УДК 004.315

Современные методы функциональной верификации RTL-моделей блоков СБИС микропроцессора

Аряшев С.И., Рогактин Б.Ю., Барских М.Е.

Учреждение Российской академии наук
Научно-исследовательский институт системных исследований РАН,

rogatkin@cs.niisi.ras.ru, barskikh@cs.niisi.ras.ru

***Аннотация* — Описана методика функциональной верификации RTL блоков СБИС микропроцессора на примере блока преобразования адресов (TLB). Рассмотренный подход к тестированию RTL-модели блока может быть использован для верификации других блоков цифровой аппаратуры, в частности в рамках широко распространенной универсальной верификационной методики (UVM).**

***Ключевые слова* — функциональная верификация, план тестирования, тестовое окружение, UVM.**

I. Введение

В условиях постоянного изменения технических требований в процессе разработки и, как следствие, необходимой постоянной доработки функциональных блоков, огромное значение приобретают технологии тестирования. На основе разных методик генерируются тестовые последовательности, проверяется функциональность блоков, собирается информация о достигнутом уровне тестового покрытия и предоставляется возможность повторного использования тестов.

Современные подходы к функциональному тестированию включают в себя методологию AVM [1] (Advanced Verification Methodology) компании Mentor Graphics, OVM [2] (Open Verification Methodology) – совместной разработки Mentor Graphics и Cadence Design Systems, и UVM (Universal Verification Methodology). Наиболее прогрессивная из них – UVM [3], построенная на основе OVM, которая, в свою очередь, является комбинацией AVM с URM (Universal Reuse Methodology). UVM также включает в себя методологии eRM (e Reuse Methodology) и VMM (Verification Methodology Manual), и является первой стандартизованной методологией функциональной верификации. На рис. 1 отражено развитие (снизу вверх) методологий функциональной верификации. Таким образом, в результате UVM вобрала в себя основные, весьма удобные и полезные свойства, такие как повторное использование кода, наработанные библиотеки классов, рекомендованные наименования файлов и др.

II. Методология UVM

С помощью методологии UVM реализуется возможность разработки сложных тестовых систем на языках SystemVerilog [4], SystemC [5] или языке «e». Выбор языка остается за верификатором, однако, язык «e» является аспектно-ориентированным языком программирования, в отличие от, например, SystemVerilog, который является объектно-ориентированным.

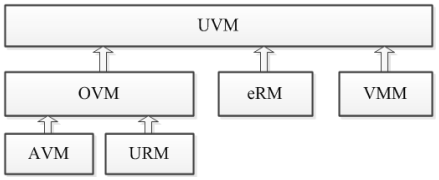

Рис. 1. Развитие методологий функциональной верификации

Концепция аспектно-ориентированного языка программирования наиболее полно подходит к использованию методов и подходов UVM, которые основаны на повторном использовании кода, что актуально при постоянных изменениях в проекте, а также позволяет расширять функциональность классов и структур без изменения уже существующего кода. Так как язык «e» разрабатывался с учетом современных требований, конструкции языка в сравнении с SystemC или SystemVerilog оказываются в большинстве случаев интуитивно понятнее и наиболее адаптированными для реализации тестового окружения UVM. Однако для использования библиотеки стандартных классов UVM, разработанной на SystemVerilog совместно с языком «e», в некоторых случаях требуется дальнейшая адаптация. В 2006 году был выпущен стандарт языка «e» [6]. В настоящее время язык «e» активно поддерживается фирмой Cadence в линейке продуктов Specman. Однако из-за относительной новизны языка «e» большинство групп разработчиков в условиях сжатых сроков проектирования предпочитают использовать в своих проектах более распространенные SystemVerilog или SystemC. Российские разработки в области

функциональной верификации включают в себя технологию UniTESK [7], разработанную в Институте системного программирования РАН. В технологии UniTESK, являющейся академической разработкой, основной упор сделан на автоматизацию создания тестов. Из плюсов UniTESK можно отметить возможность разработки спецификаций с потактовой точностью на основе пред- и пост-условий стадий операций. Что касается средств повторного использования тестов, в технологии UniTESK они менее развиты по сравнению с UVM и OVM, концепция которых хорошо отражает тенденции современной индустрии, когда основная работа сосредотачивается не на проектировании принципиально нового аппаратного обеспечения, а на интеграции аппаратуры из готовых модулей. В настоящее время индустрия функциональной верификации движется в направлении использования мультиязыковых универсальных методик верификации, в частности мультиязыковой библиотеки под методологию UVM. В данной работе рассмотрено применение методологии UVM для верификации отдельного функционального блока СБИС микропроцессора. С точки зрения верификатора блок представляет собой «черный ящик» со своим интерфейсом и набором операций, которые предполагается протестировать. Методологию функциональной верификации можно представить состоящей из основных этапов, изображенных на рис. 2. Блоки расположенные на одном уровне отражают параллельный процесс разработки. Первым этапом функциональной верификации является использование спецификаций блока в форме условий для представления требований к набору операций с интерфейсом блока или шины, а также составления плана верификация для построения тестовых последовательностей.

В спецификации могут быть описаны разного рода требования к функциональности и физическим характеристикам. Разработка плана верификации является не только начальным этапом по тестированию блока. От качества этого документа

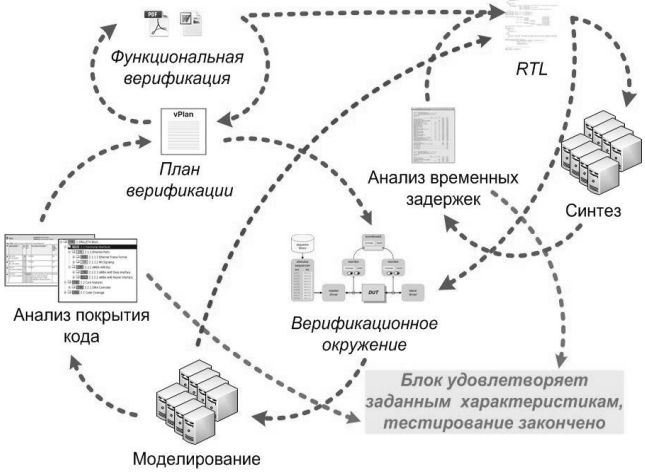

Рис. 2. Усовершенствованная методика разработки

напрямую зависит полнота тестирования, т.к. при блочном подходе к тестированию нет никаких других критериев проверки работоспособности блока кроме набора сгенерированных псевдослучайных тестов. Опытная команда верификаторов, совместно с разработчиком, создаст не только более подробный и хорошо описанный план тестирования, но и заложит в него проверки и сценарии возможных внешних условий, в которых будет работать блок, возможных воздействий, не описанных в спецификации, но зависящих от реализации модели. На основании нашего опыта был сформирован список характеристик проекта, которые должны быть в обязательном порядке отражены в плане тестирования:

1) Режим устройства и опции конфигурации.
2) Протокол обработки транзакций.
3) Протокол или схема обработки исключений.
4) Внутренние свойства блока.
5) Арбитраж использования внешних ресурсов.
6) Типичные и критические случаи использования блока.

План тестирования можно считать хорошим, если он отвечает следующим критериям:

1) План определяет конкретные, поддающиеся измерению цели «что должно быть протестировано», т.е. их можно измерить с помощью функционального покрытия.
2) План отображает точный смысл цели «что должно быть протестировано», понятный всем заинтересованным сторонам (например, системному архитектору, не знакомому с синтаксисом SV или «e»).
3) План предусматривает отслеживание того, что было протестировано (на основании анализа покрытия, соотнесенного с планом, можно точно сказать, что еще не протестировано).

Также в плане верификации могут быть ссылки на документированную спецификацию блока, отражающие те или иные тестовые сценарии, требования или конфигурацию блока.

Ситуации, не заложенные в план верификации, могут привести в дальнейшем к возникновению ошибки, поэтому содержание плана должно быть максимально полным. Конфигурация тестового окружения зависит от назначения верифицируемого блока, его интерфейса и требуемых тестовых воздействий. Выбор остается за верификатором, однако методология UVM рекомендует ряд типовых конфигураций, как для отдельных блоков, так и для шинных интерфейсов. Кроме того, некоторые САПР предоставляют средства для автоматической генерации настраиваемых тестовых окружений. Тестовые сценарии представляют собой композицию базовых операций в соответствии с назначением блока и направлены на достижение целей плана тестирования и максимального тестового покрытия. Возможны комбинации тестовых сценариев, когда успешное прохождение теста зависит от начального состояния регистров верифицируемого блока. Описание точек покрытия закладывается в план тестирования, однако последующую вставку необходимого кода покрытия и

контрольных точек удобно проводить, когда известна конфигурация тестового окружения. Методология UVM была модифицирована и расширена для целей параллельной работы при разработке блоков с её использованием. В представленном маршруте проектирования можно выделить несколько основных задач, связанных друг с другом: проектирование блока, реализация RTL модуля и синтез, проектирование тестового окружения и тестирование. В таком подходе при любых значительных изменениях RTL модуля проводится его тестирование и синтез. Не важно, чем были вызваны изменения – отладкой во время тестирования или исправлением характеристик блока под синтез. Важным условием является то, что функциональная спецификация на блок не является зафиксированным документом и может править в процессе работы над блоком. В зависимости от возникающих проблем или идей в процессе работы над блоком какие-то его характеристики могут быть изменены, например, часть его функционала может быть перенесена в другие модули. Синтез на ранних этапах разработки модуля позволяет оценить различные реализации блока и обоснованно, с учетом всех параметров, таких как быстродействие, площадь, потребление, выбрать лучшую. Разумеется, результаты синтеза на ранних этапах будут сильно зависеть от заданных граничных условий (например, задержки от входов/выходов) и правильный подбор их – тема отдельной статьи. Даже при всей начальной условности такой подход к реализации блока позволяет найти возможность достижения заданных параметров, выявить «узкие места» в реализации и в итоге сократить время на исправление проблем в характеристиках блока.

III. Верификация блока преобразования адресов

Рассмотрим более подробно применение изложенных рекомендаций на примере верификации широко используемого в СБИС микропроцессоров блока TLB. Составление спецификации проводилось поэтапно с конкретизацией требований на каждом этапе. На первом шаге разработки спецификаций была формально определена модель данных – транзакции. Были выделены базовые понятия, такие как виртуальный и физический адреса, сегмент памяти, режим работы микропроцессора и другие. Для этих понятий были определены соответствующие типы данных и функции работы с ними: вычисление сегмента памяти по виртуальному адресу, получение смещения и номера страницы адреса, проверка привилегий доступа к сегменту памяти при заданном режиме работы микропроцессора. Затем постепенно определялись типы данных для более сложных понятий: запись TLB, буферы иерархического TLB, TLB адресов данных, TLB адресов инструкций и соответствующие функции, такие как сопоставление виртуального адреса с записью TLB, поиск записи с требуемыми свойствами, преобразование адреса. В результате была определена система типов, полностью описывающая блок TLB, и набор функций, в терминах которых определяются требования.

На следующем шаге производилась структуризация требований. Для каждой операции, реализуемой в RTL-модели, были выделены микрооперации: по одной микрооперации для операций чтения, записи и проверки наличия трансляции в буфере и по несколько микроопераций для операций преобразования адресов данных и инструкций. Текстовое описание требований было разбито на атомарные требования, всего было выделено около 100 атомарных требований.

Наконец, последний шаг разработки спецификаций был связан с композицией спецификаций отдельных микроопераций в тестовые последовательности, такие как чтение, запись или поиск в буфере, операции преобразования адресов данных и инструкций. На этом шаге определялись блокировки выполнения микроопераций, условия сдвига конвейера и другие особенности совместного выполнения операций, которые оказывают влияние на наблюдаемое поведение тестируемой RTL-модели. Итерационный процесс разработки спецификации проходил при взаимодействии разработчика блока и верификатора, этот этап разработки спецификаций оказался самым трудоемким. Кроме целей верификации подобная спецификация блока может быть использована для документирования функционала блока и расширенного формального описания с учетом дальнейших модификаций блока.

Далее, для тестирования RTL-модели блока TLB был разработан план тестирования и, соответственно, четыре тестовых сценария: один сценарий для тестирования операций чтения, записи и проверки наличия трансляции в буфере и три сценария для тестирования операций преобразования адресов данных и инструкций. Нацеленность усилий на тестирование операций преобразования адресов объясняется тем фактом, что около 80% требований связаны именно с этими операциями, они составляют наибольший процент использования и являются наиболее критичными для корректной работы блока. Рассмотрим конфигурацию тестового окружения (рис. 3) в рамках методологии UVM на примере верификации TLB.

Тестовое окружение состоит из двух окружений для агентов входных и выходных воздействий, соответственно, а также верификационного окружения. Так как выходной интерфейс TLB является однонаправленным, агент выходных воздействий находится в пассивном режиме, осуществляя лишь мониторинг выходных реакций на внешние воздействия. Агент входных воздействий находится в активном режиме и через драйвер входных воздействий преобразует тестовые последовательности в дискретные цифровые воздействия на интерфейс блока TLB. Активный или пассивный режим агента, его функциональный состав и прочие настраиваемые параметры задаются в файле конфигурации тестового окружения. Предназначение драйвера тестовых последовательностей состоит в генерации пользовательских структур данных на основе ограничений, заложенных в тестовых сценариях. Связь тестового окружения с интерфейсом блока

осуществляется через карту сигналов, ставящую в соответствие именованные названия портов тестового окружения и тестируемого блока. Верификатор событий через мониторы событий осуществляет контроль ошибочных реакций тестируемого блока на внешние воздействия, для этого необходима модель сравнения. При возникновении расхождений фиксируется ошибка. Модель сравнения должна отражать эталонные реакции блока в рамках тестовых сценариев, но не обязательно должна повторять функционал блока полностью. Представленная структура тестового окружения не является единственно возможной, однако концепция является достаточно универсальной и подходит для множества поведенческих моделей устройств.

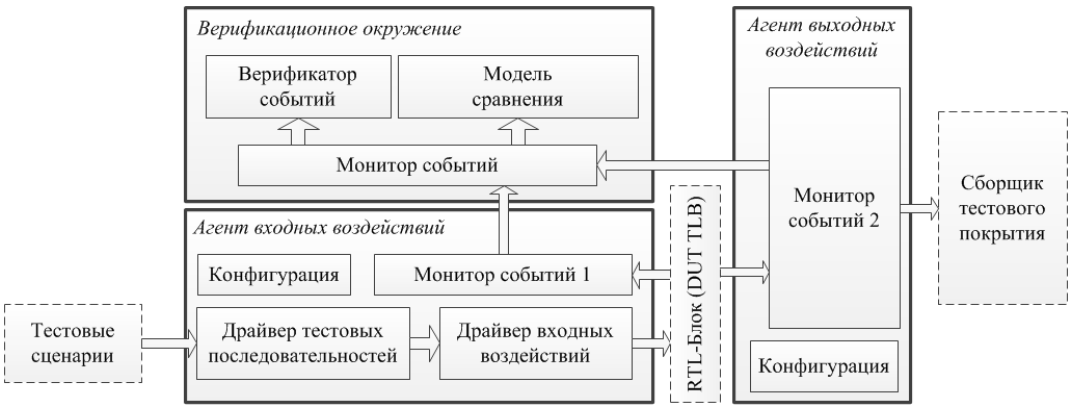

Рис. 3. Конфигурация тестового окружения TLB

IV. Заключение

В заключение можно отметить, что в рамках выполнения плана тестирования предложенная методика функциональной верификации RTL блоков СБИС микропроцессора продемонстрировала свою эффективность: при сравнительно небольшой трудоемкости разработки спецификаций и тестов для RTL-модели блока были найдены ошибки, не обнаруженные в процессе предыдущего стохастического [8] тестирования и методологии UniTESK [7]. Так, для TLB базовые спецификации были разработаны в объеме около 3000 строк кода на языке Specman *e* (около 20% от объема реализации на языке Verilog). Значительную долю спецификаций (около 40%), связанную с определением типов данных и функций работы с ними, можно повторно использовать для формального описания блока TLB любого совместимого микропроцессора. Общая трудоемкость проекта составила около 2,5 человеко-месяцев, что существенно меньше, чем трудозатраты на используемую в настоящее время регрессионную направленную базу тестов, так как последняя требует большего количества тестов и больших машинных ресурсов для моделирования микропроцессорной системы. Под тестом в данном случае понимается программный код на языке ассемблера, откомпилированный и запущенный на ядре микропроцессора, включающим в свой состав блок TLB. Уровень тестового покрытия RTL-модели, достигаемый разработанными тестами в рамках плана верификации, составляет 99%. В результате проекта было найдено около 10 расхождений между спецификацией и реализацией, среди которых 5 ошибок. Отметим, что некоторые из найденных ошибок являются критичными и могут возникнуть при моделировании загрузки операционной системы. Отладка таких ситуаций методами стохастического тестирования является весьма затруднительной и требует локализации в большом объеме данных. Созданные на основе представленной методики сценарии функционального тестирования можно повторно использовать при смешанном цифро-аналоговом моделировании заказных аналоговых блоков.

Литература

[1] A. Rose, T. Fitzpatrick, D. Rich, H. Foster. Advanced Verification Methodology Cookbook. Mentor Graphics Corporation. 2008.

[2] S. Iman. Step-by-step Functional Verification with SystemVerilog and OVM. Hansen Brown Publishing Company. 2008.

[3] Standard Universal Verification Methodology Class Reference Manual, Release 1.1. Accellera Systems Initiative. 2011.

[4] IEEE Standard for SystemVerilog — Unified Hardware Design, Specification, and Verification Language. IEEE Std 1800-2009.

[5] IEEE Standard SystemC Language Reference Manual. IEEE Std 1666. 2005.

[6] Standard for the Functional Verification Language e. IEEE Std 1647. 2010.

[7] Иванников В.П., Камкин А.С., Кулямин В.В., Петренко А.П. Применение технологии UniTESK для функционального тестирования моделей аппаратного обеспечения // Препринт 8, Институт системного программирования РАН. 2005.

[8] Грибков И.В., Захаров А.В., Кольцов П.П., Котович Н.В., Кравченко А.А., Куцаев А.С., Осипов А.С., Хисамбеев И.Ш., Коганов М.А. Развитие системы стохастического тестирования микропроцессоров INTEG // Программные продукты и системы. 2010. № 2. С. 14-23.