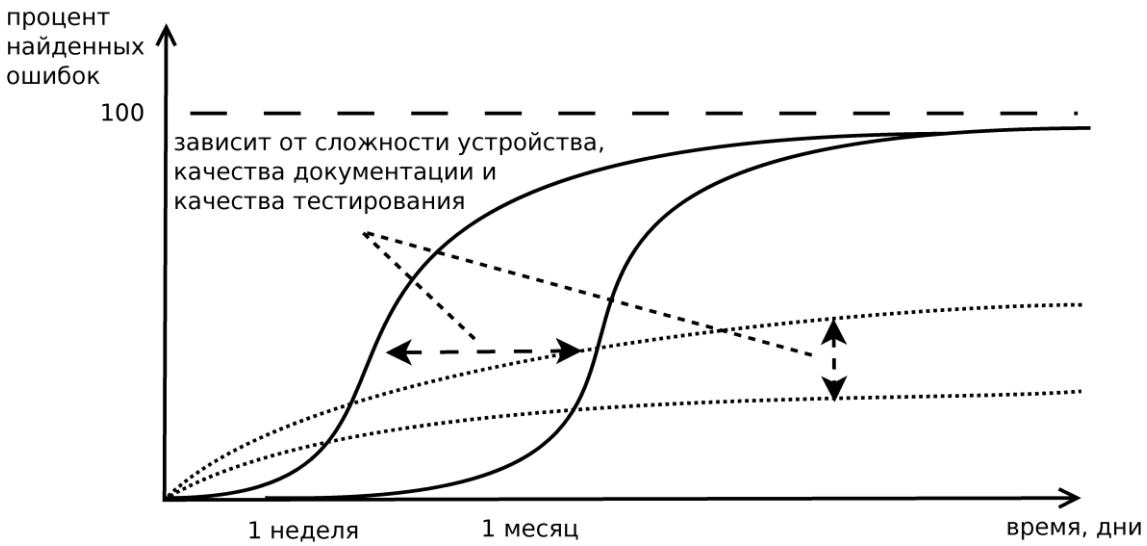
Серый цвет – *автоматически создаваемые и сторонние компоненты*

Без заполнения – *создаваемые с использованием библиотечных средств компоненты*

Рисунок 2 – Объединенные тестовые системы

Объединение оракулов заключается в объединении модулей эталонных моделей, а сопоставитель реакций и генератор вердикта становятся общими компонентами для всех объединяемых тестовых систем. Объединение модулей эталонных моделей, помещенных в их окружение, строится так: поскольку для соединения HDL-описаний некоторые их входные и выходные сигналы необходимо замкнуть между собой для передачи данных без участия тестовой системы, в результате они становятся недоступными для нее, модули эталонных моделей как раз должны повторять такое объединение. В частности, модельные реакции, соответствующие замкнутым сигналам, вместо отправки в сопоставитель, отправляются на соседние модули эталонных моделей через модельные окружения. Объединенный адаптер получается простой композицией, однако, он отключается от недоступных линий и не передает на них стимулы от генератора. Объединенный генератор, в случае его ориентации на случайную тестовую последовательность, также получается композицией прежних генераторов, а в случае создания тестовой последовательности на основе обхода конечных автоматов – требует ручного изменения структуры текущего состояния, так как простая композиция может легко привести к комбинаторному взрыву числа состояний.

Предлагаемый метод позволяет варьировать время, затрачиваемое на верификацию с учетом уровня качества, как это показано на рисунке 3: высокий уровень качества тестирования требует потактовой точности эталонной модели и, следовательно, увеличивает время начала собственно тестирования и нахождения ошибок. Первоначальное тестирование относительно низкого качества, проводимое с использованием неполных спецификаций, позволяет быстрее начать верификацию, быстрее найти более простые ошибки (а их количество обычно превосходит 50%), а затем, для нахождения сложных ошибок, если это необходимо, расширять спецификации требований.



сплошная линия – предлагаемый метод

пунктирная линия – традиционный метод

Рисунок 3 – Зависимость качества тестирования и времени его проведения

При сравнении с традиционным методом разработки тестов, когда вручную разрабатываются ограничения на тестовые последовательности и утверждения, которым должно удовлетворять поведение выполняемого в симуляторе HDL-описания, можно выделить следующие особенности, которые проявляются на практике. Традиционный метод позволяет начать находить ошибки быстрее, но затем скорость нахождения ошибок снижается. Вид графика обнаружения ошибок методом, основанным на использовании эталонной модели, следующий: сначала идет пологий участок, когда еще нет тестовой системы, затем быстрый рост и стабилизация, когда необходимо увеличивать точность модели для нахождения сложных ошибок.

Далее в главе вводится формальная модель *последовательностей стимулов*, *реакций* и отношения *соответствия* между поведением реализации и спецификации и доказывается теорема о значимости тестового набора при использовании предлагаемого тестового оракула.

Все входные и выходные сигналы HDL-описания (*реализации*) разбиваются на *входные* и *выходные интерфейсы*. Множество входных и выходных интерфейсов эталонной модели (*спецификации*) совпадают с

множеством интерфейсов реализации (In и Out). Алфавиты *стимулов* и *реакций* реализации и спецификации также совпадают (X и Y). Множества *состояний* реализации (S_{impl}) и спецификации (S_{spec}), вообще говоря, могут различаться, но и в реализации и в спецификации выделены *начальные состояния* ($s_{impl_0} \in S_{impl}$ и $s_{spec_0} \in S_{spec}$).

Стимулы, подаваемые в процессе тестирования на входной интерфейс $in \in In$, – это последовательность $\bar{X}^{in} = \langle (x_i, t_i) \rangle_{i=1}^{n}$, где $x_i \in X$ – единичный стимул, $t_i \in \mathbb{N}_0$ – время его подачи ($t_i < t_{i+1}, i = \overline{1,n-1}$). Совокупность последовательностей стимулов, подаваемых в процессе тестирования на все входные интерфейсы, будем обозначать $\bar{X} =< \bar{X}^{in_1}, \dots, \bar{X}^{in_N} >$ и называть *последовательностью стимулов*. Допустимость последовательности стимулов в начальном состоянии задается областью определения $Dom \subseteq \bigcup_{k=0}^{\infty} (X \times \mathbb{N}_0)^k$.

Реализация в ответ на последовательность стимулов $\bar{X}$ выдает на выходной интерфейс $out \in Out$ реакции $\bar{Y}_{impl}^{out}(\bar{X}) = \langle (y_i', t_i') \rangle_{i=1}^{m}$, где $y_i' \in Y$ – единичная реакция, $t_i' \in \mathbb{N}_0$ - время ее выдачи ($t_i' < t_{i+1}', i = \overline{1,m-1}$). Совокупность последовательностей реакций, выдаваемых реализацией на все выходные интерфейсы, будем обозначать $\bar{Y}_{impl} =< \bar{Y}_{impl}^{out_1}, \dots, \bar{Y}_{impl}^{out_M} >$ и называть *последовательностью реакций реализации*.

Спецификация в ответ на последовательность стимулов $\bar{X}$ выдает на выходной интерфейс $out \in Out$ реакции $\bar{Y}_{spec}^{out}(\bar{X}) = \langle (y_i, t_i) \rangle_{i=1}^{k}$, где $y_i \in Y$ – единичная реакция, $t_i \in \mathbb{N}_0$ - время ее выдачи ($t_i \leq t_{i+1}, i = \overline{1,k-1}$). Совокупность последовательностей реакций, выдаваемых спецификацией на

все выходные интерфейсы, будем обозначать $\bar{Y}_{spec} = < \bar{Y}_{spec}^{out_1}, \dots, \bar{Y}_{spec}^{out_N} >$ и называть *последовательностью реакций спецификации*.

Будем считать, что для каждого выходного интерфейса $out \in Out$ задана *максимально допустимая задержка выдачи реакций* $\Delta t^{out} \in \mathbb{N}_0$. Также будем считать, что для любой конечной последовательности стимулов существует конечная последовательность реакций. Будем обозначать *единичный элемент последовательности реакций* $\bar{Y} = \langle (y_i, t_i) \rangle_{i=1}^{k}$ как $\bar{Y}[i] = (y_i, t_i)$. Определим операцию *исключения элемента из последовательности реакций* $\bar{Y} \backslash (y, t)$: если в исходной последовательности исключаемый элемент отсутствует, результатом является эта последовательность, если исключаемый элемент присутствует, будет исключено его первое вхождение в последовательность. Обозначим длину последовательности реакций символом $m = |\bar{Y}|$.

Определение 1. Говорят, что реализация *соответствует* спецификации, если $\forall out \in Out$ и $\forall \bar{X} \in Dom$ выполняется $\left|\bar{Y}_{impl}^{out}(\bar{X})\right| = \left|\bar{Y}_{spec}^{out}(\bar{X})\right| = m^{out}$ и существует перестановка π^{out} множества $\{1, \dots, m^{out}\}$, такая что $\forall i \in \{1, \dots, m^{out}\}$ выполняется $\begin{Bmatrix} y_i' = y_j \\ t_j \le t_i' \le t_j + \Delta t^{out} \end{Bmatrix}$, где $j = \pi^{out}(i)$.

Определение 2. Говорят, что в поведении реализации *наблюдается ошибка* (*failure*), если реализация не соответствует спецификации, иными словами $\exists \bar{X} \in Dom$ и $\exists out \in Out$, такие что либо $\left|\bar{Y}_{impl}^{out}(\bar{X})\right| \neq \left|\bar{Y}_{spec}^{out}(\bar{X})\right|$, либо для любой перестановки π^{out} множества $\{1, \dots, m^{out}\}$ $\exists i \in \{1, \dots, m^{out}\}$, для которого выполняется $\begin{bmatrix} y_i' \neq y_j \\ t_j > t_i' \\ t_i' > t_j + \Delta t^{out} \end{bmatrix}$, где $j = \pi^{out}(i)$.

В случае наличия ошибки (для некоторого фиксированного соответствия реакций реализации и спецификации) используют следующую терминологию:

если $y' \neq y$, то говорят о некорректной реакции реализации; если $t > t'$, то говорят о неожиданной реакции реализации; если $t' > t + \Delta t^{out}$, то говорят о пропущенной реакции реализации.

Далее описывается алгоритм работы оракула, который работает по каждому выходному интерфейсу независимо, принимает на вход последовательность реакций реализации $\bar{Y}_{impl} = \langle (y_i', t_i') \rangle_{i=1}^{m}$ и спецификации $\bar{Y}_{spec} = \langle (y_i, t_i) \rangle_{i=1}^{k}$, полученные в ответ на одну и ту же конечную последовательность стимулов $\bar{X}$. Оракул устанавливает соответствие между последовательностями и выдает вердикт о наличии ошибки. Далее доказывается следующая теорема.

Теорема. Тестовый оракул, работающий согласно предложенному алгоритму, обеспечивает построение значимых тестовых наборов (то есть при обнаружении ошибки у оракула не бывает ложных срабатываний).

В **третьей главе** описывается реализация предложенного метода верификации с помощью библиотеки классов на языке C++ для инструмента верификации C++TESK, реализующего технологию UniTESK для языка C++. Библиотека получила название C++TESK Hardware Edition.

В **четвертой главе** приводятся результаты применения предложенного метода на практике. Так, была проведена верификация ряда модулей микропроцессоров, разрабатываемых в НИИ системных исследований РАН и ЗАО «МЦСТ». Трудозатраты на верификацию четырех из девяти модулей приведены в таблице 1 (для одного из модулей рассмотрены случаи верификации на разных стадиях проектирования). В таблице 2 приведена информация об общем количестве найденных ошибок в каждом HDL-описании и о времени нахождения первой ошибки (что позволяет судить о возможности раннего начала верификации). Можно заметить, что время нахождения первой ошибки зависит от стадии разработки, на которой находится модуль, и изменяется от порядка 60% от общего времени верификации (на поздних

этапах), до 17% для модулей, находящихся на средних этапах процесса проектирования.

Таблица 1 – Данные по применению метода верификации в 5 проектах

Название модуля	Стадия разработки	Трудозатраты и длительность верификации	Объем реализации / тестовой системы
1. Коммутатор процессор-память (1)	Завершающая	12 чел.-недель / 8 недель	5/9 KLOC
2. Коммутатор процессор-память (2)	Поздняя	16 чел.-недель / 16 недель	7/5 KLOC
3. Устройство аппаратного поиска по таблице страниц	Поздняя	48 чел.-недель / 36 недель	8/10 KLOC
4. Буфер команд	Поздняя	32 чел.-недель / 12 недель	10/6 KLOC
5. Системный контроллер прерываний	Средняя	24 чел.-недель / 12 недель	2/7.5 KLOC

Таблица 2 – Данные по найденным ошибкам

Название модуля	Стадия разработки	Первая ошибка (процент от общего времени верификации)	Найденные ошибки
1. Коммутатор процессор-память (1)	Завершающая	5 неделя (63%)	2
2. Коммутатор процессор-память (2)	Поздняя	4 неделя (25%)	23
3. Устройство аппаратного поиска по таблице страниц	Поздняя	10 неделя (28%)	25
4. Буфер команд	Поздняя	4 неделя (33%)	6
5. Системный контроллер прерываний	Средняя	2 неделя (17%)	9

График зависимости времени проведения верификации и относительного количества найденных ошибок для 1, 2, 4 и 5 проектов из таблиц 1-2 представлен на рисунке 4, такой же график для проекта 3 из таблиц 1-2 представлен на рисунке 5. При составлении графиков за 100% найденных ошибок бралось общее количество ошибок, найденных в процессе верификации.

Рисунок 4 – Нахождение ошибок в процессе верификации в проектах средней сложности

Рисунок 5 – Нахождение ошибок в процессе верификации в проекте высокой сложности

На рисунках 4-5 наглядно демонстрируется возможность быстрого начала верификации (как правило, на 2-4 неделе уже начинают находиться ошибки), показывается итеративность процессов проектирования.

В главе также представлены результаты использования композиции трех модулей эталонной модели, которая потребовалась для удобного моделирования процессов, происходящих внутри тестируемой реализации. Соединяемые модули были неравноценны: объем основного модуля эталонной модели был порядка 5000 строк кода, двух подключаемых к нему – порядка 300 строк каждый. Композиция заняла пренебрежительно малое время и потребовала незначительное количество строк дополнительного кода, в то время как при использовании ручного включения модулей, потребовалось бы фактически разработать эти два модуля заново уже в рамках основного модуля для их аккуратного объединения. Таким образом, если временные затраты на ручную композицию примерно равны затратам на разработку исходных модулей эталонных моделей, использование предложенной архитектуры построения тестовых систем позволяет существенно сократить трудоемкость композиции эталонных моделей.

В **заключении** перечисляются основные результаты работы.

Основные результаты работы

Основные научные и практические результаты, полученные в диссертационной работе и выносимые на защиту, состоят в следующем:

1. Разработан метод верификации цифровой аппаратуры, на основе формальных спецификаций в виде программных моделей на языке C++, отвечающий требованиям промышленных процессов проектирования HDL-описаний;
2. Разработан метод спецификации цифровой аппаратуры, подходящий для использования на разных уровнях абстракции;
3. Разработан метод сопоставления реакций цифровой аппаратуры и реакций эталонной модели, позволяющий автоматизировать процедуру проверки цифровой аппаратуры;
4. Реализованы инструменты, поддерживающие разработанные методы.

Предложенный метод автоматизации динамической верификации цифровой аппаратуры отвечает поставленным в работе целям. Разработанные инструменты успешно использованы в нескольких проектах по верификации сложных отечественных микропроцессоров.

Работы автора по теме диссертации

1. Камкин А.С., Чупилко М.М.. Механизмы поддержки функционального тестирования моделей аппаратуры на разных уровнях абстракции. // Труды ИСП РАН, т. 20, М., 2011, ISSN 2079-8156. с. 143-160.
2. Камкин А., Чупилко М. Обзор современных технологий имитационной верификации аппаратуры. // Программирование, № 3, 2011. с. 42–49.
3. Чупилко М. Разработка тестовых систем для многомодульных моделей аппаратуры. // Программирование, № 1, 2012, с. 47-58.

4. Chupilko M. C++TESK-SystemVerilog united approach to simulation-based verification of hardware designs. // Proceedings of EWDTS 2011: The 9th East-West Design & Test Symposium, 2011. pp. 136–139.
5. Chupilko M. Developing test systems for multi-modules hardware designs. // Proceedings of SYRCoSE 2011: The 5th Spring Young Researchers Colloquium on Software Engineering, 2011. pp. 111–116.
6. Kamkin A., Chupilko M. A TLM-based approach to functional verification of hardware components at different abstraction levels. // Proceedings of LATW 2011: The 12th Latin-American Test Workshop, 2011. pp. 1-6.
7. Chupilko M., Kamkin A. Developing cycle-accurate contract specifications for synchronous parallel-pipeline hardware: application to verification. // Proceedings of BEC 2010: The 12th Biennial Baltic Electronics Conference, 2010. pp. 185-188.
8. Chupilko M. Models of Synchronous Hardware Designs Based on FSM at Different Abstraction Levels: Application to Functional Verification. // Proceedings of EWDTS 2010: The 8th East-West Design & Test Symposium, 2010. pp. 487–490.
9. Chupilko M., Kamkin A. Contract Specification of Hardware Designs at Different Abstraction Levels: Application to Functional Verification. // Proceedings of SYRCoSE 2010: The 4th Spring Young Researchers Colloquium on Software Engineering, 2010. pp. 125–129.
10. Чупилко М.М. Автоматизация системного тестирования моделей аппаратуры на основе формальных спецификаций. // Труды ИСП РАН, т. 18, М., 2010. с. 115–128.
11. Chupilko M., Kamkin A. Specification-Driven Testbench Development for Synchronous Parallel-Pipeline Designs. // Proceedings of NorChip 2009: 27th Norchip Conference, 2009. pp. 1-4.