Механизмы поддержки функционального тестирования моделей аппаратуры на разных уровнях абстракции

А.С. Камкин, М.М. Чупилко
{kamkin,chupilko}@ispras.ru

Аннотация. На разных этапах проектирования аппаратуры используются разные представления целевой системы (общее описание архитектуры в начале проекта шаг за шагом конкретизируется вплоть до разработки топологии интегральной схемы на кристалле). Соответственно, в зависимости от зрелости проекта применяются разные методы верификации, в частности, разные методы построения эталонных моделей, используемых для оценки корректности проектируемой аппаратуры (в начале разработки используются абстрактные модели, но ближе к завершению, для повышения качества проверки, точность моделей повышается). Различие в уровнях абстракции усложняет переиспользование тестовых систем, созданных в начале проектирования, для верификации аналогичных компонентов, но на более поздних этапах. В статье предлагается подход к построению эталонных моделей аппаратуры и тестовых оракулов на их основе, который упрощает повторное использование тестовых систем, тем самым снижая затраты на верификацию.

Ключевые слова: проектирование аппаратуры, функциональное тестирование, моделирование на уровне транзакций

1. Введение

Системы аппаратуры постоянно усложняются, требуя все больше ресурсов для проектирования и верификации. Двумя основными способами преодоления сложности являются *декомпозиция* и *абстракция*. В общих словах, декомпозиция – это разделение системы на множество слабо связанных компонентов, в то время как абстракция (или абстрагирование) – это выявление существенных свойств системы относительно некоторого интересующего нас аспекта. Проектирование аппаратуры можно представить как эволюционный процесс декомпозиции системы и ее уточнения (понижения уровня абстракции), управляемый целевыми требованиями и имеющимися ресурсами.

Для того чтобы проверить соответствие результата проектирования требованиям, используется *верификация*. Часто верификация рассматривается

как завершающий этап проектирования. Однако, это не совсем так. Верификация тесно интегрирована в процесс разработки и осуществляется на всех без исключения этапах, обеспечивая своевременную обратную связь и делая проектирование более контролируемым и предсказуемым. В соответствии с вышесказанным, верификация применяется как для систем в целом, так и для отдельных компонентов. Кроме того, на разных этапах проектирования для верификации используются разные типы *эталонных моделей* (*reference models*)[1] – более абстрактные на ранних этапах, более точные в конце.

В статье рассматривается верификация *отдельных компонентов* аппаратуры. При этом предполагается, что состав и общая функциональность компонентов системы в процессе проектирования не меняются. Решаемая в работе проблема заключается в определении архитектуры *тестовой системы* (*testbench* [1]), которая, с одной стороны, является *универсальной* для разных уровней абстракции эталонных моделей, а, с другой стороны, обеспечивает *повторное использование* тестовых систем при эволюционном развитии проекта (иными словами, позволяет варьировать уровень абстракции проверяемых свойств). Если говорить более точно, в статье рассматривается архитектура не всей тестовой системы, а лишь ее части, связанной с автоматической проверкой корректности реакций целевого компонента в ответ на подаваемые стимулы (эта часть обычно называется *тестовым оракулом* или просто *оракулом* [2]).

Предлагаемый подход к построению тестовых оракулов (включающих в свой состав и эталонные модели) основан на так называемом *моделировании на уровне транзакций* (*TLM*, *Transaction Level Modeling*). Основной чертой TLM является отделение *коммуникаций* (деталей передачи данных между компонентами) от *вычислений* (функциональных преобразований данных) [3]. Связи между компонентами моделируются программно с помощью *каналов*, а *транзакции* (единичные пересылки данных) осуществляются посредством вызова интерфейсных функций каналов. Описанная в статье архитектура тестовых оракулов была апробирована в ряде промышленных проектов, где показала свою универсальность и гибкость.

Статья организована следующим образом. В разделе 2 делается обзор подходов к построению тестовых оракулов для моделей аппаратуры. Раздел 3 описывает основные понятия TLM и определяет основные уровни абстракции,

[1] Под *эталонной моделью* понимается формальным образом представленные требования, на соответствие которым проверяется результат проектирования. Как правило, эталонная модель – это программа на языке программирования общего назначения, эмулирующая работу целевой системы или ее компонента, а результат проектирования – описание (модель) схемы (системы или компонента) на специализированном языке (HDL, Hardware Description Language), например, Verilog или VHDL.

используемые при проектировании аппаратуры. В разделе 4 предлагается подход к организации тестовых оракулов и эталонных моделей. Раздел 5 описывает частные случаи предлагаемого подхода для разных уровней абстракции. В разделе 6 дается классификация ошибок в аппаратуре, основанная на рассмотренной схеме работы тестового оракула. Раздел 7 содержит обобщенное описание нашего опыта разработки тестовых систем для моделей аппаратуры. В разделе 8 делается заключение, и указываются направления дальнейших исследований.

2. Обзор подходов к построению тестовых оракулов

Существует два основных класса подходов к верификации – *формальные методы* (*formal methods*) и *методы тестирования*, основанные на имитационном моделировании (*simulation-based methods*) [4]. Известно, что формальные методы являются исчерпывающими (в некотором смысле), но они плохо масштабируются на сложные системы. Как правило, такие методы применяются для сравнительно простых компонентов, причем на достаточно поздних стадиях проектирования, когда требования достаточно стабильны. Методы тестирования не являются исчерпывающими, но они гораздо более гибкие и, следовательно, могут быть использованы на разных этапах проектирования.

Для автоматизации тестирования используют специализированные программы, называемые *тестовыми системами*. Типичная тестовая система включает в себя три базовых компонента: (1) *генератор стимулов*, (2) *тестовый оракул* и (3) *сборщик покрытия*. Генератор стимулов создает последовательность тестовых воздействий (стимулов), подаваемых на входы тестируемого компонента. Тестовый оракул оценивает корректность реакций, выдаваемых компонентом в ответ на поданные стимулы. Сборщик покрытия отслеживает достигнутый уровень тестового покрытия. Объектом исследования данной статьи являются тестовые оракулы. Существуют три основных метода построения оракулов: (1) *тесты со встроенными проверками* (*self-checking tests*), (2) *утверждения* (*assertions*) и (3) *ко-симуляция* (*co-simulation*).

Тесты со встроенными проверками – достаточно старый метод проверки реакций. Каждый стимул (точнее, тестовый пример) снабжается кодом, осуществляющим проверку выдаваемого компонентом результата на корректность [5]. Подход имеет очевидные неудобства. Во-первых, написать процедуру проверки реакций компонента для сложного теста достаточно трудно, а число тестов, как правило, велико. Во-вторых, тестовые примеры требуют постоянной поддержки, чтобы быть согласованными с изменениями в проекте (из-за большого объема тестов сопровождение может потребовать значительных ресурсов). В-третьих, для этого подхода характерна неполнота выполняемых проверок – каждый тестовый пример нацелен на достижение

определенной ситуации и обычно проверяет лишь те аспекты поведения тестируемого компонента, которые существенны для этой ситуации.

Утверждениями называются предикаты на поведение компонента, которые должны быть выполнены [6]. При таком подходе проверки отделяются от стимулов и пишутся либо в коде тестируемого компонента, либо отдельно. Это позволяет использовать для верификации автоматизированные генераторы стимулов, что является ключевым преимуществом подхода по сравнению с тестами со встроенными проверками. Как правило, утверждения описывают наиболее важные или наиболее очевидные свойства компонента. Тем самым, проверку на основе утверждений нельзя назвать полной. Следует отметить, что при использовании утверждений, встроенных в код, невозможно разработать тестовый оракул до того, как компонент будет полностью описан.

Ко-симуляция – это подход к проверке реакций, в котором вместе с моделью тестируемого компонента используется независимо разработанная эталонная модель [5]. На две модели подаются одинаковые тестовые последовательности, а результаты их работы сравниваются на соответствие. При расхождении результатов определяется, какая из моделей некорректна, ошибочная модель исправляется, после чего тестирование продолжается. Использование эталонной модели позволяет генерировать стимулы автоматически. Однако, создание программной модели, эмулирующей работу тестируемого компонента на всех допустимых тестовых последовательностях, является сложной задачей, которая в некоторых случаях равносильна повторному проектированию компонента.

Проанализируем подходы, описанные выше. Тесты со встроенными проверками не обеспечивают высокого уровня автоматизации тестирования и страдают от неполноты проверки реакций. Кроме того, их сложно сопровождать при наличии частых изменений в проекте. Утверждения являются идеальным решением для проверки небольшого числа свойств, но они не подходят для исчерпывающего тестирования сложных компонентов аппаратуры. Ко-симуляция выглядит наиболее многообещающим подходом, но разработка эталонной модели может потребовать много ресурсов. Для упрощения разработки и сопровождения эталонных моделей необходима специальная методология. На наш взгляд, за основу подобной методологии можно взять TLM.

Схожие идеи лежат в основе методологии верификации OVM (Open Verification Methodology), широко известном и применяемом на практике подходе [7, 8]. Согласно OVM, тестовая система должна быть разделена на несколько функциональных компонентов, интерфейсы между которыми должны быть определены с помощью TLM. Эта методология описывает общую архитектуру тестовой системы, но ничего не говорит о внутренней организации компонентов, в частности, не отвечает на вопрос, как генерировать тестовые последовательности и как проверять поведения сложной аппаратуры.

3. TLM и различные уровни абстракции

TLM – это подход к программному моделированию аппаратуры, в котором описание коммуникаций между компонентами отделено от их функциональности [3]. Механизмы коммуникации (такие как шины и буферы) моделируются каналами, инкапсулирующими низкоуровневые детали передачи данных. Транзакции (то есть пересылки данных) осуществляются путем вызова интерфейсных функций этих каналов. В общих словах, TLM фокусируется на функциональности передачи данных, а не ее действительной реализации. При использовании этого подхода легко экспериментировать с различными реализациями коммуникационных шин без необходимости изменения взаимодействующих компонентов.

В процессе проектирования аппаратуры используется множество промежуточных моделей, которые, с одной стороны, являются объектами верификации, а, с другой стороны, могут быть использованы в качестве эталонных моделей при последующей верификации. Для классификации уровней абстракции проектных моделей можно использовать *граф системного моделирования* (*system modeling graph*), показанный на Рис. 1 [3]. Ось X соответствует точности моделирования времени в вычислениях (функциональных преобразованиях данных), а ось Y – точности моделирования времени в коммуникациях (пересылках данных). На каждой оси отмечены три уровня абстракции: *без учета времени*, *с учетом времени* и *с потактовой точностью*.

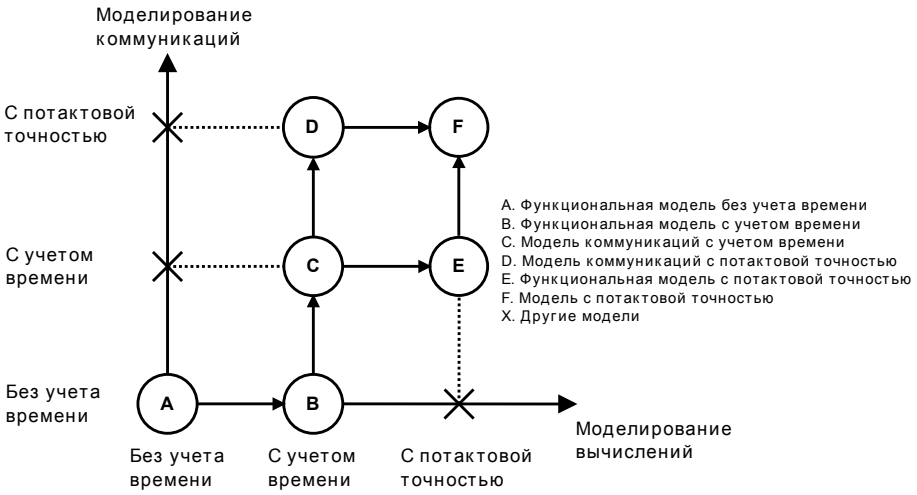

Рис. 1. Граф системного моделирования

В соответствии с [3], существуют 6 основных уровней абстракции: (A) *функциональные модели без учета времени*, (B) *функциональные модели с учетом времени*, (C) *модели коммуникаций с учетом времени*, (D) *модели коммуникаций с потактовой точностью*, (E) *функциональные модели с потактовой точностью* и (F) *модели с потактовой точностью* (терминология немного отличается от используемой в [3]). В таблице 1 приведены характеристики приведенных уровней абстракции.

Название уровня абстракции	**Моделирование времени в коммуникациях**	**Моделирование времени в вычислениях**	**Используемая схема коммуникации**
Функциональная модель без учета времени (A)	Отсутствует	Отсутствует	Общие данные
Функциональная модель с учетом времени (B)	Отсутствует	Приблизительное	Каналы передачи сообщений
Модель коммуникаций с учетом времени (C)	Приблизительное	Приблизительное	Абстрактная шина (арбитр)
Модель коммуникации с потактовой точностью (D)	Точное	Приблизительное	Уточненная шина (протокол)
Функциональная модель с потактовой точностью (E)	Приблизительное	Точное	Абстрактная шина (арбитр)
Модель с потактовой точностью (F)	Точное	Точное	Сигналы

Таблица 1. Характеристики основных уровней абстракции

4. Архитектура тестового оракула

В данной статье рассматривается моделирование и верификация отдельных компонентов аппаратуры. Это сделано намеренно для того, чтобы абстрагироваться от коммуникаций внутри тестируемого устройства. Говоря о разработке тестовой системы (точнее, об организации тестового оракула),

основной интерес представляют внешние интерфейсы (через которые осуществляется взаимодействие с тестовой системой). Обобщенная структура тестовой системы представлена на Рис 2.

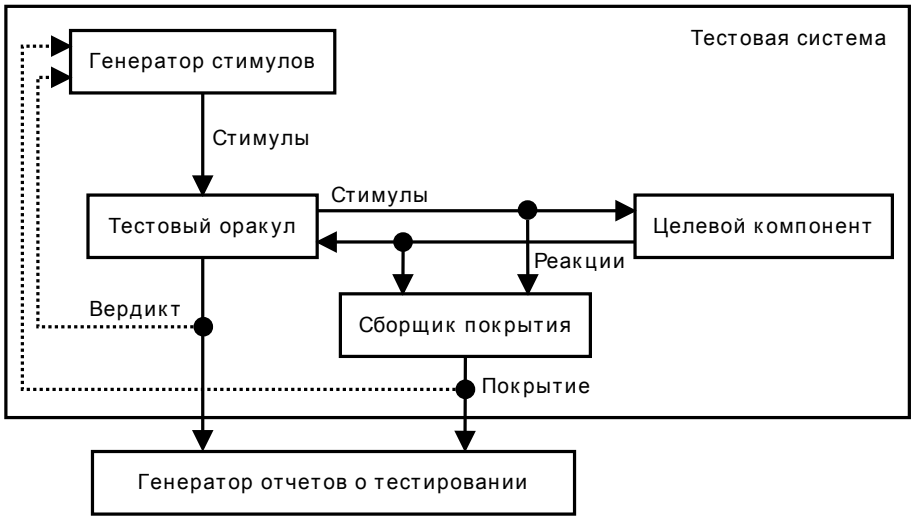

Рис. 2. Обобщенная структура тестовой системы

Генератор стимулов создает последовательность стимулов и передает их тестовому оракулу, который, в свою очередь, переводит их в представление тестируемого компонента и осуществляет их подачу. Реакции тестируемого компонента передаются оракулу, который оценивает их корректность (основываясь на утверждениях или эталонной модели). Стимулы и реакции также подаются сборщику покрытия, который оценивает завершенность тестирования, используя некоторые метрики или эвристики.

Эта схема отражает взаимодействие основных компонентов тестовой системы, не раскрывая их внутреннюю организацию. Между тем, внутренняя архитектура компонентов тестовой системы имеет решающее значение для обеспечения эффективного сопровождения и повторного использования тестов. Следует отметить, что, как правило, самые большие ресурсы, связанные с сопровождением тестовой системы, тратятся на поддержание тестового оракула в согласованном состоянии с целевым компонентом. Ниже перечислены основные требования, сформулированные нами для методов построения тестовых оракулов.

- Возможность использования абстрактной эталонной модели для тестирования компонента аппаратуры на уровне регистровых передач (RTL, Register Transfer Level).

- Простая адаптация тестовой системы к изменениям входных и выходных интерфейсов тестируемого компонента.
- Простая адаптация к изменениям временных свойств тестируемого компонента и возможность уточнения эталонной модели вплоть до потактовой точности.
- Обеспечение настолько точной диагностики ошибок, насколько это возможно для данного уровня абстракции.

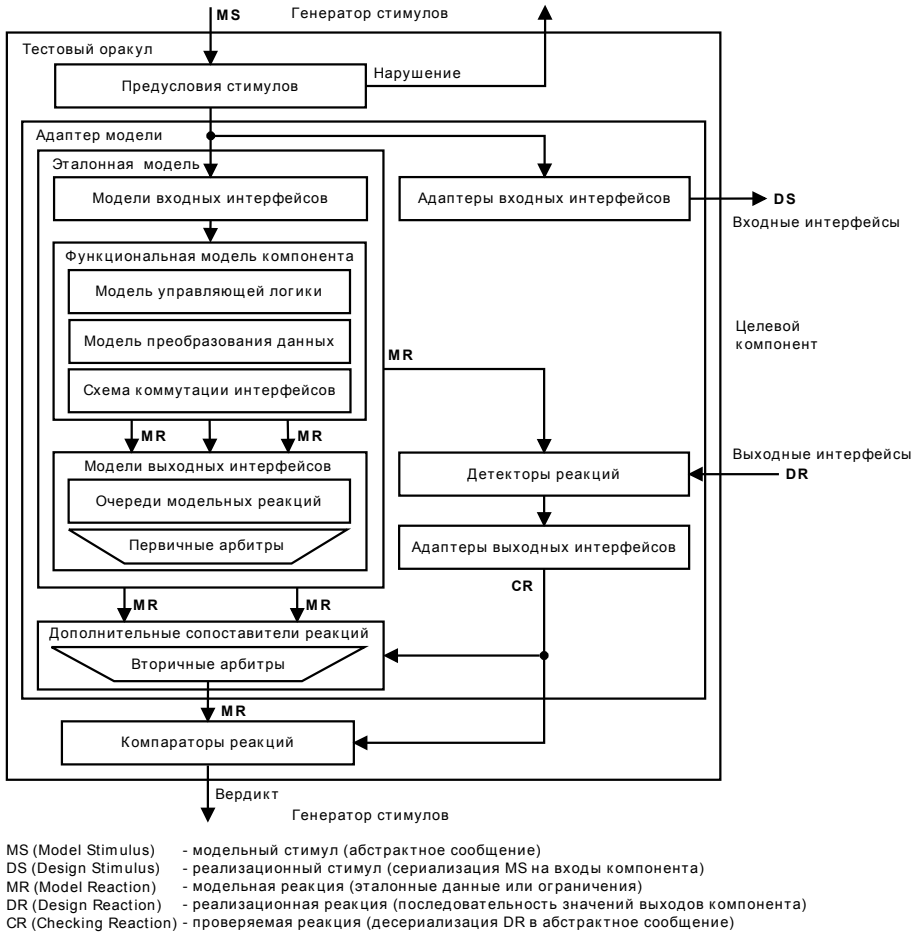

Рис. 3. Архитектура тестового оракула для компонента аппаратуры

На Рис. 3 приведена архитектура тестового оракула, учитывающая сформулированные требования. Оракул содержит *предусловия стимулов*,

компараторы реакций, *эталонную модель* и *адаптер эталонной модели*. Эталонная модель состоит из *моделей входных и выходных интерфейсов* и *функциональной модели*, а адаптер эталонной модели – из *адаптеров входных и выходных интерфейсов*. Для каждого выходного интерфейса имеется *очередь модельных реакций* (часть модели интерфейса) *детектор реакций* (часть адаптера модели) и *арбитры*, предназначенные для сопоставления реакций тестируемого компонента и реакций эталонной модели: *первичный арбитр* (часть эталонной модели) и *вторичный арбитр* (часть адаптера модели). Совокупность первичного и вторичного арбитров для выходного интерфейса называется *сопоставителем реакций* для данного интерфейса. Заметим, что предложенная архитектура основана на TLM – в ней имеется явное разделение коммуникаций (моделей входных и выходных интерфейсов) от вычислений (функциональной модели).

Теперь рассмотрим, как работает тестовый оракул. При получении очередного *модельного стимула* (*MS*, *Model Stimulus* на Рис. 3) от генератора стимулов проверяется соответствующее предусловие. Если предусловие нарушено (например, если нарушен протокол передачи входных данных), оракул фиксирует ошибку в тестовой системе. В противном случае модельный стимул передается в адаптер эталонной модели, который передает его как на эталонную модель, так и на тестируемый компонент. Во втором случае используется адаптер входного интерфейса, который *сериализует* абстрактное сообщение, представляющее модельный стимул, в потактово точную последовательность значений входных сигналов – *реализационный стимул* (*DS*, *Design Stimulus*). Эталонная модель эмулирует обработку стимула на некотором уровне абстракции, вычисляет набор *модельных реакций* (в явном виде или в форме ограничений[2]) (*MR*, *Model Reaction*) и передает их в очереди реакций соответствующих выходных интерфейсов.

Как только на каком-нибудь выходном интерфейсе тестируемого компонента детектор реакций обнаруживает реакцию (*DR*, *Design Reaction*), она с помощью адаптера интерфейса *десериализуется* в модельное представление (*CR*, *Checking Reaction*) – *проверяемая реакция*. После этого сопоставитель реакций пытается найти модельную реакцию из очереди реакций интерфейса, соответствующую проверяемой реакции. Первичный арбитр вычисляет множество реакций-кандидатов, основываясь на данных, предоставляемых эталонной моделью (временем поступления модельной реакции в очередь, ее приоритетом и др.). В некоторых ситуациях он выбирает ровно одну реакцию, но в общем случае, когда эталонная модель является достаточно абстрактной, первичный арбитр возвращает множество реакций (например, когда эталонная модель ожидает несколько реакций на одном выходном интерфейсе, но их

[2] *Ограничения* (в частности, *неопределенные значения*) используются, если требования допускают несколько корректных альтернатив для данных, содержащихся в реакции.

порядок не определен). В таких случаях дополнительно используется вторичный арбитр, который знает, как выглядит проверяемая реакция, и выбирает модельную реакцию из множества кандидатов, которая наиболее близка к проверяемой. Выбранная модельная реакция называется *эталонной реакцией* (с ней сравнивается проверяемая реакция).

Для сопоставления проверяемой реакции с одной из реакций-кандидатов вторичный арбитр использует *подсказку* – специальную функцию, определенную на множестве модельных реакций, возвращающую *идентификатор* сообщения (обычно в качестве подсказки используется поле сообщения, контрольная сумма или сообщение в целом). Пусть CR – это проверяемая реакция, $\mathbf{C} = \{MR_i\}$ – множество реакций-кандидатов, предоставленное первичным арбитром, h – функция подсказки. Если множество **C** пусто, фиксируется ошибка. Если множество **C** содержит ровно один элемент, вторичный арбитр возвращает этот элемент. В остальных случаях он вычисляет $h(CR)$ и пытается найти модельную реакцию $MR \in \mathbf{C}$ такую, что $h(MR) = h(CR)$[3]. Если таких реакций нет, фиксируется ошибка. Если найдена ровно одна реакция, она возвращается. Если найдено несколько реакций (возвращаемый подсказкой идентификатор не является уникальным), вторичный арбитр выбирает одну из них произвольным образом.

Выбранная эталонная реакция удаляется из очереди модельных реакций соответствующего выходного интерфейса и вместе с проверяемой реакцией передается компаратору реакций. В большинстве случаев компаратор проверяет равенство $CR = MR$, однако нередки ситуации, когда выполняемые проверки являются более сложными (например, когда определенность одних полей сообщения зависит от значений других полей). Если сравниваемые реакции не идентичны, фиксируется ошибка. В противном случае тестирование продолжается.

Ключевым в данной схеме являются механизмы сопоставления реакций – именно они позволяют использовать абстрактные модели для верификации компонентов на уровне RTL. Предлагаемый подход легко справляется с изменениями входных и выходных интерфейсов (для этого используются адаптеры интерфейсов). Следует особо отметить, что подход применим для всех уровней абстракции и позволяет уточнять временные свойства тестового оракула вплоть до потактовой точности (см. раздел «*Применение подхода на разных уровнях абстракции*»). Кроме того, алгоритм работы тестового оракула обеспечивает хорошую диагностику ошибок (см. раздел «*Классификация ошибок в аппаратуре*»).

[3] Вместо функции подсказки можно использовать и более общий подход, в котором на множестве модельных реакций вводится метрика ρ, а в качестве эталонной реакции выбирается $argmin_{MR \in \mathbf{C}}\{\rho(MR, CR)\}$.

5. Применение подхода на разных уровнях абстракции

В соответствии с предложенной архитектурой тестового оракула существует два основных параметра, определяющих уровень абстракции эталонной модели (и тестового оракула в целом): (1) *механизм обнаружения реакций* и (2) *механизм сопоставления реакций*. Первый из них принимает решение о том, выдал ли тестируемый компонент реакцию на том или ином выходном интерфейсе. В случае если реакции есть, второй механизм ищет модельную реакцию соответствующую обнаруженной реализационной реакции.

Механизм обнаружения реакций для одного выходного интерфейса может быть описан как булева функция d (*detection*), зависящая от состояния эталонной модели ($S \in \mathbf{MS}$) и выходов устройства, относящихся к данному интерфейсу ($O \in \mathbf{DO}$)[4]:

$$d\colon \mathbf{MS} \times \mathbf{DO} \to \{true,\ false\}.$$

В общих словах, d-функции вычисляются на каждом такте тестирования и, как правило, проверяют значения *управляющих сигналов* (*стробов*). Если некоторая функция возвращает *true*, тестовая система считает, что обнаружена реакция и запускает механизм сопоставления реакций (осуществляет поиск эталонной реакции соответствующей обнаруженной реакции). Механизм сопоставления реакций (включающий первичные и вторичные арбитры) описывается как функция a (*arbitration*), которая возвращает модельную реакцию или специальное значение *failed* (при невозможности сопоставления), в зависимости от состояния эталонной модели ($S \in \mathbf{MS}$) и проверяемой реакции ($CR \in \mathbf{MR}$):

$$a\colon \mathbf{MS} \times \mathbf{MR} \to \mathrm{MR} \cup \{failed\}.$$

Используя формализм d- и a-функций можно определить основные уровни абстракции, выделенные выше. В *моделях с потактовой точностью* для обнаружения реакций не используются выходы тестируемого компонента. Эталонная модель сама определяет моменты времени, когда на выходах устройства появляются реакции. Иными словами, d-функции выглядят следующим образом:

$$d\colon \mathbf{MS} \to \{true,\ false\}.$$

Другой отличительной чертой моделей с потактовой точностью является то, что эти модели не используют вторичные арбитры. Первичные арбитры (которые являются частью модели) всегда могут выбрать ровно одну реакцию

[4] Обычно используется подход на основе «черного ящика», поэтому внутреннее состояние тестируемого компонента здесь не рассматривается.

для того, чтобы проверить корректность обнаруженной реакции (точнее, очереди модельных реакций всех выходных интерфейсов содержат не более одной реакции, причем, как только модельная реакция заносится в очередь, сразу делается запрос на получения реакции от тестируемого компонента). Это свойство означает, что a-функции не зависят от обнаруженных реакций и никогда не возвращают *failed*:

$$a\colon \mathbf{MS} \to \mathbf{MR}.$$

На Рис. 4 показан фрагмент тестового оракула для потактово точной эталонной модели. Можно видеть, что детекторы реакций и вторичные арбитры отсутствуют.

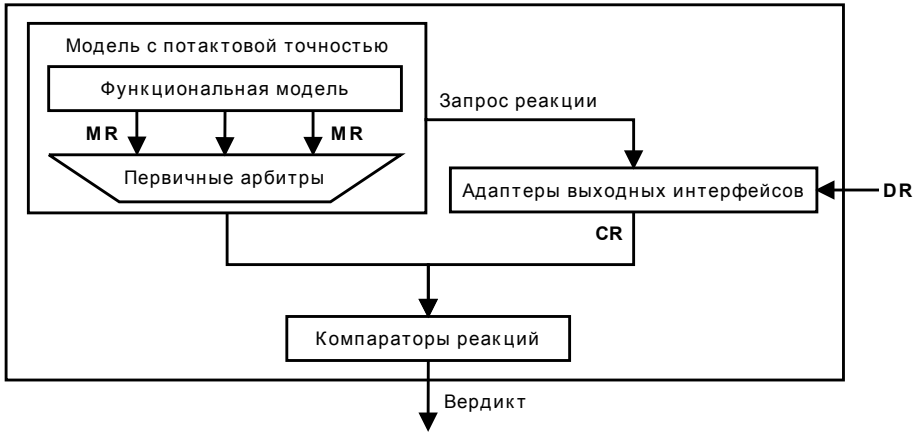

Рис. 4. Тестовый оракул для эталонной модели с потактовой точностью

В противоположность моделям с потактовой точностью, *модели без учета времени* не описывают временны*е* свойства компонентов аппаратуры. В частности, это означает, что у них нет первичных арбитров и, следовательно, вся работа по сопоставлению реакций выполняется вторичными арбитрами. Чтобы определить моменты времени, в которые возникают реакции, тестовые оракулы, основанные на таких моделях (и на моделях с приближенным моделированием времени) используют детекторы реакций. d-функции зависят только от выходов тестируемого устройства:

$$d\colon \mathbf{DO} \to \{true,\ false\}.$$

Тестовый оракул для моделей без учета времени представлен на Рис. 5. Можно видеть, что первичные арбитры отсутствуют, а все модельные реакции поступают напрямую к вторичным арбитрам.

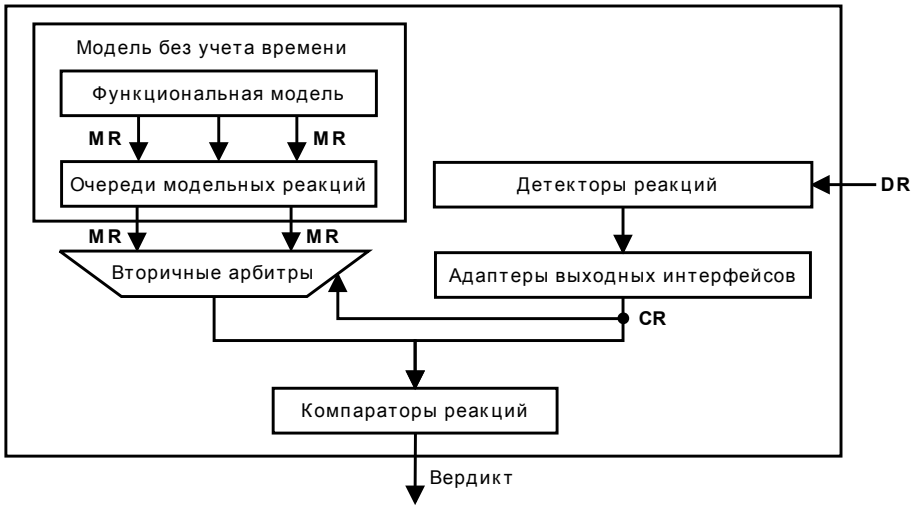

Рис. 5. Тестовый оракул для эталонных моделей без учета времени

Эталонные модели других уровней абстракции (*функциональных моделей с учетом времени*, *моделей коммуникации с учетом времени* и др.) находятся где-то посередине между моделями с потактовой точностью и моделями без учета времени. Поскольку они не являются достаточно точными в плане временны*х* свойств, они используют детекторы реакций, но в отличие от моделей без учета времени, в них применяются первичные арбитры. Разница между моделями промежуточных уровней в основном связана с различной точностью первичных арбитров.

Существует два основных класса арбитров: *детерминированные* (возвращающие не более одной реакции из очереди модельных реакций) и *недетерминированные* (которые в некоторых ситуациях возвращают несколько реакций). Детерминированные арбитры (в отличие от недетерминированных) задают линейный порядок модельных реакций для соответствующих выходных интерфейсов. Если степень недетерминизма ограничена (в некотором смысле), можно рассматривать промежуточный класс арбитров, определяющих приблизительный порядок модельных реакций.

Скажем несколько слов о переходах между уровнями абстракции. В начале процесса проектирования, когда требования существенно неполны, обычно используются функциональные модели без учета времени (A на Рис. 1). При уточнении требований эталонные модели естественным образом детализируются. Главная роль в повторном использовании тестовых систем отводится имеющейся в них модели коммуникаций (в частности, первичным арбитрам). В самом деле, коммуникации между компонентами обычно более подвержены изменениям, нежели вычисления.

В большинстве случаев эталонных моделей с приближенным учетом времени (B и C на Рис. 1) достаточно для достижения высокого качества тестирования. В некоторых случаях (например, для критичных компонентов или компонентов со сложной управляющей логикой) требуются модели с потактовой точностью (D, E, F) [9, 10]. Следует отметить, что уточнение эталонных моделей с приближенным учетом времени до потактовой точности требует значительных усилий (в этом случае необходимо учесть многие нюансы синхронизации, реализованные в тестируемом компоненте). Однако, это необходимо для ограниченного числа компонентов и только на завершающих этапах проектирования. В других случаях уточнение времени обычно связано с более точной настройкой первичных арбитров.

Рассмотрим фрагмент метода, являющегося частью эталонной модели, который описывает некоторую операцию некоторого компонента аппаратуры:

```
void DUV::operation(Interface &input, Message &stimulus, ...){
                                ...
        // Применить стимул через адаптер входного интерфейса
                        recv(input, stimulus);
        // Вычислить реакции на некотором уровне абстракции
                                ...
        // Сообщить тестовой системе об ожидаемой реакции
                        send(output, reaction);
                                ...
  }
```

У приведенного метода есть, по крайней мере, два параметра: (1) *входной интерфейс* и (2) *модельный стимул* (*входное сообщение*). В начале метода вызывается метод `recv()`, который перегружается в адаптере эталонной модели и с помощью адаптера соответствующего интерфейса сериализует входное сообщение и подает его на тестируемый компонент. После этого вычисляются модельные реакции (на некотором уровне абстракции), которые добавляются в очереди соответствующих выходных интерфейсов путем вызова метода `send()`. В моделях с потактовой точностью можно использовать конструкцию типа `cycle()` для эмуляции такта работы компонента. Вызов `send()` в таких моделях приводит к немедленному чтению (десериализации) реакции с соответствующих выходов устройства и сравнению ее с эталонным значением. Для моделей без учета времени и моделей, учитывающих время приближенно, при вызове `send()` создается отдельный процесс, который, ожидает реакцию устройства, считывает ее, находит соответствующую ей модельную реакцию и сравнивает полученную реакцию с сопоставленной ей модельной реакцией.

6. Классификация ошибок в аппаратуре

Предложенная схема проверки реакций дает основу для классификации ошибок и их диагностики. Для определения различных типов ошибок в

компонентах аппаратуры рассмотрим *граф проверки реакций*, показывающий возможные альтернативы, имеющиеся в процессе проверки (см. Рис. 6).

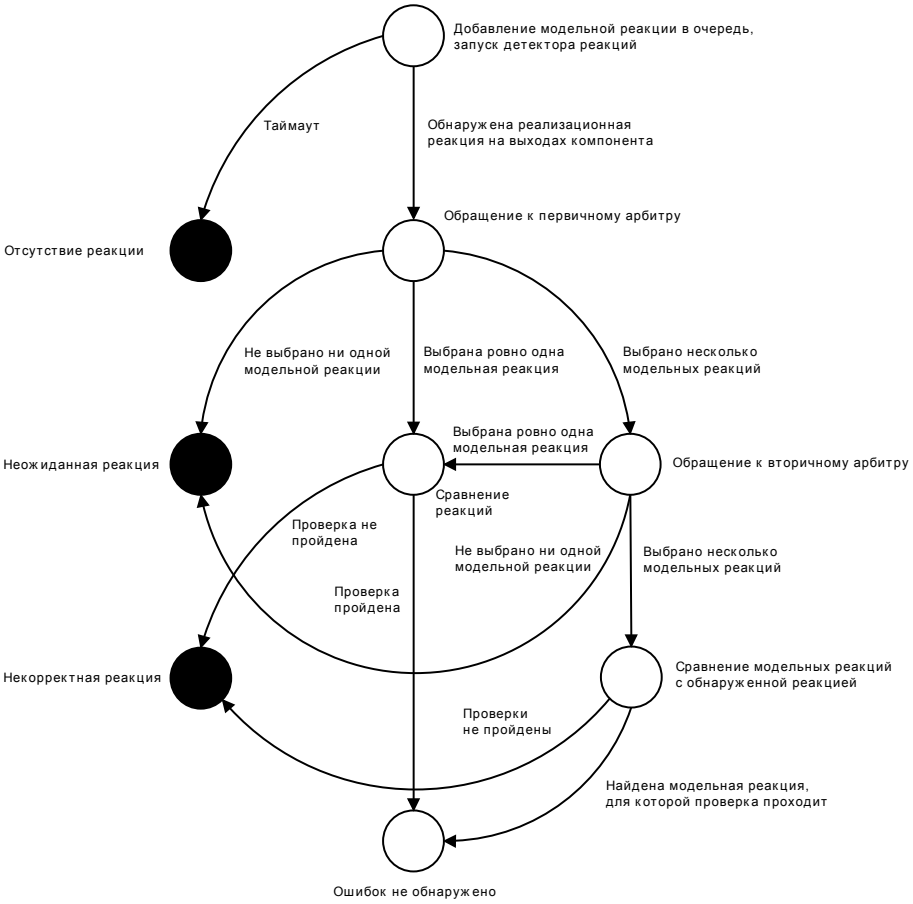

Рис. 6. Граф проверки реакций

Граф, показанный на Рис. 6, несколько упрощен, но он, тем не менее, демонстрирует основные классы ошибок в аппаратуре (они показаны черными кружками). Если тестовая система ожидает реакцию на некотором выходном интерфейсе тестируемого компонента, но реакция не возникает в течение определенного времени, фиксируется ошибка *отсутствия реакции* и сообщается информация об ожидаемой реакции (сообщение, интерфейс, ожидаемое время возникновения реакции). При обнаружении реакции, запрашивается сопоставитель реакций (первичные и вторичные арбитры).

Если он не может найти соответствующую модельную реакцию, тестовая система сообщает о *неожиданной реакции*, указывая также интерфейс и время возникновения реакции. Если сопоставитель выбирает ровно одну модельную реакцию, она считается эталонной и сравнивается с принятой реакцией. В противном случае тестовая система предупреждает о *недетерминированном поведении* и сравнивает все отобранные модельные реакции с реакцией устройства. Если есть хотя бы одно совпадение, проверка считается успешной. Иначе тестовая система сообщает о *некорректной реакции* и выводит список наиболее похожих модельных реакций.

7. Опыт применения подхода

Предложенный подход к организации тестовых оракулов был использован в ряде промышленных проектов по тестированию моделей аппаратуры (в основном отдельных модулей микропроцессоров [11]). Подход на практике показал возможность обнаружения достаточно сложных ошибок (включая трудно обнаруживаемые ошибки в управляющей логике), приемлемые трудозатраты и возможность повторного использования компонентов тестовой системы. Опыт использования подхода отражен в таблице 2.

Название тестируемого компонента	**Максимальный используемый уровень абстракции**	**Минимальный используемый уровень абстракции**	**Стадия проектирования компонента**
Буфер трансляции виртуальных адресов	Детальный учет времени	Потактовая точность	Поздняя/завершающая
Модуль арифметики для чисел с плавающей точкой	Без учета времени	—	Поздняя/завершающая
Неблокируемая кэш-память второго уровня	Приблизительный учет времени	Детальный учет времени	Средняя/поздняя
Коммутатор данных северного моста	—	Потактовая точность	Завершающая
Устройство доступа к оперативной памяти	Без учета времени	Потактовая точность	Ранняя/средняя
Системный контроллер прерываний	Без учета времени	Приблизительный учет времени	Ранняя/средняя

Таблица 2. Опыт применения предлагаемого подхода

На Рис. 7 показаны усредненные оценки трудозатрат на разработку тестовых оракулов на разных уровнях абстракции. Это достаточно грубое приближение, но оно позволяет планировать ресурсы на верификацию. Например, нам

потребовалось около 100% от начальных усилий для того, чтобы специфицировать приближенные временные свойства системного контроллера прерываний (около одного человеко-месяца). Добавленные свойства описывают порядок модельных реакций на каждом из двух выходных интерфейсов. Это позволило найти дополнительную ошибку в устройстве (одну из восьми). Другим примером является буфер трансляции адресов. Потребовалась одна человеко-неделя из трех для того, чтобы сделать эталонную модель потактово точной. При этом были найдены три дополнительные ошибки из десяти.

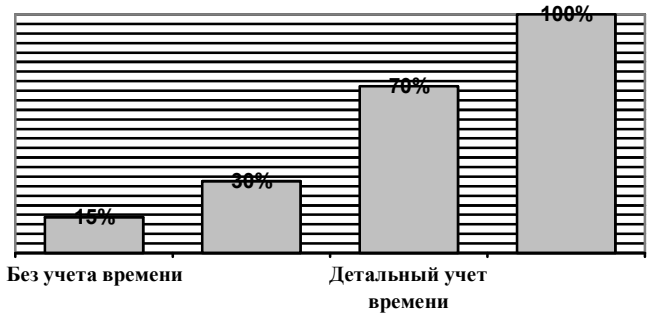

Рис. 7. Трудозатраты на разработку тестовых оракулов для разных уровней абстракции

8. Заключение

Очевидно, что использование более точных эталонных моделей позволяет находить более сложные ошибки в проектируемой аппаратуре. Также очевидно, что разработка более точных эталонных моделей требует привлечения бо́льших ресурсов. Выбор правильного уровня абстракции для тестирования того или иного компонента является трудной задачей, при решении которой необходимо учитывать множество факторов (критичность компонента, доступные ресурсы, затраты на исправление ошибок и многие другие). В ситуации, когда разные компоненты аппаратуры проверяются с разной тщательность и, более того, одни и те же компоненты тестируются по-разному в зависимости от этапа проектирования, полезно иметь универсальный подход к построению тестовых систем, применимый для разных уровней абстракции и позволяющий повторно использовать уже разработанные тестовые системы. Данная статья посвящена созданию такого подхода и рассматривает важный аспект разработки тестовых систем – создание эталонных моделей и основанных на них тестовых оракулов. Предложенная в работе архитектура тестового оракула позволяет

использовать абстрактные эталонные модели для верификации RTL-моделей аппаратуры и адаптировать тесты к изменениям интерфейсов и временны́х свойств аппаратуры. В будущем мы планируем улучшить диагностику ошибок, чтобы адекватно описывать такие ошибки, как нарушение порядка реакций на одном интерфейсе и ошибки демультиплексирования.

Литература

[1] J. Bergeron. “Writing testbenches: functional verification of HDL models”. Kluwer Academic Publishers, 2000.

[2] I. Bourdonov, A. Kossatchev, V. Kuliamin, A. Petrenko. “UniTesK test suite architecture”. In Proc. Formal Methods Europe (FME) 2002, pp. 77-88, 2002.

[3] L. Cai, D. Gajski. “Transaction level modeling: an overview”. In Proc. The International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS) 2003, pp. 19-24, 2003.

[4] W. Lam. “Hardware design verification: simulation and formal method-based approaches”. Prentice Hall, 2005.

[5] C.-M.R. Ho. “Validation tools for complex digital designs”. PhD thesis, Stanford University, 1996.

[6] H.D. Foster, A.C. Krolnik, D.J. Lacey. “Assertion-based design”. Kluwer Academic Publishers, 2004.

[7] OVM User Guide – http://www.ovmworld.org.

[8] Я.С. Губенко, А.С. Камкин, М.М. Чупилко. “Сравнительный анализ современных технологий разработки тестов для моделей аппаратного обеспечения”. Труды Института системного программирования РАН, т. 17, с. 133-143, 2009.

[9] А.С. Камкин. “Метод формальной спецификации аппаратуры с конвейерной организацией и его приложение к задачам функционального тестирования”. Труды Института системного программирования РАН, т. 16, с. 107-128, 2009.

[10] M. Chupilko, A. Kamkin. “Developing cycle-accurate contract specifications for synchronous parallel-pipeline hardware: application to verification”. In Proc. The Baltic Electronic Conference (BEC) 2010, pp. 185-188, 2010.

[11] M. Chupilko, A. Kamkin, D. Vorobyev. “Methodology and experience of simulation-based verification of microprocessor units based on cycle-accurate contract specifications”. In Proc. The Spring Young Researchers’ Colloquium on Software Engineering (SYRCoSE) 2008, vol. 2, pp. 25-31, 2008.