Средства VHDL для функциональной верификации цифровых систем

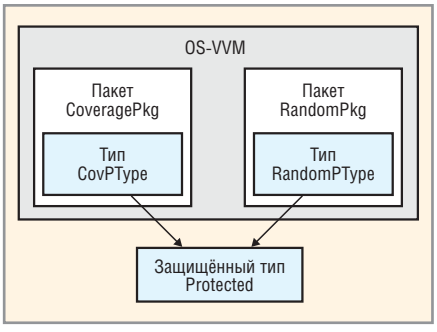
Николай Авдеев, Пётр Бибило (г. Минск, Белоруссия)

В статье описан тип protected, облегчающий написание тестирующих программ, и приведены примеры его использования в языке VHDL.

Введение

Язык VHDL давно утвердился в качестве одного из двух основных языков проектирования цифровых систем (наряду с Verilog). Потребности практики и развитие систем автоматизированного проектирования постепенно дополняли и усложняли конструкции языка VHDL, который изначально предназначался для спецификации и моделирования цифровых систем и схем. Стандарты VHDL'87, VHDL'93 были ориентированы на моделирование, а затем приспособлены и для синтеза логических схем по VHDL-описаниям проектируемых систем. Для потребностей синтеза пришлось ограничить число конструкций, т.е. выделить синтезируемое подмножество конструкций языка (IEEE Std 1076.6-1999). Ведущие фирмы – производители САПР разработали средства, позволяющие от исходного описания цифровой системы на синтезируемом подмножестве VHDL получать описания топологии кристаллов заказных СБИС либо конфигураций ПЛИС.

В настоящее время одной из важнейших является проблема получения правильной (безошибочной) исходной VHDL-модели, по которой автоматически синтезируются логические схемы, размещаемые на кристалле. Чтобы убедиться в правильности спецификации, VHDL-моделирования оказывается недостаточно в силу большой размерности задачи и невозможности перебора всех последовательностей входных тестовых векторов. Важность моделирования при этом возрастает, поскольку цена ошибки становится неприемлемо большой из-за повторения технологического цикла выпуска кристалла.



Тип *protected* используется в методологии OS-VVM

Технологии верификации исходных описаний проектов бурно развиваются. Предпринимаются попытки разработки языков более высокого уровня абстракции по сравнению с VHDL и Verilog. Но эти языки прочно занимают центральное место в проектировании, и производители САПР вынуждены учитывать данное обстоятельство. Для целей верификации в язык VHDL были добавлены новые средства и конструкции, появились стандарты VHDL'2002 и VHDL'2008. Например, в VHDL'2008 [1] было добавлено 14 ключевых слов из языка PSL (Property Specification Language), предназначенных для верификации с помощью аппарата утверждений (Assertion-Based Verification).

Проблема верификации вызвала появление методологии OS-VVM (Open Source VHDL Verification Methodology) [2], предназначенной для написания «интеллектуальных» тестирующих программ. Методология OS-VVM позволяет реализовать функциональное покрытие и управляемую генерацию псевдослучайных тестов, используемых при верификации функциональных блоков. В отличие от понятия покрытия VHDL-кода (*code coverage*) или покрытия свойств (*property coverage*), в методологии OS-VVM под функциональным покрытием понимается сбор значений переменных и сигналов VHDL-проекта при выполнении моделирования.

Важное место в пакетах *RandomPkg* и *CoveragePkg* [3], специально созданных для поддержки OS-VVM, занимает тип *protected* (см. рисунок), появившийся в стандарте VHDL'2002 и облегчающий написание тестирующих программ для функциональной верификации. Описание типа *protected* (защищённого) и его использование при написании программ VHDL приводится ниже, при этом предполагается, что читатели знакомы с основами языка VHDL.

Описание типа protected

Тип *protected* [1, 4, 5] базируется на концепции, похожей на классы в объектно-ориентированном подходе, применяемом в других языках программирования. Тип *protected* упрощает инкапсуляцию в VHDL-коде, поскольку для него осуществляется:

- объединение элементов данных и операций, которые могут выполняться над данными, в один объект;
- сокрытие деталей реализации типов данных от пользователей.

Пользователи VHDL, которые никогда не работали с защищёнными типами, могут легко представить их как специальную версию типа *record* (запись), который позволяет включать процедуры и функции дополнительно к обычным полям данных. Полное определение типа *protected* состоит из двух частей:

- декларации типа *protected*;
- тела (body) типа *protected*.

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*. В стандарте VHDL'2008 [1] подпрограммы, описанные при декларации типа *protected*, названы методами. Ниже приведён пример декларации защищённого типа COUNTER_TYPE:

```
-- декларация типа
type COUNTER_TYPE is protected
procedure Set ( num : integer);
procedure Inc;
impure function get
return integer;
end protected COUNTER_TYPE;
```

Элементы, объявленные внутри тела типа *protected*, недоступны для использования вне этого типа. Таким об-

разом, единственный способ доступа к этим элементам – использование методов, объявленных при декларации типа. Ограничением для методов является то, что их формальные параметры не могут быть типа *access* или *file*.

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, спецификации и др.

Пример тела:

```
type COUNTER_TYPE is protected
body -- тело типа
variable count : integer := 0;
procedure Set ( num : integer) is
begin
count := num ;
end procedure Set;
procedure Inc is
begin
count := count + 1 ;
end procedure Inc;
impure function Get
return integer is
begin
return count;
end function Get;
end protected body COUNTER_TYPE;
```

Если тип *protected* определён в декларативной части архитектуры, то и декларация, и тело типа должны быть размещены в этой части. Если же этот тип определён в пакете, то декларация типа должна быть размещена в декларативной части пакета, а тело типа – в теле пакета.

Все подпрограммы, определённые в теле типа *protected*, имеют полный доступ к внутренним структурам данных, поэтому они не должны указывать их в качестве формальных параметров. Это значит, что все функции типа *protected*, имеющие доступ к внутренним данным, должны быть объявлены как функции *impure*. Ключевое слово *impure* указывает на функцию с побочным эффектом (побочным эффектом является, например, чтение или запись файла). Для таких функций возвращаемые значения могут быть разными для одних и тех же значений аргументов [5, 6]. Слово *impure* является обязательным, в отличие от слова *pure* для «чистой» функции.

Правила использования типа protected

Только методы, имена которых заданы при декларации типа *protected*, являются видимыми вне данного типа. Всё, что задаётся внутри тела типа *protected*, невидимо снаружи. Однако все имена, объявленные при декларации типа, видны в теле типа *protected*. Это правило аналогично правилу видимости имён, объявленных в декларативной части пакета и в теле пакета.

Только переменные могут быть типа *protected*. Общие (shared) переменные могут быть типа *protected*. Другие переменные также могут быть типа *protected*, но, поскольку они доступны только для одного процесса, в котором они были объявлены, то их использование не имеет особого смысла.

Типы *protected* не могут быть использованы в качестве элементов файлов, элементов составных типов или типов доступа (access types).

Передача значения одной переменной типа *protected* другой перемен-

ной не допускается. Как следствие, переменная типа *protected* не должна иметь присвоения начального значения при декларации, например, недопустимой является следующая декларация:

```
shared variable Cnt1, Cnt2 :
COUNTER_TYPE := 10; -- недопустимо
 ...
Cnt1 := Cnt2; -- недопустимо
```

Аналогичным образом операторы равенства («=») и неравенства («/=») не должны использоваться для типа *protected*:

```
shared variable Cnt1, Cnt2 :
protected_type;
 ...
if (Cnt1 = Cnt2 and Cnt1 /= 5)
then -- недопустимо
```

Функции, объявляемые при декларации защищённого типа, должны объявляться с ключевым словом *impure*.

Как правило, защищённым типом *protected* являются общие переменные, которые объявляются в декларативной части архитектуры и пакетов, но не процессов и подпрограмм. Если добавить ключевое слово *shared* в декларации переменной, то такую переменную называют общей переменной, и к ней разрешается доступ более чем из одного процесса. Общие переменные могут быть продекларированы в тех местах кода VHDL, где не могут быть продекларированы обычные переменные, а именно, в декларации *entity*, в разделах деклараций архитектур, операторов блоков, операторов генерации и пакетах. В отличие от обычных переменных, могут возникать проблемы при использовании общих переменных, например, когда два процесса конкурируют при обновлении общей переменной, это может приводить к непредсказуемому итоговому значению переменной.

Пример использования типа protected

```
package new_types is
type COUNTER_TYPE is protected
procedure Set ( num : integer);
procedure Inc;
impure function get return integer;
end protected COUNTER_TYPE;
end new_types;
```

```
package body new_types is

type COUNTER_TYPE is protected
body -- тело типа
variable count : integer := 0;
procedure Set ( num : integer) is
begin
count := num ;
end procedure Set;
procedure Inc is
begin
count := count + 1 ;
end procedure Inc;
impure function Get return
integer is -- определение функции
begin
return count;
end function Get;
end protected body COUNTER_TYPE;
end new_types;

library work;
use work.new_types.all;

entity count is
end entity;

architecture beh of count is
shared variable Cnt :
counter_type;

begin
p1: process
begin
if Cnt.get = 6 then
-- wait for 0 ns;
Cnt.Set(0);
end if;
wait for 10 ns;
end process p1;

p2: process
begin
Cnt.Inc;
wait for 10 ns;
end process p2;
end architecture;
```

Для того чтобы вызвать методы (функции, процедуры) защищённого типа, следует указать имя переменной и имя метода, разделённые точкой. Например, если переменная объявлена как

```
shared variable Cnt :
COUNTER_TYPE;
```

то она может использоваться следующим образом:

```
if (Cnt.Get = 6) then . .
```

В этом выражении вызывается метод *Get* переменной *Cnt*, который возвращает значение внутренней переменной *count*.

Локальные переменные, объявленные в процессах и подпрограммах, также могут быть типа *protected*. В этом случае общий доступ к данным типа *protected* невозможен, но преимущества инкапсуляции остаются. Заметим, что нельзя продекларировать сигнал типа *protected*.

Следует отметить, что использование общих переменных защищённого типа гарантирует доступ к общим структурам данных только для одного процесса в настоящий момент, но не гарантирует порядок выполнения. Это значит, что состязание по-прежнему возможно, если обновление или чтение общей структуры данных планируется в один и тот же момент моделирования и в дельта-цикле. Наблюдать гонки (различные результаты моделирования) можно путём экспериментов, вводя дельта-задержку (с помощью оператора *wait for 0 ns*;) в процессе *p1* и переставляя в тексте программы процессы *p1*, *p2*.

Защищённые переменные *shared* можно объявлять (по сути глобальными) в другом пакете, не забыв указать ссылку на пакет, где декларируются эти переменные, и использовать эти переменные для программ сбора информации о функциональном покрытии.

Заключение

В методологии OS-VVM защищённые типы, описанные в пакетах RandomPkg и CoveragePkg, используются для того, чтобы, с одной стороны, скрыть от пользователя довольно сложные структуры данных, а с другой – с целью упрощения работы пользователя с подпрограммами и функциями, предназначенными для генерации псевдослучайных чисел и сбора данных для оценки функционального покрытия.

Литература

1. IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-2008.
2. http://osvvm.org/about-os-vvm.
3. http://www.synthworks.com/downloads/.
4. Using Protected Types in VHDL Designs. http://www.aldec.com/en/support/resources/documentation/articles/1179.
5. *Ashenden P.J., Lewis J.* The Designer's Guide to VHDL. Third Ed. Morgan Kaufmann, 2008.
6. *Суворова Е.А., Шейнин Ю.Е.* Проектирование цифровых систем на VHDL. БХВ-Петербург, 2003.