ФУНКЦИОНАЛЬНАЯ ВЕРИФИКАЦИЯ С ПОМОЩЬЮ ЯЗЫКА VHDL

Авдеев Н. А.
Объединённый институт проблем информатики Национальной академии наук Беларуси
Минск, Республика Беларусь
E-mail: avdeev_n@newman.bas-net.by

Приводится обзор функций пакетов RandomPkg и CoveragePkg, позволяющих существенно ускорить разработку тестирующих программ, использующих настраиваемую генерацию псевдослучайных тестов и функциональное покрытие, для функциональной верификации цифровых устройств.

Введение

Функциональное покрытие является одной из метрик, показывающей какая часть спецификации проверена. Качество результатов покрытия зависит от плана тестирования, т. к. 100% функциональное покрытие означает, что все функции, заданные в плане тестирования, выполнены во время моделирования. В отличии от покрытия кода, которое является функцией системы моделирования и может быть полностью автоматизировано, функциональное покрытие требует тщательной разработки тестового окружения и тестов в соответствии со спецификацией, а также анализа результатов моделирования, что не может быть полностью автоматизировано. Для осуществления функционального покрытия разработаны специальные функции в различных специализированных языках верификации, таких как SystemVerilog, ‘e’ и др.

Проблема верификации цифровых устройств привела к появлению методологии OS-VVM (Open Source VHDL Verification Methodology) [1, 2]. OS-VVM — это методология написания «интеллектуальных» тестирующих программ с использованием языка VHDL. Данная методология позволяет реализовать функциональное покрытие и управляемую генерацию псевдослучайных тестов, что используется при верификации цифровых функциональных блоков.

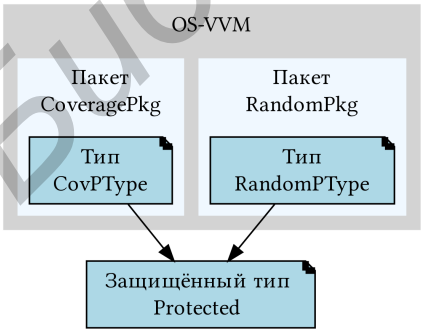

Рис. 1 – Тип *protected* используется в OS-VVM

Методология OS-VVM основывается на использовании специализированных VHDL пакетов *RandomPkg* и *CoveragePkg* [2] при разработке тестирующих программ. Функции пакетов базируются на использовании защищенного типа *protected* (рис. 1), который появился в стандарте VHDL’2002.

I. Защищенный тип protected

Защищенный тип (*protected*) [3] базируется на концепции, похожей на классы в объектно-ориентированном подходе, известном из других языков программирования. Тип *protected* позволяет объединить данные и операции, выполняемые над ними, в один объект (инкапсуляция), таким образом, скрываются детали реализации типов данных от пользователей.

Полное определение типа *protected* состоит из двух частей: декларации и тела (*body*) типа. Объявления в декларативной части типа могут включать декларации подпрограмм (процедур и функций), спецификации атрибутов и конструкции подключения, использующие ключевое слово *use*. Тела подпрограмм объявляются в теле типа *protected*. Подпрограммы, описанные при декларации типа *protected*, называются *методами*.

Элементы, объявленные внутри тела типа *protected*, не доступны для использования вне этого типа. Таким образом, единственный способ доступа к этим элементам, это использование методов, объявленных при декларации типа. Тело типа *protected* задаёт детали реализации данного типа, в теле типа могут быть описаны: декларации и тела подпрограмм, пакетов; декларации типов, подтипов, констант, переменных, файлов и *alias* (переименований); декларации атрибутов и спецификации и др.

II. Пакет RandomPkg

В пакете *RandomPkg* декларируется защищенный тип *RandomPType*, который включает в себя начальное значение (*seed*) псевдослучайного генератора и набор функций для генерации случайных чисел в различных форматах и диапазонах. Генерация псевдослучайных чисел с использованием типа *RandomPType* проходит в три этапа: декларация переменной данного типа, настройка генератора (в простейшем случае задание начального значения *seed*) и получение псевдослучайного числа, как показано в следующем примере.

```
-- декларация переменной RV
variable RV : RandomPType;
. . .
-- задание начального значения seed
RV.InitSeed(RV'instance_name);
X <= RV.RandInt(1, 10); -- получение
-- псевдослучайного числа в диапазоне [1, 10]
```

Функции генерации псевдослучайных чисел доступны не только для целых чисел, но и для векторных типов `std_logic_vector` (метод *RandSlv*), `unsigned` (*RandUnsigned*) и `signed` (*RandSigned*).

По умолчанию, функции генерации возвращают случайные числа, подчиняющиеся равномерному закону распределения (реализованному с помощью процедуры *uniform* пакета math_real из библиотеки IEEE). В пакете реализованы другие законы распределения: *FAVOR_SMALL* (распределение с преобладанием малых значений), *FAVOR_BIG* (распределение с преобладанием больших значений), *NORMAL* (нормальный закон распределения, закон Гаусса), *POISSON* (распределение Пуассона). Кроме стандартных законов распределения случайной величины в пакете существует возможность взвешенной генерации – так называют [4] генерацию по произвольному закону распределения с помощью указания списка требуемых значений целых чисел и их вероятностей появления (весов).

III. Пакет CoveragePkg

Пакет *CoveragePkg* включает описание новых типов данных и функций, которые позволяют создавать корзины для точек покрытия и перекрестного покрытия, собирать контролируемые значения переменных (сигналов), проверять полноту покрытия, выводить отчет о результатах покрытия. Но наиболее важной функцией пакета *CoveragePkg* является возможность организации интеллектуального покрытия, под которым понимается выбор псевдослучайного значения (или набора значений для перекрестного покрытия) из диапазона непокрытых значений [2].

В таблице 1 приведены основные этапы работы с функциями пакета *CoveragePkg*. Для организации покрытия в VHDL-программе необходимо создать переменную защищенного типа *CovPType* и описать модель покрытия путем задания необходимых корзин. Далее в соответствии с планом тестирования с помощью метода *ICover* осуществляется сбор значений переменных или сигналов в заданные моменты времени (или по определенным событиям). Полученные данные о покрытии можно использовать для управления псевдослучайной генерацией тестов. Метод *IsCovered* позволяет проверить достигнуто ли полное покрытие всех корзин. С помощью методов *WriteCovDb* и *ReadCovDb* можно сохранить и загрузить базу данных о покрытии, что позволяет осуществить объединение данных о покрытии по нескольким запускам моделирования, которые могут выполняться параллельно.

Заключение

Пакеты *RandomPkg* и *CoveragePkg*, входящие в методологию OS-VVM, существенно расширяют возможности функциональной верификации с помощью языка VHDL, упрощая процесс разработки тестирующих программ, использующих управляемую генерацию псевдослучайных тестов и функциональное покрытие. Исходные тексты пакетов доступны в сети Internet, продолжают обновляться и хорошо документированы. В последних версиях системы моделирования Questasim фирмы Mentor Graphics появилась библиотека OSVVM, в которой скомпилированы пакеты *RandomPkg* и *CoveragePkg*.

1. Open source VHDL verification methodology. User's Guide Rev. 1.2 [Electronic resource] / Ed. J. Lewis. – Mode of access: http://osvvm.org/downloads. – Date of access: 02.09.2013.
2. Авдеев, Н. А. Средства VHDL для функциональной верификации цифровых систем. Методология OS-VVM. / Н. А. Авдеев, П. Н. Бибило // Современная электроника. – 2013. – № 5. – С. 66–70.
3. Авдеев, Н. А. Средства VHDL для функциональной верификации цифровых систем / Н. А. Авдеев, П. Н. Бибило // Современная электроника. – 2013. – № 3. – С. 74–76.
4. Проектирование и верификация цифровых систем на кристаллах. Verilog & SystemVerilog / В. И. Хаханов [и др.]. – Харьков : ХНУРЭ, 2010. – 528 с.

Таблица 1 – Основные этапы работы с функциями пакета *CoveragePkg*

Подключение пакета	**use** work.CoveragePkg.all;
Декларация объекта покрытия	**shared variable** CovX, XYCov : CovPType;
Генерация корзин	GenBin(0, 7); -- *8 корзин, 1 значение в каждой* GenBin(0, 255, 16); -- *16 корзин одинакового размера*
Создание точек покрытия, либо перекрестного покрытия	CovX.AddBins(GenBin(0, 31, 8)); XYCov.AddCross(GenBin(0, 7), GenBin(0, 7));
Выборка (сбор) значений	CovX.ICover(X);
Проверка полноты покрытия	**if** CovX.IsCovered **then**
Оценка непокрытой области	NotCov := CovX.CountCovHoles;
Генерация псевдослучайных тестов	X := CovX.RandCovPoint; -- *выбор непокрытых значений*
Вывод отчета	CovX.WriteBin; -- *отчет может быть достаточно большим*
Сохранение базы данных	CovX.WriteCovDb("covdb.txt", OpenKind => WRITE_MODE);