Средства VHDL для функциональной верификации цифровых систем: пакет RandomPkg

Николай Авдеев, Пётр Бибило (г. Минск, Беларусь)

Статья посвящена описанию типов данных и функций пакета RandomPkg, входящего в методологию OS-VVM и предназначенного для псевдослучайной генерации чисел. На нескольких примерах показано, как можно использовать псевдослучайную генерацию в тестирующих программах.

Введение

С развитием средств сквозного автоматизированного проектирования всё больше внимания уделяется проблемам верификации описаний цифровых систем, получаемым на различных этапах их проектирования. Были разработаны методы и системы формальной верификации для описаний, представленных на языке VHDL. В настоящее время возрастание сложности проектируемых систем, реализуемых на программируемых логических схемах типа FPGA и системах на кристалле, на передний план выдвигает проблемы проверки соответствия высокоуровневых VHDL-моделей цифровых систем спецификациям на их разработку. Такая верификация реализуется на основе моделирования и требует введения в VHDL новых конструкций, создания соответствующих VHDL-пакетов для написания всё более сложных тестирующих программ. Одним из направлений такой верификации является функциональная верификация, основанная на генерации псевдослучайных тестовых наборов и функциональном покрытии.

Листинг 1

```
library work;
use work.RandomPkg.all;

entity TestRnd is
end entity TestRnd;

architecture beh of TestRnd is
 signal X : integer;
begin
 GenRnd : process is
 -- декларация переменной RV
 variable RV : RandomPType;
 begin
 -- задание значения seed
 RV.InitSeed(RV'instance_name);
 for i in 1 to 1000 loop
 -- получение числа из
 -- диапазона [1, 10]
 X <= RV.RandInt(1, 10);
 wait for 10 ns;
 end loop;
 wait;
 end process GenRnd;
end architecture beh;
```

В статьях [1, 2] рассмотрена проблема функциональной верификации исходных VHDL-описаний цифровых систем по методологии OS-VVM (Open Source VHDL Verification Methodology). Под функциональной верификацией понимается проверка соответствия VHDL-описания проекта цифровой системы заданным спецификациям. Функциональная верификация дополняет тестирование VHDL-моделей с помощью направленных тестов и аппарата ассертов [3]. В статье [1] изучается тип protected (защищённый тип), в статье [2] представлены примеры тестирующих программ. В них используются подпрограммы и типы данных, декларированные в VHDL-пакетах RandomPkg, CoveragePkg, поддерживающих методологию OS-VVM. Эти VHDL-пакеты используют стандарт VHDL'2008.

Предлагаемая статья посвящена описанию типов данных и функций пакета RandomPkg версии 2013.04 для псевдослучайной генерации чисел. Возможности генерации псевдослучайных чисел пакета RandomPkg могут быть использованы самостоятельно для создания настраиваемых псевдослучайных тестов и среды верификации (testbench). На примерах будет показано, как можно использовать псевдослучайную генерацию без использования покрытия. Следует отметить, что псевдослучайные тесты сами по себе, т.е. без сбора информации о покрытии, являются малополезными, так как неизвестно, что было проверено с их помощью. Функции интеллектуального покрытия [2] пакета CoveragePkg используют функции пакета RandomPkg, который следует рассмотреть в первую очередь, а уже затем – функции пакета CoveragePkg, предназначенные для оценки покрытия функциональности. Предполагается, что читатели знакомы с материалом статей [1, 2].

Упрощённая генерация случайных чисел

В пакете RandomPkg декларируется защищённый тип RandomPType, который включает в себя начальное значение (seed) псевдослучайного генератора и набор функций для генерации случайных чисел в различных форматах и диапазонах. Функции защищённого типа RandomPType могут не только возвращать псевдослучайное число, но и обновлять значение внутренней переменной, хранящей seed. Генерация псевдослучайных чисел с использованием типа RandomPType (см. листинг 1) проходит в три этапа: декларация переменной данного типа, настройка генератора (в простейшем случае, задание начального значения seed) и получение псевдослучайного числа.

Вызов метода (функции или процедуры) защищённого типа включает имя переменной защищённого типа (RV), имя вызываемого метода с указанием фактических параметров (в примере это записано как RV.RandInt(1, 10)).

Задание начального значения (seed) генератора

Переменные, функции и процедуры, объявленные в теле защищённого типа, являются закрытыми и доступны только через общие методы, которые объявляются в декларативной части защищённого типа. Для хранения начального значения генератора псевдослучайных чисел в теле защищённого типа RandomPType декларируется переменная RandomSeed типа RandomSeedType, который представляет собой массив из двух элементов типа integer:

```
type RandomSeedType is
 array (1 to 2) of integer;
 . . .
variable RandomSeed :
 RandomSeedType :=
 GenRandSeed(integer_vector'(1,7));
```

Пакет RandomPkg опирается на два вспомогательных пакета RandomBasePkg и SortListPkg_int. Функция GenRandSeed – это перегруженная функция, описанная в пакете RandomBasePkg, которая преобразует integer_vector, integer или string в тип RandomSeedType. По умолчанию, переменной RandomSeed назначается значение (1, 7), таким образом, генератор псевдослучайных значений готов к работе сразу после декларации переменной типа RandomPType. Но если в тестирующей программе необходимо несколько генераторов (т.е. несколько переменных типа RandomPType), причем таких, чтобы каждый из генераторов выдавал различные последовательности псевдослучайных чисел, то необходимо задать свои значения seed каждому генератору, т.е. инициализировать начальное значение seed для каждой переменной.

Для задания начального значения seed в защищённом типе RandomPType используется метод InitSeed (представляющий собой VHDL-процедуру), который преобразует значение своего аргумента к типу RandomSeedType и назначает это значение внутренней переменной RandomSeed. Метод InitSeed является перегруженным, чтобы принимать значения типа integer, integer_vector или string:

```
procedure InitSeed(I : integer);
procedure InitSeed(IV: integer_
vector);
procedure InitSeed(S : string);
```

Рекомендуется задавать уникальные значения seed с помощью строковой константы, возвращаемой атрибутом instance_name:

```
RV.InitSeed(RV'instance_name);
```

В тестирующей программе, выполняемой длительное время, можно периодически читать текущее значение seed (значение внутренней переменной RandomSeed) и выводить его в файл или в окно сообщений. Если во время выполнения программы произойдёт сбой (ошибка), то значение seed, которое было записано перед сбоем, может быть легко восстановлено. Это ускорит отладку, т.к. не понадобится повторно проводить моделирование с начала, а можно будет начать с места последнего сохранённого значения seed.

Методы GetSeed и SetSeed используются для чтения и задания значения seed: метод GetSeed возвращает значение типа RandomSeedType, а метод SetSeed берёт значение типа RandomSeedType и назначает его внутренней переменной RandomSeed, хранящей seed. Декларации этих методов приведены ниже.

```
procedure SetSeed(RandomSeedIn:
 RandomSeedType);
impure function GetSeed
 return RandomSeedType;
```

В пакете RandomBasePkg описаны процедуры write и read, позволяющие читать и записывать значение типа RandomSeedType, а также функция to_string для преобразования значения типа RandomSeedType в тип string. Эти подпрограммы не входят в защищённый тип RandomPType. Ниже будет приведён список функций и процедур ввода-вывода.

В защищённом типе RandomPType объявлены функция и процедура SeedRandom, которые являются аналогами функции GetSeed и процедуры SetSeed:

```
procedure SeedRandom
 (RandomSeedIn: RandomSeedType);
impure function SeedRandom
 return RandomSeedType;
```

Генератор псевдослучайных чисел пакета RandomPkg

В защищённом типе RandomPType базовый генератор псевдослучайных чисел (метод RandInt, см. табл. 1[1]) генерирует значение типа integer, которое либо входит в какой-то диапазон, либо в набор значений – чисел. Набор генерируемых чисел и набор исключаемых при генерации чисел имеют тип integer_vector (введённый в язык стандартом VHDL-2008).

Примеры использования базовых функций генерации псевдослучайных чисел приведены в листинге 2.

Когда задаётся массив типа integer_vector, то дополнительные круглые скобки означают, что это массив (набор) чисел, например,

```
D := RV.RandInt( (1,3,7,9) );
```

Листинг 2

```
RndGenProc : process
-- защищённый тип из RandomPkg
 variable RV : RandomPType ;
 variable D : integer ;
begin
-- Задание значения seeds
RV.InitSeed(RV'instance_name);
-- Получение значения
-- из диапазона [0, 255]
D := RV.RandInt(0, 255);
 . . .
-- Получение значения в
-- диапазоне [1, 9], исключая
-- значения 2,4,6,8
D := RV.RandInt(1,9,(2,4,6,8));
 . . .
-- Получение значения из
-- набора чисел 1,3,7,9.
D := RV.RandInt( (1,3,7,9) );
 . . .
-- Получение значения из
-- набора 1,3,7,9, исключая
-- значения 3,7
D:=RV.RandInt((1,3,7,9),(3,7));
```

Внутренняя пара скобок относится к заданию набора чисел. Внешняя пара скобок требуется, так как набор чисел (1, 3, 7, 9) является аргументом вызываемой функции.

Функции генерации доступны не только для целых чисел, но и для векторных типов std_logic_vector (RandSlv), unsigned (RandUnsigned) и signed (RandSigned). При этом значения параметров по-прежнему задаются как целые числа (integer), но при вызове этих функций следует указать дополнительный параметр – разрядность генерируемого вектора. Например, следующий вызов метода RandSlv задаёт требование генерации числа, представленного массивом – восьмиразрядным вектором типа std_logic_vector:

```
. . .
variable DataSlv :
 std_logic_vector(7 downto 0);
begin
. . .
-- Получение вектора
-- из диапазона [0,255]
DataSlv := RV.RandSlv(0,255,8);
```

Для генерации псевдослучайных чисел вещественного типа real из диапазона (Min, Max) в пакете RandomPkg имеется функция RandReal. Так же как

[1]*Указанные в статье таблицы можно скачать с сайта журнала (раздел «Дополнительные материалы к статье»)*

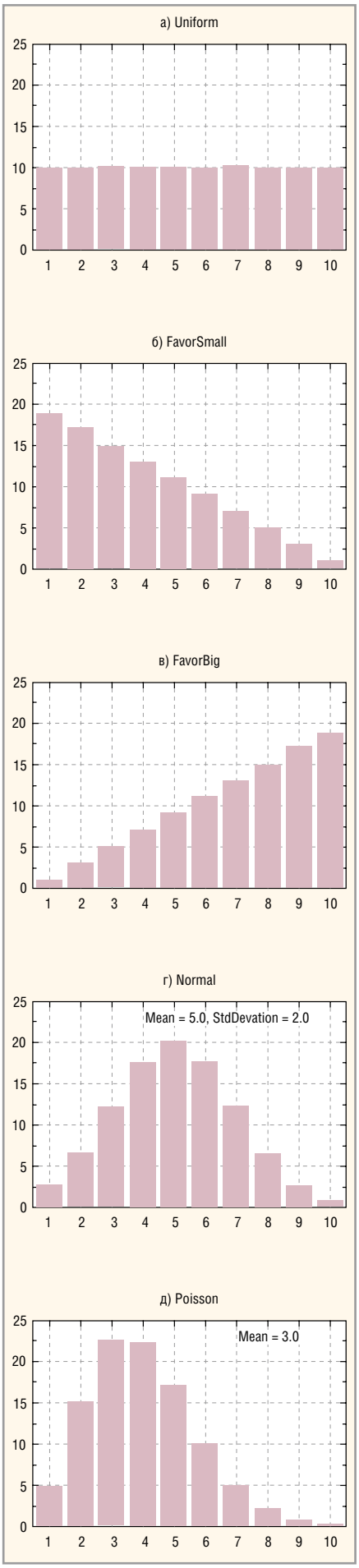

Рис. Законы распределения для разных функций генерации псевдослучайных чисел

и процедура IEEE.math_real.uniform, она никогда не генерирует свои граничные значения Min, Max. Функция RandReal, вызванная без параметров, возвратит вещественное число из диапазона (0, 1).

В пакете RandomPkg описан также ряд функций генерации псевдослучайных чисел, имеющих упрощённый вызов (сокращённый набор указываемых при вызове входных аргументов). Функция RandInt, вызываемая с одним целым параметром (Max), возвращает целое число из диапазона [0, Max]. Если при вызове функции, возвращающей псевдослучайный вектор (методы RandSlv, RandUnsigned, RandSigned), передать два целочисленных параметра (Max, Size), то будет получен вектор размерности Size, псевдослучайное значение которого лежит в диапазоне [0, Max]. Если при вызове этих функций задать только один параметр Size, то функции RandSlv, RandUnsigned возвратят псевдослучайный вектор, значение которого лежит в диапазоне [0, $2^{Size}-1$], а функция RandSigned – возвратит вектор из диапазона [-2^{Size-1}, $2^{Size-1}-1$].

В таблице 1 представлен список перегруженных базовых функций генерации псевдослучайных чисел, заданных в защищённом типе RandomPType. Символ «–» означает, что функция не имеет соответствующего параметра. Если же функция имеет параметр, то для него в таблице указывается тип данных. Через Min, Max обозначены границы диапазона генерируемых значений чисел; A – набор чисел, из которых случайным образом выбирается число; Exclude – набор чисел, исключаемых из диапазона генерируемых псевдослучайных чисел. В столбце Return указан тип значения, возвращаемого функцией. В таблице 1 (и других таблицах) приняты следующие сокращения названий типов: int соответствует типу integer; iv – integer_vector; nat – natural; sig – signed; usig – unsigned; slv – std_logic_vector.

Задание закона распределения псевдослучайных чисел

По умолчанию, функции Rand* (см. табл. 1) возвращают случайные числа, подчиняющиеся равномерному закону распределения (реализованному с помощью процедуры uniform пакета math_real из библиотеки IEEE). Через * в именах Rand* обозначено одно из пяти возможных окончаний, входящих в полное имя функции – Real, Int, Slv, Unsigned, Signed, которые обозначают тип данных возвращаемого числа real, integer, std_logic_vector, unsigned и signed соответственно.

Равномерный закон распределения можно изменить. В защищённом типе RandomPType для хранения названия текущего закона распределения и его параметров используется внутренняя переменная RandomParm, имеющая тип RandomParmType, представляющий собой запись с полями: Distribution, Mean, StdDeviation:

```
type RandomParmType is record
 Distribution : RandomDistType;
 Mean : Real;
 StdDeviation : Real;
end record ;
```

Поле Distribution задаёт название распределения и имеет тип RandomDistType – перечислимый тип со следующими возможными значениями: NONE (начальное значение), UNIFORM (равномерный закон распределения), FAVOR_SMALL (распределение с преобладанием малых значений), FAVOR_BIG (распределение с преобладанием больших значений), NORMAL (нормальный закон распределения, закон Гаусса), POISSON (распределение Пуассона). Примеры таких распределений будут приведены ниже. Декларация типа RandomDistType:

```
type RandomDistType is (NONE,
 UNIFORM, FAVOR_SMALL,
 FAVOR_BIG, NORMAL, POISSON);
```

Значение NONE соответствует равномерному закону распределения, как и UNIFORM, но позволяет проверить, было ли текущее распределение значением по умолчанию (т.е. NONE) или было изменено в ходе выполнения тестирующей программы.

Значение поля Mean в записи RandomParmType используется, если поле Distribution задано равным NORMAL или POISSON, а значение поля StdDeviation – только при Distribution = = NORMAL.

Задать и прочитать значение внутренней переменной RandomParm можно с помощью перегруженных методов SetRandomParm и GetRandomParm. Процедура SetRandomParm имеет два интерфейса. В первом процедуре передаётся один параметр типа

RandomParmType, во втором – три параметра: Distribution (тип RandomDistType), Mean и Deviation (равные по умолчанию 0.0 – вещественный ноль). Функция GetRandomParm также имеет два интерфейса и возвращает значение типа RandomParmType или RandomDistType, например,

```
RV.SetRandomParm(NORMAL, 5.0, 2.0);
RV.SetRandomParm(FAVOR_SMALL);
```

Изменить закон распределения для функций Rand* также можно с помощью вызова метода SetRandomMode, входным параметром для которого является значение типа RandomDistType. Этот метод обеспечивает совместимость с предыдущими версиями пакета. Например, чтобы задать распределение с преобладанием малых значений, необходимо выполнить следующую команду

```
RV.SetRandomMode(FAVOR_SMALL);
```

В защищённом типе RandomPType есть несколько перегруженных функций Uniform, FavorSmall, FavorBig, Normal, Poisson (см. табл. 2), которые возвращают псевдослучайные значения, подчиняющихся соответствующим законам распределения, независимо от значения внутренней переменной RandomParm (т.е. закона распределения, заданного по умолчанию). На рисунке показаны графики распределения вероятностей появления случайных чисел для этих функций, построенные для 100 000 случайных чисел. Ниже приводятся соответствующие вызовы функций:

```
X := RV.Uniform(min=>1, max=>10);
X := RV.FavorSmall(
 min => 1, max => 10);
X := RV.FavorBig(min =>1,max =>10);
X := RV.Normal(Mean => 5.0,
 StdDeviation => 2.0, Min => 1,
 max => 10, Exclude => NULL_INTV);
X := RV.Poisson(Mean=>1.0, min=>1,
 max => 10, Exclude => NULL_INTV);
```

В двух последних примерах значением параметра Exclude является пустой массив, для задания которого используется константа NULL_INTV, которая объявлена в декларативной части пакета RandomPkg следующим образом:

```
constant NULL_INTV :
 integer_vector (0 downto 1) :=
 (others => 0);
```

При задании закона распределения NORMAL (с использованием процедуры SetRandomParm) или вызове функции Normal следует учитывать, что значение StdDeviation не должно быть меньше нуля. При этом значение параметра Mean соответствует медиане (среднему значению), а значение StdDeviation – дисперсии. Для распределения Пуассона (функция Poisson) значение параметра Mean должно быть больше нуля, но при этом не может быть слишком большим (не более 709). Функция Poisson возвращает целые значения, бóльшие нуля.

Следует обратить внимание, что в более ранних версиях пакета RandomPkg функции FavorSmall и FavorBig назывались Favor_small и Favor_big.

Взвешенная генерация псевдослучайных чисел

Кроме стандартных законов распределения случайной величины в пакете существует возможность взвешенной генерации – так называют [3] генера-

цию по произвольному закону распределения с помощью указания списка требуемых значений целых чисел и их вероятностей появления (весов). Для этого в пакете RandomPkg введены следующие типы данных:

DistRecType – запись с двумя полями Value и Weight типа integer;

DistType – массив (неограниченного диапазона) элементов типа DistRecType.

```
type DistRecType is record
 Value : integer ;
 Weight : integer ;
end record ;
type DistType is
 array (natural range <>)
 of DistRecType ;
```

Для взвешенной генерации псевдослучайных чисел в типе RandomPType есть две группы перегруженных функций DistVal* и Dist* (см. табл. 3), здесь, через * обозначено одно из четырёх возможных окончаний, входящих в полное имя функции. При вызове функций группы DistVal* в качестве параметра задаётся массив пар чисел (число, вес), представляющий собой описанный выше тип DistType. При вызове функций из другой группы Dist* задаётся массив (integer_vector) весов. Например, функция DistValInt вызывается:

```
Data := RV.DistValInt(
  ((1, 7), (3, 2), (5, 1)) );
```

с массивом пар значений. Первый элемент в паре это число, а второй – его вес. Частота, с которой каждое значение будет возникать, зависит от вероятности, которая определяется по формуле [вес/(сумму всех весов)]. В приведённом ниже примере в результате многократного вызова метода DistValInt появление числа 1 будет с вероятностью 7/10 или 70%, числа 3 – 20%, а числа 5 – 10%.

Если в функции DistValSlv или DistValUnsigned задать в качестве значений отрицательные числа, то эти функции будут возвращать эти числа по абсолютному значению, т.е. положительные числа.

Функция DistInt является упрощённой версией DistValInt, в которой задаются только веса. Числа генерируются в диапазоне от 0 до N-1, где N – это количество заданных весов. Например, результат вызова:

```
Data := RV.DistInt((7,2,1));
```

функции DistInt будет следующим: вероятность выпадения числа 0 будет 70%, числа 1 – 20%, а числа 2 – 10%.

Функция DistInt имеет следующую особенность: если задать веса с помощью переменной (Weight) типа integer_vector с произвольным диапазоном индексов, то функция будет возвращать значения, соответствующие индексам [Weight'low, Weight'high], а не в диапазоне [0, N-1]:

```
signal wv : integer_vector(5 to 7)
 := (7,2,1);
...
int <= RV.DistInt(wv);
```

В данном примере функция DistInt возвратит числа из диапазона [5, 7], а не [0, 2]. В таблице 3 параметр Weight задаёт массив весов; A – массив пар (генерируемое число, вес), N – число элементов массива Weight.

Функции и процедуры ввода-вывода

Ввод-вывод текстовых файлов в VHDL осуществляется средствами пакетов Textio, Std_logic_Textio. В пакете RandomPkg имеются функции и процедуры, позволяющие осуществлять ввод-вывод для типов RandomSeedType, RandomDistType, RandomParmType:

- to_string – функция преобразования значения любого из трёх типов в строку;
- read – процедура чтения из строки (line) значения заданного типа (RandomSeedType, RandomDistType или RandomParmType);
- write – процедура записи в строку (line) значения заданного типа (RandomSeedType, RandomDistType или RandomParmType).

В таблице 4 используются следующие обозначения: L – строка (тип line) символов, читаемых либо записываемых соответствующей процедурой; A – преобразуемый тип; good – имя выходного параметра для признака успешного завершения процедур чтения; Return – тип значения, возвращаемого функцией. Для процедуры write параметр A является входным параметром, для процедуры read – выходным.

Стабильность генерации

Под стабильностью генерации обычно понимается возможность повторно запустить программу и получить точно такую же последовательность псевдослучайных чисел. Стабильность генерации необходима для верификации, так как если будет найдена ошибка в верифицируемом блоке, то после её исправления требуется повторить ту же последовательность чисел, чтобы проверить исправления.

Для обеспечения стабильности рекомендуется декларировать переменные типа RandomPType как локальные переменные процессов. Тогда данная переменная будет доступна только в этом процессе. Стабильность генерации теряется, если переменная рандомизации объявляется как общая (shared) в архитектуре и совместно используется несколькими процессами. Если переменная рандомизации является общей – значение seed также является общим. При каждой генерации псевдослучайного числа значение seed считывается и обновляется. Если процессы одновременно имеют доступ к общей переменной в течение одного и того же дельта-цикла, то генерация случайного значения зависит от порядка доступа к защищённому типу RandomPType. Этот порядок может измениться после исправления ошибки, и тестовая псевдослучайная последовательность будет другой.

Таким образом, для обеспечения стабильности генерации необходимо создавать отдельную переменную рандомизации в каждом процессе, где необходимо использовать псевдослучайные числа.

Примеры

Пример 1. Поскольку каждый результат генератора псевдослучайных чисел возвращается функцией, то результат может быть использован непосредственно в выражении, что значительно повышает удобство пользования данными функциями. Далее во всех примерах переменная RV имеет тип RandomPType.

При генерации сигнала можно рандомизировать задержки единичных и нулевых значений сигнала:

```
p1: process is
 variable RV : RandomPType;
begin -- process p1
 strob <= '1';
 wait for 10 ns;
 strob <= '0';
 wait for
 RV.RandReal(3.0, 20.0) * 10 ns;
end process p1;
```

В данном случае формируется последовательность импульсов длительностью 10 нс, пауза между импульсами задаётся в диапазоне от 30 до 200 нс. В следующем примере формируется

синхросигнал с периодом (заданным константой Period) и с переменной скважностью, изменяющейся в диапазоне от 0,4 до 0,6:

```
p2: process is
 variable RV : RandomPType;
 variable ClkWidth : time;
begin -- process p1
 ClkWidth :=
 Period * RV.RandReal(0.4, 0.6);
 clk <= '1';
 wait for ClkWidth;
 clk <= '0';
 wait for Period - ClkWidth;
end process p2;
```

С другими примерами можно ознакомиться на сайте www.soel.ru (в разделе «Дополнительные материалы к статье»).

Заключение

Тестирующие программы, разработанные по методологии OS-VVM [4], используют для функционального покрытия поочерёдный вызов подпрограмм как псевдослучайной генерации чисел из пакета RandomPkg, так и подпрограмм из пакета CoveragePkg. Каждая тестовая последовательность получается путём псевдослучайного выбора или ветви кода, или значений для операций. Управляемая рандомизация создаётся с использованием обычной техники VHDL-кодирования (например, последовательные операторы case, if, циклы и операторы присваивания). Этот подход прост и эффективен, так как псевдослучайные последовательности легко могут смешиваться с прямыми и алгоритмическими последовательностями.

Литература

1. *Авдеев Н.А., Бибило П.Н.* Средства VHDL для функциональной верификации цифровых систем. Современная электроника. 2013. № 3. С. 74–76.
2. *Авдеев Н.А., Бибило П.Н.* Средства VHDL для функциональной верификации цифровых систем. Методология OS-VVM. Современная электроника. 2013. № 5. С. 66–70.
3. *Хаханов В.И., Хаханова И.В., Литвинова Е.И., Гузь О.А.* Проектирование и верификация цифровых систем на кристаллах. Verilog & SystemVerilog, Харьков. ХНУРЭ. 2010.
4. Open source VHDL verification methodology. User's Guide Rev. 1.2 (2012–01–05), http://osvvm.org/downloads