Окончание. Начало в № 3 `2008

Проектирование в условиях временных ограничений: верификация проектов

Ростислав ГРУШВИЦКИЙ
RIGrushvitsky@mail.eltech.ru
Максим МИХАЙЛОВ
yamaksya@yandex.ru

В статье рассматриваются общие вопросы применения мониторов открытой библиотеки верификации OVL. Заключительная часть цикла посвящена также перспективам развития средств верификации и их отражению в языках описания аппаратуры.

Библиотека OVL

Библиотека **OVL**, разработанная фирмой Accellera (**www.accellera.com**), предоставляет проектировщикам, специалистам по интеграции систем и инженерам в области верификации широкие возможности для подтверждения правильности проекта с использованием моделирования, полуформального и формального методов верификации. Используя единый, четко определенный интерфейс, библиотека **OVL** ликвидирует разрыв между различными видами верификации, что делает доступным передовые методы и средства верификации даже для неопытных пользователей.

Библиотека **OVL** содержит множество мониторов утверждений, предназначенных для проверки специфических свойств систем. Существует две версии данной библиотеки, синтаксис которых определяется, прежде всего, особенностями используемых языков описания аппаратуры — **VHDL** или **Verilog**. К примеру, **Verilog**-версия подразумевает предварительную обработку (preprocessing) макросов, в то время как язык **VHDL** не поддерживает такой возможности.

С ростом сложности проектов резко возрастает необходимость их проверки. Во многих случаях успешность того или иного проекта определяется опытом проектировщика, то есть ставка делается не на то, что ошибка будет найдена и устранена, а на то, что ее попросту не будет. Соответствующий рынок изобилует различными языками верификации и интерфейсами инструментов для ее осуществления, то есть каждый пакет верифицирует по-своему, нет единых правил. Большинство этих языков и правил не вписываются в общий поток проектирования, что требует от проектировщика овладения большим числом методов и интерфейсов верификации. Кроме того, в данных обстоятельствах сложнее развивать существующие методы и средства по верификации. Библиотека **OVL** как раз и решает две основные задачи — предоставляет единый механизм как для верификации систем, так и для модернизации этого механизма за счет написания новых мониторов.

Выгода использования мониторов утверждений в проекте определяется, в частности, следующими преимуществами:

- тестирование внутренних точек системы, что существенно повышает наблюдаемость проекта;
- упрощение процесса обнаружения ошибок и определения причин возникновения неисправностей за счет ограничения возможности их появления;
- использование одних и тех же языковых конструкций как для моделирования, так и для формальной верификации.

Общие представления о мониторе

Мониторы утверждений — это объявленные в проекте компоненты, целью использования которых является гарантирование выполнения некоторых условий. По сути, мониторы утверждений представляют собой развитие оператора проверки **ASSERT** в языке **VHDL**. Преемственность можно проследить в наличии трех основных составляющих любого монитора **OVL** — event (событие), message (сообщение) и severity (важность).

- **Event** — свойство, которое проверяется утверждением. Свойство может быть постоянным или временным. Постоянное свойство должно иметь силу всегда, тогда как временное свойство справедливо только в определенные моменты времени.
- **Message** — строка, отображаемая на экране в случае возникновения ошибки утверждения.
- **Severity** — параметр, определяющий уровень важности ошибки, зафиксированной монитором.

Как уже отмечалось выше, существует две модификации библиотеки **OVL**. Далее основное внимание будет уделено **VHDL**-версии. Варианты имплементации библиотеки могут быть различны: либо в виде двух файлов **assert.vhd** и **assert_pkg.vhd**, либо в виде отдельных VHDL-файлов для каждого монитора и файлов **ovl_assert.vhd** и **ovl_assert_util.vhd**, либо каким-нибудь другим образом. Однако содержание во всех случаях остается одинаковым. Необходимо описание модели утверждения с помощью конструкции **ENTITY/ARCHITECTURE** и наличие некоторого пакета (**PACKAGE**), который включает в себя объявление компонентов (каждому монитору соответствует одна декларация **COMPONENT**). В этом же пакете должны объявляться глобальные сигналы (**ovl_reset_n**, **ovl_reset_n_enable** и т. д.) и набор служебных функций (**xorr**, **to_std** и т. д.). Для использования мониторов соответствующие **VHDL**-файлы необходимо скомпилировать в проектную директорию.

Если рассматривать монитор как конструктивный модуль на языке **VHDL**, то для него характерны следующие особенности:

- все порты являются входными (**IN**);
- монитор имеет набор настроек (**GENERIC**), часть из которых специфична для каждого отдельного монитора, но есть также три стандартных параметра:
  - **severity_lvl** — предназначен для оценки значимости ошибки,
  - **options** — 32-битный дополнительный параметр для спецификации монитора утверждений,
  - **msg** — сообщение об ошибке, которое выводится на экран при срабатывании монитора.

При моделировании срабатывание (*firing*) монитора утверждений (обычно — сообщение об ошибке) происходит при возникновении сбойной ситуации. Как правило, происходит это в следующем цикле. Однако в том случае, если проверяемое выражение *test_expr* не синхронизировано с тактовыми сигналами утверждения *clk*, то возможна недетерминированная задержка или ложное срабатывание монитора. Во избежание последствий данной ситуации необходимо всегда выравнивать выражение **test_expr** с частотой выборки монитора **clk**.

Классификация мониторов

Большинство информационных источников, связанных с библиотекой **OVL**, содержат описание свойств отдельных мониторов (перечисляемых обычно в алфавитном порядке), что не всегда удобно в прикладной сфере. Систематизация и классификация мониторов с точки зрения их целевой направленности использования в проектах представляется авторам другим и достаточно важным подходом. Поэтому далее будет приведен перечень наиболее часто употребляемых мониторов утверждений с учетом разделения на области их применения. Группы мониторов формировались, прежде всего, на основе данных, изложенных в [3], а также на основе собственного опыта, приобретенного в результате реальной работы с библиотекой **OVL**.

Авторы не считают целесообразным использовать ресурсы статьи для публикации листингов, содержащих примеры использования мониторов. Подробный разбор применения одного из мониторов **OVL** можно найти в предыдущей статье цикла. Что касается синтаксиса обращений к мониторам и других характеристик, то их детальное описание можно найти в [1]. Здесь же основное внимание будет уделено вопросам выбора одного из мониторов группы, образованной по принципу близости контролируемого свойства проектируемого устройства.

Проверка постоянных свойств

Библиотека OVL предоставляет ряд мониторов, позволяющих выполнять проверку постоянных свойств проекта. Под постоянными свойствами понимаются такие условия, которые выполняются (или наоборот не выполняются) на протяжении всех циклов.

К этой группе можно отнести мониторы: **assert_always**, **assert_never**, **assert_zero_one_hot** и **assert_range**.

Утверждение assert_always используется для проверки сохранения некоторого постоянного свойства проекта на каждом цикле работы (рис. 1). Здесь и далее на рисунках импульс на временной диаграмме будет означать возникновение события или, что есть то же самое, оценку проверяемого булевского выражения как TRUE.

Монитор **assert_never** позволяет определить постоянное свойство, которое никогда не должно быть оценено как истина.

Мониторы **assert_always** и **assert_never** удобны при определении общих постоянных свойств. Однако существует также множество специфических неизменных свойств, формулирование которых с помощью булевских выражений оказывается слишком громоздким и затруднительным. К примеру: проверка условий **one-hot** и **one-cold**, проверка на четность или нечетность, контроль допустимого диапазона переменной. Условие **one-hot** означает возможность единичного значения (**one-cold** — нулевого значения) только одного бита в наборе в каждый момент времени. Такое условие можно отследить с помощью проверки взаимного исключения отдельных битов вектора. Для этого утверждения используется логико-математическое выражение (bit_vector & (bit_vector — 1)). Результат данного выражения будет равняться нулю в том случае, если все биты вектора bit_vector являются нулями или только один из битов набора равен единице в каждый момент времени. Выполнимость этого условия можно проверить с помощью монитора **assert_always**. Однако, как уже было отмечено ранее, подобный способ является достаточно неудобным. В качестве альтернативы можно использовать **OVL** монитор **assert_zero_one_hot**, что существенно упрощает запись. Монитор **assert_zero_one_hot** наиболее подходит для схем управления. Он обеспечивает поддержку правильности функционирования конечного автомата, состояния которого закодированы по правилу **one-hot**. В схемах тракта передачи данных **assert_zero_one_hot** может гарантировать исключение конфликтной ситуации при одновременном обращении нескольких устройств к общей шине. Область возможного использования **assert_zero_one_hot** включает различные контроллеры, схемы разрешения и арбитражную логику. Наконец **assert_zero_one_hot** полезен для проверки взаимного исключения нескольких событий.

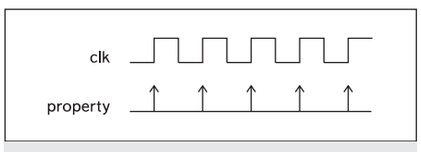

Рис. 1. Монитор assert_always

Монитор **assert_range** непрерывно контролирует выражение **test_expr** по каждому положительному фронту запускающего события или синхросигнала **clk**. Суть данного утверждения заключается в том, что значение некоторого выражения всегда должно лежать в определенном диапазоне. В противном случае будет обнаружена сбойная ситуация. Область допустимых значений выражения задается параметрами **min** и **max**. Причем значение **min** не должно превосходить значения **max**. Использование утверждения **assert_range** может обеспечивать наличие только допустимых значений, например, в схемах управления (счетчики, конечные автоматы и др.).

Проверка взаимосвязи циклов

Рассмотренные ранее мониторы постоянных свойств, без сомнения, играют существенную роль при верификации систем. Однако довольно часто возникают ситуации, требующие спецификации нескольких событий во времени (другими словами — проверки выполнимости свойств в заданных циклах). Вторая группа мониторов позволяет контролировать взаимосвязь циклов.

Здесь как наиболее типовые можно выделить мониторы: **assert_next**, **assert_frame** и **assert_cycle_sequence**.

Утверждение **assert_next** проверяет правильность временного соотношения двух событий. Так, всякий раз, когда происходит событие **A** (рис. 2), в следующем за ним цикле должно произойти событие **B** (A -> next B).

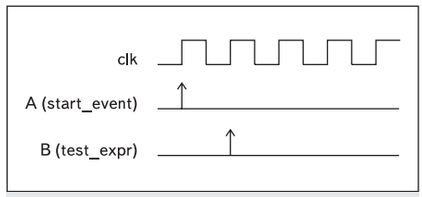

Рис. 2. Монитор assert_next

В соответствии с правилом импликации событие **A** носит название *антецедента*, а событие **B** — *консеквента*. В терминах **OVL** антецедент играет роль запускающего события (**start_event**), в то время как консеквент представлен проверяемым выражением **test_expr**. С помощью дополнительных опций монитора можно также задать, например, количество циклов, по истечении которых выражение **test_expr** должно оцениваться как истина после возникновения запускающего события (отложенная проверка — на рис. 3 оценка выражения производится через один цикл после возникновения условия запуска).

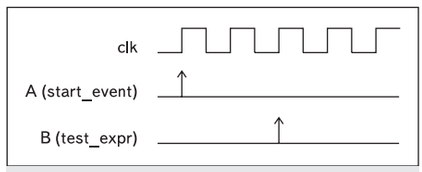

Рис. 3. Монитор assert_next: отложенная проверка

Кроме того, данный монитор поддерживает возможность наложения последовательностей событий (sequences overlapping). Например, если задано, что тестовое выражение должно быть истинным через 2 цикла после запускающего события (рис. 4), то это не означает, что необходимо ждать завершения этой цепочки событий. При включенной опции check_overlapping следующая последовательность может быть запущена ранее (второй импульс события A на рис. 4).

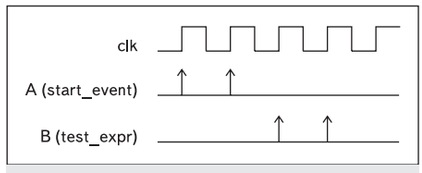

Рис. 4. Монитор assert_next: наложение последовательностей событий

Монитор **assert_next** может быть использован в системах для проверки правильности последовательности событий. Один из ярких примеров — проверка устойчивости (неизменности) данных на линии после возникновения некоторого инициирующего события (асинхронная транзакция с квитированием).

Монитор **assert_next** предоставляет возможность контроля взаимосвязи циклов с учетом точного задания времени появления проверяемого события. Однако часто возникают ситуации, когда точное значение количества циклов не определено, но известен временной интервал появления проверяемого события — ***окно***. Подобные условия можно отслеживать с помощью монитора **assert_frame**. Когда запускающее событие (**start_event**) оценивается как истина, проверяемое выражение (**test_expr**) должно также принять значение истины в пределах временного окна, которое ограничено минимальным и максимальным количеством циклов. Если проверяемое событие не возникнет в рамках допустимого окна, то произойдет срабатывание (**firing**) монитора. Монитор, так же как и предыдущий, может использоваться для проверки правильности синхронизации событий.

Мониторы **assert_next** и **assert_frame** проверяют взаимоотношение только двух событий (событием называется равенство булевского выражения значению TRUE). Если необходимо проверить появление сразу нескольких событий, то можно воспользоваться монитором **assert_cycle_sequence**. Так можно проверить, например, последовательное появление циклов операций записи (WRITE), ожидания (WAIT) и завершения (DONE) на шине.

Проверка окон, ограниченных событиями

Многие свойства реальных систем могут быть ограничены во времени некоторыми событиями. Например, разработчику может потребоваться проверка изменения значения некоторого булевского выражения в пределах окна, заданного начальным (**start_event**) и конечным (**end_event**) событиями. Иллюстрация показана на рис. 5 (изменение значения проверяемого выражения отображено изменением цвета). Также возможна обратная ситуация, когда значение выражения должно быть неизменным в течение временного окна.

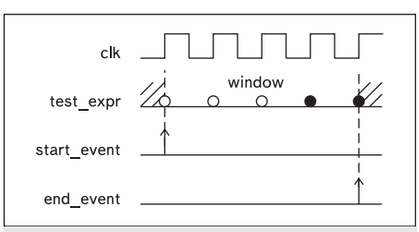

Рис. 5. Проверка окон, ограниченных событиями

Для этих целей разработаны два монитора **OVL**: **assert_win_change** и **assert_win_unchange**.

Монитор **assert_win_change** непрерывно отслеживает появление запускающего события (**start_event**) по каждому фронту тактового сигнала. При возникновении такого события монитор начинает контролировать изменение значения проверяемого булевского выражения (**test_expr**) до момента появления конечного события (**end_event**). Например, можно контролировать появление данных на шине после выдачи команды чтения и до момента возникновения следующей команды.

Монитор **assert_win_unchange** выполняет обратную функцию — следит за тем, чтобы проверяемое событие не изменялось в пределах окна, ограниченного событиями. Такое средство будет полезно, например, если необходимо удостовериться в неизменности данных при выполнении операций чтения или записи.

Проверка окон, имеющих временные ограничения

В дополнение к средствам проверки временных свойств, ограниченных событиями, существуют мониторы проверки окон, заданных с помощью временных меток (определенных циклов): **assert_change** и **assert_unchange**.

Отличие от предыдущей группы заключается только в способе задания окна. Мониторы **assert_change** и **assert_unchange** определяются запускающим событием (**start_event**) и количеством циклов (опция **num_cks**), в течение которых проверяемое значение (**test_expr**) должно измениться и оставаться неизменным, соответственно. Типовой пример использования монитора **assert_change** — сигнал подтверждения прерывания должен появиться в течение заданного числа циклов после запроса на прерывание.

Проверка переходов в автомате состояний

Библиотека OVL содержит также ряд мониторов, которые позволяют проверять правильность переходов между состояниями конечных автоматов (FSM, Finite State Machines). Среди них можно выделить пару мониторов — **assert_no_transition** и **assert_transition**.

Монитор **assert_no_transition** предназначен для проверки переходов автоматов в запрещенные состояния. По каждому фронту тактового сигнала выполняется проверка выражения **test_expr**. Если значение этого выражения совпадает со значением **start_state**, то монитор начинает следить за тем, чтобы автомат не перешел в состояние **end_state**. В случае такого перехода произойдет срабатывание монитора. Монитор **assert_transition** наоборот следит за тем, чтобы автомат перешел в состояние **end_state**.

Естественно, в рамках журнальной статьи невозможно произвести обзор и классификацию всех мониторов **OVL**. Причина этого кроется не только в ограниченности отведенных ресурсов, но и в том, что библиотека постоянно обновляется. Основная цель, которую ставили перед собой авторы, заключается в демонстрации возможностей **OVL** для верификации типовых элементов современных цифровых систем. Этой цели служит и разбиение на представленные группы. Более детальное знакомство с мониторами можно получить, посетив интернет-ресурс **http://www.accellera.org/activities/ovl/**.

Некоторые современные тенденции развития языков описания и верификации аппаратуры

Основное назначение языков описания аппаратуры с момента их появления и до последнего времени менялось в соответствии с текущими потребностями пользователей: на начальном этапе основной упор делался на возможности спецификации проектов или описания функционирования готовых устройств. Только на следующем этапе развития языков больше внимания стало уделяться автоматизации создания проектов (синтезу). Как уже отмечалось во вступительных статьях цикла, за последние годы (условно можно считать — на третьем этапе эволюции языков описания аппаратуры) резко возрос интерес к проблемам верификации аппаратуры. При этом большинство авторов отмечает, что современной тенденцией развития языков описания аппаратуры является добавление к ним средств, позволяющих при создании проектов использовать верификационные средства, написанные на том же языке, что и сам проект. Ранние версии языков описания аппаратуры (таких, как **VHDL** и **Verilog**) создавали определенные трудности для решения подобной задачи. Поэтому и получили определенное распространение языки, специализирующиеся исключительно на проблемах верификации. Как уже отмечалось ранее, среди этих языков наибольшие шансы на выживание имел язык **PSL**. Это в значительной мере определялось его сильной ориентацией на расширение возможностей традиционных языков проектирования. До последнего времени языки верификации аппаратуры разрабатывались как самостоятельные средства, хотя и ориентированные на стандартные языки.

Ситуация изменилась после 2002 года, когда фирма Accellera выпустила версию 3.0 языка **SystemVerilog**, созданного на базе переданного ей в 2001 году фирмой Co-Design Automation пакета SUPERLOG. В 2005 году комитет по стандартизации IEEE принял стандарт IEEE 1800-2005 для языка **System Verilog**. Два основных момента предопределили широкое распространение (особенно в США) языка **SystemVerilog**. Первым необходимо признать включение в язык элементов объектно-ориентированного программи-

рования и вторым — средств **PSL**. Включение в состав языка средств **PSL** резко увеличило его возможности в плане проверки верификационных свойств проектов. Перевод всех отечественных проектов на язык **SystemVerilog** казался неизбежным.

Однако после появления в 2007 году работы [5] стало понятным, что поистине революционные изменения уже не за горами и для языка **VHDL**. Новая версия языка, носящая название **VHDL-2008** и переданная в начале 2008 года в комитет стандартизации IEEE, содержит целый ряд новых средств, существенно расширяющих возможности языка. Помимо средств, добавляющих новые и усиливающих старые возможности традиционного проектирования, большое внимание в новой версии уделено верификационным проблемам.

Поскольку языку **SystemVerilog** уделено в зарубежных изданиях много внимания, а по языку **VHDL-2008** информация в отечественных изданиях весьма скудная, имеет смысл чуть подробнее остановиться на особенностях реализации последнего. Прежде всего, изменения коснулись определений языка, позволяющих более изящно и компактно записывать типовые (синтезируемые) конструкции.

К этим изменениям можно отнести расширение областей применения существующих ранее операций. Логические операции, предназначенные прежде только для скалярных или векторных операндов, стали разрешенными для смешанных (векторно-скалярных) операндов. Расширились допустимые операции: логические, сдвиговые и условные. Появились новые операции типа максимум и минимум, деление по заданному модулю и остаток для данных физического типа.

Изменения коснулись и операторов. Прежде всего, расширились возможности параллельных условных и селективных операторов присвоения сигналов (они приблизились к последовательным операторам). Расширились допустимые действия с операторами **IF GENERATE** и **CASE GENERATE**. Планируется много изменить и в системных типах данных.

Но основные изменения ориентированы на усиление верификационного аспекта языка. Спектр верификационных средств весьма широк и включает средства не только формальной, но и симуляционной верификации, а, следовательно, и произвольно комбинируемых вариантов. С одной стороны, внутри языка интегрированы основные понятия языка **PSL**. Интеграция включает ключевые слова языка и все средства, необходимые для верификации, основанной на утверждениях. С другой стороны, новая версия содержит целый ряд новых средств, облегчающих создание и использование программ Test-Bench. Поскольку основные понятия и возможности языка **PSL** уже были рассмотрены ранее, целесообразно остановиться на средствах, упрощающих создание моделирующих программ.

К ним следует отнести новое средство — внешние имена. Использование внешних имен позволяет при написании программ Test-Bench получить доступ к объектам, доступ к которым ранее был запрещен ввиду их вложенности в иерархически организованную программу. Благодаря средствам **FORCE** и **RELEASE** можно не только контролировать состояния внутренних сигналов проекта, но и управлять в тестирующих программах их значениями.

Конечно, следует учитывать, что пока **VHDL-2008** только предложен для стандартизации. Даже после приема стандарта кардинальное изменение ситуации возможно только тогда, когда откорректированный вариант языка окажется включенным в состав наиболее распространенных САПР. Однако, как показывает опыт предшествующих лет, фирмы — разработчики программного обеспечения — вносят подобные изменения достаточно оперативно. Поэтому можно надеяться на быстрое внедрение описанных верификационных средств в повседневную деятельность отечественных разработчиков.

Литература

1. Assertion Monitor Reference Manual v 03.10.14
2. Максфилд К. Проектирование на ПЛИС. Архитектура, средства и методы. Курс молодого бойца. М.: ДОДЕКА-XXI, 2007
3. Foster H., Krolnik A., Lacey D. Assertion-Based Design. Kluwer Academic Publisher, 2004.
4. Accellera Continues to Promote Increased Electronic Design Productivity with Revised VHDL Standard, Oct. 9, 2006. — **http://www.haifa.ibm.com/projects/verification/sugar/papers/accellera_vhdlVHDL_press_release_psl_inside2006.pdf**
5. Ashender P. J., Lewis J. VHDL-2008: Just the New Stuff. Morgan Kaufmann Publishers of Elsevier.